

---

# FreeRTOS

## 사용 설명서



---

## FreeRTOS: 사용 설명서

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

Amazon의 상표 및 브랜드 디자인은 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

프리어토스란 무엇인가요?	1
FreeRTOS 소스 코드 다운로드	1
FreeRTOS 버전 관리	1
FreeRTOS 장기 Support	1
FreeRTOS 확장 유지 관리 계획	2
FreeRTOS 아키텍처	2
FreeRTOS 인증 하드웨어 플랫폼	2
개발 워크플로우	3
추가 리소스	3
FreeRTOS 커널 기초	5
FreeRTOS 커널 스케줄러	5
메모리 관리	5
커널 메모리 할당	5
애플리케이션 메모리 관리	6
작업 간 조정	6
Queues	6
세마포어 및 뮤텍스	7
DTT 알림	7
스트림 버퍼	7
메시지 버퍼	8
대칭 다중 처리 (SMP) 지원	9
FreeRTOS-SMP 커널을 사용하도록 응용 프로그램 수정	9
소프트웨어 타이머	9
저전력 지원	10
FreeRTOSConfig.h	10
FreeRTOS 콘솔	11
FreeRTOS 구성	11
FreeRTOS 구성	11
빠른 연결 워크플로우	12
태그 지정 구성	12
AWS IoT Device SDK for Embedded C	13
FreeRTOS 시작하기	14
FreeRTOS 데모 애플리케이션	14
첫 번째 단계	14
사용자 설정AWS계정 및 권한	15
AWS IoT에 MCU 보드 등록	15
FreeRTOS 다운로드	17
FreeRTOS 데모 구성	18
빠른 Connect 사용한 첫 번째 단계	20
설정AWS계정 및 권한	20
FreeRTOS 다운로드 및 구성 및 MCU 보드 등록AWS IoT	21
문제 해결	21
일반 시작하기 문제 해결 팁	21
터미널 에뮬레이터 설치	22
CMak를 사용하여 프리어토스와 사용	22
사전 조건	23
타사 코드 편집기 및 디버깅 도구를 사용하여 FreeRTOS 애플리케이션 개발	24
CMak를 사용하여 FreeRTOS 빌드	24
개발자 모드 키 프로비저닝	29
소개	29
옵션 #1: AWS IoT에서 프라이빗 키 가져오기	29
옵션 #2: 온보드 프라이빗 키 생성	29
보드별 시작 안내서	31
Cypress CYW943907AEVAL1F 개발 키트	32

Cypress CYW954907AEVAL1F 개발 키트 .....	35
사이프러스 CY8CKIT-064S0S2-4343W .....	38
Windows 시뮬레이터를 사용하는 Microchip ATECC608A 보안 요소 .....	43
Espressif ESP32 시작하기 DevKitC와 ESP-WROVER-KIT .....	46
Espressif ESP32-WROOM-32SE 시작하기 .....	63
Espressif ESP32-S2 시작하기 .....	68
Infineon XMC4800 IoT 연결 키트 .....	75
Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트 .....	79
마벨 MW32xAWS IoT스타터 키트 .....	84
MediaTek MT7697Hx 개발 키트 .....	99
Microchip Curiosity PIC32MZ EF .....	103
Nordic nRF52840-DK .....	107
누보톤 NuMaker-IoT-M487 .....	111
NXP LPC54018 IoT Module .....	118
Renesas Starter Kit+ for RX65N-2MB .....	121
STMicroelectronics STM32L4 Discovery Kit IoT 노드 .....	124
Texas Instruments CC3220SF-LAUNCHXL .....	126
Windows Device Simulator .....	130
자일린스 애브넷 MicroZed 산업용 IoT Kit .....	133
FreeRTOS 다음 단계 .....	138
OTA(Over-the-Air) 업데이트 .....	140
OTA 리소스에 태그 지정 .....	140
OTA 업데이트 사전 조건 .....	140
업데이트를 저장할 Amazon S3 버킷 생성 .....	141
OTA 업데이트 서비스 역할 생성 .....	141
OTA 사용자 정책 생성 .....	143
코드 서명 인증서 생성 .....	144
Code Signing for AWS IoT에 대한 액세스 권한 부여 .....	149
OTA 라이브러리와 함께 FreeRTOS 다운로드 .....	150
MQTT를 사용한 OTA 업데이트 사전 조건 .....	151
HTTP를 사용한 OTA 업데이트 사전 조건 .....	152
OTA 자습서 .....	155
초기 펌웨어 설치 .....	155
펌웨어 버전 업데이트 .....	161
OTA 업데이트 생성(AWS IoT 콘솔) .....	162
AWS CLI를 사용하여 OTA 업데이트 생성 .....	164
OTA 업데이트 관리자 서비스 .....	180
애플리케이션에 OTA 에이전트 통합 .....	181
연결 관리 .....	181
단순 OTA 데모 .....	182
OTA 에이전트 이벤트에 응용 프로그램 콜백 사용 .....	183
OTA 보안 .....	185
Code Signing for AWS IoT .....	185
OTA 문제 해결 .....	186
설정 CloudWatch OTA 업데이트에 대한 로그 .....	186
AWS CloudTrail를 사용하여 AWS IoT OTA API 호출 로깅 .....	189
를 사용하여 CreateOTAUpdate 실패 세부 정보를 가져옵니다.AWS CLI .....	191
AWS CLI를 사용하여 OTA 오류 코드 가져오기 .....	193
여러 디바이스의 OTA 업데이트 문제 해결 .....	194
Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결 .....	194
FreeRTOS .....	195
FreeRTOS 이식 라이브러리 .....	195
FreeRTOS 애플리케이션 라이브러리 .....	196
FreeRTOS 라이브러리 구성 .....	197
백오프 알고리즘 .....	198
소개 .....	198
Bluetooth Low Energy .....	198

개요 .....	198
Architecture .....	199
종속성 및 요구 사항 .....	200
라이브러리 구성 파일 .....	200
최적화 .....	201
사용 제한 .....	201
Initialization(초기화) .....	202
API 참조 .....	203
사용 예 .....	203
이식 .....	205
FreeRTOS Bluetooth 디바이스용 모듈 .....	207
셀룰러 인터페이스 .....	208
소개 .....	208
종속성 및 요구 사항 .....	209
이식 .....	209
메모리 사용 .....	209
시작하기 .....	209
셀룰러 인터페이스 라이브러리를 MCU 플랫폼과 통합 .....	210
공통 I/O .....	210
AWS IoT Device Defender .....	211
소개 .....	211
AWS IoT Greengrass .....	211
개요 .....	211
종속성 및 요구 사항 .....	212
API 참조 .....	212
사용 예 .....	212
코레트TP .....	213
소개 .....	213
코어제이슨 .....	214
소개 .....	214
메모리 사용 .....	214
코렘큐트 .....	214
소개 .....	214
코렘QTT 에이전트 .....	215
소개 .....	215
무선 (OTA) .....	216
소개 .....	216
기능 .....	217
API 참조 .....	218
사용 예 .....	219
이식 .....	219
Memory: .....	219
코렐크CS11 .....	219
개요 .....	219
기능 .....	220
비대칭 암호화 지원 .....	221
이식 .....	222
Memory 사용 .....	222
보안 소켓 .....	223
개요 .....	223
종속성 및 요구 사항 .....	223
기능 .....	224
문제 해결 .....	224
개발자 지원 .....	224
사용 제한 .....	224
Initialization(초기화) .....	224
API 참조 .....	225

사용 예 .....	225
이식 .....	226
AWS IoT 디바이스 샘플 .....	226
소개 .....	226
AWS IoT 작업 .....	227
소개 .....	227
전송 계층 보안 .....	227
Wi-Fi .....	228
개요 .....	228
종속성 및 요구 사항 .....	228
기능 .....	228
구성 .....	229
Initialization(초기화) .....	229
API 참조 .....	230
사용 예 .....	230
이식 .....	231
FreeRTOS 데모 .....	233
FreeRTOS 데모 실행 .....	233
데모 구성 .....	233
Bluetooth Low Energy .....	233
개요 .....	233
사전 조건 .....	234
공통 구성 요소 .....	236
MQTT over Bluetooth Low Energy .....	240
Wi-Fi 프로비저닝 .....	241
일반 속성 서버 .....	243
Microchip Curiosity PIC32MZEF용 부트로더 .....	244
부트로더 상태 .....	245
플래시 디바이스 .....	246
애플리케이션 이미지 구조 .....	246
이미지 헤더 .....	246
이미지 설명자 .....	247
이미지 트레일러 .....	248
부트로더 구성 .....	248
부트로더 빌드 .....	249
AWS IoT Device Defender .....	249
소개 .....	249
기능 .....	250
AWS IoT Greengrass .....	253
Amazon EC2 .....	255
코어호트TP .....	256
CoreHTTP 상호 인증 .....	256
CoreHTTP Amazon S3 업로드 .....	257
코어HTTP S3 다운로드 .....	259
CoreHttp 멀티스레드 .....	261
AWS IoT 작업 .....	264
소개 .....	264
소스 코드 구성 .....	264
구성 AWS IoTMQTT 브로커 연결 .....	264
기능 .....	264
코렐쿼트 .....	265
CoreMQTT 상호 인증 .....	266
코렐 MQTT 에이전트 연결 공유 .....	269
OTA(Over-the-Air) 업데이트 .....	271
OTA (무선) 데모 구성 .....	274
Texas Instruments CC3220SF-LAUNCHXL .....	274
Microchip Curiosity PIC32MZEF .....	276

Espressif ESP32 .....	280
르네사스 RX65N .....	281
자습서: BLE를 이용한 에프레시프 ESP32 OTA 업데이트 .....	301
AWS IoT 디바이스 샘플 .....	309
소개 .....	309
기능 .....	309
에 연결하려면 AWS IoTMQTT 브로커 .....	310
샘플 문서 삭제 .....	310
샘플 주제 구독 .....	311
샘플 업데이트 전송 .....	311
샘플 델타 메시지 및 샘플 업데이트 메시지 처리 .....	312
보안 소켓 .....	312
AWS IoT Device TesterFreeRTOS .....	314
FreeRTOS 자격 제품군 .....	314
맞춤형 테스트 제품군 .....	314
FreeRTOS용 IDT의 지원되는 버전 .....	315
FreeRTOS용 IDT의 최신 버전 .....	315
이전 IDT 버전 .....	315
지원되지 않는 IDT 버전 .....	317
용 IDT 다운로드 .....	320
IDT를 수동으로 다운로드 .....	321
IDT를 프로그래밍 방식으로 다운로드 .....	321
FreeRTOS 자격 제품군 (FRQ) 과 함께 IDT 사용 .....	324
사전 조건 .....	325
처음으로 마이크로 컨트롤러 보드의 테스트 준비 .....	328
IDT UI를 사용하여 FreeRTOS 자격 제품군 실행 .....	341
Bluetooth Low Energy 테스트 실행 .....	348
FreeRTOS 검증 제품군 실행 .....	351
테스트 제품군 버전 .....	355
결과 및 로그 이해 .....	356
IDT를 사용하여 자체 테스트 스위트를 개발하고 실행하십시오. ....	359
FreeRTOS용 최신 버전의 IDT를 다운로드합니다. ....	359
테스트 제품군 생성 워크플로우 .....	359
자습서: 샘플 IDT 테스트 제품군 빌드 및 실행 .....	360
자습서: 간단한 IDT 테스트 제품군 개발 .....	363
문제 해결 .....	419
디바이스 구성 문제 해결 .....	419
제한 시간 오류 해결 .....	426
셀룰러 기능 및 AWS 요금 .....	426
자격 보고서 생성 정책 .....	426
지원 정책 .....	426
AWS의 보안 .....	428
Identity and Access Management .....	428
대상 .....	428
자격 증명을 통한 인증 .....	429
정책을 사용하여 액세스 관리 .....	430
자세히 알아보기 .....	432
방법 AWS IAM으로 작업하는 서비스 .....	432
자격 증명 기반 정책 예제 .....	434
문제 해결 .....	436
규정 준수 검증 .....	438
복원성 .....	439
인프라 보안 .....	439
아카이브 .....	440

# 프리어토스란 무엇인가요?

15년간 세계 유수의 칩 회사와 협력하여 개발되었으며, 현재 170초마다 다운로드되는 FreeRTOS는 마이크로컨트롤러 및 소형 마이크로프로세서를 위한 시장을 선도하는 실시간 운영 체제입니다. MIT 오픈 소스 라이선스에 따라 자유롭게 배포되는 FreeRTOS에는 커널과 모든 산업 분야에서 사용하기에 적합한 라이브러리가 포함되어 있습니다. FreeRTOS는 신뢰성과 사용 편의성에 중점을 두고 제작되었습니다.

FreeRTOS에는 연결, 보안 및 OTA (무선) 업데이트용 라이브러리가 포함되어 있습니다. FreeRTOS에는 FreeRTOS 기능을 보여주는 데모 애플리케이션도 포함되어 있습니다. [작격 보드](#).

FreeRTOS는 오픈 소스 프로젝트입니다. GitHub 사이트에서 소스 코드를 다운로드하거나 변경 사항 또는 기능 향상에 기여하거나 문제를 보고할 수 있습니다. <https://github.com/FreeRTOS/FreeRTOS>.

MIT 오픈 소스 라이선스에 따라 FreeRTOS 코드를 릴리스하므로, 상용 및 개인 프로젝트에서 코드를 사용할 수 있습니다.

또한 FreeRTOS 문서에 대한 기여도 환영합니다 (FreeRTOS 사용 설명서, FreeRTOS 이식 안내서, 및 FreeRTOS 검증 안내서). 설명서에 대한 마크다운 소스를 보려면 단원을 참조하십시오. <https://github.com/awsdocs/aws-freertos-docs>. 이는 크리에이티브 커먼즈 (CC BY-ND) 라이선스에 따라 릴리스되었습니다.

## FreeRTOS 소스 코드 다운로드

다운로드 페이지에서 최신 FreeRTOS 및 장기 Support (LTS) 패키지를 다운로드하십시오. [프리어토스 조직](#).

## FreeRTOS 버전 관리

개별 라이브러리는 의미론적 버전 관리와 유사한 x.y.z 스타일 버전 번호를 사용합니다. x는 주 버전 번호이고 y는 부 버전 번호이며 2022부터부터 z는 패치 번호입니다. 2022년 이전에 z는 포인트 릴리스 번호였으며, 첫 번째 LTS 라이브러리에는 양식의 패치 번호가 있어야 했습니다.“x.y.z LTS 패치 2”.

라이브러리 패키지는 yyyyymm.x 스타일 날짜 스템프 버전 번호를 사용합니다. yyyy는 월(년), mm, x는 해당 월의 릴리스 순서를 나타내는 선택적 시퀀스 번호입니다. LTS 패키지의 경우 x는 해당 LTS 릴리스에 대한 순차적 패치 번호입니다. 패키지에 포함된 개별 라이브러리는 해당 라이브러리의 최신 버전이 해당 날짜에 있었던 것입니다. LTS 패키지의 경우 해당 날짜에 LTS 버전으로 원래 릴리스된 LTS 라이브러리의 최신 패치 버전입니다.

## FreeRTOS 장기 Support

FreeRTOS LTS (장기 Support) 릴리스는 출시 후 최소 2년 동안 보안 및 중요 버그 수정 (필요한 경우) 을 받습니다. 이러한 지속적인 유지 관리를 통해 새로운 주요 버전의 FreeRTOS 라이브러리로 업데이트하는데 드는 중단 없이 개발 및 배포 주기 전반에 걸쳐 버그 수정을 통합할 수 있습니다.

FreeRTOS LTS를 사용하면 안전한 연결된 IoT 및 임베디드 제품을 구축하는 데 필요한 완벽한 라이브러리 세트를 얻을 수 있습니다. LTS는 이미 프로덕션 중인 디바이스의 라이브러리 업데이트와 관련된 유지보수 및 테스트 비용을 줄이는 데 도움이 됩니다.

FreeRTOS LTS에는 FreeRTOS 커널 및 IoT 라이브러리가 포함되어 있습니다 프리어토스+TCP, 코럼큐트, 코레트TP, 코렐KCS11, 코어제이슨, AWS IoTOTA, AWS IoT작업, AWS IoT Device Defender, 및 AWS IoTDevice Shadow. 자세한 내용은 FreeRTOS 단원을 참조하십시오. [LTS 라이브러리](#).

## FreeRTOS 확장 유지 관리 계획

AWS 또한 최대 10년 동안 선택한 FreeRTOS 장기 Support (LTS) 버전에서 보안 패치와 중요 버그 수정을 제공하는 EMP (확장 유지 관리 계획)도 제공합니다. FreeRTOS EMP를 사용하면 FreeRTOS 기반의 수명이 긴 기기는 기능 안정성과 수년간 보안 업데이트를 수신하는 버전을 사용할 수 있습니다. FreeRTOS 라이브러리에서 예정된 패치에 대한 알림을 적시에 수신하므로 IoT (사물 인터넷) 디바이스에 보안 패치 배포를 계획할 수 있습니다.

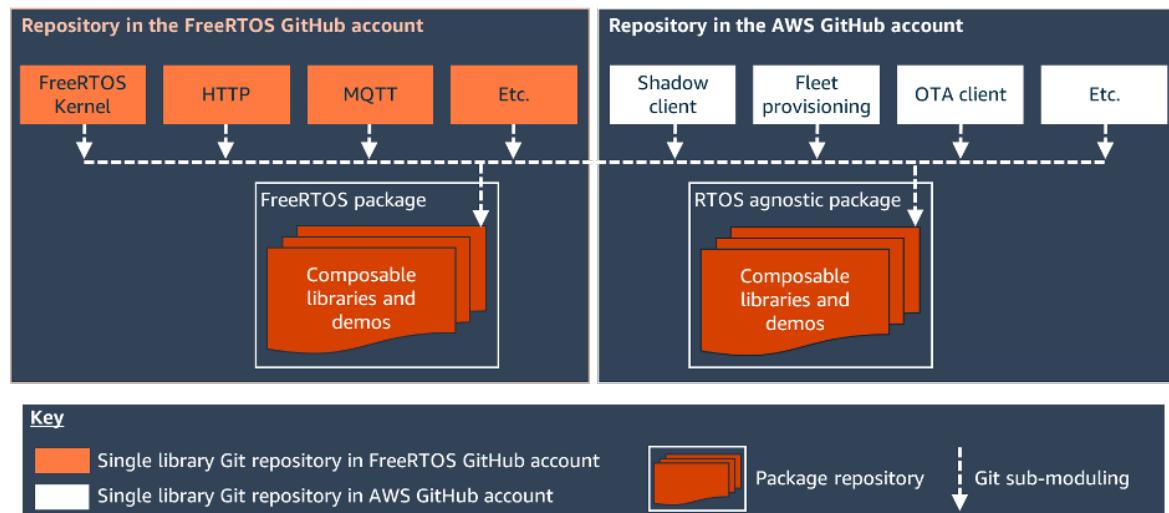
FreeRTOS EMP에 대한 자세한 내용은 단원을 참조하십시오. [기능 페이지](#).

## FreeRTOS 아키텍처

FreeRTOS에는 단일 라이브러리 리포지토리와 패키지 저장소의 두 가지 유형의 저장소가 포함되어 있습니다. 각 단일 라이브러리 저장소에는 빌드 프로젝트나 예제 없이 하나의 라이브러리에 대한 소스 코드가 포함되어 있습니다. 패키지 저장소에는 여러 라이브러리가 포함되어 있으며 라이브러리의 사용을 보여주는 미리 구성된 프로젝트를 포함할 수 있습니다.

패키지 저장소에는 여러 라이브러리가 포함되어 있지만 이러한 라이브러리의 복사본은 포함하지 않습니다. 대신 패키지 리포지토리는 포함된 라이브러리를 git 하위 모듈로 참조합니다. 서브 모듈을 사용하면 각 개별 라이브러리에 대해 단일 소스 소스를 확보할 수 있습니다.

개별 라이브러리 git 리포지토리는 두 GitHub 조직 간에 분할됩니다. FreeRTOS 특정 라이브러리 (예: Freertos+TCP) 또는 일반 라이브러리 (예: 모든 MQTT 브로커와 호환되므로 클라우드에 구애받지 않는 CoreMQTT)가 포함된 리포지토리는 FreeRTOS GitHub 조직에 있습니다. 다음을 포함하는 리포지토리 AWS IoT 특정 라이브러리 (예: AWS IoT 무선 업데이트 클라이언트)에 있습니다. AWS GitHub 조직. 다음 다이어그램에서는 구조에 대해 설명합니다.



## FreeRTOS 인증 하드웨어 플랫폼

FreeRTOS에 대해 인증되는 하드웨어 플랫폼은 다음과 같습니다.

- ATECC608A 제로 타ッチ 프로비저닝 키트 AWS IoT
- Cypress CYW943907AEVAL1F 개발 키트
- Cypress CYW954907AEVAL1F 개발 키트

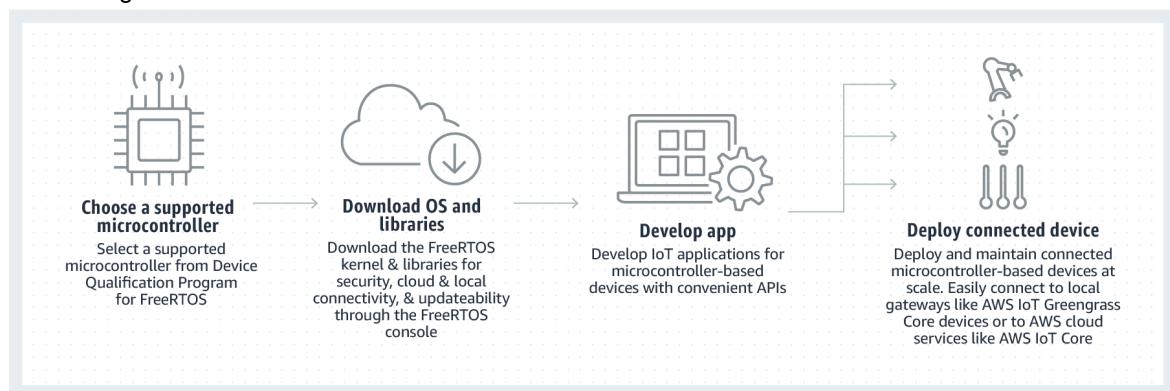
- 사이프러스 CY8CKIT-064S0S2-4343W 키트
- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- 에스프레시프 에스P32-S2-사울라-1
- Infineon XMC4800 IoT 연결 키트
- Marvell MW320 AWS IoT 시작 키트
- Marvell MW322 AWS IoT 시작 키트
- MediaTek MT7697Hx 개발 키트
- Microchip Curiosity PIC32MZEF 번들
- Nordic nRF52840-DK
- NuMaker-IoT-M487
- NXP LPC54018 IoT Module
- OPTIGA Trust X Security Solution
- Renesas RX65N RSK IoT Module
- STMicroelectronics STM32L4 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 Catalog](#)에도 나와 있습니다.

새로운 디바이스 검증에 대한 자세한 내용은 단원을 참조하십시오.[FreeRTOS 검증 안내서](#).

## 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 전체 IoT 솔루션의 한 부분으로 AWS IoT 서비스나 AWS IoT Greengrass에 연결할 수 있습니다.



## 추가 리소스

다음은 유용한 리소스입니다.

- 추가 정보[FreeRTOS 설명서](#)참조,[프리어토스조직](#).
- FreeRTOS 엔지니어링 팀의 FreeRTOS에 대한 질문이 있는 경우 문제를 개설할 수 있습니다.[FreeRTOS GitHub 페이지](#).
- FreeRTOS에 대한 기술적인 질문은[FreeRTOS 커뮤니티 포럼](#).
- 디바이스 연결에 대한 자세한 내용은[AWS IoT](#)참조,[디바이스 프로비저닝](#)의[AWS IoT Core 개발자 안내서](#).
- 에 대한 기술 지원[AWS](#)참조,[AWS 지원 센터](#).
- 에 대한 질문이 있는 경우[AWS 결제](#), 계정 서비스, 이벤트, 침해 또는 기타 문제[AWS 단원](#)을 참조하십시오.[문의처페이지](#).

# FreeRTOS 커널 기초

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)
- 멀티코어 마이크로컨트롤러에서 대칭 멀티프로세싱 (SMP) Support

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 [를 참조하십시오. 프리어토스 조직](#). FreeRTOS.org에서는 [Quick Start Guide](#) 및 상세한 [Mastering the FreeRTOS Real Time Kernel](#)을 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_4`와 유사합니다. 힙이 인접하지 않은 여러 메모리 영역에 걸쳐서 존재할 수 있습니다.

## 작업 간 조정

이 단원에는 FreeRTOS 프리미티브에 대한 정보가 포함되어 있습니다.

## Queues

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 간 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 상

태로 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 저장하는데 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 수행하는데 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.h \(p. 10\)](#)에서 sbSEND\_COMPLETED()의 자체 구현을 제공하여 이 기본 동작을 변경 할 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.h \(p. 10\)](#)에서 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, 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.h \(p. 10\)](#)에서 `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.h \(p. 10\)](#)에서 자체 구현을 제공하여 `sbRECEIVE_COMPLETED()`의 기본 동작을 변경할 수 있습니다.

## 대칭 다중 처리 (SMP) 지원

FreeRTOS 커널에서의 SMP 지원 FreeRTOS 커널의 한 인스턴스에서 여러 개의 동일한 프로세서 코어에서 작업을 예약할 수 있습니다. 핵심 아키텍처는 동일해야 하며 동일한 메모리를 공유해야 합니다.

## FreeRTOS-SMP 커널을 사용하도록 응용 프로그램 설정

FreeRTOS API는 단일 코어와 SMP 버전 간에 실질적으로 동일하게 유지됩니다. [이러한 추가 API](#). 따라서 FreeRTOS 싱글 코어 버전으로 작성된 응용 프로그램은 최소한의 노력 없이 SMP 버전으로 컴파일해야 합니다. 그러나 단일 코어 응용 프로그램에 해당되는 일부 가정은 멀티 코어 응용 프로그램의 경우 더 이상 해당되지 않을 수 있기 때문에 몇 가지 기능적 문제가 있을 수 있습니다.

한 가지 일반적인 가정은 우선 순위가 높은 작업이 실행되는 동안에는 우선 순위가 낮은 작업을 실행할 수 없다는 것입니다. 싱글 코어 시스템에서는 이것이 사실이었지만 다중 작업을 동시에 실행할 수 있기 때문에 멀티 코어 시스템에서는 더 이상 해당되지 않습니다. 응용 프로그램이 상호 제외를 제공하기 위해 상대 작업 우선 순위에 의존하는 경우 멀티 코어 환경에서 예상치 못한 결과가 나타날 수 있습니다.

다른 일반적인 가정은 ISR이 서로 동시에 실행되거나 다른 작업과 함께 실행할 수 없다는 것입니다. 멀티 코어 환경에서는 이것이 더 이상 적용되지 않습니다. 응용 프로그램 작성기는 작업과 ISR 간에 공유되는 데이터에 액세스하는 동안 적절한 상호 제외를 보장해야 합니다.

## 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 타

이미 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.h는 애플리케이션 고유의 파일이며, FreeRTOS 커널 소스 코드 디렉터리 중 하나가 아니라 애플리케이션 디렉터리 아래에 위치해야 합니다.

0|FreeRTOSConfig.hFreeRTOS 데모 및 테스트 애플리케이션용 파일은 [freertos/vendors/vendor/boards/board/aws\\_demos/config\\_files/FreeRTOSConfig.h](#)과 [freertos/vendors/vendor/boards/board/aws\\_tests/config\\_files/FreeRTOSConfig.h](#).

FreeRTOSConfig.h에서 지정할 수 있는 구성 파라미터의 목록은 [FreeRTOS.org](#)를 참조하십시오.

# FreeRTOS 콘솔

에서 [FreeRTOS 콘솔](#), 마이크로 컨트롤러 기반 디바이스용 애플리케이션을 작성하는 데 필요한 모든 것을 담은 패키지를 구성하고 다운로드할 수 있습니다.

- FreeRTOS 커널
- FreeRTOS 라이브러리
- 라이브러리 지원 플랫폼
- 하드웨어 드라이버

미리 정의된 구성을 포함하는 패키지를 다운로드하거나, 해당 애플리케이션에 필요한 하드웨어 플랫폼 및 라이브러리를 선택하여 직접 구성을 만들 수 있습니다. 이러한 구성은 AWS에 저장되며, 언제든지 다운로드할 수 있습니다.

## FreeRTOS 구성

미리 정의된 구성을 사용하면 필요한 라이브러리에 대해 고민할 필요 없이 지원되는 사용 사례를 활용하여 빠르게 시작할 수 있습니다. 미리 정의된 구성을 사용하려면 [FreeRTOS 콘솔](#) 사용할 구성 찾은 다음다운로드.

FreeRTOS 버전, 하드웨어 플랫폼 또는 라이브러리를 변경하려면 미리 정의된 구성을 사용자 지정할 수도 있습니다. 미리 정의된 구성을 사용자 지정하면 FreeRTOS 콘솔에서 미리 정의된 구성을 덮어쓰지 않고 새 사용자 지정 구성이 생성됩니다.

미리 정의된 구성에서 사용자 지정 구성을 생성하려면

1. 해당 페이지로 이동합니다.[FreeRTOS 콘솔](#).
2. 탐색 창에서 소프트웨어를 선택합니다.
3. UnderFreeRTOS 디바이스 소프트웨어, 선택다운로드 구성.
4. 사용자 지정하려는 미리 정의된 구성 옆에 있는 줄임표(...)를 선택한 다음 사용자 지정을 선택합니다.
5. 온FreeRTOS 소프트웨어 구성페이지에서 FreeRTOS 버전, 하드웨어 플랫폼 및 라이브러리를 선택하고 새 구성에 이름을 지정하고 설명을 입력합니다.
6. 페이지 맨 아래에서 생성 후 다운로드를 선택합니다.

## FreeRTOS 구성

사용자 지정 구성을 사용하여 하드웨어 플랫폼, 통합 개발 플랫폼(IDE), 컴파일러 등과 필요한 RTOS 라이브러리만 지정할 수 있습니다. 그러면 디바이스에 애플리케이션 코드를 위한 추가 공간이 확보됩니다.

사용자 지정 구성을 생성하려면

1. 해당 페이지로 이동합니다.[FreeRTOS 콘솔](#) 선택하고 새로 생성.
2. 사용할 FreeRTOS 버전을 선택합니다. 기본적으로 최신 버전이 사용됩니다.
3. 새 소프트웨어 구성 페이지에서 하드웨어 플랫폼 선택을 선택하고 사전 검증된 플랫폼 중 하나를 선택합니다.
4. 사용할 IDE 및 컴파일러를 선택합니다.

5. 필요한 FreeRTOS 라이브러리에 대해 다음을 선택합니다. 라이브러리 추가. 다른 라이브러리를 필요한 라이브러리로 선택한 경우 라이브러리가 자동으로 추가됩니다. 추가 라이브러리를 선택하려면 다른 라이브러리 추가를 선택합니다.
6. 데모 프로젝트 섹션에서 데모 프로젝트 중 하나를 활성화합니다. 그러면 프로젝트 파일에서 데모가 활성화됩니다.
7. 이름 필수에서 사용자 지정 구성에 대한 이름을 입력합니다.

#### Note

사용자 지정 구성 이름에 개인 식별 정보(OTA)를 사용하지 마십시오.

8. 설명에 사용자 지정 구성에 대한 설명을 입력합니다.
9. 페이지 맨 아래에서 생성 후 다운로드를 선택합니다.

#### 사용자 지정 구성 편집하려면

1. 해당 페이지로 이동합니다.[FreeRTOS 콘솔](#).
2. 탐색 창에서 소프트웨어를 선택합니다.
3. UnderFreeRTOS 디바이스 소프트웨어, 선택다운로드 구성.
4. 편집하려는 구성 옆에 있는 줄임표(...)를 선택한 다음 편집을 선택합니다.
5. 온FreeRTOS 소프트웨어 구성페이지에서 구성의 FreeRTOS 버전, 하드웨어 플랫폼, 라이브러리 및 설명을 변경할 수 있습니다.
6. 페이지 맨 아래에서 저장 후 다운로드를 선택합니다.

#### 사용자 지정 구성 삭제하려면

1. 해당 페이지로 이동합니다.[FreeRTOS 콘솔](#).
2. 탐색 창에서 소프트웨어를 선택합니다.
3. UnderFreeRTOS 디바이스 소프트웨어, 선택다운로드 구성.
4. 삭제하려는 구성 옆에 있는 줄임표(...)를 선택한 다음 삭제를 선택합니다.

## 빠른 연결 워크플로우

FreeRTOS 콘솔은 또한 미리 정의된 구성이 있는 모든 보드를 위한 빠른 Connect 워크플로 옵션을 포함합니다. 빠른 Connect 워크플로우는 FreeRTOS 데모 애플리케이션을 구성하고 실행하도록 도와줍니다. AWS IoT 과 AWS IoT Greengrass. 시작하려면 미리 정의된 구성 탭을 선택하고 보드를 찾은 다음 빠른 연결을 선택하고 빠른 연결 워크플로우 단계를 따릅니다.

## 태그 지정 구성

콘솔에서 구성을 생성하거나 편집할 때 FreeRTOS 구성에 태그를 적용할 수 있습니다. 구성에 태그를 적용하려면 콘솔로 이동합니다. 태그 아래에서 태그의 이름과 값을 입력합니다.

태그를 사용하여 관리하는 방법에 대한 자세한 내용은 AWS IoT 리소스, 참조[IAM 정책에 태그 사용](#)의 AWS IoT 개발자 안내서.

# AWS IoT Device SDK for Embedded C

## Note

이 SDK는 숙련된 임베디드 소프트웨어 개발자가 사용하기 위한 것입니다.

AWS IoT Device SDK for Embedded C(C-SDK)는 MIT 오픈 소스 라이선스에 따라 제공되는 C 소스 파일의 모음으로, IoT 장치를 AWS IoT Core에 안전하게 연결하기 위해 임베디드 애플리케이션에서 사용할 수 있습니다. 여기에는 MQTT 클라이언트, HTTP 클라이언트, JSON 파서 및 AWS IoTDevice Shadow, AWS IoT 작업, AWS IoT 플랫폼 브로비저닝 및 AWS IoT Device Defender 라이브러리 이 SKK는 소스 형식으로 배포되며 애플리케이션 코드, 기타 라이브러리 및 원하는 운영 체제 (OS) 와 함께 고객 펌웨어에 구축됩니다.

AWS IoT Device SDK for Embedded C는 일반적으로 최적화된 C 언어 런타임을 필요로 하는 리소스 제약 디바이스를 대상으로 합니다. 모든 운영 체제에서 SDK를 사용하고, 모든 프로세서 유형(예: MCU 및 MPU)에서 호스팅할 수 있습니다. 그러나 장치에 충분한 메모리 및 처리 리소스가 있는 경우 상위 항목 중 하나를 사용하는 것이 좋습니다.[AWS IoTDevice SDK](#).

자세한 내용은 다음 자료를 참조하세요.

- [AWS IoTDevice SDK for Embed](#)
- [AWS IoT Device SDK for Embedded C on GitHub](#)
- [AWS IoT Device SDK for Embedded C Readme](#)
- [AWS IoTDevice SDK for Embedded](#)

# FreeRTOS 시작하기

이 FreeRTOS 시작하기 자습서에서는 호스트 컴퓨터에서 FreeRTOS를 다운로드하고 구성한 후 공인 마이크로 컨트롤러 보드.

이 자습서에서는 사용자가 AWS IoT와 AWS IoT 콘솔을 잘 알고 있는 것으로 가정합니다. 그렇지 않은 경우 [AWS IoT 시작하기](#) 자습서를 완료한 후 진행하는 것이 좋습니다.

주제:

- [FreeRTOS 데모 애플리케이션 \(p. 14\)](#)
- [첫 번째 단계 \(p. 14\)](#)
- [콘솔 빠른 Connect 워크플로우의 첫 번째 단계 \(p. 20\)](#)
- [시작하기 문제 해결 \(p. 21\)](#)
- [CMak를 사용하여 프리어토스와 사용 \(p. 22\)](#)
- [개발자 모드 키 프로비저닝 \(p. 29\)](#)
- [보드별 시작 안내서 \(p. 31\)](#)
- [FreeRTOS 다음 단계 \(p. 138\)](#)

## FreeRTOS 데모 애플리케이션

이 자습서의 데모 애플리케이션은 `freertos/demos/coreMQTT_Agent/mqtt_agent_task.c` 파일입니다. 그것은 [CoreMQTT 라이브러리 \(p. 214\)](#)에 연결하려면 AWS 클라우드에서 호스팅되는 MQTT 주제에 메시지를 주기적으로 게시합니다. [AWS IoT MQTT 브로커](#).

FreeRTOS 데모 애플리케이션은 한 번에 하나만 실행할 수 있습니다. FreeRTOS 데모 프로젝트를 빌드하는 경우 첫 번째 데모가 `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` 헤더 파일은 실행되는 응용 프로그램입니다. 이 자습서에서는 어떤 데모도 활성화 또는 비활성화 할 필요가 없습니다. CoreMQTT 에이전트 데모는 기본적으로 활성화됩니다.

FreeRTOS에 포함된 데모 애플리케이션에 대한 자세한 내용은 단원을 참조하십시오. [FreeRTOS 데모 \(p. 233\)](#).

## 첫 번째 단계

를 사용하여 FreeRTOS 사용을 시작하려면 AWS IoT, 수행해야 할 사항 AWS 계정, 액세스 권한이 있는 IAM 사용자 AWS IoT 및 FreeRTOS 클라우드 서비스를 제공합니다. 또한 FreeRTOS 데모 프로젝트를 와 연동하도록 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 시작하기 DevKitC와 ESP-WROVER-KIT \(p. 46\)](#)로 이동하십시오.
- Nordic nRF52840-DK를 사용하는 경우 이러한 단계들을 건너뛰고 [Nordic nRF52840-DK 시작하기 \(p. 107\)](#)로 이동하십시오.
- 보드를 보드에 빠르게 연결하려면 AWS 클라우드, 당신은 따라갈 수 있습니다. [콘솔 빠른 Connect 워크플로우의 첫 번째 단계 \(p. 20\)](#) 여기에 표시된 단계 대신.

1. 사용자 설정AWS계정 및 권한 (p. 15)
2. AWS IoT에 MCU 보드 등록 (p. 15)
3. FreeRTOS 다운로드 (p. 17)
4. FreeRTOS 데모 구성 (p. 18)

## 사용자 설정AWS계정 및 권한

AWS 계정을 만들려면 [AWS 계정 생성 및 활성화](#)를 참조하십시오.

에 IAM 사용자를 추가하려면 AWS 계정, 참조 [IAM 사용 설명서](#). IAM 사용자 계정에 액세스 권한을 부여하려면 AWS IoT 및 FreeRTOS에서 다음 IAM 정책을 IAM 사용자 계정에 연결합니다.

- [AmazonFreeRTOSFullAccess](#)
- [AWSIoTFullAccess](#)

IAM 사용자에게 AmazonFreeRTOSFullAccess 정책을 IAM 사용자에게 연결하려면

1. 해당 위치로 이동합니다. [IAM 콘솔](#)을 차례로 선택합니다. 탐색 창에서 []를 선택합니다. 사용자.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가(Add permissions)를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 를 입력합니다. [AmazonFreeRTOSFullAccess](#)를 선택합니다. 목록에서 선택한 다음 를 선택합니다. 다음: Review)]를 선택합니다.
6. 권한 추가(Add permissions)를 선택합니다.

IAM 사용자에게 AWSIoTFullAccess 정책을 IAM 사용자에게 연결하려면

1. 해당 위치로 이동합니다. [IAM 콘솔](#)을 차례로 선택합니다. 탐색 창에서 []를 선택합니다. 사용자.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가(Add permissions)를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 를 입력합니다. [AWSIoTFullAccess](#)를 선택합니다. 목록에서 선택한 다음 를 선택합니다. 다음: Review)]를 선택합니다.
6. 권한 추가(Add permissions)를 선택합니다.

IAM 및 사용자 계정에 대한 자세한 내용은 단원을 참조하십시오. [IAM 사용 설명서](#).

정책에 대한 자세한 내용은 단원을 참조하십시오. [IAM 권한 및 정책](#).

를 설정한 후 AWS 계정 및 권한으로 계속 진행합니다. [AWS IoT에 MCU 보드 등록 \(p. 15\)](#) 또는 빠른 연결에서 워크플로 [FreeRTOS 콘솔](#).

## AWS IoT에 MCU 보드 등록

AWS 클라우드와 통신하기 위해서는 보드가 AWS IoT에 등록되어 있어야 합니다. 보드를 에 등록하려면 AWS IoT 수행해야 할 사항:

AWS IoT 정책

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

## AWS IoT 사물

AWS IoT에서 디바이스를 관리하는 데 사용할 수 있는 AWS IoT 사물. 이 정책은 AWS 클라우드에 저장됩니다.

### 프라이빗 키 및 X.509 인증서

디바이스는 프라이빗 키와 인증서를 사용하여 AWS IoT로 인증할 수 있습니다.

보드를 등록하려면 아래 절차를 수행합니다.

### AWS IoT 정책을 생성하려면

1. IAM 정책을 생성하려면 AWS 지역 및 AWS 계정 번호입니다.

를 찾으려면 AWS 계좌 번호, [AWS 관리 콘솔](#) 오른쪽 위 모서리에서 계정 이름 아래의 메뉴를 찾아 확장한 후 오른쪽 위 모서리에서 계정 이름 아래의 메뉴를 찾아 확장한 후를 선택합니다. 내 계정. 계정 ID는 Account Settings(계정 설정) 아래에 표시됩니다.

를 찾으려면 AWS 해당 리전 AWS 계정, 사용 AWS Command Line Interface. AWS CLI를 설치하려면 [AWS Command Line Interface 사용 설명서](#)의 지침을 따릅니다. AWS CLI를 설치한 후, 명령 프롬프트 창을 열고 다음 명령을 입력합니다.

```
aws iot describe-endpoint --endpoint-type=iot:Data-ATS
```

출력은 다음과 같아야 합니다.

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

이 예제에서 리전은 us-west-2입니다.

2. [AWS IoT 콘솔](#)로 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. Replace **AWS ##**과 **AWS ##** 당신과 함께 AWS 리전 및 계정 ID입니다.

```
{  
    "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 ID를 가진 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 콘솔](#)로 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 [Create]를 선택합니다.
4. AWS IoT 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. Attach a policy(정책 연결)을 선택하여 AWS IoT 작업에 대한 액세스 권한을 디바이스에 부여하는 정책을 인증서에 연결합니다.
10. 방금 생성한 정책을 선택하고 Register thing(사물 등록)을 선택합니다.

보드를 AWS IoT에 등록했으면 이어서 [FreeRTOS 다운로드 \(p. 17\)](#)를 수행할 수 있습니다.

## FreeRTOS 다운로드

FreeRTOS 콘솔 또는 에서 FreeRTOS를 다운로드할 수 있습니다.[GitHub 리포지토리](#).

#### 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 콘솔에서 프리어토스를 다운로드하려면

1. 에 로그인합니다.[FreeRTOS 콘솔](#).
2. 미리 정의된 구성에서 AWS IoT- ##### ##을 찾은 후 다운로드를 선택합니다.
3. 한 디렉터리에 다운로드한 파일의 압축을 풀고 디렉터리 경로를 복사합니다.

#### Important

- 이 주제에서 FreeRTOS 다운로드 디렉터리의 경로를 이라고 합니다.[freertos](#).
- [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 디렉터리 경로는 빌드 오류를 발생시킬 수 있습니다.
- 소스 코드에 심볼릭 링크가 포함될 수 있으므로 Windows를 사용하여 아카이브를 추출하는 경우 다음을 수행해야 할 수 있습니다.
  - 활성화 [개발자 모드](#) 또는
  - 관리자 권한으로 상승된 콘솔을 사용합니다.

이러한 방식으로 Windows는 아카이브를 추출할 때 심볼릭 링크를 옮바르게 만들 수 있습니다. 그렇지 않으면 심볼 링크의 경로가 텍스트로 포함되거나 비어 있는 일반 파일로 심볼 링크가 작성됩니다. 자세한 내용은 블로그 항목을 참조하십시오.[윈도우 10의 심볼릭 링크!](#)

Windows에서 Git을 사용하는 경우 개발자 모드를 활성화하거나 다음을 수행해야 합니다.

- Setcore.symlinks 다음 명령을 사용하여 true로 설정합니다.

```
git config --global core.symlinks true
```

- 시스템에 쓰는 git 명령을 사용할 때마다 관리자 권한으로 상승된 콘솔을 사용하십시오 (예:git pull,git clone, 및git submodule update --init --recursive).

FreeRTOS를 다운로드했으면[FreeRTOS 데모 구성 \(p. 18\)](#).

## FreeRTOS 데모 구성

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

### AWS IoT 엔드포인트를 구성하려면

FreeRTOS에 를 제공해야 합니다.AWS IoT엔드포인트를 사용하면 보드에서 실행되는 애플리케이션이 올바른 엔드포인트로 요청을 전송할 수 있습니다.

1. [AWS IoT 콘솔](#)로 이동합니다.
2. 왼쪽 탐색 창에서 설정을 선택합니다.

귀하AWS IoT엔드포인트가 에 표시됩니다.디바이스 데이터 엔드포인트. URL은 [1234567890123-ats.iot.us-east-1.amazonaws.com](#)와 같아야 합니다. 이 엔드포인트를 기록해 둡니다.

3. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다.

디바이스에 AWS IoT 사물 이름이 있어야 합니다. 이 이름을 기록해 둡니다.

4. demos/include/aws\_clientcredential.h을(를) 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 경우에 네트워크에 연결할 Wi-Fi 자격 증명을 FreeRTOS에 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

FreeRTOS는 AWS IoT 등록된 사물과 연결된 인증서 및 프라이빗 키 및 와 성공적으로 통신할 수 있는 권한 정책 및 프라이빗 키 및 프라이빗 키와 AWS IoT 디바이스를 대신하여.

#### Note

를 구성하려면 AWS IoT 자격 증명, 에서 다운로드한 프라이빗 키와 인증서가 있어야 합니다. AWS IoT 기기를 등록했을 때 콘솔을 사용합니다. 디바이스를 AWS IoT 사물로 등록한 후 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 기준 파일을 덮어씁니다.

#### Note

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

FreeRTOS를 구성했으면 해당 보드에 대한 시작 안내서를 참조하여 플랫폼의 하드웨어 및 소프트웨어 개발 환경을 설정한 후 보드에서 데모를 컴파일하고 실행할 수 있습니다. 보드별 지침은 [보드별 시작 안내서 \(p. 31\)](#) 단원을 참조하십시오. 시작하기 자습서에서 사용하는 데모 애플리케이션은 CoreMQTT 상호 인증 데모이며 다음 위치에 있습니다.`demos/coreMQTT/mqtt_demo_mutual_auth.c`.

# 콘솔 빠른 Connect 워크플로우의 첫 번째 단계

## Note

FreeRTOS 콘솔에서는 현재 다음 보드에 대해 FreeRTOS 구성 사용할 수 없습니다.

- Cypress CYW943907AEVAL1F 개발 키트
- Cypress CYW954907AEVAL1F 개발 키트

FreeRTOS 사용을 시작하려면 AWS IoT를 가져야 합니다. AWS 계정 및 액세스 권한이 있는 IAM 사용자 AWS IoT 및 FreeRTOS 클라우드 서비스를 제공합니다. 그런 다음 FreeRTOS 콘솔의 빠른 Connect 워크플로우를 사용하여 FreeRTOS를 다운로드하고, 보드에 대한 FreeRTOS 데모를 구성한 다음 AWS IoT 그래서 그것은 AWS 클라우드. 다음 단원에서는 이러한 요구 사항을 소개합니다.

1. 설정 AWS 계정 및 권한 (p. 20)
2. FreeRTOS 다운로드 및 구성 및 MCU 보드 등록 AWS IoT (p. 21)

## 설정 AWS 계정 및 권한

AWS 계정을 만들려면 [AWS 계정 생성 및 활성화](#)를 참조하십시오.

에 IAM 사용자를 추가하려면 AWS 계정, 참조 [IAM 사용 설명서](#). IAM 사용자 계정에 액세스 권한을 부여하려면 AWS IoT 및 FreeRTOS에서 다음 IAM 정책을 IAM 사용자 계정에 연결합니다.

- `AmazonFreeRTOSFullAccess`
- `AWSIoTFullAccess`

`AmazonFreeRTOSFullAccess` 정책을 IAM 사용자에 연결하려면

1. (영문) [IAM 콘솔](#) 및 탐색 창에서 를 선택합니다. 사용자.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가(Add permissions)를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 를 입력합니다. `AmazonFreeRTOSFullAccess` 목록에서 해당 항목을 선택한 다음 를 선택합니다. 다음: Review]를 선택합니다.
6. 권한 추가(Add permissions)를 선택합니다.

`AWSIoTFullAccess` 정책을 IAM 사용자에 연결하려면

1. (영문) [IAM 콘솔](#) 및 탐색 창에서 를 선택합니다. 사용자.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가(Add permissions)를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 를 입력합니다. `AWSIoTFullAccess` 목록에서 해당 항목을 선택한 다음 를 선택합니다. 다음: Review]를 선택합니다.
6. 권한 추가(Add permissions)를 선택합니다.

IAM 및 사용자 계정에 대한 자세한 내용은 단원을 참조하십시오. [IAM 사용 설명서](#).

정책에 대한 자세한 내용은 단원을 참조하십시오. [IAM 권한 및 정책](#).

# FreeRTOS 다운로드 및 구성 및 MCU 보드 등록AWS IoT

열기[FreeRTOS 콘솔](#)를 선택하고 빠른 연결MCU 보드에 대한 워크플로우입니다. 그러면 FreeRTOS를 다운로드하고, FreeRTOS 데모를 구성하고, 다음과 같이 보드를 등록할 수 있습니다. AWS IoT 그래서 그것은 AWS 클라우드. 빠른 Connect 워크플로우에서는 다음을 생성합니다.

## AWS IoT 정책

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

## AWS IoT 사물

AWS IoT에서 디바이스를 관리하는 데 사용할 수 있는 AWS IoT 사물. 이 정책은 AWS 클라우드에 저장됩니다.

## 프라이빗 키 및 X.509 인증서

디바이스는 프라이빗 키와 인증서를 사용하여 AWS IoT로 인증할 수 있습니다.

FreeRTOS를 구성했으면 해당 보드에 대한 시작 안내서를 참조하여 FreeRTOS 데모를 컴파일하고 실행할 수 있습니다. 보드별 지침은 [보드별 시작 안내서 \(p. 31\)](#) 단원을 참조하십시오. 시작하기 자습서에서 사용하는 데모 애플리케이션은 CoreMQTT 에이전트 데모이며에 있습니다.`demos/coreMQTT_Agent/mqtt_agent_task.c`.

## 시작하기 문제 해결

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

### 주제

- [일반 시작하기 문제 해결 팁 \(p. 21\)](#)
- [터미널 에뮬레이터 설치 \(p. 22\)](#)

보드별 문제 해결은 해당 보드의 [FreeRTOS 시작하기 \(p. 14\)](#) 설명서를 참조하십시오.

## 일반 시작하기 문제 해결 팁

에 메시지가 표시되지 않습니다. AWS IoT Hello World 데모 프로젝트를 실행한 후 콘솔입니다. 어떻게 해야 합니까?

다음을 수행해 보십시오.

- 터미널 창을 열고 샘플을 로깅 출력을 봅니다. 이 단계는 무엇이 잘못되었는지 확인하는데 도움이 됩니다.
- 자격 증명이 유효한지 확인합니다.

데모를 실행할 때 터미널에 표시된 로그가 잘립니다. 길이를 늘리려면 어떻게 해야 합니까?

다음 항목의 가치를 늘립니다. `configLOGGING_MAX_MESSAGE_LENGTH~255`로 `FreeRTOSconfig.h` 실행 중인 데모용 파일:

```
#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
파리티	none
종지	1비트
흐름 제어	none

## 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/ttys*
```

macOS

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

## CMak를 사용하여 프리어토스와 사용

CMak를 사용하여 FreeRTOS 애플리케이션 소스 코드에서 프로젝트 빌드 파일을 생성하고 소스 코드를 빌드하고 실행할 수 있습니다.

IDE를 사용하여 FreeTos 인증 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 시작 키트
- Marvell MW322 AWS IoT 시작 키트
- Microchip Curiosity PIC32MZEF 번들
- Nordic nRF52840 DK 개발 키트
- STMicroelectronics STM32L4 Discovery Kit IoT 노드
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows Simulator

FreeRTOS와 함께 CMak를 사용하는 방법에 대한 자세한 내용은 아래 주제를 참조하십시오.

#### 주제

- [사전 조건 \(p. 23\)](#)
- [타사 코드 편집기 및 디버깅 도구를 사용하여 FreeRTOS 애플리케이션 개발 \(p. 24\)](#)
- [CMak를 사용하여 FreeRTOS 빌드 \(p. 24\)](#)

## 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.org](#)에서 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 [scoop](#)이나 [chocolatey](#)와 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 [수식](#)에서 드립 실행형 버전을 설치하거나 Make를 번들로 제공하는 [MinGW](#)를 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 Code](#)를 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. CMak를 사용하여 FreeRTOS 애플리케이션 빌드에 대한 자세한 내용은 단원을 참조하십시오. [CMak를 사용하여 FreeRTOS 빌드 \(p. 24\)](#).

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

```
"configurations": [
  {
    "name": "Cortex Debug",
    "cwd": "${workspaceRoot}",
    "executable": "./build/st/stm321475_discovery/aws_demos.elf",
    "request": "launch",
    "type": "cortex-debug",
    "serverType": "stutil"
  }
]
```

## CMak를 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 에 대한 기본 도구 체인 파일을 제공합니다. [freertos/tools/cmake/toolchains](#). 이 파일을 CMake에 제공하는 방법은 CMake 명령줄 인터페이스를 사용하는지, 아니면 GUI를 사용하는지에 따라 다릅니다. 추가 세부 정보는 아래 [빌드 파일 생성\(CMake 명령줄 도구\) \(p. 24\)](#) 지침을 따르십시오. CMake의 교차 컴파일에 대한 자세한 내용은 공식 CMake wiki의 [CrossCompiling](#)을 참조하십시오.

CMake 기반의 프로젝트를 빌드하려면

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

[CMake 명령줄 도구](#) 또는 [CMake GUI](#)를 사용하여 네이티브 빌드 시스템용 빌드 파일을 생성할 수 있습니다.

FreeRTOS 빌드 파일 생성에 대한 자세한 내용은 단원을 참조하십시오. [빌드 파일 생성\(CMake 명령줄 도구\) \(p. 24\)](#)과 [빌드 파일 생성\(CMake GUI\) \(p. 26\)](#).

2. 네이티브 빌드 시스템을 호출하여 프로젝트를 실행 파일로 만듭니다.

FreeRTOS 빌드 파일 생성에 대한 자세한 내용은 단원을 참조하십시오. 생성된 빌드 파일에서 [FreeRTOS 빌드 \(p. 28\)](#).

## 빌드 파일 생성(CMake 명령줄 도구)

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

`cmake`에는 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. 예:

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

Or:

```
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/toolchains`에 없는 도구 체인 파일을 사용하려면 `CMAKE_TOOLCHAIN_FILE` 옵션을 사용할 수 있습니다. 예:

```
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. 예:

```
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_TYPE`을 `debug`로 설정합니다. 이 옵션을 활성화한 경우 CMake는 디버그 플래그를 컴파일 옵션에 추가하고 디버그 기호로 빌드합니다.

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

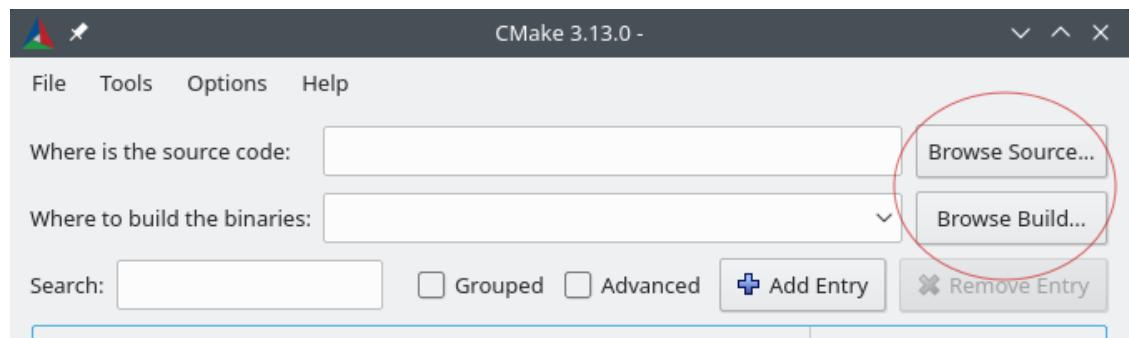
CMAKE\_BUILD\_TYPE을 release로 설정하여 컴파일 옵션에 최적화 플래그를 추가할 수도 있습니다.

## 빌드 파일 생성(CMake GUI)

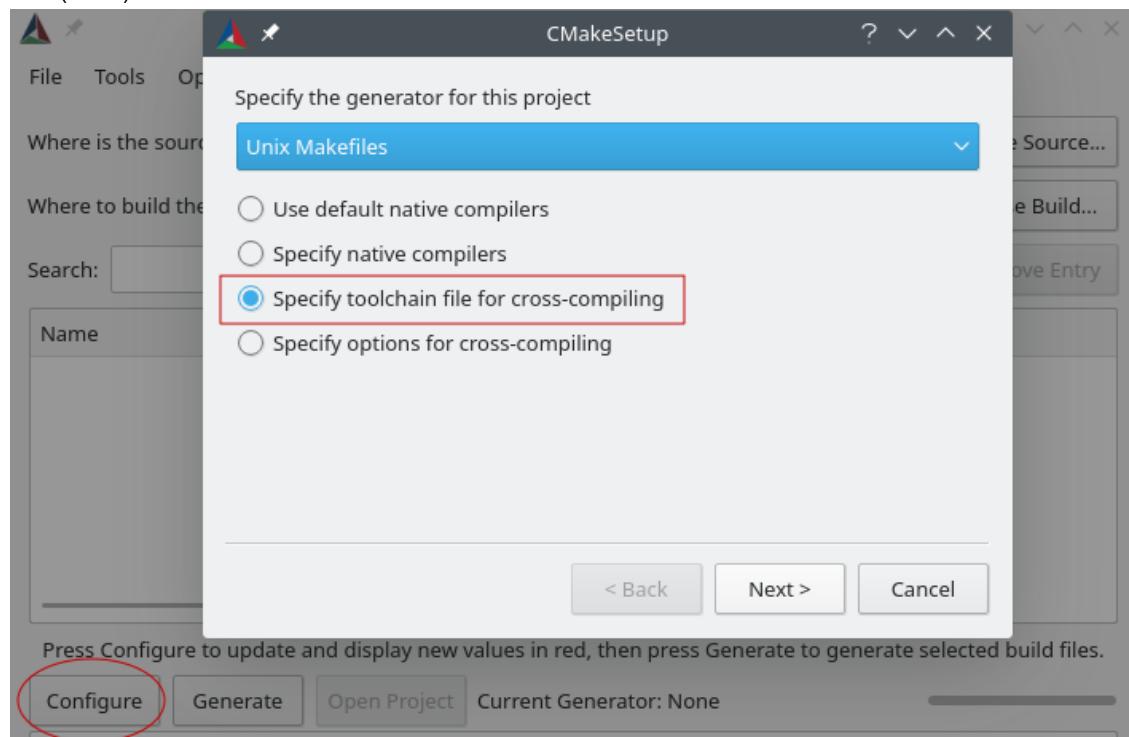
CMake GUI를 사용하여 FreeRTOS 빌드 파일을 생성할 수 있습니다.

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 를 선택하여 CMake 캐시를 삭제합니다.캐시 삭제( 사용)File 메뉴.



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 이동식 계층 대상을 제공하지 않는 템플릿 보드입니다. 결과적으로 Error in configuration process 메시지와 함께 창이 나타납니다.

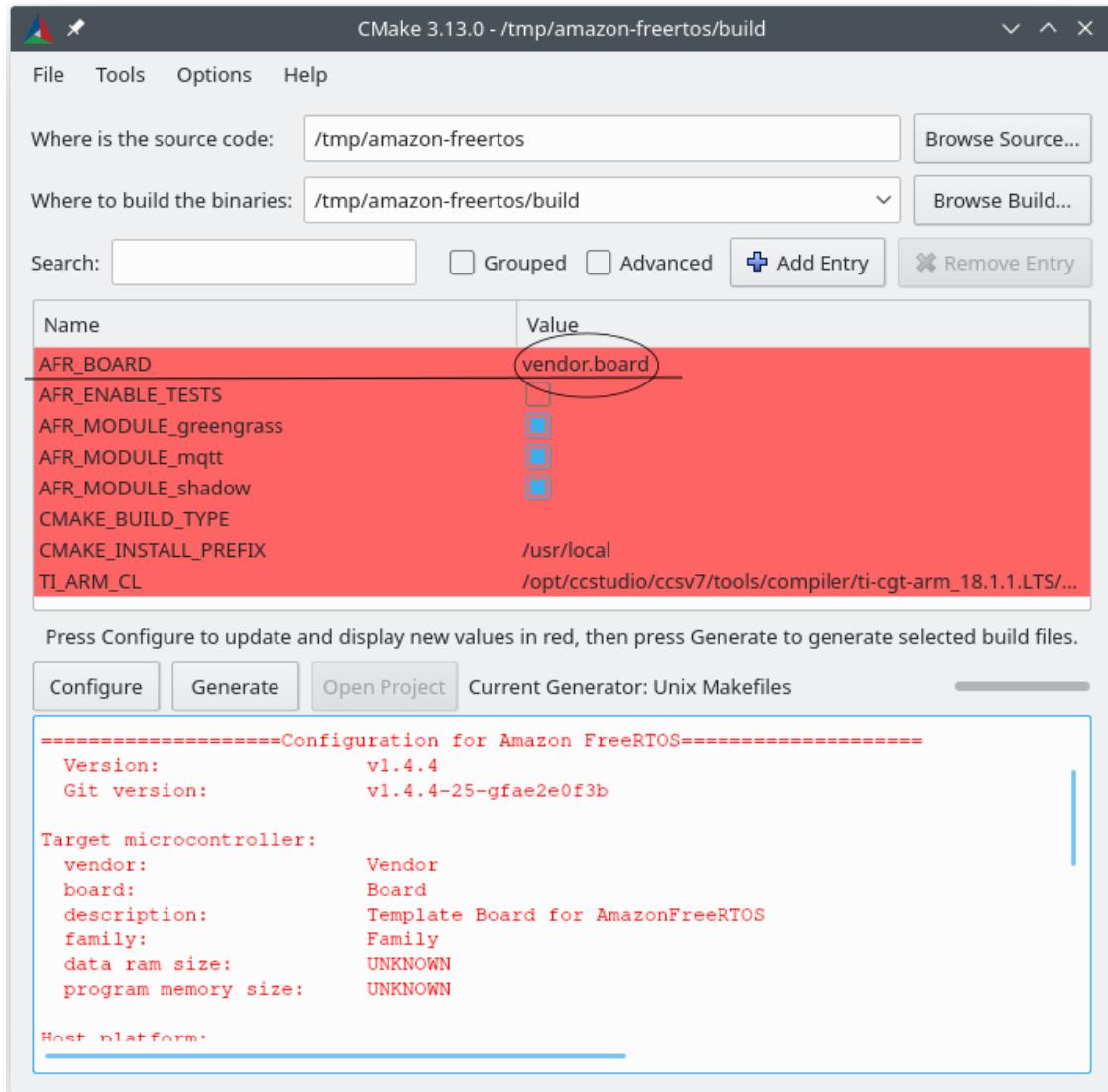
Note

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 구성은 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. 예:

```
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 설명서](#)를 참조하십시오.

# 개발자 모드 키 프로비저닝

## 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: AWS IoT에서 프라이빗 키 가져오기

랩 테스트를 위해 디바이스에서 프라이빗 키 가져오기를 허용하는 경우 [FreeRTOS 데모 구성 \(p. 18\)](#)의 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

먼저 [FreeRTOS 데모 구성 \(p. 18\)](#)의 단계를 수행하고 마지막 단계는 건너뜁니다. 즉, AWS IoT 자격 증명 포맷을 수행하지 않습니다. 최종 결과는 `demos/include/aws_clientcredential.h` 파일이 사용자 설정으로 업데이트되었지만 `demos/include/aws_clientcredential_keys.h` 파일은 업데이트되지 않은 것입니다.

### 데모 프로젝트 구성

의 보드 가이드에 설명된 대로 CoreMQTT 상호 인증 데모를 엽니다. [보드별 시작 안내서 \(p. 31\)](#). 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 프라이빗 키 및 클라이언트 인증서로 프로비저닝되지 않았으므로, 데모에서 다음과 같이 인증이 실패합니다. AWS IoT. 그러나 CoreMQTT 상호 인증 데모는 개발자 모드 키 프로비저닝을 실행하여 시작되므로 프라이빗 키가 없는 경우 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  
dcbb 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 키쌍을 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 생성되고 AWS IoT에 등록되면 다음 명령을 사용하여 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 인증

서를 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. JITP를 사용할 때 처음 AWS IoT에 대한 인증이 성공하려면 인증 기관(CA) 인증서를 가져와야 합니다. 프로젝트의 `aws_clientcredential_keys.h` 파일에서 `keyCLIENT_CERTIFICATE_PEM` 매크로를 `deviceCert.pem`의 콘텐츠로 설정하고 `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` 매크로를 `rootCA.pem`의 콘텐츠로 설정합니다.

#### 디바이스 승인

가져오기 `deviceCert.pem` 안으로 AWS IoT에 설명된 대로 레지스트리 자체 인증서 사용. 새 AWS IoT 사물을 생성하고 PENDING 인증서와 정책을 사물에 연결한 다음 인증서를 ACTIVE로 표시해야 합니다. 이러한 모든 단계는 AWS IoT 콘솔에서 수동으로 수행할 수 있습니다.

새 클라이언트 인증서가 ACTIVE이고 사물 및 정책과 연결되면 CoreMQTT 상호 인증 데모를 다시 실행합니다. 이제 AWS IoT MQTT 브로커와 성공적으로 연결됩니다.

## 보드별 시작 안내서

를 완료한 후 첫 번째 단계 (p. 14)에서 FreeRTOS 시작하기에 대한 보드별 지침은 해당 보드의 안내서를 참조하십시오.

- Cypress CYW943907AEVAL1F 개발 키트 시작하기 (p. 32)
- Cypress CYW954907AEVAL1F 개발 키트 시작하기 (p. 35)
- 사이프러스 CY8CKIT-064S0S2-4343W 키트 시작하기 (p. 38)
- Infineon XMC4800 IoT 연결 키트 시작하기 (p. 75)
- MW32x 시작하기 AWS IoT 스타터 키트 (p. 84)
- 플러그인으로 MediaTek MT7697Hx 개발 키트 (p. 99)
- Microchip Curiosity PIC32MZ EF 시작하기 (p. 103)
- Nuvoton 시작하기 NuMaker-IoT-M487 (p. 111)
- NXP LPC54018 IoT 모듈 시작하기 (p. 118)
- Renesas Starter Kit+ for RX65N-2MB 시작하기 (p. 121)
- STMicroelectronics STM32L4 Discovery Kit IoT Node 시작하기 (p. 124)
- Texas Instruments CC3220SF-LAUNCHXL 시작하기 (p. 126)
- Windows Device Simulator 시작하기 (p. 130)
- Xilinx Avnet 시작하기 MicroZed 산업용 IoT Kit (p. 133)

#### Note

를 완료할 필요가 없습니다. 첫 번째 단계 (p. 14) 다음 시작하기 자습 안내서에서 FreeRTOS.

- Windows 시뮬레이터를 사용하여 Microchip ATECC608A 보안 요소 시작하기 (p. 43)
- Espressif ESP32 시작하기 DevKitC와 ESP-WROVER-KIT (p. 46)
- Espressif ESP32-WROOM-32SE 시작하기 (p. 63)
- Espressif ESP32-S2 시작하기 (p. 68)
- Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트 시작하기 (p. 79)
- Nordic nRF52840-DK 시작하기 (p. 107)

## Cypress CYW943907AEVAL1F 개발 키트 시작하기

이 자습서에서는 Cypress CYW943907AEVAL1F 개발 키트를 시작하기 위한 지침을 제공합니다. Cypress CYW943907AEVAL1F 개발 키트가 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

### Note

이 자습서는 CoreMQTT 상호 인증 데모를 설정하고 실행하는 단계를 안내합니다. 현재, 이 보드의 FreeRTOS 포트는 TCP 서버와 클라이언트 데모를 지원하지 않습니다.

시작하려면 먼저 구성해야 합니다. AWS IoT 및 FreeRTOS 다운로드를 통해 디바이스를 AWS 클라우드에 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요.

### Important

- 이 주제에서 FreeRTOS 다운로드 디렉터리의 경로를 이라고 합니다.[freertos](#).
- [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.
- 소스 코드에 심볼릭 링크가 포함될 수 있으므로 Windows를 사용하여 아카이브를 추출하는 경우 다음을 수행해야 할 수 있습니다.
  - 활성화 [개발자 모드](#) 또는
  - 관리자 권한으로 상승된 콘솔을 사용합니다.

이러한 방식으로 Windows는 아카이브를 추출할 때 심볼릭 링크를 올바르게 만들 수 있습니다. 그렇지 않으면 심볼 링크의 경로가 텍스트로 포함되거나 비어 있는 일반 파일로 심볼 링크가 작성됩니다. 자세한 내용은 블로그 항목을 참조하십시오. [윈도우 10의 심볼릭 링크!](#)

Windows에서 Git을 사용하는 경우 개발자 모드를 활성화하거나 다음을 수행해야 합니다.

- Setcore.symlinks 다음 명령을 사용하여 true로 설정합니다.

```
git config --global core.symlinks true
```

- 시스템에 쓰는 git 명령을 사용할 때마다 관리자 권한으로 상승된 콘솔을 사용하십시오 (예:git pull, git clone, 및 git submodule update --init --recursive).
- 예 명시된 바와 같이 [FreeRTOS 다운로드 \(p. 17\)](#), Cypress용 FreeRTOS 포트는 현재에서만 사용할 수 있습니다. [GitHub](#).

## 개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

- 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
- 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
- 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

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

#### Linux 또는 macOS에서 환경 변수를 만들려면

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

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

- 시스템을 다시 시작합니다.
- 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: None
  - 정지 비트: 1
  - 흐름 제어: None

터미널 설치 및 직렬 연결 설정에 대한 자세한 내용은 [터미널 에뮬레이터 설치 \(p. 22\)](#)를 참조하십시오.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 디바이스가 전송한 메시지를 모니터링하는 AWS 클라우드 콘솔

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 **MQTT 테스트**를 선택합니다. 테스트 그런 후 선택합니다. MQTT 테스트 클라이언트 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

## 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 빌드 및 실행하려면

1. WICED Studio를 시작합니다.
2. File(파일) 메뉴에서 Import( 가져오기 )를 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 고기)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.

## 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\Win32**로 이동하고 openocd-all-brcm-libftdi.exe 파일을 두 번 클릭합니다.
  2. **WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx\_Wi-Fi\tools\drivers\CYW9WCD1EVAL1**로 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 단원을 참조하세요.[시작하기 문제 해결 \(p. 21\)](#).

## Cypress CYW954907AEVAL1F 개발 키트 시작하기

이 자습서에서는 Cypress CYW954907AEVAL1F 개발 키트를 시작하기 위한 지침을 제공합니다. Cypress CYW954907AEVAL1F 개발 키트가 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

### Note

이 자습서는 CoreMQTT 상호 인증 데모를 설정하고 실행하는 단계를 안내합니다. 현재, 이 보드의 FreeRTOS 포트는 TCP 서버와 클라이언트 데모를 지원하지 않습니다.

시작하려면 먼저 구성해야 합니다. AWS IoT 및 FreeRTOS 다운로드를 통해 디바이스를 AWS 클라우드 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 딜렉터리의 경로를 이라고 합니다.**freertos**.

### Important

- 이 주제에서 FreeRTOS 다운로드 딜렉터리의 경로를 이라고 합니다.**freertos**.
- **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 딜렉터리 경로가 길면 빌드 오류가 발생할 수 있습니다.
- 소스 코드에 심볼릭 링크가 포함될 수 있으므로 Windows를 사용하여 아카이브를 추출하는 경우 다음을 수행해야 할 수 있습니다.
  - 활성화 [개발자 모드](#) 또는
  - 관리자 권한으로 상승된 콘솔을 사용합니다.

이러한 방식으로 Windows는 아카이브를 추출할 때 심볼릭 링크를 올바르게 만들 수 있습니다. 그렇지 않으면 심볼 링크의 경로가 텍스트로 포함되거나 비어 있는 일반 파일로 심볼 링크가 작성됩니다. 자세한 내용은 블로그 항목을 참조하십시오.[윈도우 10의 심볼릭 링크!](#)

Windows에서 Git을 사용하는 경우 개발자 모드를 활성화하거나 다음을 수행해야 합니다.

- Setcore.symlinks 다음 명령을 사용하여 true로 설정합니다.

```
git config --global core.symlinks true
```

- 시스템에 쓰는 git 명령을 사용할 때마다 관리자 권한으로 상승된 콘솔을 사용하십시오 (예:git pull, git clone, 및 git submodule update --init --recursive).
- 에 명시된 바와 같이[FreeRTOS 다운로드 \(p. 17\)](#)Cypress용 FreeRTOS 포트는 현재 단에서만 사용할 수 있습니다. [GitHub](#).

## 개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

- 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
- 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
- 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-6.2의 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

- 제어판을 열고 시스템을 선택한 후 고급 시스템 설정을 선택합니다.
- 고급 탭에서 환경 변수를 선택합니다.
- 사용자 변수 아래에서 새로 만들기를 선택합니다.
- 변수 이름에 WICED\_STUDIO\_SDK\_PATH를 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: None
  - 정지 비트: 1
  - 흐름 제어: None

터미널 설치 및 직렬 연결 설정에 대한 자세한 내용은 [터미널 에뮬레이터 설치 \(p. 22\)](#)를 참조하십시오.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 디바이스가 전송한 메시지를 모니터링하는 AWS 클라우드 콘솔에서 AWS IoT 클라우드에서 MQTT 클라이언트를 실행합니다.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 `[]`을 선택합니다. 테스트를 선택합니다. MQTT 테스트 클라이언트를 눌러 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `your-thing-name/example/topic`을 입력한 다음 주제 구독을 선택합니다.

## 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 빌드 및 실행하려면

1. WICED Studio를 시작합니다.

2. File(파일) 메뉴에서 Import(가져오기)를 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 빌드 하여 보드에 다운로드하면 자동으로 실행되어야 합니다.

## 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\Win32*로 이동하고 openocd-all-brcm-libftdi.exe 파일을 두 번 클릭합니다.
  2. *WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx\_Wi-Fi\tools\drivers\CYW9WCD1EVAL1*로 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 단원을 참조하십시오. [시작하기 문제 해결 \(p. 21\)](#).

## 사이프러스 CY8CKIT-064S0S2-4343W 키트 시작하기

이 자습서에서는 를 시작하기 위한 지침을 제공합니다.[사이키트-064S0S2-4343W](#) 키트. 아직 키트를 구입하지 못했다면 해당 링크를 사용하여 키트를 구입할 수 있습니다. 이 링크를 사용하여 키트 사용 설명서에 액세스할 수도 있습니다.

### 시작하기

시작하려면 을 구성해야 합니다.AWS IoT 디바이스를 에 연결하려면 FreeRTOS를 AWS Cloud. 지침은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 사전 요구 사항을 완료하고 FreeRTOS 패키지를 사용할 수 있습니다.AWS IoT Core 자격 증명.

#### Note

이 자습서에서는 “첫 단계” 섹션에서 생성된 FreeRTOS 다운로드 디렉터리에 대한 경로는 라고 합니다.*freertos*.

## 개발 환경 설정

FreeRTOS는 CMake 또는 Make 빌드 훌름에서 작동합니다. 이 ModusToolbox Make 빌드 훌름에 적합합니다. 함께 제공되는 이클립스 IDE를 사용할 수 있습니다. ModusToolbox 또는 IAR EW-ARM, Arm MDK 또는 마이크로소프트 비주얼 스튜디오 코드와 같은 파트너 IDE 이클립스 IDE는 윈도우, macOS 및 Linux 운영 체제와 호환됩니다.

시작하려면 최신 버전을 다운로드하고 설치하려면 [ModusToolbox 소프트웨어](#). 자세한 내용은 단원을 참조하십시오. [ModusToolbox 설치 가이드](#).

### 용 도구 업데이트 ModusToolbox 2.1 이상

를 사용하는 경우 ModusToolbox 2.1 Eclipse IDE가 이 키트를 프로그래밍하려면 OpenOCD 및 펌웨어 로더 도구를 업데이트해야 합니다.

기본적으로 다음 단계에서는 **ModusToolbox** 경로:

- Windows는 `C:\Users\user_name\ModusToolbox`.
- Linux는 `user_home/ModusToolbox` 또는 아카이브 파일을 추출하도록 선택하는 위치입니다.
- macOS는 마법사에서 선택한 볼륨의 응용 프로그램 폴더 아래에 있습니다.

### OpenOCD 업데이트

이 키트는 칩을 성공적으로 지우고 프로그래밍하려면 Cypress OpenOCD 4.0.0 이상이 필요합니다.

사이프러스 오픈OCD를 업데이트하려면

- 로 이동합니다. [사이프러스 OpenOCD 출시 페이지](#).
- OS (윈도우/맥/리눅스) 용 아카이브 파일을 다운로드합니다.
- 에서 기존 파일 삭제 **ModusToolbox/tools\_2.x/openocd**.
- 파일 바꾸기 **ModusToolbox/tools\_2.x/openocd** 이전 단계에서 다운로드한 아카이브의 압축을 푼 내용과 함께 사용합니다.

### 펌웨어-로더 업데이트

이 키트에는 Cypress 펌웨어 로더 3.0.0 이상이 필요합니다.

Cypress 펌웨어-로더를 업데이트하려면

- 로 이동합니다. [사이프러스 펌웨어-로더 릴리스 페이지](#).
- OS (윈도우/맥/리눅스) 용 아카이브 파일을 다운로드합니다.
- 에서 기존 파일 삭제 **ModusToolbox/tools\_2.x/fw-loader**.
- 파일 바꾸기 **ModusToolbox/tools\_2.x/fw-loader** 이전 단계에서 다운로드한 아카이브의 압축을 푼 내용과 함께 사용합니다.

또는 CMake 를 사용하여 FreeRTOS 응용 프로그램 소스 코드에서 프로젝트 빌드 파일을 생성하고, 선호하는 빌드 도구를 사용하여 프로젝트를 빌드한 다음 OpenOCD를 사용하여 키트를 프로그래밍할 수 있습니다. CMake 훌름으로 프로그래밍하기 위해 GUI 도구를 사용하려면 Cypress 프로그래머를 다운로드하여 설치하십시오. [사이프러스 프로그래밍 솔루션 웹 페이지](#). 자세한 정보는 [CMake를 사용하여 프리어토스와 사용\(p. 22\)](#)을 참조하십시오.

## 하드웨어 설정

키트의 하드웨어를 설정하려면 다음 단계를 따르십시오.

## 1. 키트 프로비저닝

FLOWCY8CKIT-064S0S2-4343W 키트용 프로비저닝 가이드 키트를 안전하게 프로비저닝하기 위한 지침 AWS IoT.

이 키트에는 CySecure 도구 3.1.0 이상

## 2. 직렬 연결 설정

- a. 키트를 호스트 컴퓨터에 Connect.
- b. 키트의 USB 직렬 포트는 호스트 컴퓨터에서 자동으로 열거됩니다. 포트 번호를 식별합니다. Windows에서는 를 사용하여 식별할 수 있습니다. 디바이스 관리자 아래에 포트(COM & LPT).
- c. 직렬 터미널을 시작하고 다음 설정으로 연결을 엽니다.
  - 전송 속도: 115200
  - 데이터: 8비트
  - 패리티: None
  - 정지 비트: 1
  - 흐름 제어: None

## FreeRTOS 데모 프로젝트 빌드 및 실행

이 섹션에서는 데모를 빌드 및 실행합니다.

### 1. 다음 단계를 따라야 합니다. CY8CKIT-064S0S2-4343W 키트용 프로비저닝 가이드.

#### 2. FreeRTOS 데모를 빌드하십시오.

- a. 용 Eclipse IDE를 엽니다. ModusToolbox 작업 영역을 선택하거나 만들 수 있습니다.
- b. File(파일) 메뉴에서 Import( 가져오기)를 선택합니다.

확장일반, 선택기준 프로젝트를 작업 공간으로를 선택한 다음 를 선택합니다. 다음.

- c. In루트 디렉터리를 입력합니다. *freertos/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws\_demos* 프로젝트 이름을 선택합니다. aws\_demos. 기본적으로 선택되어야 합니다.
- d. 선택Finish프로젝트를 작업공간으로 가져옵니다.
- e. 다음 중 하나를 수행하여 애플리케이션을 빌드합니다.
  - 에서 쿠 패널을 선택합니다. aws\_demos 애플리케이션 빌드.
  - 선택프로젝트선택하고 모두 빌드.

오류 없이 프로젝트가 컴파일되는지 확인합니다.

#### 3. 클라우드에서 MQTT 메시지 모니터링

데모를 실행하기 전에 AWS IoT 디바이스가 에 전송하는 메시지를 모니터링하는 콘솔 AWS CloudMQTT 클라이언트를 구독하려면 AWS IoT MQTT 클라이언트는 다음 단계를 따릅니다.

- a. AWS IoT 콘솔에 로그인합니다.
- b. 탐색 창에서 [] 를 선택합니다. 테스트를 선택한 다음 MQTT 테스트 클라이언트 MQTT 클라이언트를 엽니다.
- c. 용 구독 주제를 입력합니다. *your-thing-name/example/topic*를 선택한 다음 를 선택합니다. 주제 구독.

#### 4. FreeRTOS 데모 프로젝트 실행

- a. 프로젝트를 선택합니다. aws\_demos 작업 공간에서

- b. 에서 퀄 패널을 선택합니다.aws\_demos 프로그램 (KitProg3). 이 프로그램은 프로그래밍이 완료된 후 보드 및 데모 응용 프로그램이 실행되기 시작합니다.
- c. Serial 터미널에서 실행 중인 애플리케이션의 상태를 볼 수 있습니다. 다음 그림에서는 터미널 출력의 일부를 보여 줍니다.

```

COM5 - Tera Term VT
File Edit Setup Control Window Help
VLAN MAC Address : CC:C0:79:24:DB:8B
VLAN Firmware   : v10: Jul 30 2019 01:54:48 version 7.45.98.89 <r218486 CY> FWID 01-81376c4b
VLAN CLM        : API: 12.2 Data: 9.10.39 Compiler: 1.29.4 ClmImport: 1.36.3 Creation: 2019-07-30 01:43:02
VHD VERSION     : v1.30.0-rc3-dirty : v1.30.0-rc3 : GCC 7.2 : 2019-08-27 16:29:32 +0000
1 2518 [I]nfo Soc1 IP Address assigned: 192.168.43.207
2 5083 [I]nfo Soc1 Write certificate...
3 5623 [I]nfo Soc1 Device credential provisioning succeeded.
4 5627 [iot_thread] [INFO] [IMQTT]lulu SDH successfully initialized.
5 8504 [iot_thread] [INFO] [DEMO]lulu Successfully initialized the demo. Network type for the demo: 1
6 8504 [iot_thread] [INFO] [IMQTT]lulu MQTT library successfully initialized.
7 8504 [iot_thread] [INFO] [IMQTT]lulu MQTT demo client Identifier is cy8cproto-kit <length 13>.
8 8504 [iot_thread] [INFO] [IMQTT]lulu Establishing MQTT connection...
9 13411 [iot_thread] [INFO] [IMQTT]lulu Anonymous metrics (SDK language, SDK version) will be provided to AWS IoT. Recompile with AWS_IOT_MQTT_ENABLE_METRICS set to 0 to disable.
10 13412 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> CONNECT operation 0x800b2d0> Waiting for operation completion.
11 13753 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> CONNECT operation 0x800b2d0> Wait complete with result SUCCESS.
12 13753 [iot_thread] [INFO] [IMQTT]lulu New MQTT connection 0x800c864 established.
13 13753 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> SUBSCRIBE operation scheduled.
14 14065 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> PUBLISH operation scheduled.
15 14065 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> SUBSCRIBE operation 0x800b5e0> Waiting for operation completion.
16 14065 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> SUBSCRIBE operation 0x800b5e0> Wait complete with result SUCCESS.
17 14065 [iot_thread] [INFO] [DEMO]lulu All demo topic filter subscriptions accepted.
18 14065 [iot_thread] [INFO] [DEMO]lulu Publishing messages 0 to 1.
19 14065 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
20 14065 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
21 14065 [iot_thread] [INFO] [DEMO]lulu Waiting for 2 publishes to be received.
22 14065 [iot_thread] [INFO] [DEMO]lulu 2 publishes received.
23 14424 [iot_thread] [INFO] [DEMO]lulu Incoming PUBLISH received:
Subscription topic filter: iotdemo/topic/1
Publish topic name: iotdemo/topic/1
Publish retain flag: 0
Publish QoS: 1
Publish pay24 14424 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
25 14708 [iot_thread] [INFO] [DEMO]lulu 1 publish acknowledged message for PUBLISH 0 will be sent.
26 14708 [iot_thread] [INFO] [DEMO]lulu 1 publish acknowledged message for PUBLISH 1 successfully sent.
27 14708 [iot_thread] [INFO] [DEMO]lulu Incoming PUBLISH received:
Subscription topic filter: iotdemo/topic/2
Publish topic name: iotdemo/topic/2
Publish retain flag: 0
Publish QoS: 1
Publish pay28 14708 [iot_thread] [INFO] [IMQTT]lulu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
29 14710 [iot_thread] [INFO] [DEMO]lulu 1 publish acknowledged message for PUBLISH 2 will be sent.
30 14710 [iot_thread] [INFO] [DEMO]lulu 2 publishes received.
31 14710 [iot_thread] [INFO] [DEMO]lulu Publishing messages 2 to 3.

```

MQTT 데모에서는 네 가지 주제에 대한 메시지를 게시합니다 (iotdemo/topic/*n*, 여기서 *n*=1 ~ 4) 과 같은 메시지를 다시 수신하기 위해 모든 주제를 구독합니다. 메시지가 수신되면 데모는 주제에 대한 확인 메시지를 게시합니다.iotdemo/acknowledgements. 다음 목록에서는 터미널 출력에 나타나는 디버그 메시지와 함께 메시지의 일련 번호를 참조합니다. 출력에서 WIICED 호스트 드라이버 (WHD) 드라이버 세부 정보가 일련 번호 매기기 없이 먼저 인쇄됩니다.

- 1 ~ 4 — 장치가 구성된 AP (액세스 포인트)에 연결되고 AWS 구성된 엔드포인트 및 인증서를 사용하는 서버입니다.
- 5-13 — CoreMQTT 라이브러리가 초기화되고 장치가 MQTT 연결을 설정합니다.
- 14 ~ 17 — 장치가 게시된 메시지를 다시 수신하기 위해 모든 주제를 구독합니다.
- 18 ~ 30 — 장치가 두 개의 메시지를 게시하고 메시지를 다시 수신할 때까지 기다립니다. 각 메시지가 수신되면 장치는 확인 메시지를 보냅니다.

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

## 5. CMake (CMake FreeRTOS 와 사용)

CMaKe를 사용하여 데모 애플리케이션을 빌드하고 실행할 수 있습니다. CMake 및 네이티브 빌드 시스템을 설정하려면 [사전 조건 \(p. 23\)](#).

- a. 빌드 파일을 생성할 때는 다음 명령을 사용합니다. 대상 보드를 사용하여 대상 보드를 지정합니다.-DBOARD옵션.

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

Windows를 사용하는 경우에는 를 사용하여 네이티브 빌드 시스템을 지정해야 합니다.-gCMake는 기본적으로 비주얼 스튜디오를 사용하기 때문입니다.

### Example

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

arm-none-eabi-gcc가 셸 경로에 없으면 AFR\_TOOLCHAIN\_PATH CMake 변수도 설정해야 합니다.

### Example

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

- b. CMakae를 사용하여 프로젝트를 빌드하려면 다음 명령을 사용합니다.

```
cmake --build build_dir
```

- c. 마지막으로, 프로그램 cm0.hex과 cm4.hex에서 생성된 파일 build\_dir 사이프러스 프로그래머를 사용하여

## 다른 데모 실행

다음 데모 애플리케이션은 현재 릴리스에서 작동하도록 테스트 및 검증되었습니다. 이 데모는 다음에서 찾으실 수 있습니다. [freertos/demos](#) 디렉터리. 데모 데모 실행 방법에 대한 자세한 내용은 단원을 참조하십시오. [FreeRTOS 데모 \(p. 233\)](#).

- 블루투스 Low Energy 데모
- 무선 (OTA) 업데이트 데모
- 보안 소켓 에코 클라이언트 데모
- AWS IoTDevice Shadow 데모

## Debugging

i|KitProg키트의 3은 SWD 프로토콜을 통한 디버깅을 지원합니다.

- FreeRTOS 응용 프로그램을 디버깅하려면 aws\_demos 프로젝트 작업 공간에서 다음을 선택합니다. aws\_demos 디버그 (KitProg3) (사용) 퀵 패널.

## OTA 업데이트

PSoC 64 MCU가 필요한 모든 FreeRTOS 자격 테스트를 통과했습니다. 그러나 선택 사항 over-the-air PSoC 64 표준 보안에서 구현된 (OTA) 기능 AWS 펌웨어 라이브러리가 아직 평가가 보류 중입니다. 현재 구현된 OTA 기능은 다음을 제외한 모든 OTA 자격 테스트를 통과합니다. [aws\\_ota\\_test\\_case\\_rollback\\_if\\_unable\\_to\\_connect\\_after\\_update.py](#).

PSoC64 표준 보안을 사용하여 성공적으로 검증된 OTA 이미지가 장치에 적용되는 경우 —AWSMCU 및 기기가 통신할 수 없음 AWS IoT Core, 장치가 원래 정상 작동이 확인된 이미지로 자동 롤백할 수 없습니다. 이로 인해 디바이스에 연결할 수 없는 상태가 될 수 있습니다. AWS IoT Core 추가 업데이트를 위해. 이 기능은 Cypress 팀에서 아직 개발 중입니다.

자세한 내용은 단원을 참조하십시오. [OTA 업데이트 AWS](#) 그리고 CY8CKIT-064S0S2-4343W 키트. 추가 질문이 있거나 기술 지원이 필요한 경우 [사이프러스 개발자 커뮤니티](#).

## Windows 시뮬레이터를 사용하여 Microchip ATECC608A 보안 요소 시작하기

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](#)

시작 하려면 먼저 구성해야 합니다. AWS IoT FreeRTOS 를 다운로드하여 디바이스를 에 연결합니다. AWS를 라우드. 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 디렉터리에 대한 경로는 라고 합니다.[FreeRTOS](#).

### 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 이진 이미지로 교차 컴파일합니다.
4. 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.

## Microchip ATECC608A 하드웨어 설정

Microchip ATECC608A 디바이스와 상호 작용하려면 먼저 SAMD21을 프로그래밍해야 합니다.

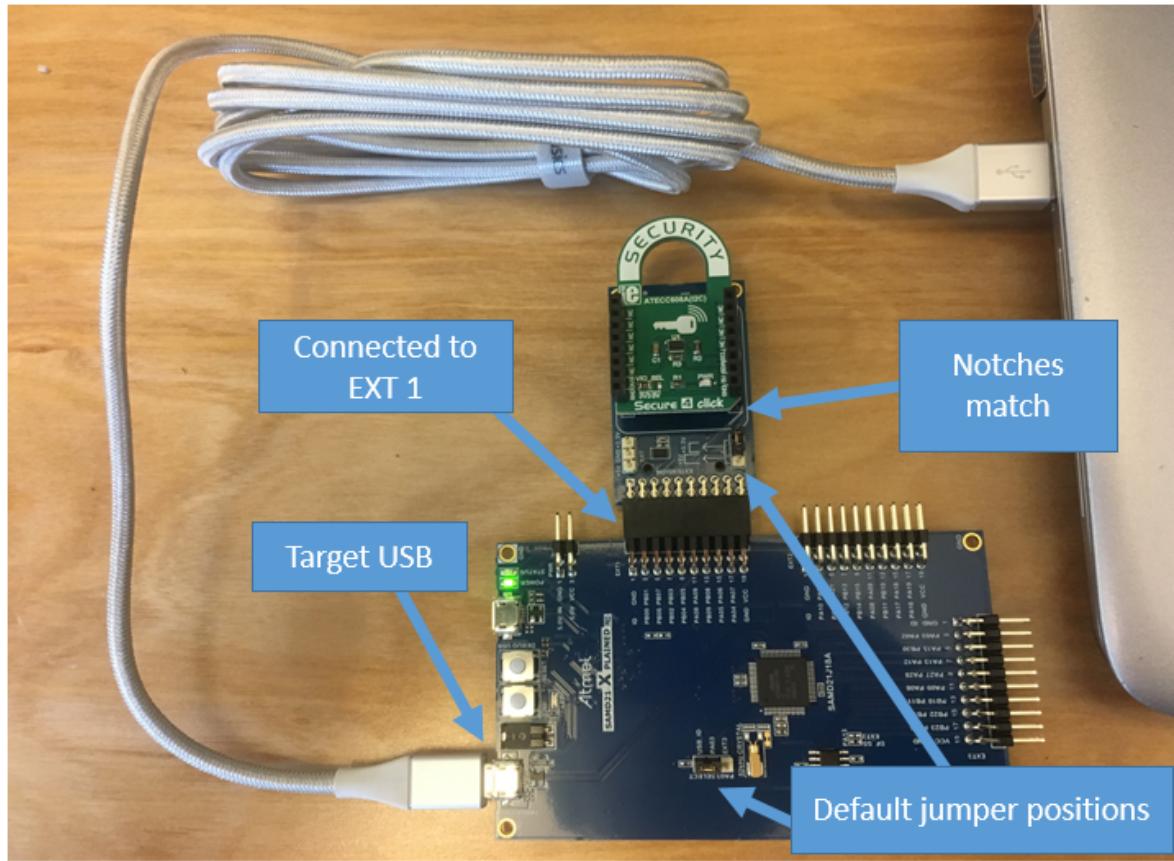
SAMD21 XPlained Pro 보드를 설정하려면

1. [CryptoAuthSSH-XSTK\(DM320109\) - 최신 펌웨어 링크](#)를 따라 지침(PDF)과 D21에 프로그래밍할 수 있는 바이너리가 들어 있는 .zip 파일을 다운로드합니다.
2. [Atmel 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 mikroBUS 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

1. 아직 설정하지 않았다면 다음과 같이 하십시오.[AWS 계정](#)을 생성합니다. IAM 사용자를 추가하려면 AWS 계정, 참조[IAM 사용 설명서](#).

IAM 사용자 계정에 액세스 권한을 부여하려면 AWS IoT 및 FreeRTOS에서는 다음 단계에서 IAM 사용자 계정에 다음 IAM 정책을 연결합니다.

- AmazonFreeRTOSFullAccess
- AWSIoTFullAccess

2. [를 연결합니다.](#)AmazonFreeRTOSFullAccess IAM 사용자에 대한 정책.

- a. 해당 위치로 이동합니다. [IAM 콘솔](#)을 선택합니다. 탐색 창에서 [\[\]](#)를 선택합니다. 사용자.
- b. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
- c. 권한 추가(Add permissions)를 선택합니다.
- d. [Attach existing policies directly]를 선택합니다.
- e. 검색 상자에 [를 입력합니다.](#)AmazonFreeRTOSFullAccess 목록에서 해당 항목을 선택한 다음 [를 선택합니다. 다음: Review](#))를 선택합니다.
- f. 권한 추가(Add permissions)를 선택합니다.

3. [를 연결합니다.](#)AWSIoTFullAccess IAM 사용자에 대한 정책.

- a. 해당 위치로 이동합니다. [IAM 콘솔](#)을 선택합니다. 탐색 창에서 [\[\]](#)를 선택합니다. 사용자.
- b. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
- c. 권한 추가(Add permissions)를 선택합니다.

- d. [Attach existing policies directly]를 선택합니다.
- e. 검색 상자에 를 입력합니다.**AWS IoT Full Access** 목록에서 해당 항목을 선택한 다음 를 선택합니다.(다음: Review)]를 선택합니다.
- f. 권한 추가(Add permissions)를 선택합니다.

IAM에 대한 자세한 내용은 단원을 참조하십시오.[IAM 권한 및 정책의 IAM 사용 설명서](#).

4. 에서 FreeRTOS 리포지토리를 다운로드하십시오.[FreeRTOS 리포지토리](#).

GitHub에서 FreeRTOS를 다운로드하려면:

1. 해당 위치로 이동합니다.[FreeRTOS 리포지토리](#).
2. Clone or download(복제 또는 다운로드)를 선택합니다.
3. 컴퓨터 명령줄에서 호스트 컴퓨터의 디렉터리로 리포지토리를 복제합니다.

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

#### Important

- 이 주제에서 FreeRTOS 다운로드 디렉터리에 대한 경로는 라고 합니다.*freertos*.
- *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.
- 소스 코드에 심볼릭 링크가 포함될 수 있으므로 Windows를 사용하여 아카이브를 추출하는 경우 다음을 수행해야 할 수 있습니다.
  - 활성화 개발자 모드 또는
  - 관리자 권한으로 상승된 콘솔을 사용합니다.

이러한 방식으로 Windows는 아카이브를 추출할 때 심볼릭 링크를 옮바르게 만들 수 있습니다. 그렇지 않으면 심볼 링크의 경로가 텍스트로 포함되거나 비어 있는 일반 파일로 심볼 링크가 작성됩니다. 자세한 내용은 블로그 항목을 참조하십시오.[윈도우 10의 심볼릭 링크!](#).

Windows에서 Git을 사용하는 경우 개발자 모드를 활성화하거나 다음을 수행해야 합니다.

- Setcore.symlinks 다음 명령을 사용하여 true로 설정합니다.

```
git config --global core.symlinks true
```

- 시스템에 쓰는 git 명령을 사용할 때마다 관리자 권한으로 상승된 콘솔을 사용하십시오 (예:git pull,git clone, 및git submodule update --init --recursive).

4. *freertos* 디렉터리에서 사용할 브랜치를 체크 아웃합니다.
5. 개발 환경을 설정합니다.

- a. 최신 버전의 [WinPCap](#)을 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\_InitToken을 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 빌드 및 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.

에서빌드메뉴, 선택솔루션 빌드를 사용하여 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 정보는 [네트워크 인터페이스 구성 \(p. 132\)](#)을 참조하십시오.

4. Microchip ATECC608A를 프로비저닝합니다.

마이크로칩은 ATECC608A 부품 설정에 도움이 되는 여러 스크립팅 도구를 제공합니다. `freertos\vendors\microchip\secure_elements\app\example_trust_chain_tool`로 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 \(p. 21\)](#) 단원을 참조하십시오.

## Espressif ESP32 시작하기|DevKitC와 ESP-WROVER-KIT

이 자습서에 따라 Espressif ESP32- 를 시작하십시오.DevKitESP32-WROOM-32, ESP32-SOLO-1 또는 ESP-WROVER 모듈과 ESP-WROVER-KIT-VB가 장착된 C. 파트너 중 하나를 구매하려면AWS파트너 장치 카탈로그에서 다음 링크를 사용하십시오.

- 에스P32 - 우룸-32DevKitC
- 에스P32-솔로-1
- 에스P32 브로버 키트

이러한 개발 보드 버전이 FreeRTOS에서 지원됩니다.

이 보드의 최신 버전에 대한 자세한 내용은 단원을 참조하십시오.[ESP32-DevKitC V4](#) 또는 [ESP-WROVER-KIT v4.1](#) 에스프레시피 웹 사이트에서 확인할 수 있습니다.

#### Note

현재 ESP32-WROVER-KIT 및 ESP용 FreeRTOS 포트 DevKitC는 대칭 다중 처리 (SMP) 기능을 지원하지 않습니다.

## 개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. FreeRTOS 데모 애플리케이션을 이진 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 를 설정해야 합니다.AWS계정 및 권한 계정을 생성하려면 단원을 참조하십시오.[생성 및 활성화AWSAccount](#).

을 추가하려면 AWS Identity and Access Management(IAM) 사용자를 계정에 연결하려면 를 참조하십시오.[IAM 사용 설명서](#). IAM 사용자 계정에 액세스 권한을 부여하려면 AWS IoT 및 FreeRTOS에서 다음 IAM 정책을 IAM 사용자 계정에 연결합니다.

- **AmazonFreeRTOSFullAccess**

IAM 사용자의 모든 FreeRTOS 리소스에 대한 전체 액세스를 허용합니다.(FreeRTOS: \*).

- **AWSIoTFullAccess**

IAM 사용자의 모든 사용자에 대한 전체 액세스를 허용합니다.AWS IoT리소스 (iot: \*).

를 연결하려면 **AmazonFreeRTOSFullAccess** IAM 사용자에게 정책

1. [IAM 콘솔](#)로 이동합니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
4. 권한 추가(Add permissions)를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. 검색 상자에 를 입력합니다.**AmazonFreeRTOSFullAccess** 목록에서 선택한 다음 를 선택합니다. 다음: Review]를 선택합니다.
7. 권한 추가(Add permissions)를 선택합니다.

를 연결하려면 AWSIoTFullAccess IAM 사용자에게 정책

1. [IAM 콘솔](#)로 이동합니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
4. 권한 추가(Add permissions)를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. 검색 상자에 [AWSIoTFullAccess](#)를 입력합니다. AWSIoTFullAccess 목록에서 선택한 다음 [선택]을 선택합니다. 다음 [Review]를 선택합니다.
7. 권한 추가(Add permissions)를 선택합니다.

IAM 및 사용자 계정에 대한 자세한 내용은 단원을 참조하십시오.[IAM 사용 설명서](#).

정책에 대한 자세한 내용은 단원을 참조하십시오.[IAM의 권한 및 정책](#).

## ESP-IDF v4.2로 시작하기

### Note

이 자습서의 Linux 명령을 사용하려면 Bash 셸을 사용해야 합니다.

1. Espressif 하드웨어를 설정합니다.

ESP32 설정에 대한 자세한 내용은 DevKitC 개발 보드 하드웨어,[ESP32-DevKitC V4 시작 안내서](#).

ESP-WROVER-KIT 개발 보드 하드웨어를 설정하는 방법에 대한 자세한 내용은[ESP-WROVER-KIT V4.1 시작하기 가이드](#).

### Important

도착했을 때 시작 Espressif 안내서의 섹션에서 중지한 다음 이 페이지의 지침으로 돌아오십시오.

2. 개발 환경을 설정합니다.

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다. 호스트 시스템의 운영 체제별 지침을 따릅니다.

- [윈도우용 툴체인의 표준 설정 및 필수 구성 요소 \(ESP-IDF v4.2\)](#).
- [macOS용 툴체인 필수 구성 요소 \(ESP-IDF v4.2\) 의 표준 설정](#).
- [Linux용 툴체인 필수 구성 요소 \(ESP-IDF v4.2\) 의 표준 설정](#).

### Important

아래의 “Get ESP-IDF” 지침에 도달하면 다음 단계를 중지한 다음 이 페이지의 지침으로 돌아옵니다.

3. 설치를 완료합니다 (리눅스/macOS).

ESP-IDF Windows 설치 프로그램은 필요한 모든 도구를 설치합니다. Linux 및 macOS 플랫폼은 설치를 완료하기 위해 추가 단계가 필요합니다. 다음 단계를 따르십시오.[FreeRTOS 다운로드 및 구성을 \(p. 49\)](#).

- a. 명령줄 창을 열니다.
- b. FreeRTOS 다운로드 디렉토리로 이동한 다음 다음 스크립트를 실행하여 플랫폼에 대한 espressif 툴체인을 다운로드하여 설치합니다.

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

- c. 다음 명령을 사용하여 터미널 경로에 ESP-IDF 툴체인 도구를 추가합니다.

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

4. 직렬 연결을 설정합니다.

호스트 시스템과 ESP32 사이에 직렬 연결을 설정하려면 DevKitC, UART Bridge VCP 드라이버에 대한 CP210x USB를 설치해야 합니다. [Silicon Labs](#)에서 이러한 드라이버를 다운로드할 수 있습니다.

호스트 머신과 ESP32-WROVER-KIT 사이에 직렬 연결을 설정하려면 FTDI 가상 COM 포트 드라이버를 설치해야 합니다. 이 드라이버는 다음에서 다운로드할 수 있습니다. [FTDI](#).

자세한 내용은 [ESP32와 직렬 연결 설정](#)을 참조하십시오. 직렬 연결을 설정한 후 보드의 연결을 위한 직렬 포트를 기록해 두십시오. 데모를 빌드할 때 이 정보가 필요합니다.

## FreeRTOS 다운로드 및 구성

환경을 설정한 후 다음 중 하나에서 FreeRTOS를 다운로드할 수 있습니다.

- [GitHub](#)(지침은 다음 () 을 참조하십시오. [Readme.md](#) 파일.)
- [FreeRTOS 콘솔](#)(올바른 항목을 선택합니다. Configuration과 하드웨어 플랫폼 예를 들어 기기와 일치하는 경우 "우"에 연결AWS IoT- ESP32-DevKitC".)

이 자습서에서는 FreeRTOS 구성 파일은 다음 위치에 있습니다.[freertos/vendors/espressif/boards/board-name/aws\\_demos/config\\_files/FreeRTOSConfig.h](#). (예: ifAFR\_BOARD espressif.esp32\_devkitc이 선택되면 구성 파일이 다음 위치에 있습니다. [freertos/vendors/espressif/boards/esp32/aws\\_demos/config\\_files/FreeRTOSConfig.h](#).)

### FreeRTOS 데모 애플리케이션 구성

1. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행 중인 경우 “ESP-IDF 4.x CMD” 앱 (ESP-IDF 툴체인을 설치할 때 이 옵션을 포함하는 경우) 또는 “명령 프롬프트” 앱을 여십시오.
2. Python3이 설치되어 있는지 확인하려면 를 실행합니다.

```
python --version
```

설치된 버전이 표시됩니다. Python 3.0.1 이상이 설치되어 있지 않으면 에서 를 설치할 수 있습니다. [파이썬 웹사이트](#).

3. 필요한 것은 AWS 실행할 명령줄 인터페이스 (CLI) AWS IoT 명령. Windows를 실행하는 경우 를 사용합니다. `easy_install awscli` 명령을 설치하려면 AWS “명령” 또는 “ESP-IDF 4.x CMD” 앱의 CLI를 사용할 수 있습니다.

macOS 또는 Linux를 실행하는 경우 단원을 참조하십시오. [다음 설치 AWS CLI](#).

4. 실행

```
aws configure
```

을 (를) 구성하고 AWS CLI를 사용하여 AWS 액세스 키 ID, 보안 액세스 키 및 기본값 AWS 리전. 자세한 내용은 단원을 참조하십시오. [구성 AWS CLI](#).

5. 다음 명령을 사용하여 을 설치합니다. AWS Python SDK (boto3):

- 윈도우의 경우 “명령” 또는 “ESP-IDF 4.x CMD” 앱에서

```
easy_install boto3
```

- macOS 또는 Linux에서는 를 실행합니다.

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

를 실행한 후

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

FreeRTOS는 다음 `SetupAWS.py`에 연결하기 위해 Espressif 보드를 더 쉽게 설정할 수 있게 해주는 스크립트 AWS IoT. 스크립트를 구성하려면 `freertos/tools/aws_config_quick_start/configure.json`을 열고 다음 속성을 설정합니다.

**afr\_source\_dir**

컴퓨터에서 `freertos` 디렉터리의 전체 경로입니다. 이 경로를 지정하기 위해 슬래시를 사용하고 있는지 확인합니다.

**thing\_name**

보드를 나타내는 AWS IoT 사물에 할당할 이름.

**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)

6. 구성 스크립트를 실행합니다.

- a. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행 중인 경우 “ESP-IDF 4.x CMD” 또는 “명령” 앱을 엽니다.
- b. 로 이동합니다. `freertos/tools/aws_config_quick_start` 디렉토리 및 실행

```
python SetupAWS.py setup
```

이 스크립트는 다음 작업을 수행합니다.

- IoT 사물, 인증서 및 정책을 생성합니다.
- 이 스크립트는 IoT 정책을 인증서에 연결하고 인증서를 AWS IoT 사물.
- 채우기 `aws_clientcredential.h`와 함께 파일 AWS IoT 엔드포인트, Wi-Fi SSID 및 자격 증명 입니다.

- 인증서와 프라이빗 키에 형식을 지정하고 예 기록합니다.`.aws_clientcredential_keys.h`에 더 파일.

#### Note

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

에 대한 자세한 내용 `SetupAWS.py` 참조, `README.md`의 `freertos/tools/aws_config_quick_start` 디렉토리입니다.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 디바이스가 예 전송하는 메시지를 모니터링하는 콘솔 AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

- AWS IoT 콘솔로 이동합니다.
- 탐색 창에서 를 선택합니다. 테스트를 선택한 후 MQTT 테스트 클라이언트.
- 구독 주제에 `your-thing-name/example/topic`을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제에 여러 번 전송되었습니다.

**idf.py 스크립트를 사용하여 FreeRTOS 데모 프로젝트를 빌드, 플래시 및 실행합니다.**

에스프레시프의 IDF 유ти리티를 사용할 수 있습니다 (`idf.py`) 프로젝트를 빌드하고 바이너리를 장치에 플래시합니다.

#### Note

일부 설정에서는 포트 옵션을 사용해야 할 수 있습니다. “`-p port-name`”와 `idf.py` 다음 예제와 같이 올바른 포트를 지정합니다.

```
idf.py -p /dev/cu.usbserial-00101301B flash
```

윈도우, 리눅스 및 macOS OS에서 프리어토스를 빌드하고 플래시 (ESP-IDF v4.2)

- FreeRTOS 다운로드 디렉토리의 루트로 이동합니다.
- 명령줄 창에서 다음 명령을 입력하여 ESP-IDF 도구를 터미널의 PATH에 추가합니다.

윈도우 (“명령” 앱)

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

윈도우 (“ESP-IDF 4.x CMD” 앱)

(이 작업은 앱을 열었을 때 이미 완료되었습니다.)

Linux/macOS

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

3. 에서 cmake를 구성하십시오.build 다음 명령을 사용하여 디렉터리를 열고 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 없으면 빌드에서 펌웨어 바이너리.bin 파일을 생성합니다.

4. 다음 명령을 사용하여 개발 보드의 플래시 메모리를 지웁니다.

```
idf.py erase_flash
```

5. 사용 idf.py 애플리케이션 바이너리를 보드에 플래시하는 스크립트.

```
idf.py flash
```

6. 다음 명령을 사용하여 보드의 직렬 포트에서 출력을 모니터링할 수 있습니다.

```
idf.py monitor
```

#### Note

다음 예에서처럼 이러한 명령을 결합할 수 있습니다.

```
idf.py erase_flash flash monitor
```

특정 호스트 시스템 설정의 경우 다음 예와 같이 보드를 플래시할 때 포트를 지정해야 합니다.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

## CMake를 사용하여 FreeRTOS 빌드 및 플래시

또한 다음과 같습니다.idf.py 코드를 빌드하고 실행하기 위해 IDF SDK에서 제공하는 스크립트는 CMake를 사용하여 프로젝트를 빌드 할 수도 있습니다. 현재는 유닉스 메이크 파일이나 닌자 빌드 시스템을 지원합니다.

### 프로젝트를 빌드 후 플래시하려면

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

Windows

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

Linux/macOS

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

3. 빌드 파일을 생성할 때는 다음 명령을 입력합니다.

Unix 메이크 파일 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
```

닌자 포함

```
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. 프로젝트를 빌드합니다.

Unix 메이크 파일 사용

```
make -C ./YOUR_BUILD_DIRECTORY -j8
```

닌자 포함

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. 플래시를 지운 다음 보드를 플래시합니다.

Unix 메이크 파일 사용

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

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

닌자 포함

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

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

## Bluetooth Low-Energy 데모 실행

FreeRTOS 지원 [Bluetooth Low Energy 라이브러리 \(p. 198\)](#) 연결성.

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 모바일 SDK 데모 애플리케이션을 설정하려면

1. [FreeRTOS Bluetooth 디바이스용 Mob \(p. 207\)](#)의 지침에 따라 모바일 플랫폼용 SDK를 다운로드하여 호스트 컴퓨터에 설치합니다.
2. [FreeRTOS 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 \(p. 236\)](#)의 지침에 따라 모바일 디바이스에서 데모 모바일 애플리케이션을 설정합니다.

보드에서 MQTT over Bluetooth Low Energy 데모를 실행하는 방법에 대한 자세한 내용은 단원을 참조하십시오.[MQTT over Bluetooth Low Energy \(p. 240\)](#).

보드에서 Wi-Fi 프로비저닝 데모를 실행하는 방법에 대한 자세한 내용은 단원을 참조하십시오.[Wi-Fi 프로비저닝 \(p. 241\)](#).

## ESP32 용 자체 CMake 프로젝트에서 FreeRTOS 사용

자체 CMake 프로젝트에서 FreeRTOS를 사용하려면 하위 디렉터리로 설정하고 애플리케이션과 함께 빌드 할 수 있습니다. 먼저 다음 중 하나에서 FreeRTOS 사본을 가져옵니다.[GitHub](#) 또는 [FreeRTOS 콘솔](#). Git를 사용하는 경우 다음 명령을 사용하여 Git 하위 모듈로 설정할 수도 있으므로 나중에 업데이트하기가 더 쉽습니다.

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

이후 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
```

그런 다음 다음은 최상위 수준의 예입니다. CMakeLists.txt FreeRTOS와 함께 애플리케이션을 빌드하는데 사용할 수 있는 파일입니다.

```
cmake_minimum_required(VERSION 3.13)  
  
project(freertos_examples)  
  
# Tell IDF build to link against this target.  
set(IDF_EXECUTABLE_SRCS "<complete_path>/src/main.c")  
set(IDF_PROJECT_EXECUTABLE my_app)  
  
# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.  
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")  
add_subdirectory(freertos)
```

```
# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

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-directory
```

애플리케이션을 보드에 플래시하려면 다음 명령을 실행합니다.

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

## FreeRTOS에서 구성 요소 사용

CMake를 실행한 후에는 요약 출력에서 사용 가능한 모든 구성 요소를 찾을 수 있습니다. 다음 예와 같아야 합니다.

```
=====Configuration for FreeRTOS=====
Version: 202107.00
Git version: 202107.00-g79ad6defb

Target microcontroller:
  vendor: Espressif
  board: ESP32-DevKitC
  description: Development board produced by Espressif that comes in two
  variants either with ESP-WROOM-32 or ESP32-WROVER module
  family: ESP32
  data ram size: 520KB
  program memory size: 4MB

Host platform:
  OS: Linux-4.15.0-66-generic
  Toolchain: xtensa-esp32
  Toolchain path: /opt/xtensa-esp32-elf
  CMake generator: Ninja

FreeRTOS modules:
  Modules to build: backoff_algorithm, common, common_io, core_http,
  core_http_demo_dependencies, core_json, core_mqtt,
  core_mqtt_agent, core_mqtt_agent_demo_dependencies,
  core_mqtt_demo_dependencies, crypto, defender, dev_mode_key_
  provisioning, device_defender, device_defender_demo_
  dependencies, device_shadow, device_shadow_demo_dependencies,
  freertos_cli_plus_uart, freertos_plus_cli, greengrass,
  http_demo_helpers, https, jobs, jobs_demo_dependencies,
  kernel, logging, mqtt, mqtt_agent_interface, mqtt_demo_
  helpers, mqtt_subscription_manager, ota, ota_demo_
  dependencies, ota_demo_version, pkcs11, pkcs11_helpers,
  pkcs11_implementation, pkcs11_utils, platform, secure_sockets,
  serializer, shadow, tls, transport_interface_secure_sockets,
  wifi

  Enabled by user: common_io, core_http_demo_dependencies, core_json,
  core_mqtt_agent_demo_dependencies, core_mqtt_demo_
  dependencies, defender, device_defender, device_defender_demo_
  dependencies, device_shadow, device_shadow_demo_dependencies,
  freertos_cli_plus_uart, freertos_plus_cli, greengrass, https,
  jobs, jobs_demo_dependencies, logging, ota_demo_dependencies,
```

```

Enabled by dependency: pkcs11, pkcs11_helpers, pkcs11Implementation, pkcs11_utils,
platform, secure_sockets, shadow, wifi
backoff_algorithm, common, core_http, core_mqtt,
core_mqtt_agent, crypto, demo_base, dev_mode_key_provisioning,
freertos, http_demo_helpers, kernel, mqtt, mqtt_agent_
interface, mqtt_demo_helpers, mqtt_subscription_manager, ota,
ota_demo_version, pkcs11_mbedtls, serializer, tls,
transport_interface_secure_sockets, utils
3rdparty dependencies: jsmn, mbedtls, pkcs11, tinycc
Available demos: demo_cli_uart, demo_core_http, demo_core_mqtt, demo_core_mqtt_
agent, demo_device_defender, demo_device_shadow,
demo_greengrass_connectivity, demo_jobs, demo_ota_core_http,
demo_ota_core_mqtt, demo_tcp
Available tests:
=====

```

에서 모든 구성 요소를 참조할 수 있습니다. Modules to build 목록. 응용 프로그램에 연결하려면 AFR::이름 앞에 있는 네임스페이스는 다음과 같이 합니다. AFR::core\_mqtt, AFR::ota 등.

## ESP-IDF를 사용하여 사용자 지정 구성 요소 추가

ESP-IDF를 사용하는 동안 더 많은 구성 요소를 추가할 수 있습니다. 예를 들어, example\_component라는 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 의 예제입니다. CMakeLists.txt 구성 요소에 대한 파일입니다.

```

add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)

```

그런 다음 최상위 레벨에서 CMakeLists.txt 파일, 바로 뒤에 다음 줄을 삽입하여 구성 요소를 추가하십시오.  
.add\_subdirectory(freertos).

```

add_subdirectory(component/example_component)

```

그런 다음 를 수정합니다. target\_link\_libraries 구성 요소를 포함합니다.

```

target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)

```

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

## FreeRTOS의 구성 재정의

현재 FreeRTOS 소스 트리 외부에서 config를 재정의하도록 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 레벨에서 CMakeLists.txt 파일, 앞에 이 줄을 추가하십시오.`add_subdirectory(freertos)` 컴파일러가 이 디렉터리를 먼저 검색합니다.

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

## ESP-IDF를 위한 자체 sdkconfig 제공

자체 sdkconfig.default를 제공하려는 경우 명령줄에서 CMake 변수 `IDF_SDKCONFIG_DEFAULTS`을 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
```

자신의 위치를 지정하지 않은 경우 sdkconfig.default 파일, FreeRTOS는 예 있는 기본 파일을 사용합니다. *freertos/vendors/espressif/boards/esp32/aws\_demos/sdkconfig.defaults*.

## 요약

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

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

set(IDF_PROJECT_EXECUTABLE my_app)
set(IDF_EXECUTABLE_SRCS "src/main.c")

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)

# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/example_component" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})

# Override the configurations for FreeRTOS.
include_directories(BEFORE freertos-configs)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
```

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

## 문제 해결

- macOS 실행하는 경우 운영 체제에서 ESP-WROVER-KIT를 인식하지 못하면 D2XX 드라이버가 설치되어 있지 않은지 확인합니다. 이를 제거하려면 [macOS X용 FTDI 드라이버 설치 가이드](#)의 지침을 따르십시오.
- ESP-IDF에서 제공하는 모니터 유ти리티 (및 make monitor를 사용하여 호출)를 사용하면 주소를 디코딩 할 수 있습니다. 이러한 이유로 이 유ти리티는 애플리케이션이 작동을 멈추는 경우 의미 있는 역추적을 얻는 데 도움이 될 수 있습니다. 자세한 내용은 단원을 참조하십시오.[자동 주소 디코딩에스프레시피 웹 사이트](#)에서 확인할 수 있습니다.
- 특별한 JTAG 하드웨어를 사용할 필요 없이 gdb와 통신하기 위해 GDBstub를 활성화할 수도 있습니다. 자세한 내용은 단원을 참조하십시오.[GDB스텁을 사용하여 GDB 시작에스프레시피 웹 사이트](#)에서 확인할 수 있습니다.
- JTAG 하드웨어 기반 디버깅이 필요한 경우 OpenOCD 기반 환경 설정에 대한 자세한 내용은 단원을 참조하십시오.[JTAG 디버깅에스프레시피 웹 사이트](#)에서 확인할 수 있습니다.
- 다음의 경우,pyserial를 사용하여 설치할 수 없습니다.pipmacOS의 경우 에서 다운로드합니다.[Pyserial 웹 사이트](#).
- 보드가 연속적으로 재설정되는 경우 터미널에 다음 명령을 입력하여 플래시를 지워 봅니다.

```
make erase_flash
```

- idf\_monitor.py를 실행할 때 오류가 나타나는 경우 Python 2.7을 사용합니다.
- ESP-IDF의 필수 라이브러리는 FreeRTOS에 포함되어 있으므로 외부에서 다운로드할 필요가 없습니다. 만약 IDF\_PATH환경 변수가 설정된 경우 FreeRTOS를 빌드하기 전에 제거하는 것이 좋습니다.
- Window에서 프로젝트를 빌드하려면 3-4분 정도 걸릴 수 있습니다. 빌드 시간을 줄이기 위해 다음을 사용 할 수 있습니다.-j4make 명령을 캡니다.

```
make flash monitor -j4
```

- 디바이스에서 AWS IoT에 연결하는 데 문제가 있으면, aws\_clientcredential.h 파일을 열고 파일에 구성 변수가 제대로 정의되어 있는지 확인합니다. clientcredentialMQTT\_BROKER\_ENDPOINT[]가 1234567890123-ats.iot.us-east-1.amazonaws.com과 같이 정의되어 있어야 합니다.
- ESP32 용 자체 CMake 프로젝트에서 FreeRTOS 사용 (p. 54)의 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 �렉터리 아래의 CMakeLists.txt 파일을 업데이트합니다.
- ESP-IDF v4.2는 엑텐사\ -ESP32\ -엘파\ -gcc 8\ .2\ .0\ 도구 체인. 이전 버전의 Xtensa 도구 체인을 사용하는 경우 필요한 버전을 다운로드하십시오.
- ESP-IDF v4.2에서 충족되지 않는 파이썬 종속성에 대해 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 파이썬 종속성을 설치합니다.

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

자세한 문제 해결 정보는 단원을 참조하십시오. [시작하기 문제 해결 \(p. 21\)](#).

## Debugging

### Espressif ESP32- 디버깅 코드 디버깅DevKitC 및 ESP-WROVER-KIT (ESP-IDF v4.2)

이 섹션에서는 ESP-IDF v4.2를 사용하여 Espressif 하드웨어를 디버깅하는 방법을 보여줍니다. 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 DevkitC 대한 연결입니다.

C232HM-DDHSL-0 Wire Color	ESP32 GPIO Pin	JTAG Signal Name
Brown (pin 5)	IO14	TMS
Yellow (pin 3)	IO12	TDI
Black (pin 10)	GND	GND
Orange (pin 2)	IO13	TCK
Green (pin 4)	IO15	TDO

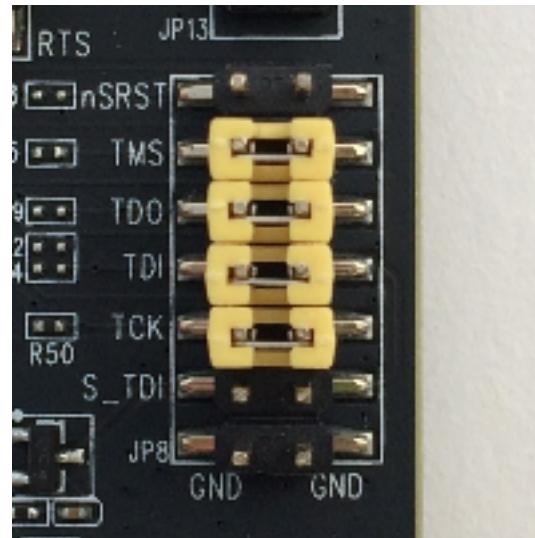
#### ESP-WROVER-KIT JTAG 설정

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

C232HM-DDHSL-0 Wire Color	ESP32 GPIO Pin	JTAG Signal Name
Brown (pin 5)	IO14	TMS
Yellow (pin 3)	IO12	TDI
Orange (pin 2)	IO13	TCK
Green (pin 4)	IO15	TDO

이러한 테이블은 [FTDI C232HM-DDHSL-0 데이터시트](#)에서 개발되었습니다. 자세한 내용은 데이터 시트의 “C232HM MPSSE 케이블 연결 및 기계 세부 정보 섹션을 참조하십시오.

ESP-WROVER-KIT에서 JTAG를 활성화하려면 TMS, TDO, TDI, TCK, 및 S\_TDI 핀의 점퍼를 다음과 같이 배치합니다.



원도우에서 디버깅 (ESP-IDF v4.2)

Windows에서 디버깅을 설정하려면

1. [Espressif ESP32- 디버깅 코드 디버깅DevKitC 및 ESP-WROVER-KIT \(ESP-IDF v4.2\) \(p. 59\)](#)의 설명과 같이 FTDI C232HM-DDHSL-0의 USB 쪽을 컴퓨터 및 다른 쪽에 연결합니다. FTDI C232HM-DDHSL-0 디바이스가 예 표시되어야 합니다. 디바이스 관리자 아래에 범용 직렬 버스 컨트롤러.
2. 범용 직렬 버스 디바이스 목록에서 C232HM-DDHSL-0 장치를 선택한 다음 속성.

#### Note

디바이스가 USB 직렬 포트로 나열될 수 있습니다.

디바이스의 속성을 보려면 속성 창에서 세부 정보 탭. 장치가 나열되지 않은 경우 [FTDI C232HM-DDHSL-0 용 원도우 드라이버](#).

3. 세부 정보 탭에서 속성을 선택한 후 하드웨어 ID를 선택합니다. 다음과 같이 보여야 합니다. 값 필드.

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 로 시작하는 줄에서 지정합니다. `ftdi_vid_pid` 공급업체 ID 및 제품 ID 뒤에로 이어집니다.

`ftdi_vid_pid 0x0403 0x6014`

4. [Windows용 OpenOCD](#)를 다운로드합니다.
5. C:\에 파일의 압축을 풀고 C:\openocd-esp32\bin을 시스템 경로에 추가합니다.
6. OpenOCD에는 libusb가 필요하며, 이 프로그램은 Windows에서 기본적으로 설치되지 않습니다. libusb를 설치하려면
  - a. [zadig.exe](#)를 다운로드합니다.
  - b. [zadig.exe](#)를 실행합니다. Options(옵션) 메뉴에서 List All Devices(모든 디바이스 나열)를 선택합니다.
  - c. 드롭다운 메뉴에서 를 선택합니다.C232HM-DDHSL-0.

- d. 녹색 화살표 오른쪽의 대상 드라이버 필드에서 WinUSB를 선택합니다.
  - e. 대상 드라이버 필드 아래의 목록에서 화살표를 선택한 다음 을 선택합니다. 드라이버 설치. Replace Driver(드라이버 바꾸기)를 선택합니다.
7. 명령 프롬프트를 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동한 후 다음 명령을 실행합니다.

```
idf.py openocd
```

이 명령 프롬프트를 열어 둡니다.

8. 새 명령 프롬프트를 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동한 다음 를 실행합니다.

```
idf.py flash monitor
```

9. 다른 명령 프롬프트를 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동한 후 해당 보드에서 데모의 실행이 시작될 때까지 기다립니다. 시작되면 를 실행합니다.

```
idf.py gdb
```

프로그램이 main 기능에서 중지해야 합니다.

#### Note

ESP32는 최대 2개의 중단점을 지원합니다.

#### macOS OS에서 디버깅 (ESP-IDF v4.2)

1. [macOS용 FTDI 드라이버](#)를 다운로드합니다.
2. [OpenOCD](#)를 다운로드합니다.
3. 다운로드한 .tar 파일의 압축을 풀고 .bash\_profile에서 경로를 OCD\_INSTALL\_DIR/openocd-esp32/bin으로 설정합니다.
4. 다음 명령을 사용하여 을 설치하십시오.libusbmacOS에서 를 선택합니다.

```
brew install libusb
```

5. 다음 명령을 사용하여 직렬 포트 드라이버를 언로드합니다.

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

6. 다음 명령을 사용하여 직렬 포트 드라이버를 언로드합니다.

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

7. 10.9 이후의 macOS 버전을 실행하는 경우 다음 명령을 사용하여 Apple FTDI 드라이버를 언로드합니다.

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

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

```
system_profiler SPUSBDataType
```

의 출력 system\_profiler 다음과 같이 보여야 합니다.

DEVICE:

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

9. projects/espressif/esp32/make/aws\_demos/esp32\_devkitj\_v1.cfg 파일을 업니다.  
디바이스의 공급업체 ID 및 제품 ID는 ftdi\_vid\_pid로 시작하는 줄에서 지정합니다. 이전 단계에서 출력한 system\_profiler의 ID와 일치하도록 ID를 변경합니다.
10. 터미널 창을 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동한 후 다음 명령을 사용하여 OpenOCD를 실행합니다.

```
idf.py openocd
```

이 터미널 창을 열린 상태로 둡니다.

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

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

12. FreeRTOS 다운로드 디렉터리의 루트로 이동한 다음 을 실행합니다.

```
idf.py flash monitor
```

13. 다른 새 터미널을 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동한 다음 를 실행합니다.

```
idf.py gdb
```

프로그램이 main에서 중지해야 합니다.

#### 리눅스에서의 디버깅 (ESP-IDF v4.2)

1. [OpenOCD](#)를 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. 이렇게 하면 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 에서 변경되었는지 확인합니다.dialout@plugdev.

```
$ls -l /dev/ttyUSB*
crw-rw---- 1 root plugdev 188, 0 Jul 10 19:04 /dev/
ttyUSB0
crw-rw---- 1 root plugdev 188, 1 Jul 10 19:04 /dev/
ttyUSB1
```

더 낮은 숫자의 /dev/ttyUSBn 인터페이스가 JTAG 통신에 사용됩니다. 다른 인터페이스는 ESP32 직렬 포트 (UART)로 라우팅되며 코드를 ESP32 플래시 메모리에 업로드하는 데 사용됩니다.

- 터미널 창에서 FreeRTOS 다운로드 디렉토리의 루트로 이동한 후 다음 명령을 사용하여 OpenOCD를 실행합니다.

```
idf.py openocd
```

- 다른 터미널을 열고 FreeRTOS 다운로드 디렉토리의 루트로 이동한 후 다음 명령을 실행합니다.

```
idf.py flash monitor
```

- 다른 터미널을 열고 FreeRTOS 다운로드 디렉토리의 루트로 이동한 후 다음 명령을 실행합니다.

```
idf.py gdb
```

프로그램이 main()에서 종지되어야 합니다.

## Espressif ESP32-WROOM-32SE 시작하기

이 자습서에서는 Espressif ESP32-WROOM-32SE를 시작하는 방법을 보여줍니다. 파트너 중 하나를 구매하려면 AWS 파트너 장치 카탈로그, 참조 [에스P32 - 우룸-32세](#).

### Note

현재 ESP32-WROOM-32SE용 FreeRTOS 포트는 대칭 멀티프로세싱 (SMP) 기능을 지원하지 않습니다.

## 개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

- 보드를 호스트 시스템에 연결합니다.
- マイ크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 머신에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 교차 컴파일합니다.
- 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.
- 직렬 연결을 사용하여 실행 중인 애플리케이션을 모니터링 및 디버깅합니다.

## 사전 조건

Espressif 보드에서 FreeRTOS를 시작하기 전에 를 설정해야 합니다. AWS 계정 및 권한

계정을 생성하려면 단원을 참조하십시오. [생성 및 활성화 AWS Account](#).

을 추가하려면 AWS Identity and Access Management(IAM) 사용자를 계정에 연결하려면 단원을 참조하십시오. [사용자 추가](#)의 IAM 사용 설명서. IAM 사용자에게 권한을 부여하려면 AWS IoT 및 FreeRTOS에서 다음 IAM 관리형 정책을 IAM 사용자에게 연결합니다.

- AmazonFreeRTOSFullAccess— IAM 사용자의 모든 FreeRTOS 리소스에 대한 전체 액세스를 허용합니다 (freertos:\*)

- **AWSIoTFullAccess**— IAM 사용자의 모든 액세스를 허용합니다.AWS IoT리소스 (iot:\*).

를 연결하려면**AmazonFreeRTOSFullAccess**IAM 사용자에게 정책

1. [IAM 콘솔](#)로 이동합니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
4. 권한 추가(Add permissions)를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. 검색 상자에 를 입력합니다.AmazonFreeRTOSFullAccess를 목록에서 선택한 다음 를 선택합니다.다음: Review)를 선택합니다.
7. 권한 추가(Add permissions)를 선택합니다.

를 연결하려면 AWSIoTFullAccess IAM 사용자에게 정책

1. [IAM 콘솔](#)로 이동합니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
4. 권한 추가(Add permissions)를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. 검색 상자에 를 입력합니다.AWSIoTFullAccess를 목록에서 선택한 다음 를 선택합니다.다음: Review)를 선택합니다.
7. 권한 추가(Add permissions)를 선택합니다.

IAM에 대한 자세한 내용은 [IAM 사용 설명서](#)를 참조하세요.

정책에 대한 자세한 내용은 단원을 참조하십시오.[IAM의 권한 및 정책](#).

## ESP-IDF v4.2로 시작하기

### Note

이 자습서에서는 Linux 명령을 사용하려면 Bash 셸을 사용해야 합니다.

1. Espressif 하드웨어를 설정합니다.

ESP32-WROOM-32SE 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 다음 () 을 참조하십시오.[ESP32-데브킷C V4 시작하기 가이드](#).

### Important

도착했을 때 설치 단계별가이드의 섹션에서 4단계 (환경 변수 설정) 를 완료할 때까지 따르십시오. 4단계를 완료한 후 중지하고 나머지 단계를 따릅니다.

2. 개발 환경을 설정합니다.

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다. 호스트 시스템의 운영 체제별 지침을 따릅니다.

- [윈도우용 툴체인의 표준 설정 및 필수 구성 요소 \(ESP-IDF v4.2\)](#)
- [macOS용 툴체인 필수 구성 요소 \(ESP-IDF v4.2\) 의 표준 설정](#)
- [Linux용 툴체인 필수 구성 요소 \(ESP-IDF v4.2\) 의 표준 설정](#)

### Important

아래의 “Get ESP-IDF” 지침에 도달하면 다음 단계를 중지하고 다음 지침으로 돌아옵니다.

- 설치를 완료합니다 (리눅스/macOS).

ESP-IDF Windows 설치 프로그램은 필요한 모든 도구를 설치합니다. Linux 및 Mac OS X 플랫폼은 설치를 완료하기 위해 추가 단계가 필요합니다. 다음 단계를 따르십시오.[FreeRTOS 다운로드 및 구성 \(p. 65\)](#).

- 명령줄 창을 열고 FreeRTOS 다운로드 디렉토리로 이동한 후 다음 스크립트를 실행하여 플랫폼에 대한 espressif 툴체인을 다운로드하여 설치합니다.

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

- 그런 다음 다음 명령을 사용하여 터미널 경로에 ESP-IDF 툴체인 도구를 추가합니다.

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

### 직렬 연결 설정

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

### FreeRTOS 다운로드 및 구성

- 환경을 설정한 후 다음 중 하나에서 FreeRTOS를 다운로드할 수 있습니다.
  - [GitHub](#)(지침은 다음 ()을 참조하십시오.[Readme.md](#)파일.)
  - [FreeRTOS 콘솔](#)(올바른 를 선택합니다.Configuration과 하드웨어 플랫폼예를 들어 기기와 일치하는 경우“에 연결AWS IoT- ESP32-데브킷 (ATECC608A 포함)“.)

### Important

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

- 의 단계에 따라 FreeRTOS 데모 프로젝트를 구성합니다.[FreeRTOS 데모 구성 \(p. 18\)](#). 마지막 단계에 도달하면 를 포맷하려면 AWS IoT 신임장을 중지하고 다음 단계를 수행합니다.
- 마이크로칩은 ATECC608A 부품 설정에 도움이 되는 여러 스크립팅 도구를 제공합니다. [freertos/vendors/microchip/example\\_trust\\_chain\\_tool](#) 디렉토리로 이동하여 README.md 파일을 엽니다.
- 디바이스를 프로비저닝하려면 의 지침을 따르십시오.[README.md](#)파일. 이 단계에는 다음 사항이 포함됩니다.
  - AWS에 인증 기관을 생성하고 등록합니다.
  - ATECC608A에서 키를 생성하고 퍼블릭 키 및 디바이스 일련 번호를 내보냅니다.

- c. 디바이스에 대한 인증서를 생성하고 해당 인증서를 AWS에 등록합니다.
5. [개발자 모드 키 프로비저닝 \(p. 29\)](#)의 지침에 따라 CA 인증서 및 디바이스 인증서를 디바이스에 로드합니다.

## AWS 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 콘솔에서 디바이스가 전송하는 메시지를 모니터링할 수 있습니다. AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 []를 선택합니다. 테스트 그런 다음 MQTT 테스트 클라이언트.
3. In구독 주제를 입력합니다. *your-thing-name/example/topic*를 선택한 다음 주제 구독.

[idf.py 스크립트를 사용하여 FreeRTOS 데모 프로젝트를 빌드 후 플래시하고 실행합니다.](#)

에스프레시프의 IDF 유ти리티를 사용할 수 있습니다 (*idf.py* 빌드 파일을 생성하고, 애플리케이션 바이너리를 빌드하고, 바이너리를 디바이스에 플래시합니다).

### Note

일부 설정에서는 포트 옵션을 사용해야 할 수 있습니다. "-p port-name" 함께 *idf.py* 다음 예에서처럼 올바른 포트를 지정합니다.

```
idf.py -p /dev/cu.usbserial-00101301B flash
```

윈도우, 리눅스 및 macOS OS에서 프리어토스를 빌드하고 플래시 (ESP-IDF v4.2)

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

### 윈도우 ("명령" 앱)

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

### 윈도우 ("ESP-IDF 4.x CMD" 앱)

(이 작업은 앱을 열었을 때 이미 완료되었습니다.)

### Linux/macOS

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

3. 에서 `cmake`를 구성하십시오. `.build` 다음 명령을 사용하여 펌웨어 이미지를 디렉터리를 빌드합니다.

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

다음 예제와 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 없으면 빌드에서 펌웨어 바이너리.bin 파일을 생성합니다.

4. 다음 명령을 사용하여 개발 보드의 플래시 메모리를 지웁니다.

```
idf.py erase_flash
```

5. 사용 idf.py 애플리케이션 바이너리를 보드에 플래시할 스크립트.

```
idf.py flash
```

6. 다음 명령을 사용하여 보드의 직렬 포트에서 출력을 모니터링할 수 있습니다.

```
idf.py monitor
```

#### Note

- 다음 예에서처럼 이러한 명령을 결합할 수 있습니다.

```
idf.py erase_flash flash monitor
```

- 특정 호스트 시스템 설정의 경우 다음 예와 같이 보드를 플래시할 때 포트를 지정해야 합니다.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

## CMake를 사용하여 FreeRTOS 빌드 및 플래시

사용하는 것 외에 idf.py 코드를 빌드하고 실행하기 위해 IDF SDK에서 제공하는 스크립트는 CMake를 사용하여 프로젝트를 빌드 할 수도 있습니다. 현재는 유닉스 메이크 파일과 닌자 빌드 시스템을 지원합니다.

### 프로젝트를 빌드 후 플래시하려면

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

#### Windows

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

### Linux/macOS

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

- 빌드 파일을 생성하려면 다음 명령을 입력합니다.

### Unix 메이크파일 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
```

### 닌자 포함

```
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
```

- 플래시를 지운 다음 보드를 플래시합니다.

### Unix 메이크파일 사용

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

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

### 닌자 포함

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

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

## 추가 정보

Espressif ESP32 보드 사용 및 문제 해결에 대한 자세한 내용은 다음 주제를 참조하십시오.

- ESP32 용 자체 CMake 프로젝트에서 FreeRTOS 사용 (p. 54)
- 문제 해결 (p. 58)
- Debugging (p. 59)

## Espressif ESP32-S2 시작하기

이 자습서는 Espressif ESP32-S2 SoC를 시작하는 방법을 보여줍니다.[ESP32-S2-사울라-1 개발 보드.](#)

### 개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

- 보드를 호스트 시스템에 연결합니다.
- マイ크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 머신에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 교차 컴파일합니다.

4. 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.
5. 직렬 연결을 사용하여 실행 중인 애플리케이션을 모니터링 및 디버깅합니다.

## 사전 조건

Espressif 보드에서 FreeRTOS를 시작하려면 먼저 를 설정해야 합니다.AWS계정 및 권한 계정을 생성하려면 단원을 참조하십시오.[생성 후 활성화AWSAccount](#).

을 추가하려면AWS Identity and Access Management(IAM) 사용자를 계정에 연결하려면 단원을 참조하십시오.[사용자 추가](#)의IAM 사용 설명서. IAM 사용자에게 권한을 부여하려면AWS IoT 및 FreeRTOS에서 다음 IAM 관리형 정책을 IAM 사용자에게 연결합니다.

- **AmazonFreeRTOSFullAccess**— IAM 사용자의 모든 FreeRTOS 리소스에 대한 전체 액세스를 허용합니다 (freertos: \*).
- **AWSIoTFullAccess**— IAM 사용자의 모든 액세스를 허용합니다.AWS IoT리소스 (iot: \*).

를 연결하려면**AmazonFreeRTOSFullAccess**IAM 사용자에게 정책

1. [IAM 콘솔](#)로 이동합니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
4. 권한 추가(Add permissions)를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. 검색 상자에 를 입력합니다.**AmazonFreeRTOSFullAccess**목록에서 선택한 다음 를 선택합니다.다음: Review)를 선택합니다.
7. 권한 추가(Add permissions)를 선택합니다.

를 연결하려면**AWSIoTFullAccess**IAM 사용자에게 정책

1. [IAM 콘솔](#)로 이동합니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
4. 권한 추가(Add permissions)를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. 검색 상자에 를 입력합니다.**AWSIoTFullAccess**목록에서 선택한 다음 를 선택합니다.다음: Review)]를 선택합니다.
7. 권한 추가(Add permissions)를 선택합니다.

IAM에 대한 자세한 내용은 [IAM 사용 설명서](#)를 참조하세요.

정책에 대한 자세한 내용은 단원을 참조하십시오.[IAM의 권한 및 정책](#).

## ESP-IDF v4.2로 시작하기

### Note

이 자습서의 Linux 명령을 사용하려면 Bash 셸을 사용해야 합니다.

1. Espressif 하드웨어를 설정합니다.

ESP32-S2 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 를 참조하십시오.[ESP32-S2-사울라-1 시작하기 가이드](#).

### Important

도착했을 때 시작 Espressif 안내선의 섹션에서 중지한 후 다음 단계를 수행합니다.

#### 2. 개발 환경을 설정합니다.

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다. 호스트 시스템의 운영 체제별 지침을 따릅니다.

- [윈도우용 툴체인의 표준 설정 및 필수 구성 요소 \(ESP-IDF v4.2\).](#)
- [macOS용 툴체인 필수 구성 요소 \(ESP-IDF v4.2\) 의 표준 설정.](#)
- [Linux용 툴체인 필수 구성 요소 \(ESP-IDF v4.2\) 의 표준 설정.](#)

### Important

아래의 “Get ESP-IDF” 지침에 도달하면 다음 단계를 중지한 다음 이 페이지의 지침으로 돌아갑니다.

#### 3. 설치를 완료합니다 (리눅스/macOS).

ESP-IDF Windows 설치 프로그램은 필요한 모든 도구를 설치합니다. Linux 및 Mac OS X 플랫폼은 설치를 완료하기 위해 추가 단계가 필요합니다. 다음 단계를 따르십시오.[FreeRTOS 다운로드 및 구성 \(p. 70\).](#)

- a. 명령줄 창을 열고 FreeRTOS 다운로드 디렉토리로 이동한 후 다음 스크립트를 실행하여 플랫폼에 대한 espressif 툴체인을 다운로드하여 설치합니다.

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

- b. 그런 다음 다음 명령을 사용하여 터미널 경로에 ESP-IDF 툴체인 도구를 추가합니다.

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

#### 4. 직렬 연결을 설정합니다.

- a. 호스트 시스템과 ESP32-S2 사이에 직렬 연결을 설정하려면 UART Bridge VCP 드라이버에 대한 CP210x USB를 설치합니다. [Silicon Labs](#)에서 이러한 드라이버를 다운로드할 수 있습니다.
- b. 다음 단계를 따르십시오.[ESP32-S2 직렬 연결 설정.](#)
- c. 직렬 연결을 설정한 후 보드의 연결을 위한 직렬 포트를 기록해 두십시오. 데모를 빌드할 때 이 정보가 필요합니다.

## FreeRTOS 다운로드 및 구성

환경을 설정한 후 다음 중 하나에서 FreeRTOS를 다운로드할 수 있습니다.

- [GitHub](#)(지침은 다음 ()[Readme.md](#) 파일.)
- [FreeRTOS 콘솔](#)(올바른 항목을 선택합니다.Configuration과 하드웨어 플랫폼 예를 들어 기기와 일치하는 경우 "우"에 연결 AWS IoT-ESP32S2-솔라-1".)

## FreeRTOS 데모 애플리케이션 구성

1. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행 중인 경우 “ESP-IDF 4.x CMD” 앱 (ESP-IDF 툴체인을 설치할 때 이 옵션을 포함하는 경우) 또는 “명령 프롬프트” 앱을 여십시오.
2. Python3이 설치되어 있는지 확인하려면 를 실행합니다.

```
python --version
```

설치된 버전이 표시됩니다. Python 3.0.1 이상이 설치되어 있지 않으면 를 사용하여 설치할 수 있습니다.[파이썬 웹 사이트](#).

- 필요한 경우 AWS 명령줄 인터페이스 (CLI) AWS IoT 명령. Windows를 실행하는 경우 easy\_install awscli 명령을 설치하려면 AWS “명령” 또는 “ESP-IDF 4.x CMD” 앱의 CLI를 사용할 수 있습니다.

macOS 또는 Linux를 실행하는 경우 단원을 참조하십시오.[다음 설치 AWSCLI](#).

- 실행

```
aws configure
```

을 (를) 구성하고 AWS CLI와 함께 AWS 액세스 키 ID, 보안 액세스 키 및 기본값 AWS 리전. 자세한 내용은 단원을 참조하십시오.[구성 AWSCLI](#).

- 다음 명령을 사용하여 다음 명령을 설치합니다. AWS Python용 SDK (boto3):

- 윈도우의 경우 “명령” 또는 “ESP-IDF 4.x CMD” 앱에서

```
easy_install boto3
```

- macOS 또는 Linux에서는 를 실행합니다.

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

그런 다음 를 실행합니다.

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

FreeRTOS 포함 SetupAWS.py에 연결하기 위해 Espressif 보드를 더 쉽게 설정할 수 있게 해주는 스크립트 AWS IoT.

구성 스크립트를 실행하려면

- 스크립트를 구성하려면 `freertos/tools/aws_config_quick_start/configure.json`을 열고 다음 속성을 설정합니다.

**afr\_source\_dir**

컴퓨터에서 `freertos` 디렉터리의 전체 경로입니다. 이 경로를 지정하기 위해 슬래시를 사용하고 있는지 확인합니다.

**thing\_name**

보드를 나타내는 AWS IoT 사물에 할당할 이름.

**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)
2. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행 중인 경우 “ESP-IDF 4.x CMD” 또는 “명령” 앱을 엽니다.
  3. 로 이동합니다.[freertos/tools/aws\\_config\\_quick\\_start](#)디렉토리 및 실행

```
python SetupAWS.py setup
```

이 스크립트는 다음 작업을 수행합니다.

- 를 생성합니다.AWS IoT사물, 인증서 및 정책
- 를 연결합니다.AWS IoT인증서에 대한 정책 및 인증서에 대한 정책AWS IoT사물.
- 채우기aws\_clientcredential.h파일을 사용하여 파일을 엽니다.AWS IoT엔드포인트, Wi-Fi SSID 및 자격 증명입니다.
- 인증서와 프라이빗 키에 형식을 지정하고aws\_clientcredential\_keys.h헤더 파일.

#### Note

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

에 대한 자세한 내용[SetupAWS.py](#)참조,[README.md](#)의[freertos/tools/aws\\_config\\_quick\\_start](#)디렉토리.

## AWS 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT디바이스가 보내는 메시지를 모니터링할 수 있는 콘솔AWS클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 []를 선택합니다.테스트그런 다음 를 선택합니다.MQTT 테스트 클라이언트.
3. 구독 주제에 [your-thing-name/example/topic](#)을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제에 여러 번 보냈습니다.

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

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

### 윈도우, 리눅스 및 macOS OS에서 프리어토스를 빌드하고 플래시 (ESP-IDF v4.2)

사용[idf.py](#)스크립트를 사용하여 프로젝트를 빌드하고 바이너리를 장치에 플래시합니다.

#### Note

일부 설정에서는 포트 옵션을 사용해야 할 수 있습니다.-p port-name와[idf.py](#)다음 예에서처럼 올바른 포트를 지정합니다.

```
idf.py -p /dev/cu.usbserial-00101301B flash
```

### 프로젝트를 빌드 후 플래시하려면

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

윈도우 (“명령” 앱)

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

윈도우 (“ESP-IDF 4.x CMD” 앱)

(이 작업은 앱을 열었을 때 이미 완료되었습니다.)

Linux/macOS

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

3. 서에서 cmake를 구성하십시오.build다음 명령을 사용하여 펌웨어 이미지를 빌드하고 빌드합니다.

```
idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
```

다음 예와 같은 결과가 표시됩니다.

```
Executing action: all (aliases: build)
Running cmake in directory /path/to/hello_world/build
Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1 -
DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -DCCACHE_ENABLE=0 /
path/to/hello_world"...
-- The C compiler identification is GNU 8.4.0
-- The CXX compiler identification is GNU 8.4.0
-- The ASM compiler identification is GNU

... (more lines of build system output)

[1628/1628] Generating binary image from built executable
esptool.py v3.0
Generated /path/to/hello_world/build/aws_demos.bin

Project build complete. To flash, run this command:
esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip
esp32s2 write_flash --flash_mode dio --flash_size detect --flash_freq 80m 0x1000
build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin
0x16000 build/ota_data_initial.bin 0x20000 build/aws_demos.bin
or run 'idf.py -p (PORT) flash'
```

오류가 없으면 빌드에서 펌웨어 바이너리.bin 파일을 생성합니다.

4. 다음 명령을 사용하여 개발 보드의 플래시 메모리를 지웁니다.

```
idf.py erase_flash
```

5. 사용 idf.py 애플리케이션 바이너리를 보드에 플래시하는 스크립트입니다.

```
idf.py flash
```

6. 다음 명령을 사용하여 보드의 직렬 포트에서 출력을 모니터링할 수 있습니다.

```
idf.py monitor
```

#### Note

- 다음 예에서처럼 이러한 명령을 결합할 수 있습니다.

```
idf.py erase_flash flash monitor
```

- 특정 호스트 시스템 설정의 경우 다음 예와 같이 보드를 플래시할 때 포트를 지정해야 합니다.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

## CMake를 사용하여 FreeRTOS 빌드 및 플래시

사용하는 것 외에 idf.py 코드를 빌드하고 실행하기 위해 IDF SDK에서 제공하는 스크립트는 CMake를 사용하여 프로젝트를 빌드 할 수도 있습니다. 현재는 유닉스 메이크 파일과 닌자 빌드 시스템을 지원합니다.

프로젝트를 빌드 후 플래시하려면

- 명령줄 창에서 FreeRTOS 다운로드 디렉터리의 루트로 이동합니다.
- 다음 스크립트를 실행하여 셀의 PATH에 ESP-IDF 도구를 추가합니다.

- Windows

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

- Linux/macOS

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

- 빌드 파일을 생성할 때는 다음 명령을 입력합니다.

- Unix 메이크 파일

```
cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
```

- 닌자와 함께

```
cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
```

- 프로젝트를 빌드합니다.

- Unix 메이크 파일

```
make -C ./YOUR_BUILD_DIRECTORY -j8
```

- 닌자와 함께

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

- 플래시를 지운 다음 보드를 플래시합니다.

- Unix 메이크 파일

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

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

- 닌자와 함께

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

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

## 추가 정보

Espressif ESP32 보드 사용 및 문제 해결에 대한 자세한 내용은 다음 주제를 참조하십시오.

- ESP32용 자체 CMake 프로젝트에서 FreeRTOS 사용 (p. 54)
- 문제 해결 (p. 58)
- Debugging (p. 59)

## Infineon XMC4800 IoT 연결 키트 시작하기

이 자습서에서는 Infineon XMC4800 IoT 연결 키트를 시작하기 위한 지침을 제공합니다. Infineon XMC4800 IoT 연결 키트가 없는 경우 AWS 파트너 디바이스 카탈로그를 구입하시기 바랍니다. [파트너](#).

보드와 직렬 연결을 열고 로깅 및 디버깅 정보를 보려는 경우 XMC4800 IoT Connectivity Kit 외에 3.3V USB/직렬 변환기도 필요합니다. CP2104는 Adafruit의 [CP2104 Friend](#)와 같은 보드에서 널리 사용되는 일반적인 USB/직렬 변환기입니다.

시작하려면 먼저 구성해야 합니다. AWS IoT 및 FreeRTOS 다운로드를 다운로드하여 디바이스를 AWS 클라우드 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 딜렉터리에 대한 경로는 라고 합니다. [freertos](#).

## 개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

- マイクロ 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
- 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
- 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

## 개발 환경 설정

FreeRTOS는 Infineon의 DAVE 개발 환경을 사용하여 XMC4800 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 가이드](#)를 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 3.3V 가 필요합니다. 보드의 IOREF 점퍼를 사용하여 보드의 신호를 5V로 변경하지 마십시오.

케이블이 연결된 상태로 [GNU Screen](#)과 같은 터미널 에뮬레이터에서 직렬 연결을 엽니다. 보드 속도는 기본적으로 115200로 설정되며 8 데이터 비트, 패리티 없음, 1 정지 비트가 설정됩니다.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모를 실행하기 전에 MQTT 클라이언트를 AWS IoT 디바이스가 보낸 메시지를 모니터링하는 콘솔 AWS 클라우드

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.

2. 탐색 창에서 []를 선택합니다. 테스트 그런 다음 선택합니다. MQTT 테스트 클라이언트 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 "Hello World!" 구독한 주제에 여러 번 전송되었습니다.

## FreeRTOS 데모 프로젝트 빌드 및 실행

### DAVE로 FreeRTOS 데모를 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 폰 디렉터리에서 데모 프로젝트는 `projects/infineon/xmc4800_iotkit/dave4/aws_demos`.

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

4. [마침]을 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 클라이언트에 디바이스가 전송한 MQTT 메시지가 표시되어야 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----
```

## CMak로 FreeRTOS 데모 빌드

용 IDE FreeRTOS 개발을 사용하지 않으려는 경우 CMak를 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 [CMak를 사용하여 프리어토스와 사용 \(p. 22\)](#) 단원을 참조하십시오. (MinGW는 네이티브 Microsoft Windows 애플리케이션을 위한 최소한의 개발 환경입니다.)

### CMak로 FreeRTOS 데모를 빌드하려면

1. GNU Arm Embedded Toolchain을 설정합니다.

- a. [Arm Embedded Toolchain 다운로드 페이지](#)에서 Windows 버전의 도구 체인을 다운로드합니다.

### Note

"8-2018-q4-major" 버전에는 "objcopy" 유ти리티에 대해 [보고된 버그](#)가 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 조건 \(p. 23\)](#)을 참조하십시오.

3. 생성된 빌드 파일을 포함할 폴더(**build-folder**)를 생성합니다.
4. 디렉터리를 FreeRTOS 다운로드 디렉터리로 변경합니다.**freertos**) 및 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. [JLINK \(p. 76\)](#)를 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. JLINK 실행 파일을 사용하여 이미지를 플래시합니다.

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

애플리케이션 로그는 보드에서 설정한 [직렬 연결 \(p. 76\)](#)을 통해 볼 수 있어야 합니다.

## 문제 해결

아직 구성하지 않은 경우 반드시 구성하십시오. AWS IoT 및 FreeRTOS 다운로드를 다운로드하여 디바이스를 AWS 클라우드 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요.

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 단원을 참조하십시오. [시작하기 문제 해결 \(p. 21\)](#).

## Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트 시작하기

이 자습서에서는 Infineon OPTIGA Trust X 보안 요소 및 XMC4800 IoT 연결 키트를 시작하기 위한 지침을 제공합니다. [Infineon XMC4800 IoT 연결 키트 시작하기 \(p. 75\)](#) 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 연결 키트 AWS 파트너 디바이스 카탈로그를 사용하여 디바이스를 구입하시기 바랍니다. [파트너](#).
2. 보안 확장 팩:

- 보안 요소 - Infineon OPTIGA Trust X.

방문하기 AWS 파트너 디바이스 카탈로그를 통해 구입하시기 바랍니다. [파트너](#).

- 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 세 가지가 필요합니다. male-to-male 점퍼 와이어 Infineon에 직렬 케이블을 연결하도록 (수신 (RX), 전송 (TX) 및 접지 (GND))) MyIoT 어댑터 보드.

시작하려면 먼저 구성해야 합니다. AWS IoT 및 FreeRTOS 다운로드를 사용하여 디바이스를 AWS 클라우드에 등록합니다. 옵션 #2: 온보드 프라이빗 키 생성 (p. 29) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 디렉토리에 대한 경로는 라고 합니다. [freertos](#).

## 개요

이 자습서에 포함된 단계는 다음과 같습니다.

- マイクロコントローラー ボード用 内蔵型 アプリケーションを開発し、デバッグするための ソフトウェアをホスト マシンにインストールします。
- FreeRTOS デモ アプリケーションを ZIP 形式で 提供されるイメージとして 提供されます。
- アプリケーションイメージを ボードにロードする後、アプリケーションを実行します。
- モニタリングとデバッグを実行するため、直列接続によって ボード上で実行されるアプリケーションと並行して動作します。

## 개발 환경 설정

FreeRTOS는 Infineon의 DAVE 개발 환경을 사용하여 XMC4800 프로그래밍을 수행합니다. 온보드 디버거와 통신하려면 시작하기 전에 DAVE와 몇 가지 J-Link 드라이버를 다운로드하여 설치합니다.

### DAVE 설치

- Infineon의 [DAVE 소프트웨어 다운로드](#) 페이지로 이동합니다.
- 운영 체제에 맞는 DAVE 패키지를 선택하고 등록 정보를 제출합니다. 등록한 후 .zip 파일의 다운로드 링크가 포함된 확인 이메일을 수신해야 합니다.
- 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.

- DAVE를 시작하려면 압축을 푸는 DAVE\_version\_os\_date.zip 폴더에 있는 실행 파일을 실행합니다.

자세한 내용은 [DAVE 빠른 시작 가이드](#)를 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 3.3V 가 필요합니다. 보드의 IOREF 점퍼를 사용하여 보드의 신호를 5V로 변경하지 마십시오.

케이블이 연결된 상태로 [GNU Screen](#)과 같은 터미널 에뮬레이터에서 직렬 연결을 엽니다. 보드 속도는 기본적으로 115200로 설정되며 8 데이터 비트, 패리티 없음, 1 정지 비트가 설정됩니다.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 디바이스가 AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트 그런 다음 선택합니다. MQTT 테스트 클라이언트를 눌러 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제로 여러 번 전송되었습니다.

## FreeRTOS 데모 프로젝트 빌드 및 실행

### DAVE로 FreeRTOS 데모 가져오기

1. DAVE를 시작합니다.
2. DAVE에서 파일을 선택한 후 가져오기를 선택합니다. Infineon 폴더를 확장하고 DAVE Project(DAVE 프로젝트)를 선택한 후 다음을 선택합니다.
3. Import DAVE Projects(DAVE 프로젝트 가져오기) 창에서 Select Root Directory(루트 디렉토리 선택)를 선택하고 Browse(가져오기)를 선택한 다음 XMC4800 데모 프로젝트를 선택합니다.

FreeRTOS 다운로드의 압축을 폰 디렉토리에서 데모 프로젝트는 `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`.

Copy Projects Into Workspace(Workspace에 프로젝트 복사)가 선택 해제된 상태인지 확인합니다.

4. [마침]을 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

- 프로젝트 메뉴에서 Rebuild Active Project(활성 프로젝트 다시 빌드)를 선택하여 aws\_demos를 다시 빌드하고 구성 변경 사항이 적용되었는지 확인합니다.
- 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.
- GDB SEGGER J-Link Debugging(GDB SEGGER J-Link 디버깅)을 두 번 클릭하여 디버그 확인을 생성합니다. Debug(디버그)를 선택합니다.
- 디버거가 main()의 종단점에서 중지되면 실행 메뉴에서 다시 시작을 선택합니다.

여기서 옵션 #2: 온보드 프라이빗 키 생성 (p. 29)의 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----
```

## CMak로 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 [CMak를 사용하여 프리](#)

어토스와 사용 (p. 22) 단원을 참조하십시오. ([MinGW](#)는 네이티브 Microsoft Windows 애플리케이션을 위한 최소한의 개발 환경입니다.)

용 IDE FreeRTOS 개발을 사용하지 않으려는 경우 CMak를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

### CMak로 FreeRTOS 데모를 빌드하려면

1. GNU Arm Embedded Toolchain을 설정합니다.

- a. [Arm Embedded Toolchain 다운로드 페이지](#)에서 Windows 버전의 도구 체인을 다운로드합니다.

#### Note

objcopy 유ти리티에 [보고된 버그](#)가 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 조건 \(p. 23\)](#)을 참조하십시오.

3. 생성된 빌드 파일을 포함할 폴더([build-folder](#))를 생성합니다.
4. 디렉터리를 FreeRTOS 다운로드 디렉터리로 변경합니다.[freertos](#)) 및 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. [JLINK \(p. 76\)](#)를 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. JLINK 실행 파일을 사용하여 이미지를 플래시합니다.

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

애플리케이션 로그는 보드에서 설정한 [직렬 연결 \(p. 76\)](#)을 통해 볼 수 있어야 합니다. [옵션 #2: 온보드 프라이빗 키 생성 \(p. 29\)](#)의 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 \(p. 21\)](#) 단원을 참조하십시오.

## MW32x 시작하기|AWS IoT스타터 키트

이 AWS IoT 단일 마이크로 컨트롤러 칩에 802.11b/g/n Wi-Fi를 통합하는 NXP의 최신 통합 Cortex M4 마이크로 컨트롤러인 88MW320/88MW322를 기반으로 하는 개발 키트입니다. 개발 키트는 FCC 인증을 받았습니다. 자세한 내용은 단원을 참조하십시오.[AWS 파트너 디바이스 카탈로그](#) 파트너로부터 하나를 구매하십시오. 또한 88MW320/88MW322 모듈은 FCC 인증을 받았으며 사용자 지정 및 대량 판매가 가능합니다.

이 시작 안내서는 호스트 컴퓨터의 SDK와 함께 애플리케이션을 교차 컴파일한 다음 SDK와 함께 제공된 도구를 사용하여 생성된 바이너리 파일을 보드에 로드하는 방법을 보여줍니다. 응용 프로그램이 보드에서 실행되기 시작하면 호스트 컴퓨터의 Serial 콘솔에서 디버깅하거나 상호 작용할 수 있습니다.

Ubuntu 16.04는 개발 및 디버깅을 위해 지원되는 호스트 플랫폼입니다. 다른 플랫폼을 사용할 수 있지만 공식적으로 지원되지는 않습니다. 호스트 플랫폼에 소프트웨어를 설치할 수 있는 권한이 있어야 합니다. SDK를 빌드하는 데 필요한 외부 도구는 다음과 같습니다.

- Ubuntu 16.04 호스트 플랫폼
- ARM 툴체인 버전 4\_9\_2015q3
- Eclipse 4.9.0 IDE

ARM 툴체인은 애플리케이션과 SDK를 교차 컴파일하는 데 필요합니다. SDK는 최신 버전의 툴체인을 활용하여 이미지 공간을 최적화하고 더 적은 공간에 더 많은 기능을 제공합니다. 이 가이드에서는 도구 체인의 버전 4\_9\_2015q3를 사용하고 있다고 가정합니다. 이전 버전의 툴체인을 사용하는 것은 권장되지 않습니다. 개발 키트는 무선 마이크로 컨트롤러 데모 프로젝트 펌웨어로 미리 플래시됩니다.

### 주제

- [하드웨어 설정 \(p. 84\)](#)
- [개발 환경 설정 \(p. 84\)](#)
- [FreeRTOS 데모 프로젝트 빌드 및 실행 \(p. 87\)](#)
- [Debugging \(p. 98\)](#)
- [문제 해결 \(p. 99\)](#)

## 하드웨어 설정

미니 USB 대 USB 케이블을 사용하여 MW32x 보드를 노트북에 Connect. 미니 USB 케이블을 보드에 있는 유일한 미니 USB 커넥터에 Connect. 젠더를 변경할 필요가 없습니다.

보드가 랩톱 또는 데스크탑 컴퓨터에 연결되어 있으면 외부 전원 공급 장치가 필요하지 않습니다.

이 USB 연결은 다음을 제공합니다.

- 보드에 대한 콘솔 액세스. 가상 tty/com 포트는 콘솔에 액세스하는 데 사용할 수 있는 개발 호스트에 등록됩니다.
- JTAG 액세스. 이는 펌웨어 이미지를 보드의 RAM 또는 플래시로 로드하거나 언로드하거나 디버깅 목적으로 사용할 수 있습니다.

## 개발 환경 설정

개발 목적을 위해 최소 요구 사항은 ARM 툴체인과 SDK와 함께 번들로 제공되는 툴입니다. 다음 단원에는 ARM 도구 체인 설정에 대한 세부 정보가 나와 있습니다.

### GNU 툴체인

SDK는 공식적으로 GCC 컴파일러 툴체인을 지원합니다. GNU ARM용 크로스 컴파일러 툴체인은 [GNU 암 임 베디드 툴체인 4.9-2015-Q3 업데이트](#).

빌드 시스템은 기본적으로 GNU 도구 체인을 사용하도록 구성됩니다. Makefiles는 GNU 컴파일러 툴체인 바이너리가 사용자의 PATH에서 사용할 수 있으며 Makefiles에서 호출될 수 있다고 가정합니다. Makefiles는 또한 GNU 툴체인 바이너리의 파일 이름에 접두사가 arm-none-eabi-.

GCC 툴체인은 GDB와 함께 사용하여 OpenOCD (SDK와 함께 번들로 제공)로 디버깅할 수 있습니다. JTAG와 인터페이스하는 소프트웨어를 제공합니다.

gcc-암 내장 툴체인의 버전 4\_9\_2015q3을 사용하는 것이 좋습니다.

## Linux 툴체인 설정 프로시저

Linux에서 GCC 도구 체인을 설정하려면 다음 단계를 따릅니다.

- 에서 제공되는 도구 체인 tarball을 다운로드합니다.[GNU 암 임베디드 툴체인 4.9-2015-Q3 업데이트](#). 파일은 다음과 같습니다.gcc-arm-none-eabi-4\_9-2015q3-20150921-linux.tar.bz2.
- 선택한 디렉토리에 파일을 복사합니다. 디렉토리 이름에 공백이 없어야 합니다.
- 다음 명령을 사용하여 파일을 풀 수 있습니다.

```
tar -vxf filename
```

- 설치된 툴체인의 경로를 시스템 PATH에 추가합니다. 예를 들어, 의 끝에 다음 행을 추가합니다..profile에 있는 파일/home/**user-name**디렉토리.

```
PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
```

### Note

Ubuntu의 최신 배포에는 데비안 버전의 GCC 크로스 컴파일러가 함께 제공될 수 있습니다. 그렇다면 네이티브 크로스 컴파일러를 제거하고 위의 설정 절차를 따라야 합니다.

## Linux 개발 호스트로 작업하기

Ubuntu 또는 Fedora와 같은 모든 최신 Linux 데스크톱 배포판을 사용할 수 있습니다. 하지만 최신 릴리스로 업그레이드하는 것이 좋습니다. 다음 단계는 Ubuntu 16.04에서 작동하도록 확인되었으며 해당 버전을 사용하고 있다고 가정합니다.

### 패키지 설치

SDK에는 새로 설치된 Linux 시스템에서 개발 환경을 빠르게 설정할 수 있는 스크립트가 포함되어 있습니다. 이 스크립트는 머신 유형을 자동으로 감지하고 C 라이브러리, USB 라이브러리, FTDI 라이브러리, ncurses, python 및 라텍스를 포함한 적절한 소프트웨어를 설치하려고 시도합니다. 이 섹션에서는 일반 디렉토리 이름 amzsd़k\_bundle-x.y.z를 나타냅니다.AWSSDK 루트 디렉토리. 실제 디렉토리 이름은 다를 수 있습니다. 루트 권한이 있어야 합니다.

- 로 이동합니다.**amzsd़k\_bundle-x.y.z/**디렉토리를 선택하고 이 명령을 실행합니다.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/install_pkgs.sh
```

### Sudo 방지

이 가이드의 내용flashprog작업은 다음을 사용합니다.flashprog.py아래에 설명 된대로 보드의 NAND를 플래시하는 스크립트. 마찬가지로ramload작업은 다음을 사용합니다.ramload.pyNAND를 깜박이지 않고 호스트에서 마이크로 컨트롤러의 RAM으로 펌웨어 이미지를 직접 복사하는 스크립트입니다.

Linux 개발 호스트를 구성하여 flashprog과 ramload 필요 없는 작업 sudo 매번 명령을 실행합니다. 이를 위해 다음 명령을 실행합니다.

```
./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. 이렇게 하면 장치 감지가 트리거됩니다. 다음과 같은 메시지가 표시되어야 합니다./var/log/messages 파일 또는 실행 후 dmesg 명령.

```
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
```

두 개의 TyUSB 장치가 생성되어 있는지 확인합니다. 두 번째 ttyUSB는 직렬 콘솔입니다. 위 예제에서는 이름이 "ttyUSB1"입니다.

이 안내서에서는 미니 콤을 사용하여 직렬 콘솔 출력을 확인합니다. 다음과 같은 다른 직렬 프로그램을 사용할 수도 있습니다.putty. 다음 명령을 실행하여 설정 모드에서 minicom을 실행합니다.

```
minicom -s
```

미니콤에서 로 이동합니다. 직렬 포트 설정 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 키. 화면에 해시 (#) 가 표시됩니다.

#### Note

개발 보드에는 FTDI 실리콘 장치가 포함됩니다. FTDI 디바이스는 호스트에 대해 두 개의 USB 인터페이스를 제공합니다. 첫 번째 인터페이스는 MCU의 JTAG 기능과 연결되며 두 번째 인터페이스는 MCU의 물리적 UARTRX 포트와 연결됩니다.

## OpenOCD 설치

OpenOCD는 임베디드 대상 장치에 대한 디버깅, 시스템 내 프로그래밍 및 경계 스캔 테스트를 제공하는 소프트웨어입니다.

OpenOCD 버전 0.9가 필요합니다. Eclipse 기능에도 필요합니다. Linux 호스트에 버전 0.7과 같은 이전 버전이 설치된 경우 현재 사용 중인 Linux 배포판에 적합한 명령을 사용하여 해당 저장소를 제거합니다.

표준 리눅스 명령을 실행하여 OpenOCD를 설치합니다.

```
apt-get install openocd
```

위의 명령으로 버전 0.9 이상을 설치하지 않으면 다음 절차에 따라 openocd 소스 코드를 다운로드하여 컴파일하십시오.

OpenOCD를 설치하려면

1. 다음 명령을 실행하여 libusb-1.0을 설치합니다.

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

2. 에서 openocd 0.9.0 소스 코드를 다운로드합니다.<http://openocd.org/>.
3. openocd를 추출하고 압축을 폰 디렉터리로 이동합니다.
4. 다음 명령으로 openocd를 구성합니다.

```
./configure --enable-ftdi --enable-jlink
```

5. make 유ти리티를 실행하여 openocd를 컴파일합니다.

```
make install
```

## Eclipse를 설정하기

### Note

이 단원에서는 의 단계를 완료했다고 가정합니다.[Sudo 방지 \(p. 85\)](#)

Eclipse는 애플리케이션 개발 및 디버깅에 선호되는 IDE입니다. 스레드 인식 디버깅을 포함하여 통합 디버깅 지원을 갖춘 풍부하고 사용자 친화적인 IDE를 제공합니다. 이 섹션에서는 지원되는 모든 개발 호스트에 대한 일반적인 Eclipse 설정에 대해 설명합니다.

Eclipse를 설정하려면

1. JRE (Java Runtime Environment) 를 다운로드하여 설치합니다.

Eclipse에서는 JRE를 설치해야 합니다. Eclipse를 설치한 후에 설치할 수 있지만 먼저 설치하는 것이 좋습니다. JRE 버전 (32/64비트) 은 Eclipse 버전 (32/64비트) 과 일치해야 합니다. JRE는 에서 다운로드할 수 있습니다.[Java SE 런타임 환경 8 다운로드](#)오라클 웹 사이트에서

2. 에서 “C/C++ 개발자용 Eclipse IDE”를 다운로드하여 설치합니다.<http://www.eclipse.org>. Eclipse 버전 4.9.0 이상이 지원됩니다. 설치하려면 다운로드한 아카이브를 추출하기만 하면 됩니다. 플랫폼별 Eclipse 실행 파일을 실행하여 애플리케이션을 시작합니다.

## FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모 프로젝트를 실행하는 방법은 두 가지가 있습니다.

- 명령줄을 사용합니다.
- Eclipse IDE를 사용합니다.

이 주제에서는 두 옵션에 대해 설명합니다.

### 프로비저닝

- 테스트 애플리케이션 또는 데모 애플리케이션을 사용하는지에 따라 다음 파일 중 하나로 프로비저닝 데이터를 설정합니다.
  - ./tests/common/include/aws\_clientcredential.h
  - ./demos/common/include/aws\_clientcredential.h

예:

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"  
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"  
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

#### Note

다음과 같은 Wi-Fi 보안 값을 입력할 수 있습니다.

- eWiFiSecurityOpen
- eWiFiSecurityWEP
- eWiFiSecurityWPA
- eWiFiSecurityWPA2

SSID와 암호는 큰따옴표로 둑어야 합니다.

### 명령줄을 사용하여 FreeRTOS 데모 빌드 및 실행

1. 다음 명령을 사용하여 데모 응용 프로그램 빌드를 시작합니다.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -  
DAFR_ENABLE_TESTS=0
```

다음 예제와 동일한 출력을 얻어야 합니다.

```

marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -Bbuild -DAFR_ENABLE_TESTS=0

=====Configuration for Amazon FreeRTOS=====
Version:          vl.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, demo_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$ 

```

- 빌드 디렉터리로 이동합니다.

```
cd build
```

- make 유ти리티를 실행하여 응용 프로그램을 빌드합니다.

```
make all -j4
```

다음 그림과 동일한 출력을 얻어야 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$ 

```

- 다음 명령을 사용하여 테스트 애플리케이션을 빌드합니다.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -  
DAFR_ENABLE_TESTS=1  
cd build  
make all -j4
```

를 실행합니다.cmake명령 사이를 전환할 때마다aws\_demos project및aws\_tests project.

5. 개발 보드의 플래시에 펌웨어 이미지를 씁니다. 개발 보드가 재설정되면 펌웨어가 실행됩니다. 이미지를 마이크로 컨트롤러로 플래시하기 전에 SDK를 빌드해야 합니다.
  - a. 펌웨어 이미지를 플래시하기 전에 공통 구성 요소 Layout 및 Boot2를 사용하여 개발 보드의 플래시를 준비하십시오. 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
```

이flashprog명령은 다음을 시작합니다.

- 레이아웃 — flashprog 유ти리티는 먼저 플래시에 레이아웃을 쓰도록 지시합니다. 레이아웃은 플래시의 파티션 정보와 유사합니다. 기본 레이아웃은 다음 위치에 있습니다./lib/third\_party/mcu\_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt.
- Boot2 — WMSDK에서 사용하는 부트 로더입니다. 이flashprog또한 명령은 플래시에 부트로더를 씁니다. 마이크로 컨트롤러의 펌웨어 이미지를 플래시한 후 로드하는 것은 부트로더의 작업입니다. 아래 그림과 동일한 출력을 얻었는지 확인하십시오.

```
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 플래시에 있어야 하는 자체 펌웨어가 있습니다. 이flashprog.pyBoot2 부트 로더 및 MCU 펌웨어를 플래시하는 것과 동일한 방식으로 Wi-Fi 펌웨어를 플래시하는 유ти리티입니다. 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
```

명령의 출력이 아래 그림과 유사하게 출력되는지 확인합니다.

```
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 표시되어야 합니다.  
e. 테스트 앱을 실행하려면 aws\_tests.binバイ너리가 같은 디렉토리에 위치합니다.

```
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 출력은 아래 그림에 표시된 것과 유사하게 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 a2wtm15blvjj8-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 유ти리티를 사용하여 호스트에서 마이크로 컨트롤러 RAM으로 직접 마이크로 컨트롤러 이미지를 복사합니다. 이미지는 플래시에 복사되지 않으므로 마이크로 컨트롤러를 재부팅하면 손실됩니다.

실행 파일을 즉시 시작하기 때문에 펌웨어 이미지를 SRAM에 로드하는 것이 더 빠른 작업입니다. 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
```

아래 그림에 명령 출력이 나와 있습니다.

```
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: wmcrc.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcrc.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcrc.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
75072 bytes written at address 0x00100000
8 bytes written at address 0x00112540
468 bytes written at address 0x20000040
downloaded 75548 bytes in 0.636127s (115.979 KiB/s)
verified 75548 bytes in 0.959023s (76.930 KiB/s)
shutdown command invoked
```

명령 실행이 완료되면 데모 앱의 로그가 표시되어야 합니다.

### 아이클립스 IDE를 사용하여 FreeRTOS 데모 빌드 및 실행

1. Eclipse 작업 영역을 설정하기 전에 다음을 실행해야 합니다.cmake명령.

다음 명령을 실행하여aws\_demosEclipse 프로젝트.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -
DAFR_ENABLE_TESTS=0
```

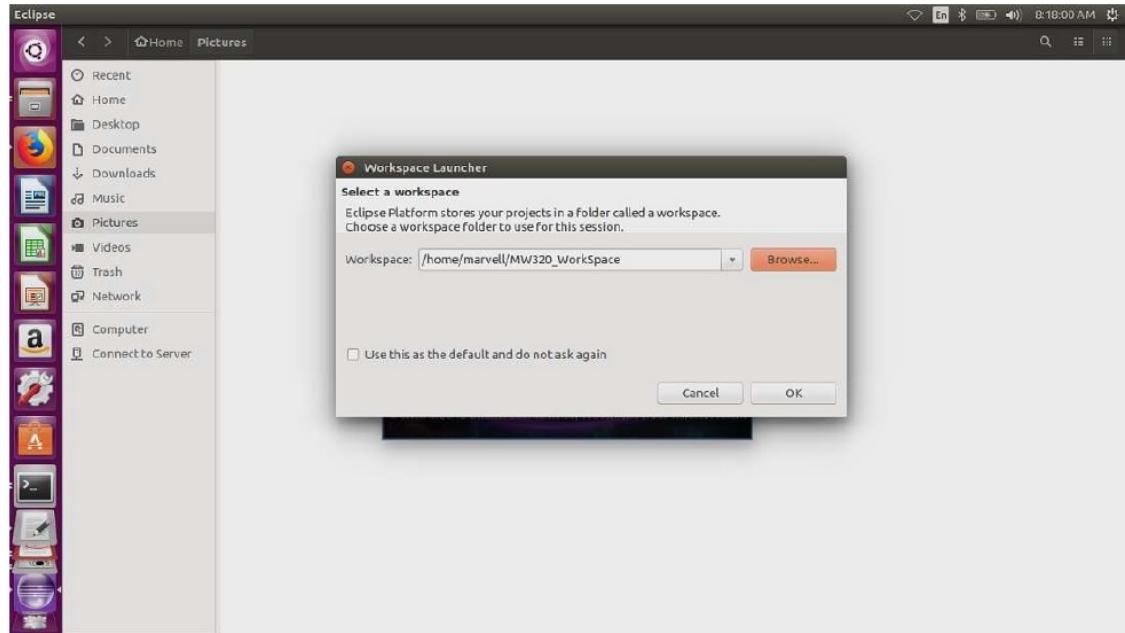
다음 명령을 실행하여aws\_testsEclipse 프로젝트.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -
DAFR_ENABLE_TESTS=1
```

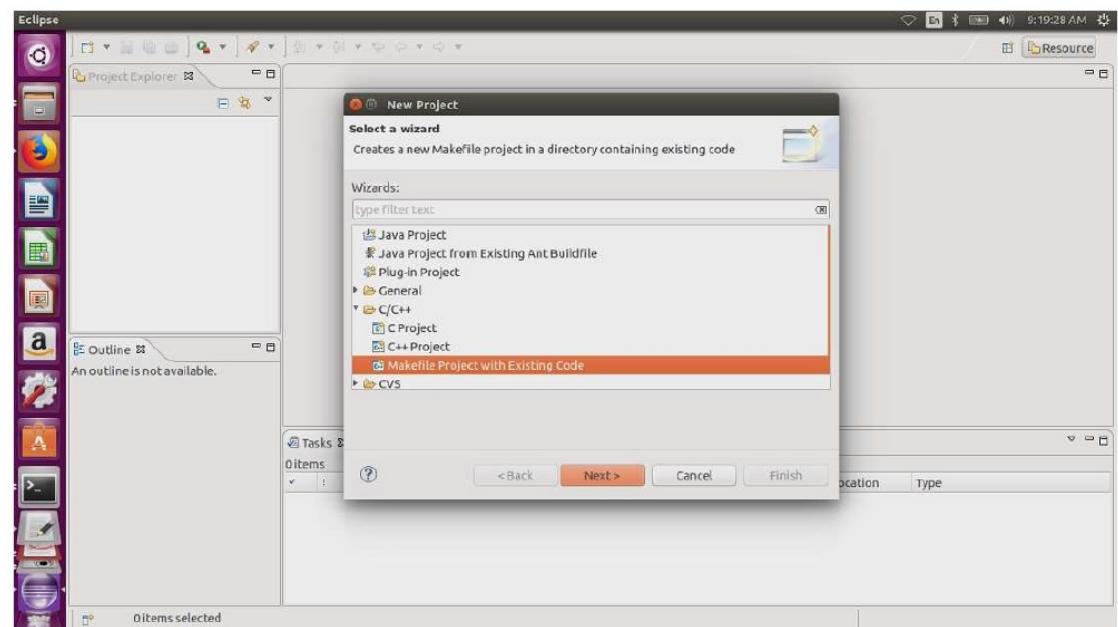
#### Tip

를 실행합니다.cmake명령 사이를 전환할 때마다aws\_demos프로젝트 및aws\_tests프로젝트.

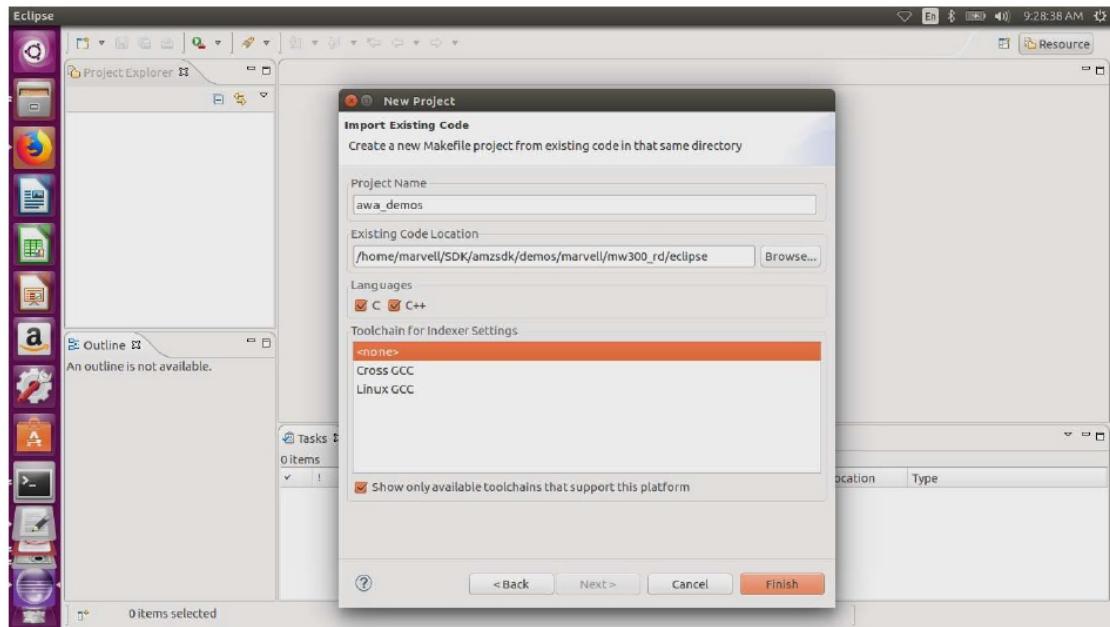
2. Eclipse를 열고 메시지가 표시되면 아래 그림과 같이 Eclipse 작업 영역을 선택합니다.



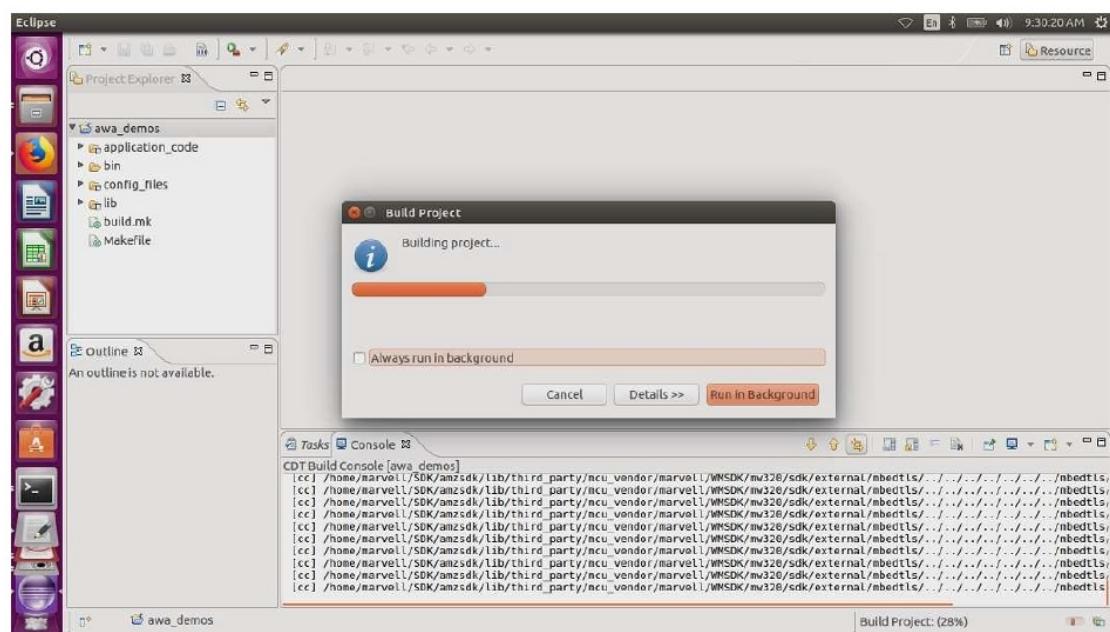
3. 를 생성할 옵션을 선택합니다.메이크 파일 프로젝트: 기존 코드아래 그림과 같습니다.



4. 선택검색기존 코드의 디렉토리를 지정한 다음Finish.



5. 탐색 창에서 를 선택합니다. aws\_demos 프로젝트 탐색기에서 오른쪽 클릭 aws\_demos 메뉴를 연 다음 빌드.



빌드가 성공하면 build/cmake/vendors/marvell/mw300\_rd/aws\_demos.bin 파일.

6. 명령줄 도구를 사용하여 레이아웃 파일을 플래시합니다 (layout.txt), Boot2 바이너리 (boot2.bin), MCU 펌웨어 바이너리 (aws\_demos.bin) Wi-Fi 펌웨어를 연결합니다.
  - 펌웨어 이미지를 플래시하기 전에 공통 구성 요소인 Layout 및 Boot2로 개발 보드의 플래시를 준비하십시오. 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
```

0|flashprog명령은 다음을 시작합니다.

- 레이아웃 — flashprog 유ти리티는 먼저 플래시에 레이아웃을 쓰도록 지시합니다. 레이아웃은 플래시의 파티션 정보와 유사합니다. 기본 레이아웃은 /lib/third\_party/mcu\_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt.
- Boot2 — WMSDK에서 사용하는 부트 로더입니다. 또한 flashprog 명령은 플래시에 부트로더를 씁니다. 플래시된 후 마이크로 컨트롤러의 펌웨어 이미지를 로드하는 것은 부트로더의 작업입니다. 아래 그림과 동일한 출력을 얻었는지 확인하십시오.

```
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 플래시에 있어야 하는 자체 펌웨어가 있습니다. 이 flashprog.pyboot2 부트 로더 및 MCU 펌웨어를 플래시하는 것과 동일한 방식으로 Wi-Fi 펌웨어를 플래시하는 유ти리티입니다. 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
```

명령의 출력이 아래 그림과 유사하게 출력되는지 확인합니다.

```
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 표시되어야 합니다.
- e. 테스트 앱을 실행하려면 aws\_tests.bin 바이너리가 같은 디렉토리에 위치합니다.

```
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 출력은 아래 그림에 표시된 것과 유사하게 출력되어야 합니다.

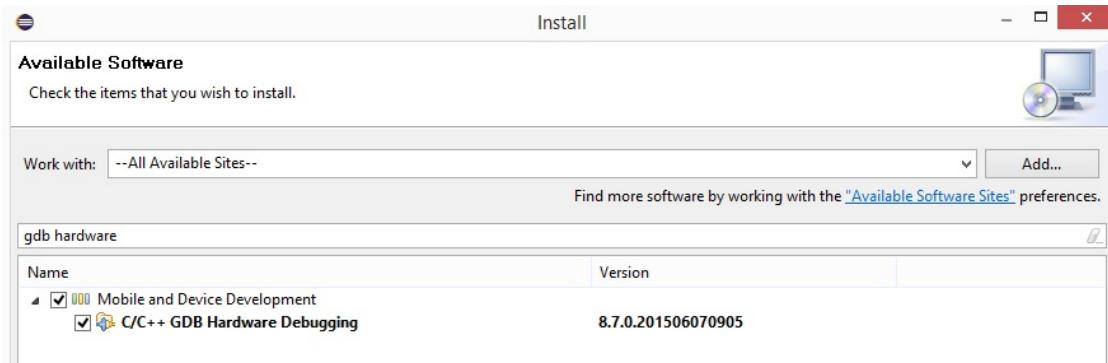
```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

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_nrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
      select <transport>'.
jtag_nrst_delay: 100
cortex_m_reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmc0re.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmc0re.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmc0re.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

## Debugging

- Eclipse를 시작하고 Help를 선택한 다음 새 소프트웨어 설치.에서 작업메뉴, Select사용 가능한 모든 사이트. 필터 텍스트 입력 GDB Hardware. 를 선택합니다.C/C++ GDB 하드웨어 디버깅플러그인을 설치하고 설치합니다.



## 문제 해결

### 네트워크 문제

네트워크 자격 증명을 확인합니다. 의 “프로비저닝”을 참조하십시오.[FreeRTOS 데모 프로젝트 빌드 및 실행 \(p. 87\)](#).

### 추가 로그 활성화

- 보드별 로그를 활성화합니다.

통화 활성화wmstdio\_init(UART0\_ID, 0) 함수에서prvMiscInitialization의main.c테스트 또는 데모용 파일

- Wi-Fi 로그 활성화

매크로 활성화CONFIG\_WLCMGR\_DEBUG의[freertos/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h](#)파일.

### GDB 사용

를 사용할 것을 권장합니다.arm-none-eabi-gdb과gdbSDK와 함께 패키지된 명령 파일 디렉터리로 이동합니다.

```
cd freertos/lib/third\_party/mcu\_vendor/marvell/WMSDK/mw320
```

다음 명령을 한 줄로 실행하여 GDB에 연결합니다.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../build/cmake/vendors/marvell/mw300 _rd/aws_demos.axf
```

## 플러그인으로 MediaTekMT7697Hx 개발 키트

이 자습서에서는 을 시작하기 위한 지침을 제공합니다.MediaTekMT7697Hx 개발 키트. 이 구성되지 않은 경 우MediaTekMT7697Hx 개발 키트, 다음을 방문하십시오.AWS파트너 디바이스 카탈로그를 통해 파트너에서 구입하시기 바랍니다.[파트너](#).

시작 하려면 먼저 구성해야 합니다.AWS IoT 그리고 FreeRTOS 다운로드를 통해 디바이스를AWS클라우드. 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 디렉터리에 대한 경로는 라고 합니다.[freertos](#).

## 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 이진 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 USB 포트에 연결하십시오. MediaTek MT7697Hx 개발 키트.

### 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. 설치했는지 확인합니다. MediaTek 디바이스 팩 (MT76x7 시리즈).

## 직렬 연결 설정

USB 케이블을 사용해 보드를 호스트 컴퓨터에 Connect. COM 포트가 Windows 장치 관리자에 나타납니다. 디버깅을 위해 다음과 같은 터미널 유ти리티 도구를 사용하여 포트에 대한 세션을 열 수 있습니다. HyperTerminal 또는 TeraTerm.

### 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 콘솔에서 디바이스가 보낸 메시지를 모니터링하기 위한 콘솔 AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 **MQTT 테스트**를 선택합니다. 테스트 그런 다음 **MQTT 테스트 클라이언트**를 선택합니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제에 여러 번 전송되었습니다.

### Keil MDK로 FreeRTOS 데모 프로젝트를 빌드하고 실행하려면

Keil µVision에서 FreeRTOS 데모 프로젝트를 빌드하려면

1. **Start** 메뉴에서 Keil µVision을 엽니다.
2. `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx` 프로젝트 파일을 엽니다.

- 메뉴에서 프로젝트를 선택한 후 Build target(빌드 대상)을 선택합니다.

코드가 빌드되면 projects/mediatek/mt7697hx-dev-kit/uvision/aws\_demos/out/Objects/aws\_demo.axf에서 데모 실행 파일을 볼 수 있습니다.

#### FreeRTOS 데모 프로젝트를 실행하려면

- 다음을 설정합니다.MediaTekMT7697Hx 개발 키트를 PROGRAM 모드로 전환합니다.

키트를 PROGRAM 모드로 설정하려면 PROG 버튼을 길게 누릅니다. PROG 버튼을 계속 누른 상태에서 재설정 버튼에서 손을 뗀 후 PROG 버튼에서 손을 뗅니다.

- 메뉴에서 Flash(플래시)를 선택한 후 Configure Flash Tools(플래시 도구 구성)를 선택합니다.
- 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.
- 메뉴에서 Flash(플래시)를 선택한 후 다운로드를 선택합니다.

다운로드가 완료되면 μVision에서 알립니다.

- 터미널 유ти리티를 사용하여 직렬 콘솔 창을 엽니다. 직렬 포트를 115200bps, 패리티 없음, 8비트 및 1정지 비트로 설정합니다.
- 를 선택합니다.재설정합니다버튼을 클릭합니다.MediaTekMT7697Hx 개발 키트.

## 문제 해결

### Keil μVision에서 FreeRTOS 프로젝트

현재 를 편집해야 합니다.MediaTekFreeRTOS 데모 프로젝트를 디버깅하기 전에 Keil μVision에 포함된 패키지MediaTekKeil μVision이 포함된

를 편집하려면MediaTekFreeRTOS 프로젝트 디버깅용 패키지

- Keil MDK 설치 폴더에서 Keil\_v5\ARM\PACK\ .Web\MediaTek.MTx.pdsc 파일을 검색하여 엽니다.
- flag = Read32(0x20000000);의 인스턴스를 flag = Read32(0x0010FBFC);로 모두 바꿉니다.
- Write32(0x20000000, 0x76877697);의 인스턴스를 Write32(0x0010FBFC, 0x76877697);로 모두 바꿉니다.

#### 프로젝트 디버깅을 시작하려면

- 메뉴에서 Flash(플래시)를 선택한 후 Configure Flash Tools(플래시 도구 구성)를 선택합니다.
- 대상 탭을 선택한 후 Read/Write Memory Areas(읽기/쓰기 메모리 영역)를 선택합니다. IRAM1 및 IRAM2가 모두 선택되었는지 확인합니다.
- 디버깅 탭을 선택한 후 CMSIS-DAP Debugger(CMSIS-DAP 디버거)를 선택합니다.
- vendors/mediatek/boards/mt7697hx-dev-kit/aws\_demos/application\_code/main.c를 열고 매크로 MTK\_DEBUGGER를 1로 설정합니다.
- 데모 프로젝트를 μVision으로 다시 빌드합니다.
- 다음을 설정합니다.MediaTekMT7697Hx 개발 키트를 PROGRAM 모드로 전환합니다.

키트를 PROGRAM 모드로 설정하려면 PROG 버튼을 길게 누릅니다. PROG 버튼을 계속 누른 상태에서 재설정 버튼에서 손을 뗀 후 PROG 버튼에서 손을 뗅니다.

- 메뉴에서 Flash(플래시)를 선택한 후 다운로드를 선택합니다.

다운로드가 완료되면 μVision에서 알립니다.

- 를 누릅니다.재설정합니다버튼을 클릭합니다.MediaTekMT7697Hx 개발 키트.

9. µVision 메뉴에서 를 선택합니다. 디버그를 선택한 다음 를 선택합니다. 디버그 세션 시작/중지. 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 ) ; }
```

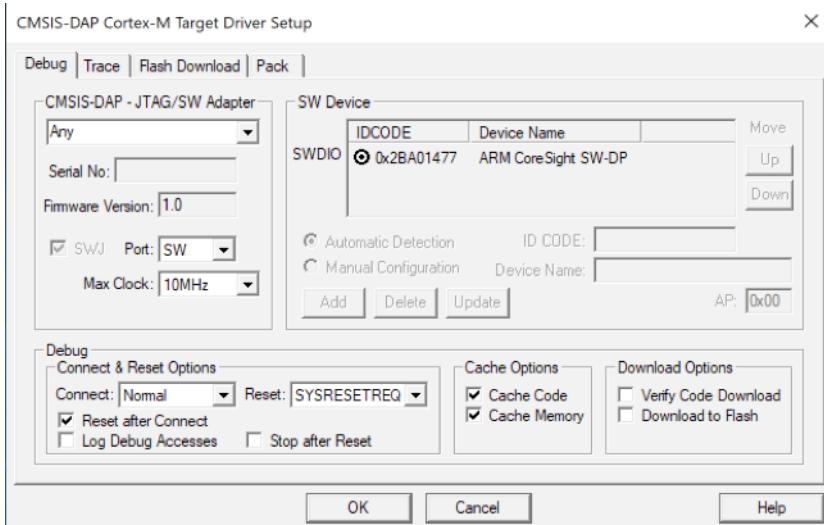
11. Call Stack + Locals(스택 + 로컬 호출) 창에서 wait\_ice 값을 0로 변경합니다.
12. 프로젝트의 소스 코드에 중단점을 설정하고 코드를 실행합니다.

## IDE 디버거 설정 문제 해결

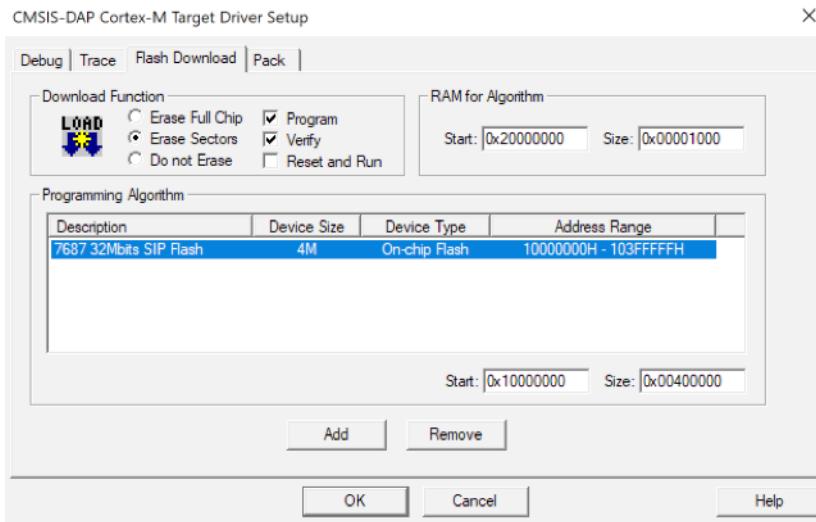
애플리케이션 디버깅에 문제가 있는 경우 디버거 설정이 잘못된 것일 수 있습니다.

디버거 설정이 올바른지 확인하려면

1. Keil µVision을 엽니다.
2. 오른쪽 클릭 aws\_demos 프로젝트, 선택옵션, 그리고 아래유ти리티탭에서 을 선택합니다. 설정옆에 “— 디버그 드라이버 사용 —”.
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 단원을 참조하십시오.[시작하기 문제 해결 \(p. 21\)](#).

## Microchip Curiosity PIC32MZ EF 시작하기

### Note

마이크로칩과 일치하여 FreeRTOS 레퍼런스 통합 리포지토리 메인 브랜치에서 Curiosity PIC32MZEF (DM320104) 를 제거하고 더 이상 새 릴리스에서 수행하지 않을 것입니다. 마이크로칩은 [공식 공지](#)새로운 설계에는 PIC32MZEF (DM320104) 가 더 이상 권장되지 않습니다. PIC32MZEF 프로젝트와 소스 코드는 이전 릴리스 태그를 통해 계속 액세스할 수 있습니다. Microchip은 고객이 호기심을 사용할 것을 권장합니다.[PIC32MZ-EF-2.0 개발 보드 \(DM320209\)](#)새로운 디자인을 위해. Pic32mzv1 플랫폼은 여전히 에서 찾을 수 있습니다.[v202012.00](#)FreeRTOS 참조 통합 리포지토리의 그러나 플랫폼은 더 이상 에서 지원되지 않습니다.[v202107.00](#)FreeRTOS 레퍼런스의

이 자습서에서는 Microchip Curiosity PIC32MZ EF를 시작하기 위한 지침을 제공합니다. Microchip Curiosity PIC32MZ EF 번들이 없는 경우 AWS파트너 디바이스 카탈로그를 구입하려면[파트너](#).

번들에는 다음 항목이 포함됩니다.

- Curiosity PIC32MZ EF Development Board
- MikroElectronikaUSB UART 클릭 보드
- MikroElectronika WiFi7 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](#)

시작하려면 먼저 구성해야 합니다.AWS IoTFreeRTOS 다운로드를 다운로드하여 디바이스를 AWS클라우드. 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요.

### Important

- 이 주제에서 FreeRTOS 다운로드 디렉터리의 경로를 이라고 합니다.[freertos](#).

- *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 경로는 빌드 오류가 발생할 수 있습니다.
- 소스 코드에 심볼릭 링크가 포함될 수 있으므로 Windows를 사용하여 아카이브를 추출하는 경우 다음을 수행해야 할 수 있습니다.
  - 활성화 개발자 모드 또는
  - 관리자 권한으로 상승된 콘솔을 사용합니다.

이러한 방식으로 Windows는 아카이브를 추출할 때 심볼릭 링크를 올바르게 만들 수 있습니다. 그렇지 않으면 심볼 링크의 경로가 텍스트로 포함되거나 비어 있는 일반 파일로 심볼 링크가 작성됩니다. 자세한 내용은 블로그 항목을 참조하십시오.[윈도우 10의 심볼릭 링크!](#)

Windows에서 Git을 사용하는 경우 개발자 모드를 활성화해야 합니다. 그렇지 않으면 다음을 수행해야 합니다.

- Setcore.symlinks 다음 명령을 사용하여 true로 설정합니다.

```
git config --global core.symlinks true
```

- 시스템에 쓰는 git 명령을 사용할 때마다 관리자 권한으로 상승된 콘솔을 사용하십시오 (예:git pull, git clone, 및 git submodule update --init --recursive).

## 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 이진 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. 을 Connect.MikroElectronikaUSB 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. 을 Connect.MikroElectronikaUSB A - USB 미니 B 케이블을 사용하여 USB UART Click Board를 컴퓨터에 연결합니다.
4. 보드를 인터넷에 연결하려면 다음 옵션 중 하나를 사용하십시오.
  - 를 사용하려면 Wi-Fi를 연결합니다. MikroElectronika Wi-Fi 7 Click Board를 Microchip Curiosity PIC32MZ EF의 microBUS 2 커넥터에 연결합니다. [FreeRTOS 데모 구성 \(p. 18\)](#) 섹션을 참조하세요.
  - 이더넷을 사용하여 Microchip Curiosity PIC32MZ EF 보드를 인터넷에 연결하려면, PIC32 LAN8720 PHY 부속 보드를 Microchip Curiosity PIC32MZ EF의 J18 헤더에 연결합니다. 이더넷 케이블의 한쪽 끝을 LAN8720 PHY 도터 보드에 연결합니다. 다른 쪽 끝을 라우터 또는 다른 인터넷 포트에 연결합니다. 전처리기 매크로도 정의해야 합니다. [PIC32\\_USE\\_ETHERNET](#).
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 단원을 참조하세요.[MPLAB Snap In-Circuit Debugger](#).

## PKOB(PICkit On Board)를 사용하여 Microchip Curiosity PIC32MZ EF 하드웨어 설정

이전 섹션의 설정 절차를 따르는 것이 좋습니다. 그러나 다음 단계에 따라 통합 PickIt On Board (PickIt On Board) 프로그래머/디버거를 사용하여 기본 디버깅으로 FreeRTOS 데모를 평가하고 실행할 수 있습니다.

1. 을 Connect.MikroElectronikaUSB UART Click Board를 Microchip Curiosity PIC32MZ EF의 microBUS 1 커넥터에 연결합니다.
2. 보드를 인터넷에 연결하려면 다음 중 하나를 수행합니다.
  - 를 사용하려면Wi-Fi를 연결합니다.MikroElectronikaWi-Fi 7 Click Board를 Microchip Curiosity PIC32MZ EF의 microBUS 2 커넥터에 연결합니다. ([FreeRTOS 데모 구성 \(p. 18\)](#)의 "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. 전처리기 매크로도 정의해야 합니다.[PIC32\\_USE\\_ETHERNET](#).
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. 을 Connect.MikroElectronikaUSB A - USB 미니 B 케이블을 사용하여 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

FreeRTOSAWS참조 통합 v202007.00은 현재 MPLabv5.35에서만 지원됩니다. 이전 버전의 FreeRTOSAWS참조 통합은 MPLabv5.40에서 지원됩니다.

### MPLABV5.35 다운로드

- [Windows용 MPLAB X 통합 개발 환경](#)
- [macOS용 MPLAB X 통합 개발 환경](#)
- [Linux용 MPLAB X 통합 개발 환경](#)

### 최신 MPLab 다운로드 (MPLabv5.40)

- 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: None
  - 정지 비트: 1
  - 흐름 제어: None

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 디바이스가 전송한 메시지를 모니터링하는 콘솔 AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 탐색 창에서 []를 선택합니다. 테스트를 선택한 후 선택합니다. MQTT 테스트 클라이언트 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제로 여러 번 전송되었습니다.

## FreeRTOS 데모 프로젝트 빌드 및 실행

MPLAB IDE에서 FreeRTOS 데모를 엽니다.

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.

이더넷을 사용하여 연결하려면 전처리기 매크로를 정의해야 합니다. `PIC32_USE_ETHERNET`.

이더넷을 사용하여 MPLAB IDE를 사용하여 연결하려면

1. MPLAB IDE에서 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 속성.
2. 에서 프로젝트 속성 대화 상자에서 선택 ##### ##(글로벌 옵션) 확장한 후 ##### ##-gcc.
3. 용 옵션 범주, 선택 사전 처리 및 메시지를 (를) 추가한 다음 PIC32\_USE\_ETHERNET 문자열로 변환 사전 처리기 매크로.

## 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 종단점에서 중지되면 실행 메뉴에서 다시 시작을 선택합니다.

## CMak로 FreeRTOS 데모 빌드

FreeRTOS 개발을 위해 IDE를 사용하지 않으려는 경우 CMaker를 사용하여 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 데모 애플리케이션 또는 애플리케이션을 빌드 및 실행할 수 있습니다.

CMak로 FreeRTOS 데모를 빌드하려면

1. 생성된 빌드 파일을 포함할 디렉터리를 생성합니다 (예: ## #####).
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. C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab\_platform\bin과 C:\Program Files \Microchip\xc32\v2.40\bin 경로.

3. 디렉터리를 빌드 디렉터리로 변경 (## #####)를 실행한 다음 실행합니다. make 해당 디렉토리에서

자세한 정보는 [CMak를 사용하여 프리어토스와 사용 \(p. 22\)](#)을 참조하십시오.

이더넷을 사용하여 연결하려면 전처리기 매크로를 정의해야 합니다. PIC32\_USE\_ETHERNET.

## 문제 해결

문제 해결 정보는 [시작하기 문제 해결 \(p. 21\)](#)을 참조하십시오.

## Nordic nRF52840-DK 시작하기

이 자습서에서는 Nordic nRF52840-DK를 시작하기 위한 지침을 제공합니다. Nordic nRF52840-DK가 없는 경우 AWS 파트너용 디바이스 카탈로그를 통해 구입하시기 바랍니다. [파트너](#).

시작하려면 설정 AWS IoT FreeRTOS 블루투스 저에너지용 Amazon Cognito (p. 234) 단원을 수행해야 합니다.

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.

- 보드를 호스트 시스템에 연결합니다.
- マイクロ 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
- 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
- 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 설명서](#)를 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

- [Segger Embedded Studio Downloads](#) 페이지로 이동하고 해당 운영 체제의 Embedded Studio for ARM 옵션을 선택합니다.
- 설치 관리자를 실행하고 표시되는 메시지에 따라 완료합니다.

### FreeRTOS Bluetooth Low Energy Bluetooth Low Energy Mobile

Bluetooth Low Energy에서 FreeRTOS 데모 프로젝트를 실행하려면 모바일 디바이스에서 FreeRTOS Bluetooth Low Energy 데모 애플리케이션을 실행해야 합니다.

#### FreeRTOS Bluetooth Low Energy Mobile

- [FreeRTOS Bluetooth 디바이스용 Mob \(p. 207\)](#)의 지침에 따라 모바일 플랫폼용 SDK를 다운로드하여 호스트 컴퓨터에 설치합니다.
- [FreeRTOS 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 \(p. 236\)](#)의 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 설치 (p. 22)의 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 설치 (p. 22)의 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 페이지](#) 리포지토리를 복제합니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.

#### Important

- 이 주제에서 FreeRTOS 다운로드 디렉터리의 경로를 이라고 합니다.[freertos](#).
- [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.
- 소스 코드에 심볼릭 링크가 포함될 수 있으므로 Windows를 사용하여 아카이브를 추출하는 경우 다음을 수행해야 할 수 있습니다.
  - 활성화 개발자 모드 또는
  - 관리자 권한으로 상승된 콘솔을 사용합니다.

이러한 방식으로 Windows는 아카이브를 추출할 때 심볼릭 링크를 올바르게 만들 수 있습니다. 그렇지 않으면 심볼 링크의 경로가 텍스트로 포함되거나 비어 있는 일반 파일로 심볼 링크가 작성됩니다. 자세한 내용은 블로그 항목을 참조하십시오. [윈도우 10의 심볼릭 링크!](#)

Windows에서 Git을 사용하는 경우 개발자 모드를 활성화해야 합니다. 그렇지 않으면 다음을 수행해야 합니다.

- Setcore.symlinks 다음 명령을 사용하여 true로 설정합니다.

```
git config --global core.symlinks true
```

- 시스템에 쓰는 git 명령을 사용할 때마다 관리자 권한으로 상승된 콘솔을 사용하십시오 (예:git pull, git clone, 및 git submodule update --init --recursive).

## 프로젝트 구성

데모를 활성화하려면 로 작업하도록 프로젝트를 구성해야 합니다. AWS IoT. AWS IoT로 작업하도록 프로젝트를 구성하려면 디바이스를 AWS IoT 사물로 등록해야 합니다. [설정AWS IoTFreeRTOS 블루투스 저에너지용 Amazon Cognito \(p. 234\)](#)을 수행할 때 디바이스가 등록되어 있어야 합니다.

### AWS IoT 엔드포인트를 구성하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 설정(Settings)을 선택합니다.

구하AWS IoT엔드포인트가 에 표시됩니다.디바이스 데이터 엔드포인트텍스트 상자 URL은 [1234567890123-ats.iot.us-east-1.amazonaws.com](https://1234567890123-ats.iot.us-east-1.amazonaws.com)와 같아야 합니다. 이 엔드포인트를 기록해 듭니다.

3. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다. 디바이스의 AWS IoT 사물 이름을 기록해 듭니다.
4. AWS IoT 엔드포인트와 AWS IoT 사물 이름을 기록했으면 IDE에서 `freertos/demos/include/aws_clientcredential.h`를 열고 다음 `#define` 상수의 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.h`로 이동하여 `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )`를 정의문 목록에 추가합니다.
2. `Openvendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h` 다음 중 하나를 정의합니다. `CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED` 또는 `CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED`이 예제와 같습니다.

```
/* To run a particular demo you need to define one of these.
 * Only one demo can be configured at a time
 *
 *      CONFIG_BLE_GATT_SERVER_DEMO_ENABLED
 *      CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *      CONFIG_SHADOW_BLE_TRANSPORT_DEMO_ENABLED
 *      CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *      CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED
 *      CONFIG_POSIX_DEMO_ENABLED
 *
 *      These defines are used in iot_demo_runner.h for demo selection */
#define CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
```

3. 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.h`에서 다음 속성을 재정의합니다.

- `NRF_SDH_BLE_VS_UUID_COUNT`

공급업체별 UUID의 수입니다. 새 공급업체별 UUID를 추가하면 이 개수를 1씩 늘립니다.

- `NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE`

속성 테이블 크기(바이트)입니다. 크기는 4의 배수여야 합니다. 이 값은 특성 테이블 전용 메모리(특성 크기 포함)에 대해 설정된 메모리 양을 나타내므로 프로젝트마다 다릅니다. 속성 테이블의 크기를 초과하면 `NRF_ERROR_NO_MEM` 오류가 발생합니다. 일반적으로 `NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE`를 수정하는 경우 RAM 설정도 다시 구성해야 합니다.

(테스트의 경우 파일의 위치는 [freertos/vendors/nordic/boards/nrf52840-dk/aws\\_tests/config\\_files/sdk\\_config.h](#)입니다.)

## 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.emProject`를 사용합니다.

### Segger Embedded Studio에서 FreeRTOS Bluetooth Low Energy 데모를

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. 오른쪽 클릭 `aws_demos`의 데모 프로젝트프로젝트 탐색기를 선택하고 빌드.

### 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. [무료 라이선스 요청](#) 그런 다음 설정 중에 무료 라이선스 활성화를 클릭하고 지침을 따릅니다.

4. 디버깅을 선택한 다음 이동을 선택합니다.

데모가 시작되면 Bluetooth Low Energy를 통해 모바일 장치와 페어링되기를 기다립니다.

5. 의 지침을 따릅니다.[MQTT over Bluetooth Low Energy](#)FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 모바일 MQTT 프록시로 사용하여 데모를 완료합니다.

## 문제 해결

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 단원을 참조하세요.[시작하기 문제 해결 \(p. 21\)](#).

## Nuvoton 시작하기|NuMaker-IoT-M487

이 자습서에서는 Nuvoton 시작하기 지침을 제공합니다.NuMaker-IoT-M487 개발 보드. 이 시리즈 마이크로 컨트롤러와 기본 제공 RJ45 이더넷 및 Wi-Fi 모듈이 포함되어 있습니다. Nuvoton이 없는 경우NuMaker-IoT-M487, 방문[AWS파트너 디바이스 카탈로그](#)파트너로부터 하나를 구매하십시오.

시작하려면 먼저 구성해야 합니다.AWS IoT 및 FreeRTOS 소프트웨어를 사용하여 개발 보드를 AWS 클라우드. 지침은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리에 대한 경로는 라고 합니다.[freertos](#).

## 개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

1. 마이크로컨트롤러 보드용 임베디드 애플리케이션을 개발하고 디버깅하기 위한 소프트웨어를 호스트 머신에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 이진 이미지로 교차 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.

에 대한 개발 도구를 설치하려면 NuMaker-IoT-M487

1. Keil MDK 웹 사이트에서 [Keil MDK Nuvoton 에디션](#)을 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 [Nuvoton](#)에 지원을 요청하십시오.
3. 설치 누-링크\_케일\_드라이버\_V3.06.7215R(또는 최신 버전), [누보튼 개발 도구](#) 페이지.

## 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.uvproj`를 선택합니다.
  - 이더넷 데모 프로젝트를 열려면 `freertos\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth` 디렉터리에서 대상 프로젝트 `aws_demos_eth.uvproj`를 선택합니다.
4. 보드를 플래시하기 위한 설정이 올바른지 확인하려면 aws\_demo 프로젝트를 마우스 오른쪽 버튼으로 클릭한 다음 Options(옵션)를 선택합니다. (자세한 내용은 [문제 해결 \(p. 114\)](#) 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 모니터링 할 수 있습니다.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트를 선택합니다. MQTT 테스트 클라이언트를 눌러 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 종단점에서 중지되면 Run(실행) 메뉴를 열고 Run(실행)(F5)을 선택합니다.

디바이스에서 전송한 MQTT 메시지가 AWS IoT 콘솔의 MQTT 클라이언트에 나타나야 합니다.

## CMake와 FreeRTOS 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. [CMake를 사용하여 프리어토스와 사용 \(p. 22\)](#)의 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.

에서 MQTT 클라이언트를 사용할 수 있습니다. AWS IoT 디바이스가 전송되는 메시지를 모니터링하는 콘솔 AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트를 선택합니다. MQTT 테스트 클라이언트를 눌러 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 머신에서 명령 프롬프트를 열고 다음으로 이동합니다. **FreeRTOS** folder.
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
```

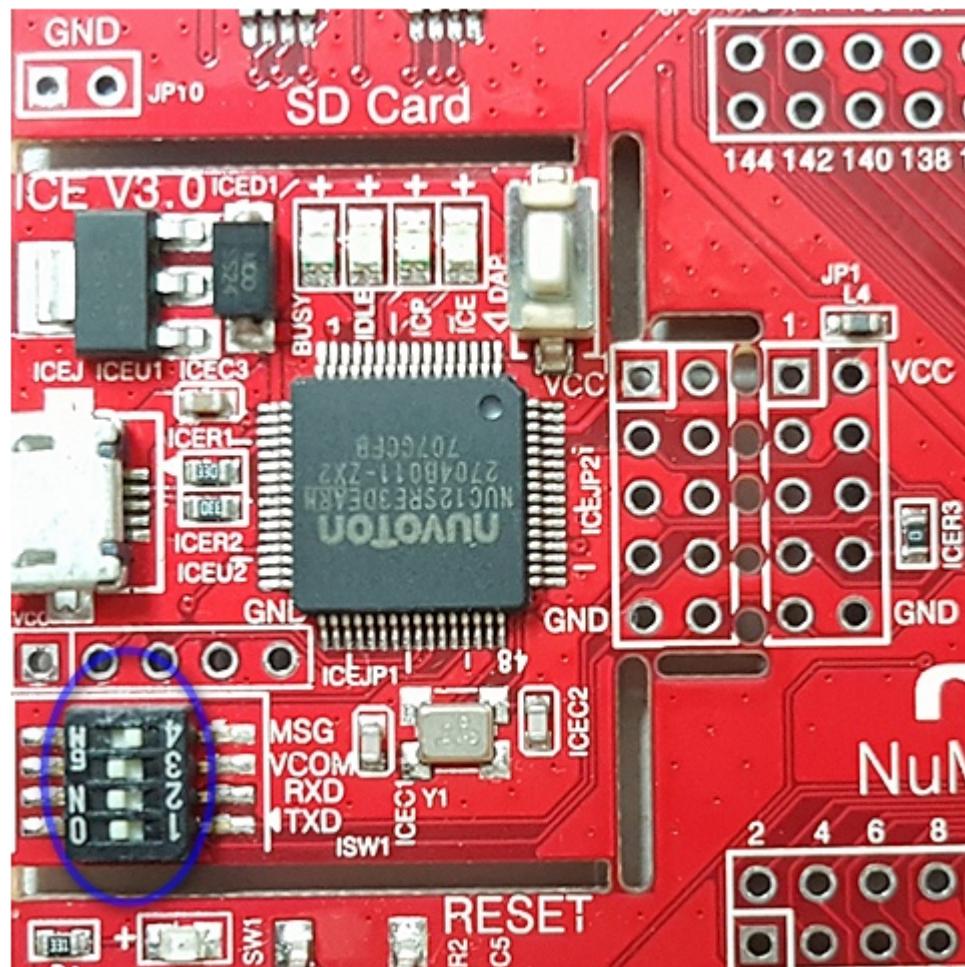
이 시점에서는 이진 파일 aws\_demos.bin이 **BUILD\_FOLDER**/vendors/Nuvoton/boards/numaker\_iot\_m487\_wifi 폴더에 있어야 합니다.

5. 플래시 모드에 맞게 보드를 구성하려면 MSG 스위치(ICE에서 ISW1의 4번)가 ON으로 전환되어 있는지 확인합니다. 보드에 플러그인하면 창(드라이브)이 할당됩니다. ([문제 해결 \(p. 114\)](#) 섹션을 참조하세요.)
6. 터미널 에뮬레이터를 열어 UART를 통해 메시지를 봅니다. [터미널 에뮬레이터 설치 \(p. 22\)](#)의 지침을 따릅니다.
7. 생성된 이진수를 디바이스에 복사하여 데모 프로젝트를 실행합니다.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독한 경우 AWS IoT 콘솔에서 디바이스가 전송한 MQTT 메시지를 볼 수 있어야 합니다.

## 문제 해결

- 창에서 디바이스 이 인식되지 않는 경우 VCOM를 설치합니다. NuMaker 링크에서 Windows 직렬 포트 드라이버 [Nu-link USB 드라이버 v1.6](#).
- Nu-Link를 통해 디바이스를 Keil MDK(IDE)에 연결하는 경우 그림과 같이 MSG 스위치(ICE에서 ISW1의 4 번)가 OFF로 전환되어 있는지 확인합니다.



개발 환경을 설정하거나 보드에 연결하는 동안 문제가 발생하는 경우 [Nuvoton](#)에 문의하십시오.

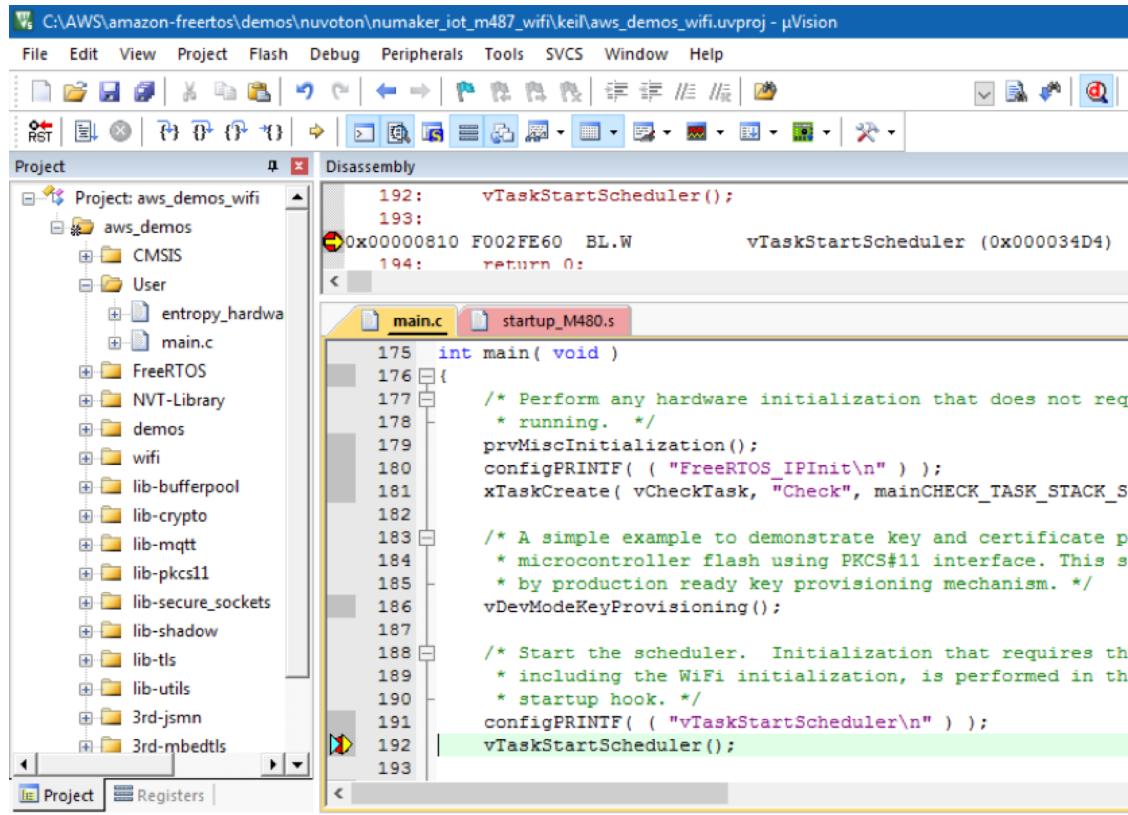
### Keil µVision에서 FreeRTOS 프로젝트

Keil µVision에서 디버깅 세션을 시작하려면

1. Keil µVision을 엽니다.
2. 단계에 따라 FreeRTOS 데모 프로젝트를 빌드하려면 [FreeRTOS 데모 프로젝트 빌드 및 실행 \(p. 112\)](#).
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.

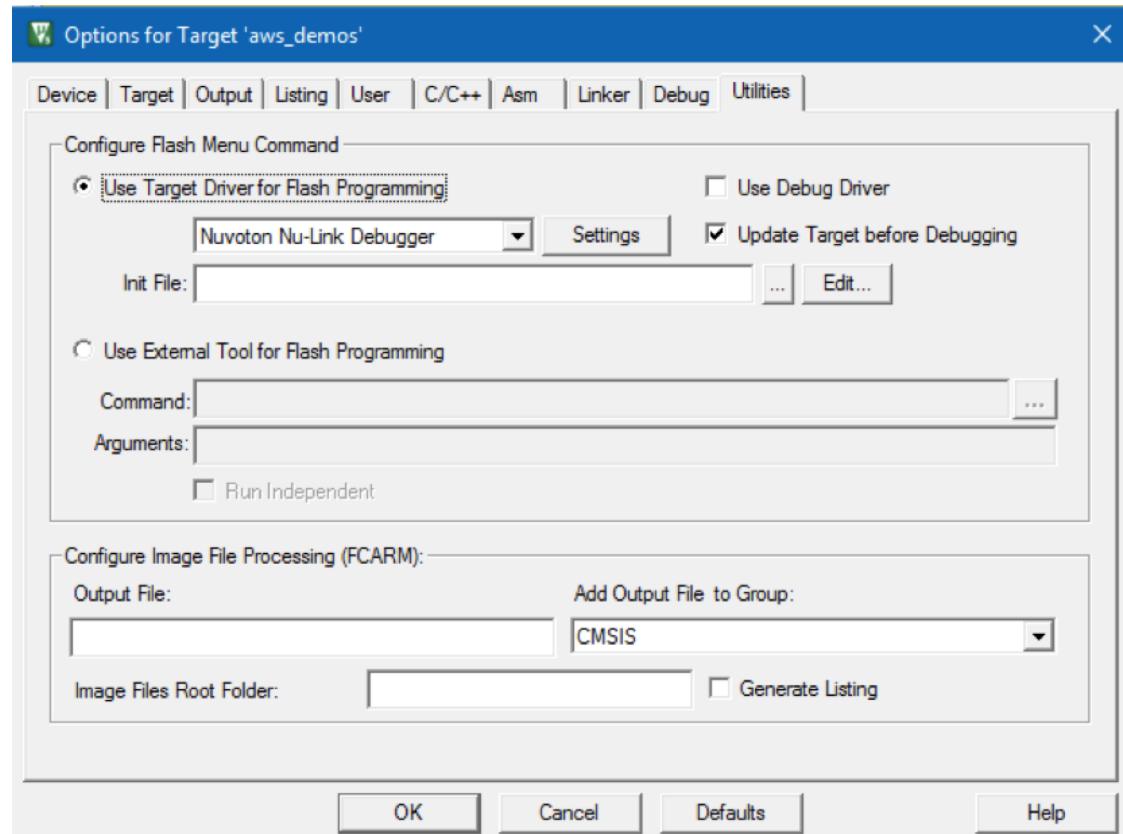


## μVision 디버그 설정 문제 해결

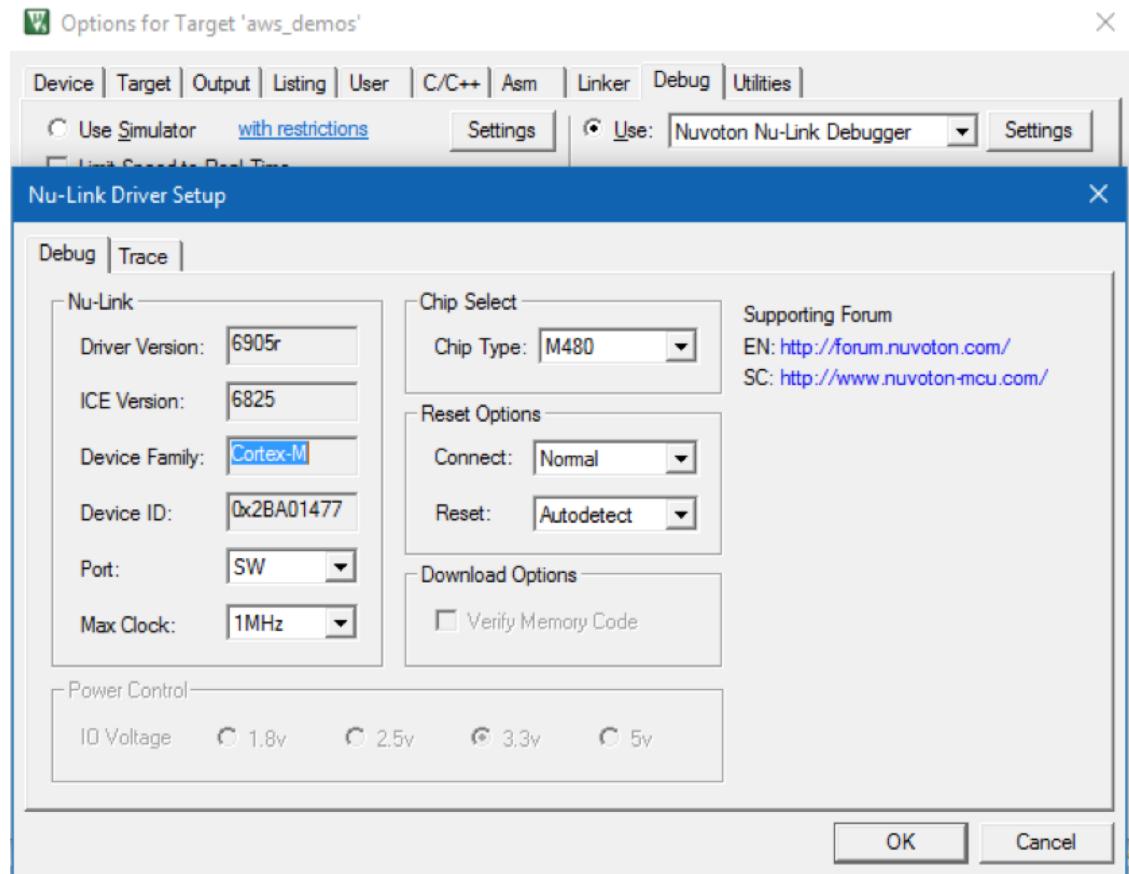
애플리케이션을 디버깅하는 동안 문제가 발생하는 경우 Keil μVision에서 디버그 설정이 올바르게 설정되었는지 확인합니다.

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(유필리티) 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

이 자습서에서는 NXP LPC54018 IoT 모듈을 시작하기 위한 지침을 제공합니다. NXP LPC54018 IoT 모듈이 없는 경우 단원을 참조하십시오. AWS파트너 디바이스 카탈로그를 구입하시기 바랍니다. [파트너](#). USB 케이블을 사용하여 NXP LPC54018 IoT Module을 컴퓨터에 연결합니다.

시작하려면 먼저 구성해야 합니다. AWS IoT 및 FreeRTOS 다운로드를 다운로드하여 디바이스를 AWS 클라우드에 대한 경로는 라고 합니다. [freertos](#).

### 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 이진 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 단원을 참조하십시오.NXP LPC54018 IoT 모듈 사용 설명서는 에서 사용할 수 있습니다.[OM40007 LPC54018 IoT 모듈제품 페이지](#).

- OM40006 IoT 모듈 디버거를 사용하는 경우 디버거의 20핀 커넥터를 NXP IoT 모듈의 10핀 커넥터에 연결하려면 변환기 케이블을 사용합니다.
- 미니 USB-USB 케이블을 사용하여 NXP LPC54018과 OM40006 IoT 모듈 디버거를 컴퓨터의 USB 포트에 연결합니다.

## 개발 환경 설정

FreeRTOS는 NXP LPC54018 IoT 모듈을 위한 두 개의 IDE를 지원합니다. IAR Embedded Workbench 및 MCUXpresso.

시작하기 전에 이러한 IDE 중 하나를 설치합니다.

IAR Embedded Workbench for ARM을 설치하려면

- 찾아보기|[IAR Embedded Workbench](#)소프트웨어를 다운로드합니다.

### Note

IAR Embedded Workbench for ARM에는 Microsoft Windows가 필요합니다.

- 설치 관리자를 실행하고 표시되는 메시지에 따릅니다.
- License Wizard(라이선스 마법사)에서 Register with IAR Systems to get an evaluation license(IAR Systems에 등록하여 평가 라이선스 받기)를 선택합니다.
- 데모를 실행하기 전에 디바이스에 부트로더를 배치합니다.

NXP의 MCUXpresso를 설치하려면

- [NXP](#)에서 MCUXpresso 설치 관리자를 다운로드하여 설치합니다.

### Note

버전 10.3.x 이상이 지원됩니다.

- MCUXpresso SDK로 이동하여 Build your SDK(SDK 빌드)를 선택합니다.

### Note

버전 2.5 이상이 지원됩니다.

- Select Development Board(개발 보드 선택)를 선택합니다.
- Select Development Board(개발 보드 선택)의 Search by Name(이름으로 검색)에 **LPC54018-IoT-Module**를 입력합니다.
- Boards(보드)에서 LPC54018-IoT-Module을 선택합니다.
- 하드웨어 세부 정보를 확인한 다음 Build MCUXpresso SDK(MCUXpresso 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.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 콘솔에서 디바이스가 보내는 메시지를 모니터링합니다. AWS 클라우드

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트를 선택합니다. MQTT 테스트 클라이언트를 눌러 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 "Hello World!" 구독한 주제에 여러 번 전송되었습니다.

## FreeRTOS 데모 프로젝트 빌드 및 실행

### IDE로 FreeRTOS 데모 가져오기

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\_demos를 입력하고 aws\_demos.eww를 선택합니다.
3. Project(프로젝트) 메뉴에서 Rebuild All(모두 다시 빌드)을 선택합니다.

MCUXpresso IDE로 FreeRTOS 샘플 코드를 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\_demos를 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 main의 종단점에서 종지되면 Debug(디버그) 메뉴에서 Go(이동)를 선택합니다.

### Note

J-Link Device Selection(J-Link 디바이스 선택) 대화 상자가 열리면 OK(확인)를 선택하여 계속 합니다. 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.

### MCUXpresso IDE로 FreeRTOS 데모 프로젝트를 실행하려면

- IDE의 프로젝트 메뉴에서 빌드를 선택합니다.
- 처음 디버깅하는 경우 aws\_demos 프로젝트를 선택하고 디버깅 도구 모음에서 파란색 디버그 버튼을 선택합니다.
- 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).

- 디버거가 main()의 중단점에서 중지되면 Debug(디버그) 메뉴에서 Go(이동)를 선택합니다.

## 문제 해결

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 단원을 참조하십시오. [시작하기 문제 해결 \(p. 21\)](#).

## Renesas Starter Kit+ for RX65N-2MB 시작하기

이 자습서에서는 Renesas Starter Kit+ for RX65N-2MB를 시작하기 위한 지침을 제공합니다. RSK+ for RX65N-2MB 가 없는 경우 AWS파트너 장치 카탈로그 및 당사의 제품 중 하나를 구입하십시오. [파트너](#).

시작하려면 단원을 구성해야 합니다. AWS IoTFreeRTOS 다운로드를 다운로드하여 디바이스를 AWS 클라우드. 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 디렉터리의 경로를 이라고 합니다. [freertos](#).

## 개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

- 보드를 호스트 시스템에 연결합니다.
- マイ크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
- 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

## Renesas 하드웨어 설정

RSK+ for RX65N-2MB를 설정하려면

- 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 컴퓨터를 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-2MB for FreeRTOS 구성을 설정하려면 Renesas e<sup>2</sup>스튜디오 IDE 및 CC-RX 컴파일러.

### Note

Renesas e<sup>2</sup>studio IDE 및 CC-RX 컴파일러는 Windows 7, 8, 10 운영 체제에서만 지원됩니다.

### e<sup>2</sup>studio 다운로드 및 설치

1. 로 이동 [Renesas e<sup>2</sup>스튜디오 설치 프로그램](#) 페이지를 다운로드하고 오프라인 설치 관리자를 다운로드합니다.
2. 그러면 Renesas 로그인 페이지로 이동합니다.

Renesas에 계정이 있는 경우 사용자 이름과 암호를 입력한 후 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<sup>2</sup>studio 설치 관리자를 컴퓨터에 다운로드합니다.
4. 설치 관리자를 열고 단계에 따라 완료합니다.

자세한 내용은 단원을 참조하십시오. [e<sup>2</sup>Studio](#)로네사스 웹 사이트에서 확인할 수 있습니다.

## RX Family C/C++ 컴파일러 패키지 다운로드 및 설치

1. [로 이동RX Family C/C++ 컴파일러](#)페이지를 다운로드하고 V3.00.00 패키지를 다운로드합니다.
2. 실행 파일을 열고 컴파일러를 설치합니다.

자세한 내용은 Renesas 웹사이트에서 [RX Family용 C/C++ 컴파일러 패키지](#)를 참조하십시오.

### Note

컴파일러는 평가판에 대해서만 무료로 제공되며 60일 동안 유효합니다. 61일째는 라이선스 키를 받아야 합니다. 자세한 내용은 [평가 소프트웨어 도구](#)를 참조하십시오.

## FreeRTOS 샘플 빌드 및 실행

이제 데모를 구성했으므로 보드에서 데모 프로젝트를 빌드 및 실행할 준비가 된 것입니다.

### e<sup>2</sup>Studio에서 FreeRTOS 데모 빌드

#### e<sup>2</sup>studio에서 데모를 가져와서 빌드하기

1. 시작 메뉴에서 e<sup>2</sup>studio를 시작합니다.
2. Select a directory as a workspace(작업 영역으로 디렉터리 선택) 창에서 작업하려는 폴더를 찾아 시작을 선택합니다.
3. 처음 e<sup>2</sup>studio를 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 e<sup>2</sup>studio를 열 경우 Code Generator Registration(코드 생성기 등록) 창이 나타납니다. OK(확인)을 선택합니다.
5. Code Generator COM component register(코드 생성기 COM 구성요소 등록)창이 나타납니다. Please restart e<sup>2</sup>studio to use Code Generator에서 확인을 선택합니다.
6. e<sup>2</sup>studio 다시 시작 창이 나타납니다. OK(확인)을 선택합니다.
7. e<sup>2</sup>studio가 다시 시작됩니다. Select a directory as a workspace(작업 영역으로 디렉터리 선택) 창에서 시작을 선택합니다.
8. e<sup>2</sup>studio 시작 화면에서 e<sup>2</sup>studio 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 컴파일러 용 라이선스 키가 없다면 이 메시지를 무시할 수 있습니다. 라이선스 관리자를 설치하려면 [License Manager\(라이선스 관리자\)](#) 다운로드 페이지를 참조하십시오.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 콘솔에서 디바이스가 전송한 메시지를 모니터링합니다. AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.

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

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제로 여러 번 전송되었습니다.

## FreeRTOS 프로젝트 실행

e<sup>2</sup>studio에서 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. 확장르네사스 GDB 하드웨어 디버깅을 선택하고 aws\_demosHardwareDebug.
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.exe에 대한 방화벽 경고 메시지가 표시될 수 있습니다. Private networks, such as my home or work network(프라이빗 네트워크(예: 흄 또는 직장 네트워크))를 선택한 다음 Allow access(액세스 허용)를 선택합니다.

6. e<sup>2</sup>studio가 Renesas Debug Perspective(Renesas 디버그 관점)로 변경하도록 요청할 수 있습니다. 예(Yes)를 선택합니다.

E2 Lite 디버거에서 녹색 'ACT' LED가 커집니다.

7. 코드가 보드에 다운로드되면 다시 시작을 선택하여 첫 번째 줄의 main 함수에 대해 코드를 실행합니다. 다시 시작을 다시 선택하여 나머지 코드를 실행합니다.

Renesas가 출시한 최신 프로젝트는 [renesas-rx](#) 포크 오브 amazon-freertos 리포지토리 [GitHub](#).

## 문제 해결

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 단원을 참조하십시오. [시작하기 문제 해결 \(p. 21\)](#).

## STMicroelectronics STM32L4 Discovery Kit IoT Node 시작하기

이 자습서에서는 STMicroelectronics STM32L4 Discovery Kit IoT 노드를 시작하기 위한 지침을 제공합니다. STMicroelectronics STM32L4 Discovery Kit IoT 노드가 없는 경우 AWS파트너 디바이스 카탈로그를 구입하시기 바랍니다. [파트너](#).

최신 Wi-Fi 펌웨어를 설치했는지 확인합니다. 최신 Wi-Fi 펌웨어를 다운로드하려면 [STM32L4 Discovery 키트 IoT 노드, 저전력 무선, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi](#)를 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.

시작하려면 구성해야 합니다. AWS IoT, FreeRTOS 다운로드 및 Wi-Fi를 사용하여 장치를 AWS클라우드 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 디렉터리에 대한 경로는 라고 합니다. [freertos](#).

## 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 이진 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

## 개발 환경 설정

### System Workbench for STM32 설치

1. [OpenSTM32.org](#)로 이동합니다.
2. OpenSTM32 웹 페이지에서 등록합니다. System Workbench를 다운로드하려면 로그인해야 합니다.
3. [System Workbench for STM32 설치 관리자](#)로 이동하여 System Workbench를 다운로드하고 설치합니다.

설치 중에 문제가 발생하면 [System Workbench 웹 사이트](#)의 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. File(파일) 메뉴에서 Import(가져오기)를 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.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 콘솔에서 디바이스가 보내는 메시지를 모니터링합니다. AWS 클라우드

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트를 선택합니다. MQTT 테스트 클라이언트를 눌러 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제에 여러 번 보냈습니다.

## FreeRTOS 데모 프로젝트 실행

1. USB 케이블을 사용하여 STMicroelectronics STM32L4 Discovery Kit IoT Node를 컴퓨터에 연결합니다. (사용할 올바른 USB 포트는 보드와 함께 제공된 제조업체 설명서를 참조하십시오.)
2. Project Explorer에서 aws\_demos를 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.

## CMak를 사용하여 프리어토스와 사용

용 IDE FreeRTOS 개발을 사용하지 않으려는 경우 CMaak를 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=stm321475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

arm-none-eabi-gcc가 셸 경로에 없으면 AFR\_TOOLCHAIN\_PATH CMake 변수도 설정해야 합니다. 예:

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

CMak를 사용하는 방법에 대한 자세한 내용은 단원을 참조하십시오.[CMak를 사용하여 프리어토스와 사용 \(p. 22\)](#).

## 문제 해결

데모 애플리케이션의 UART 출력에 다음이 나타나면 Wi-Fi 모듈의 펌웨어를 업데이트해야 할 수 있습니다.

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

최신 Wi-Fi 펌웨어를 다운로드 하려면 [STM32L4 Discovery 키트 IoT 노드, 저전력 무선, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi](#)를 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 단원을 참조하십시오.[시작하기 문제 해결 \(p. 21\)](#).

## Texas Instruments CC3220SF-LAUNCHXL 시작하기

이 자습서에서는 Texas Instruments CC3220SF-LAUNCHXL을 시작하기 위한 지침을 제공합니다. Texas Instruments (TI) CC3220SF-LAUNCHXL 개발 키트가 없는 경우 AWS Partner Device Catalog을 사용하여 [파트너](#).

시작하려면 단원을 구성해야 합니다. AWS IoT FreeRTOS 다운로드를 통해 디바이스를 AWS 클라우드. 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 딜렉터리에 대한 경로는 라고 합니다.[freertos](#).

## 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 이진 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

## 개발 환경 설정

아래 단계에 따라 FreeRTOS를 시작하려면 개발 환경을 설정합니다.

FreeRTOS는 TI CC3220SF-LAUNCHXL 개발 키트에 대해 두 개의 IDE를 지원합니다. 코드 컴포저 스튜디오 및 IAR Embedded Workbench 버전 8.32. 이 중 하나의 IDE를 사용하여 시작할 수 있습니다.

### Install Code Composer 설치

1. [TI Code Composer Studio](#)로 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. 용설치할 제품군, 선택SimpleLink와 이파이 CC32xx 무선 MCU.
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 해결](#)을 참조하십시오.

### IAR Embedded Workbench 설치

1. ARM용 IAR Embedded Workbench의 버전 8.32용 [Windows 설치 관리자](#)를 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.

### 설치SimpleLinkCC3220 SDK

설치[SimpleLinkCC3220 SDK](#). 이SimpleLinkWi-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 사용 중인 경우LaunchPad보드가 아래에 나타납니다. 감지된 디바이스, 선택Start. 보드가 감지되지 않으면CC3220SF 런치XL아래의 보드 목록에서새 구성률 선택한 다음을 선택합니다. 이미지 크리에이터 시작.

3. New Project(새 프로젝트)를 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. 선택검색을 클릭한 후 CC3220SF 를 설치한 위치로 이동합니다. SimpleLink의 SDK. 서비스 팩은 `ti/simplelink_cc32xx_sdk_<VERSION>/tools/cc32xx_tools/servicepack-cc3x20/servicepack-<VERSION>.bin`에 있습니다.
8.  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.

- 예 설명된 FreeRTOS 데모 애플리케이션 구성[FreeRTOS 데모 구성 \(p. 18\)](#).
- 사용[SmartConfig](#) Texas Instruments

## FreeRTOS 데모 프로젝트 빌드 및 실행

### TI Code Composer에서 FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모를 TI 코드 컴포저로 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 프로젝트가 컴파일되는지 확인합니다.

TI 코드 컴포저에서 FreeRTOS 데모를 실행하려면

1. Texas Instruments CC3220SF-LAUNCHXL의 Sense On Power(SOP) 점퍼가 0 위치에 있는지 확인합니다. 자세한 내용은 단원을 참조하십시오.[SimpleLink Wi-Fi CC3x20, CC3x3x 네트워크 프로세서 사용자 안내서](#).
2. USB 케이블을 사용하여 Texas Instruments CC3220SF-LAUNCHXL을 컴퓨터에 연결합니다.
3. 프로젝트 탐색기에서 `CC3220SF.ccxm1`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 종단점에서 중지되면 실행 메뉴로 이동하여 다시 시작을 선택합니다.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 콘솔에서 디바이스가 보내는 메시지를 모니터링할 콘솔 AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트를 선택한 다음 를 선택합니다 MQTT 테스트 클라이언트를 놀려 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제로 여러 번 전송되었습니다.

## IAR Embedded Workbench에서 FreeRTOS 데모 프로젝트 빌드 및

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.eww`를 선택한 다음 확인을 선택합니다.
3. 프로젝트 이름(`aws_demos`)을 마우스 오른쪽 버튼으로 클릭한 다음 Make(만들기)를 선택합니다.

IAR Embedded Workbench에서 FreeRTOS 데모를 시작하려면

1. Texas Instruments CC3220SF-LAUNCHXL의 Sense On Power(SOP) 점퍼가 0 위치에 있는지 확인합니다. 자세한 내용은 단원을 참조하십시오. [SimpleLinkWi-Fi CC3x20, CC3x3x 네트워크 프로세서 사용자 안내서](#).
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. “경고: 초기화하지 못했습니다. EnergyTrace, “이 (가) 표시되면. 에 대한 자세한 내용 EnergyTrace를 참조하십시오. [MSPEnergyTrace 기술](#).
5. 디버거가 `main()`의 종단점에서 중지되면 디버깅 메뉴로 이동하여 이동을 선택합니다.

## CMaak와 프리어토스와 사용

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

CMaak로 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 정보는 [CMake를 사용하여 프리어토스와 사용 \(p. 22\)](#)을 참조하십시오.

## 문제 해결

AWS IoT 콘솔의 MQTT 클라이언트에 메시지가 나타나지 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 단원을 참조하십시오.[시작하기 문제 해결 \(p. 21\)](#).

## Windows Device Simulator 시작하기

이 자습서에서 FreeRTOS Windows Device Simulator를 시작하기 위한 지침을 제공합니다.

시작하려면 먼저 구성해야 합니다. AWS IoT 및 FreeRTOS 다운로드를 통해 디바이스를 AWS 클라우드. 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 딜렉터리의 경로를 이라고 합니다.[freertos](#).

FreeRTOS는 FreeRTOS 라이브러리와 지정한 플랫폼에 대한 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. 의 최신 버전을 설치하려면 [Npcap](#). “를 선택합니다. WinPcap 설치 중 API 호환 모드”.
2. [Microsoft Visual Studio](#)를 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. 아래에서 선택할 수 있습니다. 선택의 단원 C++로 데스크톱 개발 구성 요소.

3. 활성 유선 이더넷 연결이 있는지 확인합니다.
4. (선택 사항) CMake 기반 빌드 시스템을 사용하려면 최신 버전의 를 설치하십시오.[CMake](#). FreeRTOS를 사용하려면 CMake 버전 3.13 이상이 필요합니다.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 디바이스가 AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트를 선택하고 MQTT 테스트 클라이언트를 눌러 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

데모 프로젝트가 기기에서 성공적으로 실행되면 “Hello World!” 구독한 주제에 여러 번 전송되었습니다.

## FreeRTOS 데모 프로젝트 빌드 및 실행

Visual Studio 또는 CMake를 사용하여 FreeRTOS 프로젝트를 빌드할 수

### 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 버전을 선택하고 (드롭다운의 초기 값) 확인.

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. 첫 번째 실행에서 다음을 수행해야 합니다.[네트워크 인터페이스 선택 \(p. 132\)](#).

### 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 디렉토리를 설정합니다.

GUI에서 FreeRTOS 소스 코드 디렉토리를 설정합니다 (`freertos`)에 대한 소스 코드는 어디에 있습니까?.

Set `freertos/build...`에 대한 바이너리를 빌드 할 위치.

2. CMake 프로젝트를 구성합니다.

CMake GUI에서 항목 추가를 선택하고 Add Cache Entry(캐시 항목 추가) 창에서 다음 값을 설정합니다.

이름

AFR\_BOARD

## 유형

STRING

값

pc.windows

설명

(선택 사항)

3. 구성은 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 발전기 옵션을 그대로 두고 Finish.
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 프로젝트 및 선택 다음으로 설정 StartUp 프로젝트. 이를 통해 프로젝트를 빌드하고 실행할 수 있습니다. 첫 번째 실행에서 다음을 수행해야 합니다. [네트워크 인터페이스 선택 \(p. 132\)](#).

FreeRTOS에서 CMake 사용에 대한 자세한 내용은 단원을 참조하십시오. [CMake를 사용하여 프리어토스와 사용 \(p. 22\)](#).

## 네트워크 인터페이스 구성

데모 프로젝트의 첫 번째 실행 시 사용할 네트워크 인터페이스를 선택해야 합니다. 프로그램에서 네트워크 인터페이스가 계산됩니다. 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. 앞의 예에서 사용할 번호는 다음과 같습니다.1.

FreeRTOSConfig.h를 열고 configNETWORK\_INTERFACE\_TO\_USE를 유선 네트워크 인터페이스에 해당하는 번호로 설정합니다.

### Important

이더넷 인터페이스만 지원됩니다. Wi-Fi는 지원되지 않습니다.

## 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 일반적인 문제 해결 정보는 단원을 참조하십시오.[시작하기 문제 해결 \(p. 21\)](#).

## Xilinx Avnet 시작하기|MicroZed산업용 IoT Kit

이 자습서에서는 Xilinx Avnet을 시작하기 위한 지침을 제공합니다. MicroZed산업용 IoT Kit Xilinx Avnet이 없는 경우 MicroZed산업용 IoT Kit, AWS파트너 디바이스 카탈로그를 방문하여[파트너](#).

시작하려면 먼저 구성해야 합니다. AWS IoT FreeRTOS 다운로드를 통해 디바이스를 AWS 클라우드. 자세한 내용은 [첫 번째 단계 \(p. 14\)](#) 단원을 참조하세요. 이 자습서에서 FreeRTOS 다운로드 딜렉터리에 대한 경로는 라고 합니다.[freertos](#).

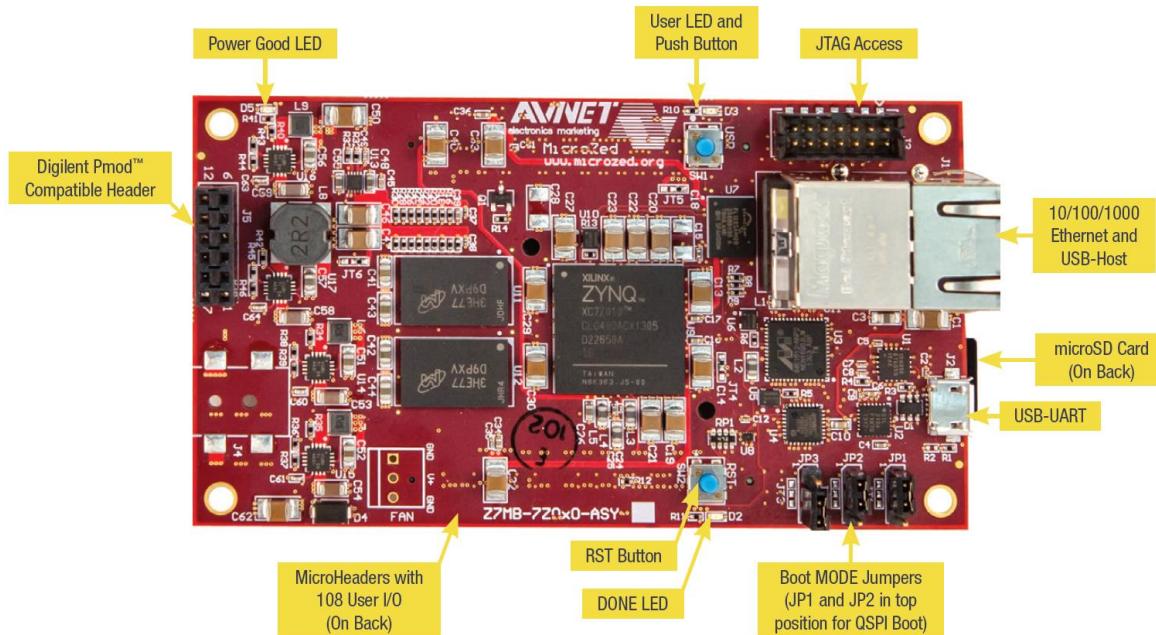
### 개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

- 보드를 호스트 시스템에 연결합니다.
- マイ크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
- FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
- 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

## 설정MicroZed하드웨어

다음 다이어그램은 을 설정할 때 유용할 수 있습니다. MicroZed하드웨어:



### 를 설정하려면MicroZed보드

1. 컴퓨터를 USB-UART 포트에 Connect.MicroZed보드.
2. 컴퓨터를 JTAG Access 포트에 Connect.MicroZed보드.
3. 라우터 또는 인터넷에 연결된 이더넷 포트를 이더넷 및 USB 호스트 포트에 Connect.MicroZed보드.

## 개발 환경 설정

에 대한 FreeRTOS 구성 설정하려면MicroZed키트를 사용하려면 Xilinx 소프트웨어 개발 키트 (XSDK) 를 사용해야 합니다. XSDK는 Windows 및 Linux에서 지원됩니다.

### XSDK 다운로드 및 설치

Xilinx 소프트웨어를 설치하려면 무료 Xilinx 계정이 필요합니다.

XSDK를 다운로드하려면

1. 로 이동합니다.[소프트웨어 개발 키트 독립형WebInstall클라이언트다운로드 페이지](#).
2. 운영 체제에 적합한 옵션을 선택합니다.
3. Xilinx 로그인 페이지로 이동합니다.

Xilinx에 계정이 있는 경우 사용자 이름과 암호를 입력한 후 Sign in을 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 감지되지 않는 경우 MicroZed의 USB-UART 연결에서 CP210x USB-to-UART Bridge VCP 드라이버를 수동으로 설치합니다. 지침은 [Silicon Labs CP210x USB-to-UART Installation Guide](#)를 참조하십시오.

XSDK에 대한 자세한 내용은 Xilinx 웹 사이트에서 [Getting Started with Xilinx SDK](#)를 참조하십시오.

## 클라우드에서 MQTT 메시지 모니터링

FreeRTOS 데모 프로젝트를 실행하기 전에 MQTT 클라이언트를 AWS IoT 콘솔에서 디바이스가 전송되는 메시지를 모니터링합니다. AWS 클라우드.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 를 선택합니다. 테스트를 선택합니다. MQTT 테스트 클라이언트를 눌러 MQTT 클라이언트를 엽니다.
3. 구독 주제에 **your-thing-name/example/topic**을 입력한 다음 주제 구독을 선택합니다.

## FreeRTOS 데모 프로젝트 빌드 및 실행

XSDK IDE에서 FreeRTOS 데모를 엽니다.

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. 온프로젝트 가져오기페이지 선택루트 디렉터리 선택데모 프로젝트의 루트 디렉터리를 입력합니다. `freertos/projects/xilinx/microzed/xsdk/aws_demos`. 디렉터리를 찾아보려면 검색.

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`, `fsb1` 및 `MicroZed_hw_platform_0` 프로젝트가 표시되지 않는 경우 이전 단계의 #3부터 반복하되, 루트 디렉터리를 `freertos/vendors/xilinx`로 설정하고 `aws_bsp`, `fsb1` 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. 추가(Add)를 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. 을 설정하려면 MicroZed보드의 부팅 모드 점퍼를 JTAG 부팅 모드로 이동합니다.

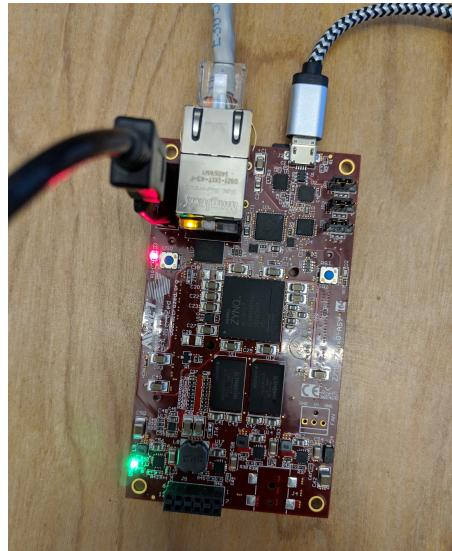


2. USB-UART 포트 바로 아래에 있는 MicroSD 카드 슬롯에 MicroSD 카드를 삽입합니다.

### Note

디버깅하기 전에 MicroSD 카드에 있는 모든 콘텐츠를 백업하십시오.

보드는 다음과 같은 모습이어야 합니다.



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.

부팅하도록 선택할 수 있습니다. MicroZedMicroSD 카드 또는 QSPI 플래시에서 보드를 사용하여 FreeRTOS 데모 프로젝트를 실행합니다. 자첨은 [FreeRTOS 데모 프로젝트의 부팅 이미지 생성 \(p. 136\)](#) 및 [FreeRTOS 데모 프로젝트 실행 \(p. 137\)](#) 단원을 참조하십시오.

## FreeRTOS 데모 프로젝트 실행

FreeRTOS 데모 프로젝트를 실행하려면 MicroZedMicroSD 카드 또는 QSPI 플래시에서 보드를 실행합니다.

을 설정할 때 MicroZedFreeRTOS 데모 프로젝트를 실행하기 위한 보드, 다이어그램을 참조하십시오. [설정 MicroZed 하드웨어 \(p. 133\)](#). 을 연결했는지 확인합니다. MicroZed 컴퓨터에 탑승합니다.

### MicroSD 카드에서 FreeRTOS 프로젝트 부팅

Xilinx와 함께 제공되는 MicroSD 카드를 포맷합니다. MicroZed 산업용 IoT Kit

1. BOOT.bin 파일을 MicroSD 카드에 복사합니다.
2. USB-UART 포트 바로 아래의 MicroSD 카드 슬롯에 카드를 삽입합니다.
3. 다음을 설정합니다. MicroZed 부팅 모드 점퍼를 SD 부팅 모드로 이동합니다.

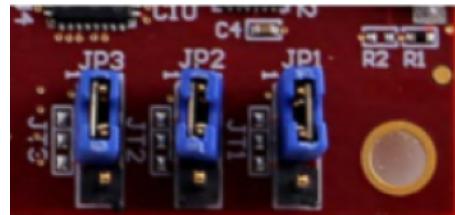
SD Card



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. 을 설정하려면 MicroZed보드의 부팅 모드 점퍼를 JTAG 부팅 모드로 이동합니다.



2. 컴퓨터가 USB-UART 및 JTAG Access 포트에 연결되어 있는지 확인합니다. 녹색 Power Good LED 조명이 커져야 합니다.
3. XSDK IDE의 메뉴에서 Xilinx를 선택한 후 Program Flash를 선택합니다.
4. Program Flash Memory에서 하드웨어 플랫폼이 자동으로 채워져야 합니다. 용연결을 선택합니다. MicroZed보드를 호스트 컴퓨터와 연결하는 하드웨어 서버.

#### 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 fsbl1.e1f 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. 을 설정하려면 MicroZed보드의 부팅 모드 점퍼를 QSPI 부팅 모드로 이동합니다.
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.

## 문제 해결

잘못된 경로와 관련된 빌드 오류가 발생한 경우 [FreeRTOS 데모 프로젝트 빌드 \(p. 136\)](#)에 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 단원을 참조하십시오. [시작하기 문제 해결 \(p. 21\)](#).

## FreeRTOS 다음 단계

FreeRTOS 데모 프로젝트를 빌드, 플래시 및 실행한 후에는 FreeRTOS 웹 사이트를 방문하여 자세히 알아 볼 수 있습니다. [새로운 FreeRTOS 프로젝트 만들기](#). 중요한 작업을 수행하고 상호 작용하는 방법을 보여주

는 많은 FreeRTOS 라이브러리에 대한 데모도 있습니다. AWS IoT 서비스 및 프로그램 보드 관련 기능 (예: 셀룰러 모뎀) 자세한 내용은 단원을 참조하십시오. [FreeRTOS 라이브러리 카테고리](#)페이지

FreeRTOS.org 웹 사이트에는 다음과 같은 심층적인 정보가 있습니다. [FreeRTOS 커널기본](#) 실시간 운영 체제 개념도 제공합니다. 자세한 내용은 단원을 참조하십시오. [FreeRTOS 커널 개발자 문서](#)과 [FreeRTOS 커널 보조 문서](#)페이지

# FreeRTOS 무선 업데이트

무선(OTA) 업데이트를 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 서명](#)을 사용하여 파일에 서명하거나 자체 코드 서명 도구를 사용하여 파일에 서명할 수 있습니다.

OTA 업데이트를 생성할 경우 [OTA 업데이트 관리자 서비스 \(p. 180\)](#)에서는 디바이스에 업데이트가 사용 가능함을 알려주는 [AWS IoT 작업](#)을 생성합니다. OTA 데모 애플리케이션은 디바이스에서 실행되고 알림 주제를 구독하는 FreeRTOS 작업을 생성합니다. AWS IoT 작업 및 업데이트 메시지를 수신합니다. 업데이트를 사용할 수 있는 경우 OTA 에이전트가 요청을 게시합니다. AWS IoT 선택한 설정에 따라 HTTP 또는 MQTT 프로토콜을 사용하여 업데이트를 수신합니다. OTA 에이전트는 다운로드한 파일의 디지털 서명을 확인하고 유효할 경우 펌웨어 업데이트를 설치합니다. FreeRTOS OTA 업데이트 데모 애플리케이션을 사용하지 않을 경우 [AWS IoT무선 \(OTA\) 라이브러리 \(p. 216\)](#) 펌웨어 업데이트 기능을 사용하려면 자체 애플리케이션에 연결할 수 있습니다.

FreeRTOS over-the-air 업데이트를 통해 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 방식으로 분류할 수 있습니다(예: 용도, 소유자 또는 환경 기준). 이는 동일한 유형의 리소스가 많을 때 유용합니다. 지정한 태그를 기반으로 리소스를 신속하게 식별할 수 있습니다.

자세한 내용은 [AWS IoT 리소스에 태그 지정](#) 단원을 참조하십시오.

## OTA 업데이트 사전 조건

를 사용하려면 over-the-air (OTA) 업데이트에서 다음을 수행합니다.

- [HTTP를 사용한 OTA 업데이트 사전 조건 \(p. 152\)](#) 또는 [MQTT를 사용한 OTA 업데이트 사전 조건 \(p. 151\)](#)를 확인합니다.
- 업데이트를 저장할 Amazon S3 버킷 생성 ([p. 141](#)).
- OTA 업데이트 서비스 역할 생성 ([p. 141](#)).

- OTA 사용자 정책 생성 (p. 143).
- 코드 서명 인증서 생성 (p. 144).
- Code Signing for AWS IoT를 사용할 경우 Code Signing for AWS IoT에 대한 액세스 권한 부여 (p. 149)를 선택하십시오.
- OTA 라이브러리와 함께 FreeRTOS 다운로드 (p. 150).

## 업데이트를 저장할 Amazon S3 버킷 생성

OTA 업데이트 파일은 Amazon S3 버킷에 저장됩니다.

AWS IoT용 코드 서명을 사용할 경우 코드 서명 작업을 생성하는 데 사용하는 명령에서는 서명되지 않은 펌웨어 이미지가 있는 소켓 버킷과 서명된 펌웨어가 기록되는 대상 버킷을 사용합니다. 소스와 대상 모두에 동일한 버킷을 지정할 수 있습니다. 원래 파일을 덮어쓰지 않도록 파일 이름이 GUID로 변경됩니다.

Amazon S3 버킷을 생성하려면

1. 에서 Amazon S3 콘솔에 로그인합니다.<https://console.aws.amazon.com/s3/>.
2. 버킷 만들기를 선택합니다.
3. 를 입력합니다.버킷 이름.
4. Under퍼블릭 액세스 차단에 대한 버킷 설정유지모든 퍼블릭 액세스 차단기본 권한을 수락하도록 선택되었습니다.
5. Under버킷 버전 관리를 선택합니다.활성화모든 버전을 동일한 버킷에 유지합니다.
6. 버킷 만들기를 선택합니다.

Amazon S3 대한 자세한 내용은 단원을 참조하십시오.[Amazon 심플 스토리지 서비스 사용 설명서](#).

## OTA 업데이트 서비스 역할 생성

OTA 업데이트 서비스에서는 이 역할을 사용하여 고객 대신 OTA 업데이트 작업을 생성하고 관리한다고 가정합니다.

OTA 서비스 역할을 생성하려면

1. 에 로그인합니다.<https://console.aws.amazon.com/iam/>.
2. 탐색 창에서 역할을 선택합니다.
3. 역할 생성(Create role)을 선택합니다.
4. [신뢰할 수 있는 유형의 엔터티 선택>Select type of trusted entity] 아래에서 AWS 서비스를 선택합니다.
5. 선택IoT목록에서AWS서비스.
6. 사용 사례 선택>Select your use case 아래에서 IoT를 선택합니다.
7. [다음: 권한(Next: Permissions)]을 선택합니다.
8. [다음: 권한(Next: Tags)]를 선택합니다.
9. [다음: 권한(Next: Review)]를 선택합니다.
10. 역할 이름과 설명을 입력한 후 역할 생성을 선택합니다.

IAM 역할에 대한 자세한 내용은 단원을 참조하십시오.[IAM 역할](#).

### Important

흔란스러운 부국장 보안 문제를 해결하려면[AWS IoT Core](#)안내서.

#### OTA 서비스 역할에 OTA 업데이트 권한을 추가하려면

1. IAM 콘솔 페이지의 검색 상자에 역할 이름을 입력한 후 목록에서 해당 역할을 선택합니다.
2. 정책 연결(Attach policies)을 선택합니다.
3. 검색 상자에 "AmazonFreeRTOSOTAUpdate"를 입력하고 필터링된 정책 목록에서 AmazonFreeRTOSOTAUpdate를 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 해당 AWS 계정 ID로 바꾸고 **your\_role\_name**을 OTA 서비스 역할 이름으로 바꿔야 합니다.

5. 정책 검토(Review policy)를 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 정책 AmazonFreeRTOSOTAUpdate에 필요한 권한이 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:::your_bucket_name/*"  
        }  
    ]  
}
```

```

        "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 검토(Review policy)를 선택합니다.
6. 정책 이름을 입력한 후 정책 생성을 선택합니다.

## OTA 사용자 정책 생성

IAM 사용자에게 수행 권한을 부여해야 합니다. over-the-air 업데이트. IAM 사용자에게 다음에 대한 권한이 있어야 합니다.

- 펌웨어 업데이트를 저장할 S3 액세스
- AWS Certificate Manager에 저장된 인증서 액세스
- AWS IoT MQTT 기반 파일 전송 기능.
- FreeRTOS OTA 업데이트에 액세스
- AWS IoT 작업 액세스
- IAM에 액세스.
- Code Signing for AWS IoT 액세스 [Code Signing for AWS IoT에 대한 액세스 권한 부여 \(p. 149\)](#) 단원을 참조하세요.
- FreeRTOS 하드웨어 플랫폼을 나열합니다.

IAM 사용자에게 필요한 권한을 부여하려면 OTA 사용자 정책을 생성한 후 IAM 사용자에게 연결합니다. 자세한 내용은 단원을 참조하십시오.[IAM 정책](#).

OTA 사용자 정책을 생성하려면

1. 열기<https://console.aws.amazon.com/iam/>콘솔.
2. 탐색 창에서 사용자를 선택합니다.
3. 목록에서 IAM 사용자를 선택합니다.
4. 권한 추가(Add permissions)를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. 정책 생성(Create policy)을 선택합니다.
7. JSON 탭을 선택하고 다음 정책 문서를 복사하여 정책 편집기에 붙여 넣습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket",
                "s3>ListAllMyBuckets",
                "s3>CreateBucket",
                "s3>PutBucketVersioning",
                "s3>GetBucketLocation",
                "s3>GetObjectVersion",
                "s3>ListBucketVersions",
                "s3>DeleteObject"
            ]
        }
    ]
}
```

```
        "acm:ImportCertificate",
        "acm>ListCertificates",
        "iot:*",
        "iam>ListRoles",
        "freertos>ListHardwarePlatforms",
        "freertos>DescribeHardwarePlatform"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject"
    ],
    "Resource": "arn:aws:s3:::example-bucket/*"
},
{
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::your-account-id:role/role-name"
}
]
}
```

Replace## ##OTA 업데이트 펌웨어 이미지가 저장되는 Amazon S3 버킷의 이름을 사용합니다.  
Replace## ID당신과 함께 AWS 계정 ID입니다. 를 찾을 수 있습니다. AWS 콘솔의 오른쪽 상단에 있는 계정 ID를 입력할 때 대시(-)를 제거합니다. Replace## ##을 사용하여 방금 생성한 IAM 서비스 역할의 이름입니다.

8. 정책 검토(Review policy)를 선택합니다.
9. 새 OTA 사용자 정책의 이름을 입력하고 정책 생성을 선택합니다.

#### IAM 사용자에게 OTA 사용자 정책을 연결하려면

1. IAM 콘솔의 탐색 창에서 사용자를 선택한 후 사용자를 선택합니다.
2. 권한 추가(Add permissions)를 선택합니다.
3. [Attach existing policies directly]를 선택합니다.
4. 방금 생성한 OTA 사용자 정책을 검색하고 옆에 있는 확인란을 선택합니다.
5. [다음: 권한(Next: Review)]를 선택합니다.
6. 권한 추가(Add permissions)를 선택합니다.

## 코드 서명 인증서 생성

펌웨어 이미지에 디지털 방식으로 서명하려면 코드 서명 인증서와 프라이빗 키가 필요합니다. 테스트를 위해 자체 서명 인증서와 프라이빗 키를 생성할 수 있습니다. 프로덕션 환경의 경우 잘 알려진 CA(인증 기관)를 통해 인증서를 구입합니다.

플랫폼에 따라 필요한 코드 서명 인증서 유형이 다릅니다. 다음 단원에서는 서로 다른 FreeRTOS 인증 플랫폼에 대한 코드 서명 인증서를 생성하는 방법을 설명합니다.

#### 주제

- Texas Instruments CC3220SF-LAUNCHXL을 위한 코드 서명 인증서 생성 (p. 145)
- Espressif ESP32에 대한 코드 서명 인증서 생성 (p. 146)
- Nordic nrf52840-dk에 대한 코드 서명 인증서 생성 (p. 147)
- FreeRTOS Windows 시뮬레이터에 대한 코드 서명 인증서 생성 (p. 148)

- 사용자 지정 하드웨어에 대한 코드 서명 인증서 생성 (p. 149)

## Texas Instruments CC3220SF-LAUNCHXL을 위한 코드 서명 인증서 생성

이 SimpleLink Wi-Fi CC3220SF 무선 마이크로 컨트롤러 런치패드 개발 키트는 펌웨어 코드 서명을 위한 두 가지 인증서 체인을 지원합니다.

- 프로덕션(인증서 카탈로그)

프로덕션 인증서 체인을 사용하려면 상용 코드 서명 인증서를 구매하고 [TI Uniflash 도구](#)를 사용하여 보드를 프로덕션 모드로 설정해야 합니다.

- 테스트 및 개발(인증서 실습)

놀이터 인증서 체인을 사용하면 자체 서명 코드 서명 인증서로 OTA 업데이트를 시험해 볼 수 있습니다.

AWS Command Line Interface를 사용하여 코드 서명 인증서, 프라이빗 키, 인증서 체인을 AWS Certificate Manager로 가져와야 합니다. 자세한 내용은 [다음 설치AWS CLI](#)의 AWS Command Line Interface 사용 설명서를 참조하십시오.

최신 버전의 [SimpleLink CC3220 SDK](#)를 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)
```

에 있는 인증서 SimpleLink CC3220 SDK는 DER 형식입니다. 자체 서명된 코드 서명 인증서를 만들려면 해당 인증서를 PEM 형식으로 변환해야 합니다.

다음 단계에 따라 Texas Instruments 실습 인증서 계층 구조에 연결되고 AWS Certificate Manager 및 Code Signing for AWS IoT 기준을 충족하는 코드 서명 인증서를 생성합니다.

### Note

코드 서명 인증서를 생성하려면 컴퓨터에 [OpenSSL](#)을 설치합니다. OpenSSL을 설치한 후 명령 프롬프트나 터미널 환경에서 openssl이 OpenSSL 실행 파일에 할당되었는지 확인합니다.

자체 서명된 코드 서명 인증서를 생성하려면

- 명령 프롬프트 창이나 터미널을 관리자 권한으로 엽니다.
- 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
```

- 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 Manager으로 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 Code Signing for AWS IoT를 사용하여 펌웨어 이미지에 서명한다는 가정하에 작성되었습니다. Code Signing for AWS IoT를 사용하는 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수도 있습니다.

## Espressif ESP32에 대한 코드 서명 인증서 생성

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 openssl이 OpenSSL 실행 파일에 할당되었는지 확인합니다. AWS Command Line Interface를 사용하여 코드 서명 인증서, 프라이빗 키, 인증서 체인을 AWS Certificate Manager로 가져와야 합니다. AWS CLI 설치에 대한 자세한 내용은 [AWS CLI 설치](#)를 참조하세요.

- 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
```

- 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
```

- 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
```

- 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 AWS Certificate Manager으로 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 Code Signing for AWS IoT를 사용하여 펌웨어 이미지에 서명한다는 가정하에 작성되었습니다. Code Signing for AWS IoT를 사용하는 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수도 있습니다.

## Nordic nrf52840-dk에 대한 코드 서명 인증서 생성

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 openssl이 OpenSSL 실행 파일에 할당되었는지 확인합니다. AWS Command Line Interface를 사용하여 코드 서명 인증서, 프라이빗 키, 인증서 체인을 AWS Certificate Manager로 가져와야 합니다. AWS CLI 설치에 대한 자세한 내용은 [AWS CLI 설치](#)를 참조하세요.

- 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 Manager으로 가져옵니다.

```
aws acm import-certificate --certificate file:///ecdsasigner.crt --private-key file:///  
ecdsasigner.key
```

이 명령은 인증서에 대한 ARN을 표시합니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

Note

이 단계는 Code Signing for AWS IoT를 사용하여 펌웨어 이미지에 서명한다는 가정하에 작성되었습니다. Code Signing for AWS IoT를 사용하는 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수도 있습니다.

## FreeRTOS Windows 시뮬레이터에 대한 코드 서명 인증서 생성

FreeRTOS 윈도우 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 [OpenSSL](#)을 설치합니다. OpenSSL을 설치한 후 명령 프롬프트나 터미널 환경에서 openssl이 OpenSSL 실행 파일에 할당되었는지 확인합니다.

AWS Command Line Interface를 사용하여 코드 서명 인증서, 프라이빗 키, 인증서 체인을 AWS Certificate Manager로 가져와야 합니다. AWS CLI 설치에 대한 자세한 내용은 [AWS CLI 설치](#)를 참조하세요.

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 Manager으로 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 Code Signing for AWS IoT를 사용하여 펌웨어 이미지에 서명한다는 가정하에 작성되었습니다. Code Signing for AWS IoT를 사용하는 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수도 있습니다.

## 사용자 지정 하드웨어에 대한 코드 서명 인증서 생성

적절한 도구 세트를 사용하여 하드웨어에 대해 자체 서명된 인증서와 프라이빗 키를 생성합니다.

AWS Command Line Interface를 사용하여 코드 서명 인증서, 프라이빗 키, 인증서 체인을 AWS Certificate Manager로 가져와야 합니다. AWS CLI 설치에 대한 자세한 내용은 [AWS CLI 설치](#)를 참조하세요.

코드 서명 인증서를 생성한 후 를 사용할 수 있습니다.AWS CLIACM으로 가져오려면 다음을 수행합니다.

```
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.

ACM은 특정 알고리즘 및 키 크기를 사용하려면 인증서가 필요합니다. 자세한 내용은 [인증서를 가져오기 위한 사전 조건](#)을 참조하십시오. ACM에 대한 자세한 내용은 단원을 참조하세요.[로 인증서 가져오기AWS Certificate Manager](#).

코드 서명 인증서의 내용을 복사하여 에 붙여 넣고 서식을 지정해야 합니다.`vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h`나중에 다운로드한 FreeRTOS 코드의 일부인 파일입니다.

## Code Signing for AWS IoT에 대한 액세스 권한 부여

프로덕션 환경에서 펌웨어 업데이트에 디지털 방식으로 서명하여 업데이트의 신뢰성과 무결성을 확인해야 합니다. 업데이트를 수동으로 서명하거나 Code Signing for AWS IoT를 사용하여 코드에 서명할 수 있습니다. FreeRTOS용 코드 서명을 사용하려면 IAM 사용자 계정에 FreeRTOS용 코드 서명에 대한 액세스 권한을 부여해야 합니다.

IAM 사용자 계정에 Code Signing for 에 대한 권한을 부여하려면 AWS IoT

1. 에 로그인합니다.<https://console.aws.amazon.com/iam/>.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 생성(Create Policy)을 선택합니다.
4. JSON 탭에서 다음 JSON 문서를 복사하여 정책 편집기에 붙여 넣습니다. 이 정책을 사용하면 IAM 사용자가 모든 코드 서명 작업에 액세스할 수 있습니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "signer:*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

5. 정책 검토(Review policy)를 선택합니다.
6. 정책 이름과 설명을 입력한 다음 정책 생성을 선택합니다.
7. 탐색 창에서 사용자를 선택합니다.
8. IAM 사용자 계정을 선택합니다.
9. 권한 탭에서 권한 추가를 선택합니다.
10. Attach existing policies directly(기존 정책 직접 연결)을 선택한 후 방금 생성한 코드 서명 정책 옆에 있는 확인란을 선택합니다.
11. [다음: 권한(Next: Review)]를 선택합니다.
12. 권한 추가(Add permissions)를 선택합니다.

## OTA 라이브러리와 함께 FreeRTOS 다운로드

에서 FreeRTOS를 다운로드할 수 있습니다. FreeRTOS 콘솔 (p. 11) 또는 에서 FreeRTOS를 복제하거나 다운로드할 수 있습니다. GitHub. 자세한 내용은 README.md 파일을 참조하십시오.

콘솔에서 다운로드한 FreeRTOS 구성에 OTA 라이브러리를 포함하려면 사전 정의된 구성을 사용자 지정하거나 OTA 기능을 지원하는 플랫폼에 대해 구성을 생성하면 됩니다. 온FreeRTOS 소프트웨어 구성구성 속성 페이지, 아래 Libraries, 선택 OTA 업데이트. 데모 프로젝트에서 OTA 데모를 활성화하도록 선택할 수 있습니다. 나중에 수동으로 데모를 활성화할 수도 있습니다.

OTA 데모 애플리케이션 설정 및 실행에 대한 자세한 내용은 OTA(Over-the-Air) 업데이트 데모 애플리케이션 (p. 271) 단원을 참조하십시오.

### Important

- 이 주제에서 FreeRTOS 다운로드 디렉터리의 경로를 이라고 합니다. *freertos*.
- *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.
- 소스 코드에 심볼릭 링크가 포함될 수 있으므로 Windows를 사용하여 아카이브를 추출하는 경우 다음을 수행해야 할 수 있습니다.
  - 활성화 개발자 모드 또는

- 관리자 권한으로 상승된 콘솔을 사용합니다.

이러한 방식으로 Windows는 아카이브를 추출할 때 심볼릭 링크를 올바르게 만들 수 있습니다. 그렇지 않으면 심볼 링크의 경로가 텍스트로 포함되거나 비어 있는 일반 파일로 심볼 링크가 작성됩니다. 자세한 내용은 블로그 항목을 참조하십시오. [윈도우 10의 심볼릭 링크!](#).

Windows에서 Git을 사용하는 경우 개발자 모드를 활성화해야 합니다. 그렇지 않으면 다음을 수행해야 합니다.

- Setcore.symlinks 다음 명령을 사용하여 true로 설정합니다.

```
git config --global core.symlinks true
```

- 시스템에 쓰는 git 명령을 사용할 때마다 관리자 권한으로 상승된 콘솔을 사용하십시오 (예:git pull, git clone, 및 git submodule update --init --recursive).

## MQTT를 사용한 OTA 업데이트 사전 조건

이 단원에서는 MQTT를 사용하여 수행하기 위한 일반적인 요구 사항에 대해 설명합니다. over-the-air (OTA 업데이트).

### 최소 요구 사항

- 디바이스 펌웨어에는 필요한 FreeRTOS 라이브러리 (CoreMQTT 에이전트, OTA 업데이트 및 해당 종속성) 가 포함되어 있어야 합니다.
- FreeRTOS 버전 1.4.0 이상이 필요합니다. 하지만 가능한 경우 최신 버전을 사용하는 것이 좋습니다.

### Configurations

버전 201912.00부터 FreeRTOS OTA는 HTTP 또는 MQTT 프로토콜을 사용하여 펌웨어 업데이트 이미지를 전송할 수 있습니다. AWS IoT 디바이스로 이동합니다. FreeRTOS에서 OTA 업데이트를 생성할 때 두 프로토콜을 모두 지정하면 각 디바이스에서 이미지를 전송하는 데 사용되는 프로토콜을 결정합니다. 자세한 내용은 [HTTP를 사용한 OTA 업데이트 사전 조건 \(p. 152\)](#) 섹션을 참조하세요.

기본적으로 의 OTA 프로토콜 구성은 `ota_config.h` MQTT 프로토콜을 사용하는 것입니다.

### 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 장치는 AWS IoT에 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 리소스](#)에서 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/*"  
    ]  
}  
}
```

## 참고

- iot:Connect 권한을 사용하면 MQTT를 통해 디바이스를 AWS IoT에 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/\*) 항목에 대한 iot:Subscribe 및 iot:Publish 권한을 사용하면 연결된 디바이스에서 AWS IoT로부터 OTA 업데이트 데이터를 가져올 수 있습니다. MQTT를 통해 펌웨어 업데이트를 수행하려면 이러한 권한이 필요합니다.
- iot:Receive 권한을 사용하면 AWS IoT Core에서 해당 주제에 대한 메시지를 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.

## HTTP를 사용한 OTA 업데이트 사전 조건

이 단원에서는 HTTP를 사용하여 수행하는 일반적인 요구 사항에 대해 설명합니다. over-the-air (OTA) 업데이트. 버전 201912.00부터 FreeRTOS OTA는 HTTP 또는 MQTT 프로토콜을 사용하여 펌웨어 업데이트 이미지를 전송할 수 있습니다. AWS IoT 디바이스로 이동합니다.

### Note

- 펌웨어 이미지를 전송하는 데 HTTP 프로토콜을 사용할 수도 있지만 외의 다른 상호 작용에서 CoreMQTT 에이전트 라이브러리가 여전히 필요합니다. AWS IoT Core 작업 실행 알림, 작업 문서, 실행 상태 업데이트 송수신 등 CoreMqtt 에이전트 라이브러리를 사용합니다.
- 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 에이전트, HTTP, OTA 에이전트 및 해당 종속성) 가 포함되어 있어야 합니다.
- FreeRTOS 버전 201912.00 이상이 HTTP를 통한 OTA 데이터 전송을 활성화하려면 OTA 프로토콜의 구성 을 변경해야 합니다.

## Configurations

\vendors\boards\board\aws\_demos\config\_files\ota\_config.h 파일에서 다음과 같은 OTA 프로토콜 구성은 참조하십시오.

HTTP를 통한 OTA 데이터 전송을 활성화하려면

- configENABLED\_DATA\_PROTOCOLS를 OTA\_DATA\_OVER\_HTTP로 변경합니다.
- OTA가 업데이트되면 MQTT 또는 HTTP 프로토콜을 사용할 수 있도록 두 프로토콜을 모두 지정할 수 있습니다. configOTA\_PRIMARY\_DATA\_PROTOCOL을 OTA\_DATA\_OVER\_HTTP로 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\_WIFI를 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 FreeRTOS를 사용하여 계산된 모든 지원되는 플랫폼의 힙 메모리 사용량 데이터입니다. xPortGetFreeHeapSize API. 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): 43 KB

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): 82 KB

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 IoT에 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 리소스](#)에서 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/${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

```
        "Resource": [
            "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
        ]
    }
}
```

### 참고

- iot:Connect 권한을 사용하면 MQTT를 통해 디바이스를 AWS IoT에 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.
- iot:Receive 권한을 사용하면 AWS IoT Core에서 해당 주제에 대한 메시지를 현재 연결된 디바이스에게 게시할 수 있습니다. MQTT 메시지를 전송할 때마다 이 권한을 확인합니다. 이 권한을 사용하면 주제를 현재 구독 중인 클라이언트에 대한 액세스 권한을 최소화할 수 있습니다.

## OTA 자습서

이 단원에는 OTA 업데이트를 사용하여 FreeRTOS를 실행하는 디바이스에서 펌웨어를 업데이트하기 위한 자습서가 펌웨어 이미지 외에도 OTA 업데이트를 사용하여 모든 유형의 파일을 AWS IoT에 연결된 디바이스로 보낼 수 있습니다.

AWS IoT 콘솔 또는 AWS CLI를 사용하여 OTA 업데이트를 생성할 수 있습니다. 콘솔을 사용하면 많은 작업이 자동으로 수행되므로 OTA를 가장 쉽게 시작할 수 있습니다. 이 AWS CLI는 OTA 업데이트 작업을 자동화하거나, 많은 수의 디바이스로 작업하거나, FreeRTOS에 적격하지 않은 디바이스를 사용 중인 경우에 유용합니다. FreeRTOS용 디바이스 검증에 대한 자세한 내용은 단원을 참조하십시오.[FreeRTOS 파트너 웹 사이트](#).

### OTA 업데이트를 생성하려면

- 초기 펌웨어 버전을 하나 이상의 디바이스에 배포합니다.
- 펌웨어가 올바르게 실행되는지 확인합니다.
- 펌웨어를 업데이트해야 하는 경우 코드를 변경하고 새 이미지를 빌드합니다.
- 펌웨어에 수동으로 서명할 경우 서명된 펌웨어 이미지를 Amazon S3 버킷에 업로드합니다. 예 대한 코드 서명을 사용하는 경우 AWS IoT를 클릭하고 서명되지 않은 펌웨어 이미지를 Amazon S3 버킷에 업로드합니다.
- 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

펌웨어를 업데이트하려면 OTA 에이전트 라이브러리를 사용하여 OTA 업데이트 작업을 수신하는 초기 펌웨어 버전을 설치해야 합니다. FreeRTOS를 실행하지 않을 경우 이 단계를 건너뜁니다. 대신 OTA 에이전트 구현을 디바이스에 복사해야 합니다.

### 주제

- [Texas Instruments CC3220SF-LAUNCHXL에 초기 펌웨어 버전 설치 \(p. 156\)](#)

- Espressif ESP32에 초기 펌웨어 버전 설치 (p. 158)
- Nordic nRF52840 DK에 초기 펌웨어 버전 설치 (p. 160)
- Windows 시뮬레이터의 초기 펌웨어 (p. 161)
- 사용자 지정 보드에 초기 펌웨어 버전 설치 (p. 161)

## Texas Instruments CC3220SF-LAUNCHXL에 초기 펌웨어 버전 설치

이 단계는 Texas Instruments CC3220SF-LAUNCHXL에 대해 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행 (p. 274)에 설명된 대로 aws\_demos 프로젝트를 이미 빌드했다는 가정하에 작성되었습니다.

1. Texas Instruments CC3220SF-LAUNCHXL에서 SOP 점퍼를 중간 핀 집합(위치 = 1)에 두고 보드를 재 설정합니다.
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. New Project(새 프로젝트)를 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 아이콘 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 아이콘 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. [Open]을 선택합니다.
  - a. 파일 대화 상자에서 파일 이름이 mcuflashimg.bin으로 설정되어 있는지 확인합니다.
  - b. 공급업체 확인란을 선택합니다.
  - c. File Token(파일 토큰)에 1952007250을 입력합니다.
  - d. Private Key File Name(프라이빗 키 파일 이름)에서 찾아보기를 선택한 후 코드 서명 인증서와 프라이빗 키를 생성한 작업 디렉터리에서 tisigner.key를 선택합니다.
  - e. Certification File Name(인증서 파일 이름)에서 tisigner.crt.der을 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-cc3x20으로 이동하여 sp\_3.7.0.1\_2.0.0.0\_2.2.0.6.bin을 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 접합(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

```

## Espressif ESP32에 초기 펌웨어 버전 설치

이 안내서는 [Espressif ESP32-DevKitC](#) 및 [ESP-WROVER-KIT 시작하기](#) 및 [무선\(OTA\) 업데이트 사전 조건](#)의 단계를 이미 수행했다는 가정하에 작성되었습니다. OTA 업데이트를 시도하기 전에 예 설명된 MQTT 데모 프로젝트를 실행할 수 있습니다.[FreeRTOS 시작하기](#)보드 및 툴 체인이 올바르게 설정되어 있는지 확인합니다.

### 초기 출고 시 이미지를 보드에 플래시하려면

1. Open `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`를 주석으로 처리합니다.`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`를 열고 를 정의합니다.`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` 또는 `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
2. [OTA 업데이트 사전 조건 \(p. 140\)](#)에서 생성한 SHA-256/ECDSA PEM 형식 코드 서명 인증서를 `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h`에 복사합니다. 다음 방식으로 형식을 지정해야 합니다.

```

#define otapalconfigCODE_SIGNING_CERTIFICATE \
"-----BEGIN CERTIFICATE-----\n" \
"....base64 data...\n" \
"-----END CERTIFICATE-----\n";

```

3. OTA 업데이트 데모를 선택한 상태에서 [ESP32 시작하기](#)에 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`을 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
BLKO of EFUSE
I (123) system_api: Base MAC address is not set, read default base MAC address from
BLKO 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] vDHCPPProcess: 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] vDHCPPProcess: 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+J를 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

이 안내서는 [Nordic nRF52840-DK 시작하기 \(p. 107\)](#) 및 [무선\(OTA\) 업데이트 사전 조건](#)의 단계를 이미 수행했다는 가정하에 작성되었습니다. OTA 업데이트를 시도하기 전에 예 설명된 MQTT 데모 프로젝트를 실행할 수 있습니다. [FreeRTOS 시작하기](#) 보드 및 툴체인이 올바르게 설정되어 있는지 확인합니다.

### 초기 출고 시 이미지를 보드에 플래시하려면

1. `freertos/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/ aws_demo_config.h` 을(를) 업니다.
2. Replace#define CONFIG\_CORE\_MQTT\_MUTUAL\_AUTH\_DEMO\_ENABLED 와 CONFIG\_OTA\_MQTT\_UPDATE\_DEMO\_ENABLED 또 는 CONFIG\_OTA\_HTTP\_UPDATE\_DEMO\_ENABLED.

3. OTA 업데이트 데모를 선택한 상태에서 [Nordic nRF52840-DK 시작하기 \(p. 107\)](#)에 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

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

IDE를 사용하여 `aws_demos` 프로젝트를 빌드합니다. 이때 OTA 라이브러리를 포함해야 합니다. FreeRTOS 소스 코드의 구조에 대한 자세한 내용은 단원을 참조하십시오.[FreeRTOS 데모 \(p. 233\)](#).

FreeRTOS 프로젝트 또는 디바이스에 코드 서명 인증서, 프라이빗 키 및 인증서 트러스트 체인을 포함해야 합니다.

적절한 도구를 사용하여 애플리케이션을 보드에 번인하고 올바르게 실행되는지 확인합니다.

## 펌웨어 버전 업데이트

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. 파일 찾기 `/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` 다음 값을 증가시킵니다. 앱\_버전\_빌드.
3. 0이 아닌 파일 유형이 아닌 Renesas rx65n 플랫폼에 대한 업데이트를 예약하려면 Renesas Secure Flash Programmer 도구를 사용하여 파일에 서명해야 합니다. 그렇지 않으면 장치에서 서명 확인이 실패합니다. 이 도구는 확장자를 사용하여 서명된 파일 패키지를 만듭니다..rsuRenesas의 소유자 파일 유

형입니다. 도구는 다음에서 찾아볼 수 있습니다.[GitHub](#). 다음 예제 명령을 사용하여 이미지를 생성할 수 있습니다.

```
"Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure  
Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
```

4. 프로젝트를 다시 빌드합니다.

에 설명된 대로 펌웨어 업데이트를 생성한 Amazon S3 버킷에 복사해야 합니다.[업데이트를 저장할 Amazon S3 버킷 생성 \(p. 141\)](#). Amazon S3 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
- Espressif ESP32: vendors/espressif/boards/esp32/aws\_demos/make/build/aws\_demos.bin

## OTA 업데이트 생성(AWS IoT 콘솔)

1. AWS IoT 콘솔의 탐색 창에서 관리를 선택한 다음 작업을 선택합니다.
2. 생성(Create)을 선택합니다.
3. UnderFreeRTOS OTA (OTA) 업데이트 작업 생성, 선택OTA 업데이트 작업 생성.
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. 다음(Next)을 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. 비적격 플랫폼을 사용하는 경우 ECDSA P-256 SHA-256 이외의 다른 ciphersuite를 서명에 사용하면 AWS IoT용 코드 서명을 사용하거나 펌웨어 업데이트에 직접 서명할 수 있습니다. 자세한 정보는 [펌웨어 업데이트에 디지털 방식으로 서명 \(p. 164\)](#)을 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. 대부분의 디바이스에서 이 필드를 비워 둘 수 있습니다. Windows 시뮬레이터 및 인증서를 특정 파일 위치에 배치하는 장치의 경우 여기에 경로 이름을 입력합니다.

### 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. Under파일 형식에서 0-255 범위의 정수 값을 입력합니다. 입력한 파일 형식이 MCU로 전달되는 Job 문서에 추가됩니다. MCU 펌웨어/소프트웨어 개발자는 이 가치를 어떻게 처리해야 하는지에 대한 완전한 소유권을 가지고 있습니다. 가능한 시나리오에는 기본 프로세서와 독립적으로 펌웨어를 업데이트할 수 있는 보조 프로세서가 있는 MCU가 있습니다. 장치가 OTA 업데이트 작업을 수행하면 File Type 을 사용하여 업데이트되는 프로세서를 식별할 수 있습니다.
13. OTA 업데이트 작업의 IAM 역할에서 [OTA 업데이트 서비스 역할 생성 \(p. 141\)](#)의 지침에 따라 역할을 선택합니다.
14. 다음(Next)을 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 룰아웃, 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. 대부분의 디바이스에서 이 필드를 비워 둘 수 있습니다. Windows 시뮬레이터 및 인증서를 특정 파일 위치에 배치하는 장치의 경우 여기에 경로 이름을 입력합니다.
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. UnderAmazon S3 펌웨어 이미지 선택을 클릭하고 Amazon S3 버킷에서 Amazon S3 버킷과 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 콘솔에서는 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 \(p. 271\)](#) 단원을 참조하십시오.

## AWS CLI를 사용하여 OTA 업데이트 생성

AWS CLI를 사용하여 OTA 업데이트를 생성하는 경우,

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 CLI를 사용하여 OTA 업데이트를 수행할 때 AWS IoT용 코드 서명을 사용하거나 펌웨어 업데이트에 직접 서명할 수 있습니다. AWS IoT용 코드 서명에서 지원되는 암호화 서명 및 해싱 알고리즘 목록은 [SigningConfigurationOverrides](#)를 참조하세요. 코드 서명에서 지원되지 않는 암호화 알고리즘을 사용하려는 경우 AWS IoT를 통해 Amazon S3 업로드하려면 먼저 펌웨어 바이너리에 서명해야 합니다.

#### AWS IoT용 코드 서명을 사용하여 펌웨어 이미지에 서명

용 코드 서명을 사용하여 펌웨어 이미지에 서명하려면 AWS IoT 중 하나를 사용할 수 있습니다. [AWSSDK 또는 명령줄 도구](#). 용 코드 서명에 대한 자세한 내용 AWS IoT를 참조하십시오. [Code Signing for AWS IoT](#).

코드 서명 도구를 설치하고 구성한 후 서명되지 않은 펌웨어 이미지를 Amazon S3 버킷에 복사하고 다음을 사용하여 코드 서명 작업을 시작합니다. AWS CLI 명령 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

## ## ##과## ## ##Amazon S3 버킷이 될 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 선택한 후 페이지 상단의 다음 항목을 찾을 수 있습니다. 버전, 선택표시.

#### destination

S3 버킷에 서명된 펌웨어가 복사될 디바이스의 대상입니다. 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 platformId입니다.

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 버킷

을 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. AWS IoT용 코드 서명에 대한 자세한 내용은 [AWS IoT용 코드 서명](#)을 참조하십시오.

## 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 S3 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. 버킷의 내용을 조사하여 Amazon S3 콘솔에서 이 값을 찾을 수 있습니다.

Code Signing for AWS IoT를 사용 중인 경우 파일 이름은 Code Signing for AWS IoT에서 생성되는 GUID입니다.

사용 `aws iot create-stream` AWS CLI 명령을 사용하여 스트림을 생성합니다.

```
aws iot create-stream \  
  --stream-id your_stream_id \  
  --description your_description \  
  --files file://stream.json \  
  
```

```
--role-arn your_role_arn
```

이것들은create-stream AWS CLI명령:

**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.

Code Signing for AWS IoT를 사용하는 경우 이 키는 GUID입니다.

다음은 예제 stream.json 파일입니다.

```
[  
  {  
    "fileId":123,  
    "s3Location": {  
      "bucket": "codesign-ota-bucket",  
      "key": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"  
    }  
  }  
]
```

**role-arn**

이[OTA 서비스 역할](#) (p. 141)또한 펌웨어 이미지가 저장되는 Amazon S3 버킷에 대한 액세스 권한을  
부여합니다.

서명된 펌웨어 이미지의 Amazon S3 객체 키를 찾으려면aws signer describe-signing-job --job-id *my-job-id* 명령 위치*my-job-id*은 (는)에서 표시하는 작업 ID입니다.create-signing-job AWS CLI명령. 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

사용 `create-ota-update` AWS CLI OTA 업데이트 작업을 생성하는 명령을 선택합니다.

### 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

이름	유형	설명
otaUpdateId	문자열 (최대:128분:1)	생성할 OTA 업데이트의 ID입니다.
description	문자열 (최대:2028)	OTA 업데이트에 대한 설명입니다.
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. 유효한 값: 연속   스냅샷.  열거형: 지속적   스냅샷
awsJobExecutionsRolloutConfig		OTA 업데이트의 룰아웃에 대한 구성.
maximumPerMinute	integer (최대:1000분:1)	분당 시작한 OTA 업데이트 작업 실행 최대 수.
exponentialRate		작업 룰아웃 증가 속도입니다. 이 파라미터를 사용하면 작업 룰아웃에 대한 기하급수적 속도 증가를 정의할 수 있습니다.

이름	유형	설명
<code>baseRatePerMinute</code>	integer (최대:1000분:1)	작업 률아웃 시작 시 대기 중인 작업에 대한 알림을 받을 분당 최소 사를 수입니다. 이는 률아웃의 초기 속도입니다.
<code>rateIncreaseCriteria</code>		작업에 대한 률아웃 속도 증가를 시작하는 기준입니다.  AWS IoT는 소수점 이하 최대 1 자리까지 지원합니다. 예를 들어, 1.5는 지원하지만 1.55는 지원하지 않습니다.
<code>numberOfNotifiedThings</code>	integer (최소:1)	이 사물 수에 대한 알림을 받으면 률아웃 속도가 증가하기 시작합니다.
<code>numberOfSucceededThings</code>	integer (최소:1)	이 사물 수가 작업 실행에서 성공하면 률아웃 속도가 증가하기 시작합니다.
<code>awsJobPresignedUrlConfig</code>		미리 서명된 URL의 구성 정보입니다.
<code>expiresInSec</code>	bigint	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.
<code>awsJobAbortConfig</code>		작업 중단이 발생하는 시기와 방법을 결정하는 기준입니다.
<code>abortCriteriaList</code>	list	작업을 중지할 시기와 방법을 결정하는 기준 목록입니다.
<code>failureType</code>	문자열	작업 중단을 시작할 수 있는 작업 실행 실패 유형입니다.  열거형: 실패   거부됨   시간_OUT   모두
<code>action</code>	문자열	작업 중단을 시작하기 위해 수행 할 작업 유형입니다.  열거형: CANCEL
<code>minNumberOfExecutedThings</code>	integer (최소:1)	작업을 중지하기 전에 작업 실행 알림을 받아야 하는 최소 사를 수입니다.

이름	유형	설명
<code>awsJobTimeoutConfig</code>		각 디바이스가 작업 실행을 마쳐야 하는 시간을 지정합니다. 타이머는 작업 실행 상태가 <code>IN_PROGRESS</code> 에 설정되면 시작합니다. 타이머가 만료하기 전에 작업 실행 상태가 다른 터미널 상태로 설정되어 있지 않으면 <code>TIMED_OUT</code> 로 자동으로 설정됩니다.
<code>inProgressTimeoutInMinutes</code>	<code>bigint</code>	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 <code>IN_PROGRESS</code> 상태를 유지할 때마다 작업 실행은 실패하며 터미널 <code>TIMED_OUT</code> 상태로 전환합니다.
<code>files</code>	<code>list</code>	OTA 업데이트에서 스트리밍되는 파일입니다.
<code>fileName</code>	문자열	파일 이름입니다
<code>fileType</code>	<code>integer</code> 최대 범위: 255 최소 범위: 0	장치가 클라우드에서 수신한 파일 유형을 식별할 수 있도록 작업 문서에 포함할 수 있는 정수 값입니다.
<code>fileVersion</code>	문자열	파일 버전입니다.
<code>fileLocation</code>		업데이트된 펌웨어 위치
<code>stream</code>		OTA 업데이트를 포함하는 스트림.
<code>streamId</code>	문자열 (최대:128문:1)	스트림 ID입니다.
<code>fileId</code>	<code>integer</code> (최대:255문:0)	스트림과 연결된 파일의 ID입니다.
<code>s3Location</code>		S3에 업데이트된 펌웨어 위치.
<code>bucket</code>	문자열 (최소:1)	S3 버킷
<code>key</code>	문자열 (최소:1)	S3 키입니다.

이름	유형	설명
<code>version</code>	문자열	S3 버킷 버전
<code>codeSigning</code>		파일의 코드 서명 메서드입니다.
<code>awsSignerJobId</code>	문자열	의 ID입니다. AWSSignerJob 파일 서명을 위해 생성된 것입니다.
<code>startSigningJobParameter</code>		코드 서명 작업을 설명합니다.
<code>signingProfileParameter</code>		코드 서명 프로파일을 설명합니다.
<code>certificateArn</code>	문자열	인증서 ARN.
<code>platform</code>	문자열	디바이스 하드웨어 플랫폼.
<code>certificatePathOnDevice</code>	문자열	디바이스의 코드 서명 인증서의 위치.
<code>signingProfileName</code>	문자열	코드 서명 프로파일 이름.
<code>destination</code>		코드 서명한 파일을 쓸 위치.
<code>s3Destination</code>		업데이트된 펌웨어의 S3의 위치를 설명합니다.
<code>bucket</code>	문자열 (최소:1)	업데이트된 펌웨어를 포함하는 S3 버킷.
<code>prefix</code>	문자열	S3 접두사.
<code>customCodeSigning</code>		파일 코드 서명을 위한 사용자 지정 메서드입니다.
<code>signature</code>		파일 서명입니다.
<code>inlineDocument</code>	blob	코드 서명에 대한 base64 인코딩 바이너리 표현입니다.
<code>certificateChain</code>		인증서 체인입니다.
<code>certificateName</code>	문자열	인증서 이름입니다
<code>inlineDocument</code>	문자열	코드 서명 인증서 체인에 대한 base64 인코딩 바이너리 표현입니다.
<code>hashAlgorithm</code>	문자열	파일의 코드 서명에 사용되는 해시 알고리즘입니다.
<code>signatureAlgorithm</code>	문자열	파일의 코드 서명에 사용되는 서명 알고리즘입니다.
<code>attributes</code>	맵	이름/속성 페이의 목록입니다.

이름	유형	설명
roleArn	문자열 (최대:2048문:20)	부여하는 IAM 역할 AWS IoTAmazon S3에 대한 액세스 권한AWS IoTJob과 AWSOTA 업데이트 작업을 생성하는 코드 서명 리소스입니다.
additionalParameters	맵	이름-값 페어로 추가되는 OTA 업데이트 파라미터의 목록입니다.
tags	list	업데이트 관리에 사용할 수 있는 메타데이터입니다.
Key	문자열 (최대:128문:1)	태그 키.
Value	문자열 (최대: 256문:1)	태그 값.

#### 결과

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

#### AWS CLI 출력 필드

이름	유형	설명
otaUpdateId	문자열 (최대:128문:1)	OTA 업데이트 ID입니다.
awsIotJobId	문자열	이AWS IoTOTA 업데이트와 연결된 작업 ID입니다.
otaUpdateArn	문자열	OTA 업데이트 ARN입니다.
awsIotJobArn	문자열	이AWS IoTOTA 업데이트와 연결된 작업 ARN입니다.
otaUpdateStatus	문자열	OTA 업데이트 상태입니다.  열거형: 생성_보류 중   생성_진행   생성_완료   생성_실패

다음은 AWS IoT용 코드 서명을 사용하는 create-ota-update 명령에 전달되는 JSON 파일의 예입니다.

```
[
  {
    "fileName": "firmware.bin",
```

```

    "fileType": 1,
    "fileLocation": {
        "stream": {
            "streamId": "004",
            "fileId": 123
        }
    },
    "codeSigning": {
        "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
}
]

```

다음은 예 전달되는 JSON 파일의 예입니다.`create-ota-update` AWS CLI인라인 파일을 사용하여 사용자 지정 코드 서명 자료를 제공하는 명령입니다.

```

[
{
    "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 예 전달되는 JSON 파일의 예입니다.`create-ota-update` AWS CLIFreeRTOS OTA에서 코드 서명 작업을 시작하고 코드 서명 프로필 및 스트림을 생성하는 데 사용되는 명령

```

[
{
    "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",
                "profileArn": "your_profile_arn"
            }
        }
    }
}
]

```

```
        "certificatePathOnDevice": "certificate_path"
    },
    "destination": {
        "s3Destination": {
            "bucket": "your_destination_bucket"
        }
    }
}
]
```

다음은 예 전달되는 JSON 파일의 예입니다.create-ota-update AWS CLI기존 프로필로 코드 서명 작업을 시작하고 지정된 스트림을 사용하는 OTA 업데이트를 생성하는 명령

```
[
{
    "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 예 전달되는 JSON 파일의 예입니다.create-ota-update AWS CLIFreeRTOS OTA에서 기존 코드 서명 작업 ID로 스트림을 생성할 수 있는 명령입니다.

```
[
{
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning": {
        "awsSignerJobId": "your_signer_job_id"
    }
}]
```

다음은 예 전달되는 JSON 파일의 예입니다.create-ota-update AWS CLIOTA 업데이트를 생성하는 명령입니다. 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 업데이트의 목록을 가져옵니다.

```
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 OTA 업데이트의 생성 또는 삭제 상태를 가져오는 명령을 사용합니다.

```
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": "AvP3bfJC9gyqnwoxPHuTqM5GWENT4iiii"  
                    }  
                },  
                "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 단원을 참조하십시오.[작업 실행 설명](#).

## OTA 관련 데이터 삭제

지금은 AWS IoT 콘솔을 사용하여 스트림 또는 OTA 업데이트를 삭제할 수 없습니다. AWS CLI를 사용하여 스트림, OTA 업데이트, OTA 업데이트 중에 생성되는 AWS IoT 작업을 삭제할 수 있습니다.

### OTA 스트림 삭제

MQTT를 사용하는 OTA 업데이트를 생성할 때 명령줄이나 AWS IoT 콘솔을 사용하여 펌웨어를 청크로 분할하여 MQTT를 통해 전송할 수 있도록 스트림을 생성할 수 있습니다. AWS CLI를 사용하여 이 스트림을 삭제할 수 있습니다.`delete-stream` AWS CLI 다음 예와 같이 명령을 입력합니다.

```
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 수행할 AWS IoT 작업
- 업데이트 중인 각 디바이스에 대해 AWS IoT 작업 실행

`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

FreeRTOS 생성 AWS IoTTOTA 업데이트를 생성할 때 작업을 수행합니다. 또한 작업을 처리하는 각 디바이스에 대해 작업 실행이 생성됩니다. `aws iot delete-job` AWS CLI 작업 및 관련 작업 실행을 삭제할 수 있는 명령을 선택합니다.

```
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.

`aws iot delete-job-execution`을 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
```

와 마찬가지로 `aws iot delete-job` AWS CLI 명령을 전달할 수 있습니다. `--force`에 대한 매개 변수 `aws iot delete-job-execution` 작업 실행을 강제로 삭제합니다. 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 \(p. 271\)](#) 단원을 참조하십시오.

## OTA 업데이트 관리자 서비스

이 over-the-air (OTA) 업데이트 관리자 서비스를 사용하여 다음을 수행할 수 있습니다.

- AWS IoT 작업, AWS IoT 스트림 및 코드 서명을 포함하여 OTA 업데이트와 해당 업데이트가 사용하는 리소스를 생성합니다.
- OTA 업데이트에 대한 정보를 가져옵니다.
- 와 연결된 모든 OTA 업데이트 나열 AWS 계정.
- 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 스냅샷입니다. 단원을 참조하십시오.[작업의 단원AWS IoT개발자 안내서](#)필요한 업데이트 유형에 대한 논의입니다.
- OTA 업데이트를 수행하는 데 사용되는 프로토콜은 다음과 같습니다. [MQTT], [HTTP] 또는 [MQTT, HTTP]입니다. MQTT 및 HTTP를 지정하면 디바이스 설정에 따라 사용되는 프로토콜이 결정됩니다.
- 대상 디바이스로 전송할 파일 목록
- 부여하는 IAM 역할AWS IoTAmazon S3에 대한 액세스 권한AWS IoTjob과 AWSOTA 업데이트 작업을 생성하는 코드 서명 리소스입니다.
- 사용자 정의 이름-값 페어 목록(선택 사항)

OTA 업데이트는 디바이스 펌웨어를 업데이트하도록 설계되었지만, OTA 업데이트를 사용하여 AWS IoT에 등록된 하나 이상의 디바이스에 원하는 파일을 전송할 수 있습니다. 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. 다음과 같이 펌웨어 업데이트에 서명할 수 있습니다.[FreeRTOS 코드 서명](#) 또는 자체 코드 서명 도구를 사용할 수 있습니다.

프로세스를 더 자세히 제어하려면 [CreateStream API](#)를 사용하여 MQTT를 통해 업데이트를 전송할 때 스트림을 생성할 수 있습니다. 경우에 따라 FreeRTOS 에이전트를 수정할 수 있습니다.[암호](#)를 눌러 보내고 받는 블록의 크기를 조정합니다.

OTA 업데이트를 생성하면 OTA 관리자 서비스에서 디바이스에 업데이트가 사용 가능함을 알려주는 [AWS IoT 작업](#)을 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 않을 경우 자체 OTA 에이전트를 구현하여 업데이트를 수신 및 다운로드하고 설치 작업을 수행해야 합니다.

## 애플리케이션에 OTA 에이전트 통합

이 over-the-air (OTA) 에이전트는 OTA 업데이트 기능을 제품에 추가하기 위해 작성해야 하는 코드 양을 간소화하도록 설계되었습니다. 통합 작업에서는 기본적으로 OTA 에이전트를 초기화하고 OTA 에이전트 이벤트 메시지에 응답하기 위한 사용자 지정 콜백 함수를 생성합니다. OS 초기화 중에 MQTT, HTTP (파일 다운로드에 HTTP가 사용되는 경우) 및 PAL (플랫폼별 구현) 인터페이스가 OTA 에이전트로 전달됩니다. 버퍼를 초기화하여 OTA 에이전트로 전달할 수도 있습니다.

### Note

OTA 업데이트 기능을 애플리케이션에 통합하는 작업은 다소 간단하지만, OTA 업데이트 시스템에서 디바이스 코드 통합보다 더 많은 것을 이해해야 합니다. 이를 구성하는 방법을 알아보려면 AWS 계정 AWS IoT 사물, 자격 증명, 코드 서명 인증서, 프로비저닝 디바이스 및 OTA 업데이트 작업 [FreeRTOS 사전 조건](#).

## 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.

## 단순 OTA 데모

다음은 에이전트가 MQTT 브로커에 연결하고 OTA 에이전트를 초기화하는 방법을 보여주는 간단한 OTA 데모에서 발췌한 내용입니다. 이 예에서는 기본 OTA 애플리케이션 콜백을 사용하고 일부 통계를 초당 하나씩 반환하도록 데모를 구성합니다. 간결하게 나타내기 위해 이 데모에서는 일부 세부 정보를 생략합니다.

OTA 데모에서는 연결 끊기 콜백을 모니터링하고 연결을 다시 설정하여 MQTT 연결 관리를 보여 줍니다. 연결이 끊어지면 데모에서 먼저 OTA 에이전트 작업을 일시 중단한 다음 MQTT 연결을 다시 설정하려고 시도 합니다. MQTT 재연결 시도는 최대 값까지 기하 급수적으로 증가하고 지터도 추가됩니다. 연결이 다시 설정되면 OTA 에이전트가 작업을 계속합니다.

AWS IoT MQTT 브로커를 사용하는 작업의 예는 `demos/ota` 디렉터리의 OTA 데모 코드를 참조하십시오.

OTA 에이전트는 자체 작업이므로 이 예의 의도적인 1초 지연은 이 애플리케이션에만 영향을 줍니다. 따라서 에이전트의 성능에는 영향을 주지 않습니다.

```
static BaseType_t prvRunOTADemo( void )
{
    /* Status indicating a successful demo or not. */
    BaseType_t xStatus = pdFAIL;

    /* OTA library return status. */
    OtaErr_t xOtaError = OtaErrUninitialized;

    /* OTA event message used for sending event to OTA Agent.*/
    OtaEventMsg_t xEventMsg = { 0 };

    /* OTA interface context required for library interface functions.*/
    OtaInterfaces_t xOtaInterfaces;

    /* OTA library packet statistics per job.*/
    OtaAgentStatistics_t xOtaStatistics = { 0 };

    /* OTA Agent state returned from calling OTA_GetState.*/
    OtaState_t xOtaState = OtaAgentStateStopped;

    /* Set OTA Library interfaces.*/
    prvSetOtaInterfaces( &xOtaInterfaces );

    /***** Init OTA Library. *****/
    if( ( xOtaError = OTA_Init( &xOtaBuffer,
                                &xOtaInterfaces,
                                ( const uint8_t * ) ( democonfigCLIENT_IDENTIFIER ),
                                prvOtaAppCallback ) ) != OtaErrNone )
    {
        LogError( ( "Failed to initialize OTA Agent, exiting = %u.",
                    xOtaError ) );
    }
    else
    {
        xStatus = pdPASS;
    }

    /***** Create OTA Agent Task. *****/
    if( xStatus == pdPASS )
    {
        xStatus = xTaskCreate( prvOTAAgentTask,
                              "OTA Agent Task",
                              otaexampleAGENT_TASK_STACK_SIZE,
                              NULL,
                              otaexampleAGENT_TASK_PRIORITY,
```

```

        NULL );

    if( xStatus != pdPASS )
    {
        LogError( ( "Failed to create OTA agent task:" ) );
    }
}

/***** Start OTA *****/
if( xStatus == pdPASS )
{
    /* Send start event to OTA Agent.*/
    xEventMsg.eventId = OtaAgentEventStart;
    OTA_SignalEvent( &xEventMsg );
}

/***** Loop and display OTA statistics *****/
if( xStatus == pdPASS )
{
    while( ( xOtaState = OTA_GetState() ) != OtaAgentStateStopped )
    {
        /* Get OTA statistics for currently executing job. */
        if( xOtaState != OtaAgentStateSuspended )
        {
            OTA_GetStatistics( &xOtaStatistics );

            LogInfo( ( " Received: %u    Queued: %u    Processed: %u    Dropped: %u",
                       xOtaStatistics.otaPacketsReceived,
                       xOtaStatistics.otaPacketsQueued,
                       xOtaStatistics.otaPacketsProcessed,
                       xOtaStatistics.otaPacketsDropped ) );
        }

        vTaskDelay( pdMS_TO_TICKS( otaexampleEXAMPLE_TASK_DELAY_MS ) );
    }
}

return xStatus;
}

```

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.
- 지수 지연과 지터로 다시 연결해 보십시오.
- 다시 연결되면 OTA 에이전트 작업을 재개합니다.
- 에이전트가 중지하면 1초 지연된 다음 연결을 다시 시도합니다.

## OTA 에이전트 이벤트에 응용 프로그램 콜백 사용

이전 예제가 사용됨prvOtaAppCallbackOTA 에이전트 이벤트에 대한 콜백 핸들러입니다. (네 번째 매개 변수 참조)OTA\_InitAPI 호출). 완료 이벤트에 대한 사용자 지정 처리를 구현하려면 OTA 데모/애플리케이션에서 기본 처리를 변경해야 합니다. OTA 프로세스 중에 OTA 에이전트는 다음 이벤트 열거형 중 하나를 콜

백 핸들러에 전송할 수 있습니다. 이벤트를 처리하는 방법과 시기는 애플리케이션 개발자가 결정해야 합니다.

```
/** @ingroup ota_enum_types
 * @brief OTA Job callback events.
 *
 * After an OTA update image is received and authenticated, the agent calls the user
 * callback (set with the @ref OTA_Init API) with the value OtaJobEventActivate to
 * signal that the device must be rebooted to activate the new image. When the device
 * boots, if the OTA job status is in self test mode, the agent calls the user callback
 * with the value OtaJobEventStartTest, signaling that any additional self tests
 * should be performed.
 *
 * If the OTA receive fails for any reason, the agent calls the user callback with
 * the value OtaJobEventFail instead to allow the user to log the failure and take
 * any action deemed appropriate by the user code.
 *
 * See the OtaImageState_t type for more information.
 */
typedef enum OtaJobEvent
{
    OtaJobEventActivate = 0,           /*!< @brief OTA receive is authenticated and ready to
                                     activate. */
    OtaJobEventFail = 1,              /*!< @brief OTA receive failed. Unable to use this
                                     update. */
    OtaJobEventStartTest = 2,          /*!< @brief OTA job is now in self test, perform user
                                     tests. */
    OtaJobEventProcessed = 3,          /*!< @brief OTA event queued by OTA_SignalEvent is
                                     processed. */
    OtaJobEventSelfTestFailed = 4,      /*!< @brief OTA self-test failed for current job. */
    OtaJobEventParseCustomJob = 5,      /*!< @brief OTA event for parsing custom job document.
                                     */
    OtaJobEventReceivedJob = 6,          /*!< @brief OTA event when a new valid AFT-OTA job is
                                     received. */
    OtaJobEventUpdateComplete = 7,      /*!< @brief OTA event when the update is completed. */
    OtaLastJobEvent = OtaJobEventStartTest
} OtaJobEvent_t;
```

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.

#### **OtaJobEventActivate**

콜백 핸들러가 이 이벤트를 수신하면 즉시 디바이스를 재설정하거나 나중에 디바이스를 재설정하기 위한 호출을 예약할 수 있습니다. 따라서 필요한 경우 디바이스 재설정 및 자체 테스트 단계를 연기할 수 있습니다.

#### **OtaJobEventFail**

콜백 핸들러가 이 이벤트를 수신하면 업데이트가 실패한 것입니다. 이 경우 아무것도 수행할 필요가 없습니다. 로그 메시지를 출력하거나 애플리케이션 관련 작업을 수행할 수 있습니다.

#### **OtaJobEventStartTest**

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 OtaJobEventStartTest 이벤트를 콜백 함수에 전송합니다. 개발자는 업데이트 후 디바이스 펌웨어가 올바르게 작동하는지 확인하는 데 필요한 테스트를 추가할 수 있습니다. 자체 테스트에서 디바이스 펌웨어가 안정적인 것으로 간주

될 경우 코드는 `OTA_SetImageState( OtaImageStateAccepted )` 함수를 호출하여 펌웨어를 새 영구 이미지로 커밋해야 합니다.

**OtaJobEventProcessed**

대기 중인 OTA 이벤트 `OTA_SignalEvent`가 처리되므로 OTA 버퍼를 해제하는 것과 같은 정리 작업을 수행 할 수 있습니다.

**OtaJobEventSelfTestFailed**

현재 작업에 대해 OTA 자체 테스트가 실패했습니다. 이 이벤트의 기본 처리는 OTA 에이전트를 종료하고 다시 시작하여 디바이스가 이전 이미지로 롤백되도록 하는 것입니다.

**OtaJobEventUpdateComplete**

OTA 작업 업데이트 완료에 대한 알림 이벤트입니다.

## OTA 보안

다음은 세 가지 측면입니다. over-the-air (OTA) 보안:

연결 보안

OTA 업데이트 관리자 서비스는 AWS IoT에서 사용되는 전송 계층 보안(TLS) 상호 인증과 같은 기준 보안 메커니즘을 사용합니다. OTA 업데이트 트래픽은 AWS IoT 디바이스 게이트웨이를 통해 전달되며 AWS IoT 보안 메커니즘을 사용합니다. 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 업데이트 관리자 서비스는 용 코드 서명을 사용합니다. AWS IoT 펌웨어에 자동으로 서명합니다. 자세한 내용은 [AWS IoT용 코드 서명](#)을 참조하십시오.

디바이스에서 실행되는 OTA 에이전트는 펌웨어가 디바이스에 수신되면 무결성 검사를 수행합니다.

운영자 보안

제어 플레인 API를 통해 생성되는 모든 API 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 ACL에 읽기 권한을 부여해야 합니다. AWS IoT 스트리밍 중에 Amazon S3 저장된 펌웨어 업데이트에 액세스할 수 있도록 서비스 보안 주체입니다.

## Code Signing for AWS IoT

AWS IoT 콘솔은 [AWS IoT용 코드 서명](#)을 사용하여 AWS IoT에서 지원되는 디바이스에 대한 펌웨어 이미지에 자동으로 서명합니다.

Code Signing for AWS IoT는 ACM으로 가져오는 인증서와 프라이빗 키를 사용합니다. 자체 서명된 인증서를 테스트에 사용할 수 있지만 잘 알려진 상용 CA (인증 기관)에서 인증서를 얻는 것이 좋습니다.

코드 - 서명 인증서에는 X.509 버전 3이 사용됩니다. `Key Usage`과 `Extended Key Usage` 확장. `Key Usage` 확장은 `Digital Signature`로 설정되어 있고, `Extended Key Usage` 확장은 `Code Signing`으로 설정되어 있습니다. 코드 이미지에 서명하는 방법에 대한 자세한 내용은 [AWS IoT용 코드 서명 개발자 안내서](#) 및 [AWS IoT용 코드 서명 API 참조](#)를 참조하십시오.

Note

용 코드 서명을 다운로드할 수 있습니다.[AWS IoT SDK 출처Amazon Web Services](#)용 도구.

## OTA 문제 해결

다음 단원에서는 OTA 업데이트 관련 문제를 해결하는 데 유용한 정보를 제공합니다.

주제

- 설정 CloudWatch OTA 업데이트에 대한 로그 (p. 186)
- AWS CloudTrail를 사용하여 AWS IoT OTA API 호출 로깅 (p. 189)
- 를 사용하여 CreateOTAUpdate 실패 세부 정보를 가져옵니다.AWS CLI (p. 191)
- AWS CLI를 사용하여 OTA 오류 코드 가져오기 (p. 193)
- 여러 디바이스의 OTA 업데이트 문제 해결 (p. 194)
- Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결 (p. 194)

## 설정 CloudWatch OTA 업데이트에 대한 로그

OTA 업데이트 서비스에서는 Amazon CloudWatch로 로깅을 지원합니다. 이AWS IoTAmazon 활성화 및 구성을 위한 콘솔 CloudWatch OTA 업데이트에 대한 로깅 자세한 내용은 [Cloudwatch Logs](#)를 참조하십시오.

로깅을 활성화하려면 IAM 역할을 생성하고 OTA 업데이트 로깅을 구성해야 합니다.

Note

OTA 업데이트 로깅을 활성화하기 전에 CloudWatch 액세스 권한을 기록합니다. 액세스 권한이 있는 사용자 CloudWatch 로그는 디버깅 정보를 볼 수 있습니다. 자세한 정보는 단원을 참조하십시오.[Amazon에 대한 인증 및 액세스 제어 CloudWatch 로그](#).

## 로깅 역할 생성 및 로깅 활성화

[AWS IoT 콘솔](#)을 사용하여 로깅 역할을 생성하고 로깅을 활성화합니다.

- 탐색 창에서 설정을 선택합니다.
- 로그에서 편집을 선택합니다.
- 세부 사항 수준에서 디버깅을 선택합니다.
- Under역할 설정, 선택새로 생성을 사용하여 로깅용 IAM 역할을 생성합니다.
- 이름 아래에 역할의 고유한 이름을 입력합니다. 모든 필요한 권한을 가진 역할이 생성됩니다.
- 업데이트(Update)를 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 생성된 경우
- AWS IoT 작업이 완료된 경우
- 스트림이 생성된 경우

에서 로그를 볼 수 있습니다.[CloudWatch 콘솔](#).

에서 OTA 업데이트를 보려면 CloudWatch 로그

1. 탐색 창에서 로그를 선택합니다.
2. 로그 그룹에서 AWSIoTLogsV2를 선택합니다.

OTA 업데이트 로그는 다음 속성을 포함할 수 있습니다.

accountId

이 AWS로그가 작성된 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 전송되었습니다.AWS IoTMQTT 기반 파일 전송 기능.
- DescribeStream: 스트림에 대한 정보 요청이 으로 전송되었습니다.AWS IoTMQTT 기반 파일 전송 기능.

awsJobId

로그를 생성한 AWS IoT 작업 ID입니다.

clientId

로그를 생성한 요청을 보낸 MQTT 클라이언트 ID입니다.

clientToken

로그를 생성한 요청과 연결된 클라이언트 토큰입니다.

세부 정보

로그를 생성한 작업에 대한 추가 정보입니다.

logLevel

로그의 로깅 수준입니다. OTA 업데이트 로그의 경우 이 값은 항상 DEBUG로 설정됩니다.

otaUpdateId

로그를 생성한 OTA 업데이트의 ID입니다.

protocol

로그를 생성한 요청을 만드는 데 사용된 프로토콜입니다.

상태

로그를 생성한 작업의 상태입니다. 유효한 값은 다음과 같습니다.

- 성공
- 실패

streamId

로그를 생성한 AWS IoT 스트림 ID입니다.

timestamp

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 작업을 만들 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."
}
```

## AWS CloudTrail를 사용하여 AWS IoT OTA API 호출 로깅

FreeRTOS는 캡처하는 서비스인 CloudTrail과 통합됩니다. AWS IoT OTA API는 사용자가 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. CloudTrail 코드에서 로 전달되는 API 호출을 캡처합니다. AWS IoT OTA API, CloudTrail에서 수집한 정보를 사용하여 어떤 요청이 이루어졌는지 확인할 수 있습니다. AWS IoT OTA, 어떤 소스 IP 주소에서 요청했는지, 누가 언제 요청했는지 등을 알 수 있습니다.

이 서비스를 구성하고 사용하는 방법을 포함한 CloudTrail에 대한 자세한 내용은 [AWS CloudTrail 사용 설명서](#)를 참조하세요.

## CloudTrail의 FreeRTOS 정보

일시 CloudTrail에서 로깅이 활성화되었습니다. AWS 계정, API 호출 AWS IoT OTA 액션은 다음에서 추적됩니다. CloudTrail 로그 파일이 다른 파일과 함께 작성된 로그 파일 AWS 서비스 레코드. CloudTrail는 기간 및 파일 크기를 기준으로 새 파일을 만들고 기록하는 시점을 결정합니다.

다음 AWS IoT CloudTrail에서 OTA 컨트롤 플레이인 작업은에서 로깅합니다.

- [CreateStream](#)
- [DescribeStream](#)
- [ListStreams](#)
- [UpdateStream](#)
- [DeleteStream](#)
- [CreateOTAUpdate](#)
- [GetOTAUpdate](#)
- [ListOTAUpdates](#)
- [DeleteOTAUpdate](#)

### Note

AWS IoT OTA 데이터 영역 작업 (디바이스 측)은 CloudTrail에서 로그하지 않습니다. 사용 CloudWatch 이를 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 자격 증명으로 했는지 여부.
- 역할 또는 연합된 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 단원을 참조하십시오. [CloudTrail userIdentity 요소](#). AWS IoT OTA 작업은 다음에 나와 있습니다. [AWS IoT OTA API 참조](#).

원하는 기간만큼 Amazon S3 버킷에 로그 파일을 저장할 수 있습니다. 그러나 Amazon S3 수명 주기 규칙을 정의하여 자동으로 로그 파일을 아카이브하거나 삭제할 수도 있습니다. 기본적으로 로그 파일은 Amazon S3 서버 측 암호화 (SSE)를 통해 암호화됩니다.

로그 파일이 전송될 때 알림을 받으려면을 구성할 수 있습니다. CloudTrail Amazon SNS 알림을 게시할 수 있습니다. 자세한 내용은 단원을 참조하십시오. [CloudTrail에 대한 Amazon SNS 알림 구성](#).

집계할 수도 있습니다. AWS IoT 여러 개의 OTA 로그 파일 AWS 여러 개의 리전 AWS 단일 Amazon S3 버킷에 계정을 생성합니다.

자세한 내용은 단원을 참조하십시오. [수신 CloudTrail 여러 리전에서 로그 파일과 수신 CloudTrail 여러 계정의 로그 파일](#).

## 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 호출의 주문 스택 트레이스가 아니기 때문에 특정 순서로 표시되지 않습니다.

다음 예제는 CloudTrail에 대한 호출의 로그를 보여 주는 로그 항목 CreateOTAUpdateaction.

```
{  
    "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 정보를 가져 옵니다.AWS CLI

OTA 업데이트 작업을 만드는 프로세스가 실패하면 문제를 해결하기 위해 취할 수 있는 조치가 있을 수 있  
습니다. OTA 업데이트 작업을 생성하면 OTA 관리자 서비스가 IoT 작업을 생성하고 대상 장치에 맞게 예약

하며 이 프로세스는 다른 유형의 작업을 생성하거나 사용합니다.AWS 계정의 리소스 (코드 서명 작업,AWS IoT Amazon S3 객체)입니다. 오류가 발생하면 프로세스를 만들지 않고 실패할 수 있습니다.AWS IoT 작업. 이 문제 해결 섹션에서는 오류의 세부 정보를 검색하는 방법에 대한 지침을 제공합니다.

1. [AWS CLI](#)를 설치하고 구성합니다.
2. 실행 aws configure 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 단원을 참조하십시오.[를 통한 빠른 구성aws configure.](#)

3. 실행합니다.

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

위치##\_####\_##\_001를 생성할 때 OTA 업데이트에 부여한 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": {}
                }
            }
        ],
        "codeSigning": {
            "signingProfileParameter": {},
            "signingProfileName": "signing_profile_name",
            "destination": {
                "s3Destination": {
                    "bucket": "bucket_name",
                    "prefix": "SignedImages/"
                }
            }
        }
    }
}
```

```

        "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)"
        }
    }
}

```

생성에 실패하면 otaUpdateStatus 명령 출력의 필드에는 다음이 포함됩니다.  
CREATE\_FAILED 및 errorInfo 필드에는 실패에 대한 세부 정보가 포함됩니다.

## AWS CLI를 사용하여 OTA 오류 코드 가져오기

1. AWS CLI를 설치하고 구성합니다.
2. 실행 aws config 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 단원을 참조하십시오. [통한 빠른 구성aws configure.](#)

3. 실행합니다.

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

위치 **JobID** 상태를 가져오려는 작업에 대한 전체 작업 ID 문자열입니다 (OTA 업데이트 작업이 생성되었을 때). **ThingName**입니다 AWS IoT 디바이스가 다음과 같이 등록되어 있는 사물 이름 AWS IoT

4. 출력값은 다음과 같습니다.

```

{
    "execution": {
        "jobId": "AFR_OTA-*****",
        "status": "FAILED",
        "statusDetails": {
            "detailsMap": {
                "reason": "0xFFFFFFFF: 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. "0xFFFFFFFF"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.html](https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws_ota_agent_8h.html)에 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

동일한 펌웨어 이미지를 사용해 여러 디바이스(사물)에서 OTA를 수행하려면 비휘발성 메모리에서 `clientcredentialIOT_THING_NAME`을 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 헤더를 사용한다면 디바이스 인증서의 일반 이름에서 사물 이름을 읽을 수 있습니다.

## Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결

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 `restore-to-factory` 출고 시 이미지로 되돌릴 수 있는 프로세스를 선택합니다. `network_if.c`에서 비동기 이벤트 핸들러를 업데이트하여 원하는 동작을 프로그래밍해야 합니다.

# FreeRTOS

FreeRTOS 라이브러리는 FreeRTOS 커널 및 내부 라이브러리에 추가 기능을 제공합니다. FreeRTOS 라이브러리를 사용하여 Embedded 애플리케이션의 네트워킹 및 보안을 위해 사용할 수 있습니다. FreeRTOS 라이브러리를 사용하면 애플리케이션과 상호 작용할 수도 있습니다. AWS IoT 서비스. FreeRTOS에는 다음을 위한 라이브러리가 포함되어 있습니다.

- MQTT 및 디바이스 샐도우를 사용하여 AWS IoT 클라우드에 디바이스를 안전하게 연결합니다.
- AWS IoT Greengrass 코어를 검색하고 연결합니다.
- Wi-Fi 연결을 관리합니다.
- [FreeRTOS 무선 업데이트 \(p. 140\)](#)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

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 단원을 참조하십시오. [FreeRTOS 이식](#).

### FreeRTOS 이식 라이브러리

라이브러리	API 참조	설명
Bluetooth Low Energy	<a href="#">Bluetooth Low Energy API 참조</a>	FreeRTOS 블루투스 저에너지 라이브러리를 사용하여 마이크로컨트롤러는 AWS IoT 게이트웨이 디바이스를 통한 MQTT 브로커 자세한 정보는 <a href="#">Bluetooth Low Energy 라이브러리 (p. 198)</a> 을 참조하십시오.
OTA(Over-the-Air) 업데이트	<a href="#">AWS IoTOTA (무선) 업데이트 API 참조</a>	FreeRTOS AWS IoTOTA (Over-the-Air) 업데이트 라이브러리를 사용하면 FreeRTOS 장치에서 업데이트 알림을 관리하고 업데이트를 다운로드하고 펌웨어 업데이트의 암호화 확인을 수행할 수 있습니다. 자세한 정보는 <a href="#">AWS IoT 무선 (OTA) 라이브러리 (p. 216)</a> 을 참조하십시오.
FreeRTOS+POSIX	<a href="#">FreeRTOS+POSIX API 참조</a>	FreeRTOS+POSIX 라이브러리를 사용하여 POSIX 호환 애플리케이션을 FreeRTOS 에코 시스템에 이식할 수 있습니다. 자세한 내용은 <a href="#">FreeRTOS+POSIX</a> 를 참조하십시오.

라이브러리	API 참조	설명
보안 소켓	<a href="#">Secure Sockets API 참조</a>	자세한 정보는 <a href="#">보안 소켓 라이브러리 (p. 223)</a> 을 참조하십시오.
FreeRTOS+TCP	<a href="#">FreeRTOS+TCP API 참조</a>	FreeRTOS+TCP는 FreeRTOS를 위한 확장 가능한 오픈 소스 및 스레드 세이프 TCP/IP 스택입니다.  자세한 내용은 <a href="#">FreeRTOS+TCP</a> 을 참조하십시오.
Wi-Fi	<a href="#">Wi-Fi API 참조</a>	FreeRTOS Wi-Fi 라이브러리를 사용하면 마이크로컨트롤러의 하위 수준 무선 스택과 연결할 수 있습니다.  자세한 내용은 <a href="#">Wi-Fi 라이브러리 (p. 228)</a> 섹션을 참조하세요.
코렐크CS11		CorePKCS11 라이브러리는 프로비저닝 및 TLS 클라이언트 인증을 지원하기 위한 공개 키 암호화 표준 #11 의 참조 구현입니다.  자세한 내용은 <a href="#">코어프KCS11 라이브러리 (p. 219)</a> 섹션을 참조하세요.
TLS		자세한 정보는 <a href="#">전송 계층 보안 (p. 227)</a> 을 참조하십시오.
공통 I/O	공통 I/O API 참조	자세한 정보는 <a href="#">공통 I/O (p. 210)</a> 을 참조하십시오.
셀룰러 인터페이스	셀룰러 인터페이스 API 참조	셀룰러 인터페이스 라이브러리는 통일된 API를 통해 널리 사용되는 몇 가지 셀룰러 모뎀의 기능을 제공합니다. 자세한 내용은 <a href="#">셀룰러 인터페이스 라이브러리 (p. 208)</a> 섹션을 참조하세요.

## FreeRTOS 애플리케이션 라이브러리

필요에 따라 FreeRTOS 구성에 다음 독립 실행형 애플리케이션 라이브러리를 포함 시킬 수 있습니다.AWS IoT 클라우드에서의 서비스.

### Note

일부 애플리케이션 라이브러리는 의 라이브러리와 동일한 API를 사용합니다.AWS IoTDevice SDK for Embedded C [AWS IoT디바이스 SDK C API 레퍼런스](#). 에 대한 자세한 내용AWS IoTDevice SDK for Embedded C [AWS IoT Device SDK for Embedded C \(p. 13\)](#).

### FreeRTOS 애플리케이션 라이브러리

라이브러리	API 참조	설명
AWS IoT Device Defender	<a href="#">디바이스 Defender C SDK API 참조</a>	FreeRTOSAWS IoT Device DefenderFreeRTOS 디바이스를 에 연결합니다.AWS IoT Device Defender.

라이브러리	API 참조	설명
		자세한 정보는 <a href="#">AWS IoT Device Defender 라이브러리 (p. 211)</a> 을 참조하십시오.
AWS IoT Greengrass	<a href="#">Greengrass API 참조</a>	FreeRTOS AWS IoT Greengrass FreeRTOS 디바이스를 연결합니다. AWS IoT Greengrass. 자세한 정보는 <a href="#">AWS IoT Greengrass Discovery 라이브러리 (p. 211)</a> 을 참조하십시오.
MQTT	<a href="#">MQTT (v1.x.x) 라이브러리 API 참조</a> <a href="#">MQTT(v1) 에이전트 API 참조</a> <a href="#">MQTT (v2.x.x) C SDK API 참조</a>	CoremQTT 라이브러리는 FreeRTOS 디바이스가 MQTT 주제를 게시 및 구독 할 수 있도록 클라이언트를 제공합니다. MQTT는 디바이스가 AWS IoT와 상호 작용하기 위해 사용하는 프로토콜입니다. CoremQTT 라이브러리 버전 3.0.0에 대한 자세한 내용은 단원을 참조하십시오. <a href="#">CoremQTT 라이브러리 (p. 214)</a> .
CoremQTT	<a href="#">코렘QTT 에이전트 라이브러리 API 레퍼런스</a>	CoreMqtt 에이전트 라이브러리는 CoreMqtt 라이브러리에 스레드 안전성을 추가하는 고급 API입니다. 백그라운드에서 MQTT 연결을 관리하고 다른 작업의 개입이 필요하지 않은 전용 MQTT 에이전트 작업을 만들 수 있습니다. 라이브러리는 CoreMqtt의 API와 동등한 스레드 안전 기능을 제공하므로 다중 스레드 환경에서 사용할 수 있습니다. CoremQTT 에이전트 라이브러리에 대한 자세한 내용은 단원을 참조하십시오. <a href="#">CoremQTT 에이전트 라이브러리 (p. 215)</a> .
AWS IoT 디바이스 새 도우	<a href="#">디바이스 새도우 C SDK API 참조</a>	이 AWS IoT FreeRTOS 디바이스가 와상호 작용할 수 있도록 지원하는 Device Shadow AWS IoT 장치 그림자. 자세한 정보는 <a href="#">AWS IoT 디바이스 새도우 라이브러리 (p. 226)</a> 을 참조하십시오.

## FreeRTOS 라이브러리 구성

FreeRTOS 및 AWS IoT Device SDK는 C 전처리기 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 다음을 사용합니다. #defineFreeRTOS 라이브러리, 데모 및 테스트를 구성하는 지시문입니다.

지원되는 전역 구성 옵션에 대한 자세한 내용은 [전역 구성 파일 참조](#)를 참조하십시오.

## 백오프 알고리즘

### 소개

이 [백오프 알고리즘](#) library는 네트워크 혼잡을 피하기 위해 동일한 데이터 블록의 반복 재전송을 공간화하는 데 사용되는 유ти리티 라이브러리입니다. 이 라이브러리는 다음을 사용하여 네트워크 작업 (예: 서버와의 네트워크 연결 실패)을 재시도하는 백오프 기간을 계산합니다. [지터를 사용한 지수 백오프 알고리즘](#)

지터가 있는 지수 백오프는 일반적으로 네트워크 혼잡 또는 서버의 높은 부하로 인해 발생하는 서버에 실패 한 연결 또는 네트워크 요청을 다시 시도할 때 사용됩니다. 네트워크 연결을 동시에 시도하는 여러 장치에서 만든 재시도 요청의 타이밍을 분산시키는데 사용됩니다. 연결이 좋지 않은 환경에서는 클라이언트는 언제든지 연결이 끊어질 수 있습니다. 따라서 백오프 전략은 성공할 가능성이 없을 때 반복적으로 다시 연결을 시도 하지 않아서 배터리를 절약하는 데 도움이 됩니다.

라이브러리는 C로 작성되었으며 다음을 준수하도록 설계되었습니다. [이소 C90과 미스라 C:2012](#). 라이브러리는 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없으며 힙 할당이 없으므로 IoT 마이크로 컨트롤러에 적합하지만 다른 플랫폼에도 완벽하게 이식할 수 있습니다.

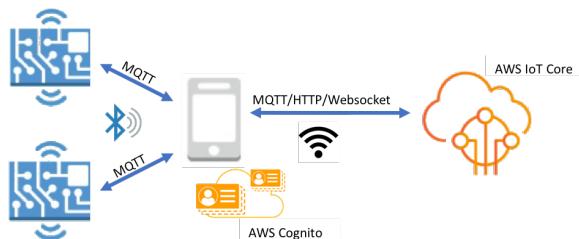
이 라이브러리는 자유롭게 사용할 수 있으며 [MIT 오픈 소스 라이선스](#).

코드 백오프 알고리즘의 크기 (ARM Cortex-M용 GCC로 생성된 예제)		
백오프_알고리즘.c	0.1K	0.1K
추정 총	0.1K	0.1K

## Bluetooth Low Energy 라이브러리

### 개요

FreeRTOS는 프록시 디바이스 (예: 휴대폰)를 통해 Bluetooth Low Energy를 이용한 MQTT 주제 게시 및 구독을 지원합니다. FreeRTOS [Bluetooth Low Energy](#) 라이브러리는 마이크로 컨트롤러와 안전하게 통신할 수 있습니다. AWS IoT MQTT 브로커.



Bluetooth 디바이스용 Mobile SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로 컨트롤러의 내장형 애플리케이션과 통신하는 기본 모바일 애플리케이션을 작성할 수 있습니다. 모바일 SDK에 대한 자세한 내용은 [FreeRTOS Bluetooth 디바이스용 Mob \(p. 207\)](#)을 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.

## Architecture

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) 서비스로 구성됩니다. 디바이스 정보, Wi-Fi 프로비저닝, 네트워크 추상화 및 대용량 객체 전송이 가능합니다.

### 디바이스 정보

디바이스 정보 서비스는 마이크로컨트롤러에 대한 다음 정보를 수집합니다.

- 디바이스가 사용 중인 FreeRTOS 버전
- 디바이스가 등록된 계정의 AWS IoT 엔드포인트
- 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

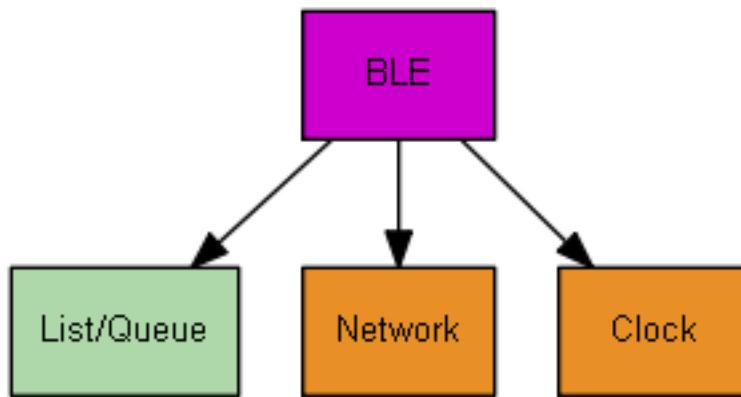
하위 수준 FreeRTOS Bluetooth Low Energy 래퍼는 제조업체의 Bluetooth Low Energy 스택의 추상화입니다. 하위 수준 래퍼는 하드웨어를 통한 직접 제어를 위해 공통 API 세트를 제공합니다. 하위 수준 API는 RAM 사용량을 최적화하지만 기능이 제한됩니다.

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.

- [선행 컨테이너](#)도서관
- 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	<a href="#">Wi-Fi 라이브러리 (p. 228)</a>

AWS IoT MQTT 브로커와 통신하려면 AWS 계정이 있어야 하고 디바이스를 AWS IoT 사물로 등록해야 합니다. 설정에 대한 자세한 내용은 [AWS IoT 개발자 안내서](#)를 참조하십시오.

FreeRTOS Bluetooth Low Energy는 Amazon Cognito를 사용하여 모바일 디바이스의 사용자 인증을 MQTT 프록시 서비스를 사용하려면 Amazon Cognito 자격 증명과 사용자 풀을 만들어야 합니다. 각 Amazon Cognito 자격 증명에는 적절한 정책이 연결되어 있어야 합니다. 자세한 정보는 [Amazon Cognito 개발자 안내서](#)를 참조하세요.

## 라이브러리 구성 파일

Bluetooth Low Energy 서비스를 이용한 FreeRTOS MQTT 서비스를 사용하는 애플리케이션은 `iot_ble_config.h` 구성 파라미터가 정의된 헤더 파일입니다. 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 참조](#)를 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 블루투스 저에너지 라이브러리는 eBT`propertySecureConnectionOnly` 속성을 TRUE로 설정하여 디바이스를 보안 연결만 허용 모드에 둡니다. [Bluetooth 핵심 규격 v5.0, Vol 3, Part C, 10.2.4](#)에 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, 10.2.1](#)을 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()`을 사용하여 미들웨어를 초기화합니다.

#### Note

를 실행하는 경우에는 이 초기화 단계가 필요하지 않습니다. AWS데모. 데모 초기화는 `freertos/demos/network_manager`에 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 참조](#)를 참조하십시오.

## 사용 예

아래의 예는 새로운 서비스를 광고하고 생성하기 위해 Bluetooth Low Energy 라이브러리를 사용하는 방법을 보여줍니다. FreeRTOS Bluetooth Low Energy 데모 애플리케이션은 단원을 [Bluetooth Low Energy 데모 애플리케이션](#).

### 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\_CUSTOMADVERTISEMENT\_MSG를 1로 설정합니다. 이렇게 하면 IoTBle\_SetCustomAdvCb 콜백이 트리거됩니다.

## 새 서비스 추가

서비스의 전체 예제는 [freertos/.../ble/services](#)를 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,  
    .pushHandlesBuffer = 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.

- vendors/*vendor*/boards/*board*/aws\_demos/config\_files/iot\_ble\_config.h에서 IOT\_BLE\_ADD\_CUSTOM\_SERVICES를 1로 설정합니다.
- 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

플랫폼에 FreeRTOS Bluetooth Low Energy 라이브러리를 이식하는 방법에 대한 자세한 내용은 단원을 참조 하십시오.[Bluetooth Low Energy 라이브러리 이식](#)FreeRTOS 포팅 안내서를 참조하십시오.

## FreeRTOS Bluetooth 디바이스용 Mob

FreeRTOS Bluetooth 디바이스용 Mobile SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로 컨트롤러와 상호 작용하는 모바일 애플리케이션을 만들 수 있습니다. Mobile SDK도 통신할 수 있습니다.AWS서비스, 사용자 인증을 위해 Amazon Cognito를 사용합니다.

## FreeRTOS 블루투스 디바이스용 Android SDK

Bluetooth Low Energy를 통해 마이크로 컨트롤러와 상호 작용하는 Android 모바일 애플리케이션을 빌드할 수 있습니다. SDK는 [GitHub](#)에서 사용 가능합니다.

FreeRTOS 블루투스 장치용 Android SDK를 설치하려면 프로젝트의 “SDK 설정”에 대한 지침을 따르십시오.[읽어보기. md파일](#).

SDK에 포함된 데모 모바일 애플리케이션 설정 및 실행에 대한 정보는 [사전 조건 \(p. 234\)](#) 및 [FreeRTOS 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 \(p. 236\)](#)을 참조하십시오.

## FreeRTOS 디바이스용 iOS SDK

Bluetooth Low Energy를 통해 마이크로 컨트롤러와 상호 작용하는 iOS 모바일 애플리케이션을 빌드할 수 있습니다. SDK는 [GitHub](#)에서 사용 가능합니다.

iOS SDK를 설치하려면

1. [CocoaPods](#)를 다음과 같이 설치합니다.

```
$ gem install cocoapods  
$ pod setup
```

### Note

`sudo`를 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 \(p. 234\)](#) 및 [FreeRTOS 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 \(p. 236\)](#)을 참조하십시오.

## 셀룰러 인터페이스 라이브러리

### 소개

셀룰러 인터페이스 라이브러리는 통일 된 API를 통해 널리 사용되는 몇 가지 셀룰러 모뎀의 기능을 제공합니다. 현재 이 프로젝트에는 이 세 가지 셀룰러 모뎀에 대한 라이브러리가 포함되어 있습니다.

- [奎泰爾 BG96](#)
- [시에라 와이어리스 HL7802](#)
- [유블록스 사라-R4](#)

셀룰러 인터페이스 라이브러리의 현재 버전은 해당 셀룰러 모뎀에서 제공하는 TCP 스택을 캡슐화합니다. 그들은 모두 동일한 유니폼을 구현합니다. [셀룰러 인터페이스 라이브러리 API](#). 이 API는 AT 명령의 복잡성을 숨기고 소켓과 같은 인터페이스를 C 프로그래머에게 공개합니다.

애플리케이션이 셀룰러 인터페이스 API를 직접 사용하도록 선택할 수 있지만 이러한 목적으로 설계된 것은 아닙니다. 일반적인 FreeRTOS 시스템에서 응용 프로그램은 다음과 같은 상위 수준의 라이브러리를 사용합니다. [코렐QTT](#)라이브러리 및 [코레트TP](#)라이브러리, 다른 끝점과 통신합니다. 이러한 상위 레벨 라이브러리는 추상 인터페이스를 사용합니다. [전송 인터페이스](#)를 사용하여 데이터를 보내고 수신합니다. 전송 인터페이스는 셀룰러 인터페이스 라이브러리 위에 구현할 수 있습니다. [셀룰러 인터페이스 데모](#)에서는 이러한 구현을 사용합니다.

대부분의 셀룰러 모뎀은 다음에 의해 정의된 AT 명령을 다소 구현합니다. [3지PP TS v27.007](#) 표준. 이 프로젝트는 [이행](#) 이러한 표준 AT 명령의 [재사용 가능한 공통 구성 요소](#). 이 프로젝트의 세 가지 셀룰러 인터페이스 라

이브러리는 모두 공통 코드를 활용합니다. 각 모뎀의 라이브러리는 공급업체별 AT 명령만 구현한 다음 전체 셀룰러 인터페이스 라이브러리 API를 노출합니다.

3GPP TS v27.007 표준을 구현하는 공통 구성 요소는 다음 코드 품질 기준에 따라 작성되었습니다.

- GNU 복잡도 점수는 8을 넘지 않습니다
- 미스라 C:2012 코딩 표준입니다. 표준과의 편차는 “coverity”로 표시된 소스 코드 주석에 설명되어 있습니다.

## 종속성 및 요구 사항

셀룰러 인터페이스 라이브러리에는 직접적인 종속성이 없습니다. 그러나 이더넷, Wi-Fi 및 셀룰러는 FreeRTOS 네트워크 스택에 공존할 수 없습니다. 개발자는 다음과 통합할 네트워크 인터페이스 중 하나를 선택해야 합니다.[보안 소켓 라이브러리](#).

## 이식

플랫폼에 셀룰러 인터페이스 라이브러리를 이식하는 방법에 대한 자세한 내용은 단원을 참조하십시오.[셀룰러 인터페이스 라이브러리 포팅](#)의 FreeRTOS 이식 안내서.

## 메모리 사용

코드 셀룰러 인터페이스 라이브러리의 크기 (ARM Cortex-M용 GCC로 생성된 예제)		
셀룰러_3gpp_아피.c	6.3K	5.7K
셀룰러_3gpp_urc_핸들러.c	0.9K	0.8K
셀룰러_at_코어.c	1.4K	1.2K
셀룰러_커먼_API.c	0.5K	0.5K
셀룰러_커먼.c	1.6K	1.4K
셀룰러_pkthandler.c	1.4K	1.2K
셀룰러_pktio.c	1.8K	1.6K
총 추정	13.9K	12.4K

## 시작하기

### 소스 코드 다운로드

소스 코드는 FreeRTOS 라이브러리의 일부로 또는 자체적으로 다운로드 할 수 있습니다.

HTTPS를 사용하여 Github에서 라이브러리를 복제하려면

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

SSH를 사용한 사용:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

## 폴더 구조

이 저장소의 루트에 다음 폴더가 표시됩니다.

- source: 3GPP TS v27.007에 의해 정의된 표준 AT 명령을 구현하는 재사용 가능한 공통 코드
- modules: 각 셀룰러 모뎀에 대해 3GPP 이외의 AT 명령을 구현하는 공급업체별 코드
- doc: 설명서
- test: 단위 테스트 및 cbmc
- tools: 커버리티 정적 분석 및 CMock을 위한 도구

## 라이브러리 구성 및 빌드

셀룰러 인터페이스 라이브러리는 애플리케이션의 일부로 빌드해야 합니다. 이렇게 하려면 특정 구성을 제공해야 합니다. [이프리어토스\\_셀룰러\\_인터페이스\\_윈도우\\_시뮬레이터](#) 프로젝트는 다음을 제공합니다. 예 빌드를 구성하는 방법에 대해 설명합니다. 추가 정보는 단원을 참조하십시오. [셀룰러 API 레퍼런스](#).

단원을 참조하십시오. [셀룰러 인터페이스](#) 자세한 내용은 페이지를 참조하십시오.

## 셀룰러 인터페이스 라이브러리를 MCU 플랫폼과 통합

셀룰러 인터페이스 라이브러리는 추상화된 인터페이스를 사용하여 MCU에서 실행됩니다. [Comm 인터페이스](#) 셀룰러 모뎀과 통신합니다. Comm 인터페이스는 MCU 플랫폼에서도 구현되어야 합니다. Comm Interface의 가장 일반적인 구현은 UART 하드웨어를 통해 통신하지만 SPI와 같은 다른 물리적 인터페이스에서도 구현할 수 있습니다. Comm 인터페이스에 대한 설명서는 [셀룰러 라이브러리 API 레퍼런스](#). 다음과 같은 Comm 인터페이스의 구현 예제를 사용할 수 있습니다.

- FreeRTOS 윈도우 시뮬레이터 통신 인터페이스
- FreeRTOS 공통 IO UART 통신 인터페이스
- STM32 L475 디스커버리 보드 통신 인터페이스
- 시에라 센서 허브 보드 통신 인터페이스

## 공통 I/O

### 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/O](#)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

## 지원되는 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 단원을 참조하십시오.[FreeRTOS 이식 안내서](#).

# AWS IoT Device Defender 라이브러리

## 소개

이 AWS IoT Device Defender IoT 디바이스에서 보안 메트릭을 보내는 라이브러리 AWS IoT Device Defender. 이 AWS IoT Device Defender 각 장치에 대해 적절한 동작으로 정의한 내용과 편차가 있는지 장치에서 이러한 보안 메트릭을 지속적으로 모니터링합니다. 무언가가 제대로 보이지 않는다면 AWS IoT Device Defender 문제를 해결하기 위한 조치를 취할 수 있도록 경고를 보냅니다. 와의 상호 작용 AWS IoT Device Defender 용도 MQTT, 경량 게시-구독 프로토콜입니다. 이 라이브러리는 에서 사용하는 MQTT 주제 문자열을 작성하고 인식하는 API를 제공합니다. AWS IoT Device Defender.

자세한 내용은 AWS IoT 개발자 안내서의 [AWS IoT Device Defender 섹션](#)을 참조하세요.

라이브러리는 C로 작성되었으며 다음을 준수하도록 설계되었습니다. [이소 C90과 미스라 C:2012](#). 라이브러리는 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없습니다. 또한 스레딩 또는 동기화와 같은 플랫폼 종속성이 없습니다. 모든 MQTT 라이브러리 및 모든 라이브러리와 함께 사용할 수 있습니다. [JSON](#) 또는 [CBOR](#) 라이브러리 이 라이브러리는 [교정쇄](#) 안전한 메모리 사용과 힐 할당이 없으므로 IoT 마이크로 컨트롤러에 적합하지만 다른 플랫폼에도 완벽하게 휴대 할 수 있습니다.

이 AWS IoT Device Defender 라이브러리는 자유롭게 사용할 수 있으며 [MIT 오픈 소스 라이선스](#).

코드 크기(AWS IoT ARM Cortex-M용 GCC로 생성된 예제)

수비수.c	1.1K	0.6K
추정 총 추정	1.1K	0.6K

# AWS IoT Greengrass Discovery 라이브러리 개요

이 AWS IoT Greengrass 검색 라이브러리는 마이크로 컨트롤러 디바이스가 네트워크에서 Greengrass 코어를 찾는 데 사용됩니다. AWS IoT Greengrass Discovery API를 사용하면 디바이스가 코어의 앤드포인트를 찾은 후 Greengrass 코어에 메시지를 보낼 수 있습니다.

## 종속성 및 요구 사항

Greengrass Discovery 라이브러리를 사용하려면 AWS IoT에서 인증서, 정책 등과 같은 사물을 생성해야 합니다. 자세한 내용은 [AWS IoT 시작하기](#)를 참조하십시오.

`freertos/demos/include/aws_clientcredential.h` 파일에서 다음 상수의 값을 설정해야 합니다.

**clientcredentialMQTT\_BROKER\_ENDPOINT**

AWS IoT 엔드포인트입니다.

**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 [AWS IoT Greengrass 시작하기](#) 섹션을 참조하세요.

CoreMQTT 라이브러리가 Greengrass 연결에 필요하지 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 참조](#)를 참조하십시오.

## 사용 예

### Greengrass 워크플로우

MCU 디바이스는 AWS IoT에서 Greengrass 코어 연결 파라미터를 포함하는 JSON 파일을 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.h`에 정의되어 있습니다.

Greengrass 코어가 하나만 존재하는 경우 `GGD_GetGGCIPandCertificate`를 호출하여 Greengrass 코어 연결 정보로 JSON 파일을 요청합니다. `GGD_GetGGCIPandCertificate`가 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**

AWS IoT에 대한 HTTP GET 요청을 생성하여 Greengrass 코어 검색을 위한 검색 요청을 시작합니다. `GD_SecureConnect_Send`는 AWS IoT에 요청을 전송하는 데 사용됩니다.

### **GGD\_JSONRequestGetSize**

HTTP 응답에서 JSON 파일의 크기를 가져옵니다.

### **GGD\_JSONRequestGetFile**

JSON 객체 문자열을 가져옵니다. `GGD_JSONRequestGetSize` 및 `GGD_JSONRequestGetFile`은 `GGD_SecureConnect_Read`를 사용하여 소켓에서 JSON 데이터를 가져옵니다. AWS IoT에서 JSON 데이터를 수신하려면 `GGD_JSONRequestStart`, `GGD_SecureConnect_Send`, `GGD_JSONRequestGetSize`를 호출해야 합니다.

### **GGD\_GetIPandCertificateFromJSON**

JSON 데이터에서 IP 주소 및 Greengrass 코어 인증서를 추출합니다. `xAutoSelectFlag`를 `True`로 설정하여 자동 선택을 볼 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.

## 코어HTTP 라이브러리

소형 IoT 디바이스 (MCU 또는 소형 MPU) 용 HTTP C 클라이언트 라이브러리

## 소개

CoreHttp 라이브러리는 클라이언트 구현의 하위 집합입니다.[HTTP/1.1](#) 표준. HTTP 표준은 TCP/IP를 기반으로 실행되는 상태 비저장 프로토콜을 제공하며 분산, 협업, 하이퍼텍스트 정보 시스템에서 자주 사용됩니다.

CoreHttp 라이브러리는 [HTTP/1.1](#) 프로토콜 표준 이 라이브러리는 낮은 메모리 공간을 위해 최적화되었습니다. 라이브러리는 애플리케이션이 동시성을 완전히 관리할 수 있도록 완전 동기식 API를 제공합니다. 고정 버퍼만 사용하므로 애플리케이션이 메모리 할당 전략을 완벽하게 제어할 수 있습니다.

라이브러리는 C로 작성되었으며 다음을 준수하도록 설계되었습니다. [이소 C90과 MISRA C:2012](#). 라이브러리의 유일한 종속성은 표준 C 라이브러리입니다. [HTTP-파서의 LTS 버전 \(v12.19.1\)](#) Node.js 부터. 라이브러리는 [교정쇄](#) 안전한 메모리 사용과 힙 할당이 없으므로 IoT 마이크로 컨트롤러에 적합하지만 다른 플랫폼에도 완벽하게 휴대 할 수 있습니다.

IoT 애플리케이션에서 HTTP 연결을 사용할 때는 에서 설명한 대로 TLS 프로토콜을 사용하는 것과 같은 보안 전송 인터페이스를 사용하는 것이 좋습니다. [CoreHTTP 상호 인증 데모 \(p. 256\)](#).

이 라이브러리는 자유롭게 사용할 수 있으며 [MIT 오픈 소스 라이선스](#).

코드 크기 CoreHttp (ARM Cortex-M용 GCC로 생성된 예제)		
코어_HTTP_클라이언트.c	3.2K	2.6K
http_파서.c (HTTP 파서)	15.7K	13.0K
추정 합계	18.9K	15.6K

## 코어JSON 라이브러리

### 소개

JSON (JavaScript Object Notation) 은 사람이 읽을 수 있는 데이터 직렬화 형식 다음과 같은 데이터를 교환하는 데 널리 사용됩니다. [AWS IoT 디바이스 샘플 서비스](#)는 GitHub REST API와 같은 많은 API의 일부입니다. JSON은 Ecma 인터내셔널의 표준으로 유지됩니다.

CoreJSON 라이브러리는 키 조회를 지원하는 파서를 제공하면서 엄격하게 [ECMA-404 표준 JSON 데이터 교환 구문](#). 이 라이브러리는 C로 작성되었으며 ISO C90 및 MISRA C:2012를 준수하도록 설계되었습니다. 가 됩니다. [교정쇄](#) 안전한 메모리 사용과 힙 할당이 없으므로 IoT 마이크로 컨트롤러에 적합하지만 다른 플랫폼에도 완벽하게 휴대 할 수 있습니다.

### 메모리 사용

CoreJSON 라이브러리는 내부 스택을 사용하여 JSON 문서의 중첩된 구조를 추적합니다. 스택은 단일 함수 호출 기간 동안 존재하며 보존되지 않습니다. 매크로를 정의하여 스택 크기를 지정할 수 있습니다. `JSON_MAX_DEPTH` 기본값은 32개 레벨입니다. 각 레벨은 단일 바이트를 소비합니다.

CoreJSON 코드 크기 (ARM Cortex-M용 GCC로 생성된 예제)		
코어_json.c	2.9K	2.4K
총 추정	2.9K	2.4K

## CoreMQTT 라이브러리

### 소개

CoreMqtt 라이브러리는 클라이언트 구현입니다. [MQTT](#)(메시지 큐 원격 분석 전송) 표준입니다. MQTT 표준은 경량 게시/구독을 제공합니다 (또는 [펍서브](#)) TCP/IP 위에서 실행되며 M2M (기계 대 기계) 및 사물 인터넷 (IoT) 사용 사례에서 자주 사용되는 메시징 프로토콜.

CoreMqtt 라이브러리는 MQTT 3.1.1 프로토콜 표준. 이 라이브러리는 낮은 메모리 공간을 위해 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. 라이브러리는 특정 사용 사례의 요구에 정확하게 맞도록 선택 및 결합할 수 있는 컴포저블 함수 메뉴를 제공합니다.

라이브러리는 다음과 같이 작성되었습니다. C규정을 준수하도록 설계 [이소 C90과 미스라 C:2012](#). 이 MQTT 라이브러리는 다음을 제외하고 추가 라이브러리에 대한 종속성이 없습니다.

- 표준 C 라이브러리
- 고객이 구현한 네트워크 전송 인터페이스
- (선택 사항) 사용자가 구현한 플랫폼 시간 함수

라이브러리는 간단한 송신 및 수신 전송 인터페이스 사양을 제공함으로써 기본 네트워크 드라이버와 분리됩니다. 응용 프로그램 작성기는 기존 전송 인터페이스를 선택하거나 해당 응용 프로그램에 맞게 고유한 전송 인터페이스를 구현할 수 있습니다.

라이브러리는 MQTT 브로커에 연결하고, 주제를 구독/구독 취소하고, 주제에 메시지를 게시하고, 수신 메시지를 수신하는 고급 API를 제공합니다. 이 API는 위에서 설명한 전송 인터페이스를 매개 변수로 사용하며 이를 사용하여 MQTT 브로커와 메시지를 주고받습니다.

라이브러리는 또한 저수준 시리얼라이저/디시리얼라이저 API를 노출합니다. 이 API는 다른 오버헤드 없이 필요한 MQTT 기능의 하위 집합으로 구성된 간단한 IoT 애플리케이션을 구축하는데 사용할 수 있습니다. 시리얼 라이저/디시리얼라이저 API는 소켓과 같은 사용 가능한 전송 계층 API와 함께 사용하여 브로커와 메시지를 주고받을 수 있습니다.

IoT 애플리케이션에서 MQTT 연결을 사용할 때는 TLS 프로토콜을 사용하는 것과 같은 보안 전송 인터페이스를 사용하는 것이 좋습니다.

이 MQTT 라이브러리에는 스레딩 또는 동기화와 같은 플랫폼 종속성이 없습니다. 이 라이브러리는 [교정쇄](#) 안전한 메모리 사용과 힐 할당이 없음을 보여 주므로 IoT 마이크로 컨트롤러에 적합하지만 다른 플랫폼에도 완벽하게 이식 할 수 있습니다. 자유롭게 사용할 수 있으며 아래에 배포됩니다. [MIT 오픈 소스 라이선스](#).

코드 크기 (ARM Cortex-M용 GCC로 생성된 예제)		
코어_mqtt.c	3.0K	2.6K
코어_mqtt_state.c	1.4K	1.1K
코어_mqtt_시리얼라이저.c	2.5K	2.0K
추정 총	6.9K	5.7K

## CoremQTT 에이전트 라이브러리

### 소개

CoreMqtt 에이전트 라이브러리는 스레드 안전성을 추가하는 높은 수준의 API입니다. [CoremQTT 라이브러리 \(p. 214\)](#). 백그라운드에서 MQTT 연결을 관리하고 다른 작업의 개입이 필요하지 않은 전용 MQTT 에이전트 작업을 만들 수 있습니다. 라이브러리는 CoreMqtt의 API와 동등한 스레드 안전 기능을 제공하므로 다중 스레드 환경에서 사용할 수 있습니다.

MQTT 에이전트는 독립 작업 (또는 실행 스레드)입니다. MQTT 라이브러리의 API에 액세스할 수 있는 유일한 작업이므로 스레드 안전성을 달성합니다. 모든 MQTT API 호출을 단일 작업으로 격리하여 액세스를 직렬화하고 세마포어 또는 기타 동기화 프리미티브가 필요하지 않습니다.

라이브러리는 스레드 안전 메시징 대기열 (또는 기타 프로세스 간 통신 메커니즘)을 사용하여 MQTT API를 호출하는 모든 요청을 직렬화합니다. 메시징 구현은 메시징 인터페이스를 통해 라이브러리에서 분리되므로 라이브러리를 다른 운영 체제로 이식할 수 있습니다. 메시징 인터페이스는 에이전트의 명령 구조에 대한 포인터를 보내고 수신하는 함수와 이러한 명령 객체를 할당하는 함수로 구성되어 있으므로 응용 프로그램 작성자가 애플리케이션에 적합한 메모리 할당 전략을 결정할 수 있습니다.

라이브러리는 C로 작성되었으며 다음을 준수하도록 설계되었습니다. [이소 C90과 미스라 C:2012](#). 라이브러리에는 다음 이외의 추가 라이브러리에 대한 종속성이 없습니다. [CoreMQTT 라이브러리 \(p. 214\)](#) 표준 C 라이브러리가 있습니다. 라이브러리 [교정쇄](#) 안전한 메모리 사용과 힙 할당이 없으므로 IoT 마이크로 컨트롤러에 사용할 수 있지만 다른 플랫폼에서도 완벽하게 이식 할 수 있습니다.

이 라이브러리는 자유롭게 사용할 수 있으며 [MIT 오픈 소스 라이선스](#).

코드 CoreMQTT 에이전트 크기 (ARM Cortex-M용 GCC로 생성된 예제)		
코어_mqtt_에이전트.c	1.7K	1.5K
코어_mqtt_에이전트_명령_함수.c	0.3K	0.2K
코어_mqtt.c (코렐쿼트)	3.0K	2.6K
코어_mqtt_state.c (코렐QTT)	1.4K	1.1K
코어_mqtt_시리얼라이저.c (코렐QTT)	2.5K	2.0K
추정 합계	8.9K	7.4K

## AWS IoT무선 (OTA) 라이브러리

### 소개

이 [AWS IoT무선 \(OTA\) 업데이트 라이브러리](#) 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 공유하여 잠재적으로 많은 양의 RAM을 절약할 수 있습니다. 또한 OTA 에이전트 라이브러리를 사용하여 펌웨어 업데이트를 테스트, 커밋 또는 롤백하기 위한 애플리케이션별로 직을 정의할 수 있습니다.

사물인터넷 (IoT)은 인터넷 연결을 전통적으로 연결되지 않은 임베디드 장치로 확장합니다. 이러한 장치는 인터넷을 통해 사용 가능한 데이터를 전달하도록 프로그래밍할 수 있으며 원격으로 모니터링하고 제어할 수 있습니다. 기술의 발전과 함께 이러한 전통적인 임베디드 장치는 소비자, 산업 및 기업 공간에서 빠른 속도로 인터넷 기능을 제공하고 있습니다.

IoT 디바이스는 일반적으로 대량으로 배포되며, 종종 작업자가 액세스하기 어렵거나 비실용적이지 않은 장소에 배포됩니다. 데이터를 노출할 수 있는 보안 취약성이 발견되는 시나리오를 상상해보십시오. 이러한 시나리오에서는 영향을 받는 장치를 보안 수정으로 빠르고 안정적으로 업데이트하는 것이 중요합니다. OTA 업데이트를 수행할 수 없으면 지리적으로 분산된 장치를 업데이트하는 것도 어려울 수 있습니다. 기술자가 이러한 장치를 업데이트하는 것은 비용이 많이 들고 시간이 많이 걸리며 때로는 비실용적입니다. 이러한 장치를 업데이트하는 데 필요한 시간은 보안 취약성에 더 오랜 기간 동안 노출됩니다. 업데이트를 위해 이러한 장치를 리콜하는 것은 비용이 많이 들고 가동 중지 시간으로 인해 소비자에게 심각한 중단을 초래할 수 있습니다.

Over the Air (OTA) 업데이트를 사용하면 비용이 많이 드는 리콜 또는 기술자를 방문하지 않고도 장치 펌웨어를 업데이트할 수 있습니다. 이 방법을 사용하면 다음과 같은 이점이 있습니다.

- 보안- 디바이스가 현장에 배포된 후 발견되는 보안 취약점 및 소프트웨어 버그에 신속하게 대응할 수 있는 기능
- 혁신- 새로운 기능이 개발됨에 따라 제품을 자주 업데이트하여 혁신 사이클을 주도할 수 있습니다. 업데이트는 기존 업데이트 방법에 비해 가동 중지 시간을 최소화하면서 신속하게 적용될 수 있습니다.
- 비용- OTA 업데이트는 전통적으로 이러한 장치를 업데이트하는 데 사용되는 방법에 비해 유지 보수 비용을 크게 줄일 수 있습니다.

OTA 기능을 제공하려면 다음과 같은 설계 고려 사항이 필요합니다.

- 통신 보안- 업데이트는 전송 중에 다운로드가 변조되지 않도록 암호화된 통신 채널을 사용해야 합니다.
- 복구- 간헐적인 네트워크 연결 또는 잘못된 업데이트 수신으로 인해 업데이트가 실패할 수 있습니다. 이러한 시나리오에서는 장치가 안정된 상태로 돌아가고 벽돌이 생기지 않도록 해야 합니다.
- 작성자 확인- 버전 및 호환성 확인과 같은 다른 유효성 검사와 함께 신뢰할 수 있는 소스에서 업데이트를 확인해야 합니다.

FreeRTOS로 OTA 업데이트 설정에 대한 자세한 내용은 단원을 참조하십시오.[FreeRTOS 무선 업데이트 \(p. 140\)](#).

## AWS IoT무선 (OTA) 라이브러리

이 AWS IoTOTA 라이브러리를 사용하면 새로 사용 가능한 업데이트의 알림을 관리하고 다운로드하고 펌웨어 업데이트의 암호화 확인을 수행할 수 있습니다. OTA (Over-the-air) 클라이언트 라이브러리를 사용하면 펌웨어 업데이트 메커니즘을 장치에서 실행 중인 응용 프로그램과 논리적으로 분리할 수 있습니다. OTA (Over-the-air) 클라이언트 라이브러리는 응용 프로그램과 네트워크 연결을 공유하여 리소스가 제한된 장치에 메모리를 절약할 수 있습니다. 또한 OTA (Over-the-Air) 클라이언트 라이브러리를 사용하여 펌웨어 업데이트를 테스트, 커밋 또는 를 백하기 위한 애플리케이션별로 직을 정의할 수 있습니다. 라이브러리는 메시지 큐 원격 분석 전송 (MQTT) 및 HTTP (하이퍼텍스트 전송 프로토콜) 와 같은 다양한 응용 프로그램 프로토콜을 지원하며 네트워크 유형 및 조건에 맞게 미세 조정할 수 있는 다양한 구성 옵션을 제공합니다.

이 라이브러리의 API는 다음과 같은 주요 기능을 제공합니다.

- 사용 가능한 새 업데이트 요청에 대한 알림 또는 폴링에 등록하십시오.
- 업데이트 요청을 수신, 구문 분석 및 검증합니다.
- 업데이트 요청의 정보에 따라 파일을 다운로드하고 확인합니다.
- 수신된 업데이트를 활성화하기 전에 자체 테스트를 실행하여 업데이트의 기능적 유효성을 확인합니다.
- 디바이스의 상태를 업데이트합니다.

이 라이브러리는 AWS 펌웨어 업데이트 전송, 여러 지역에서 많은 수의 장치 모니터링, 잘못된 배포의 폭발 반경 감소, 업데이트 보안 확인과 같은 다양한 클라우드 관련 기능을 관리하는 서비스입니다. 이 라이브러리는 모든 MQTT 또는 HTTP 라이브러리와 함께 사용할 수 있습니다.

이 라이브러리의 데모는 CoreMQTT 라이브러리를 사용하여 완전한 무선 업데이트를 보여줍니다. AWS FreeRTOS 디바이스의 서비스

## 기능

전체 OTA 에이전트 인터페이스는 다음과 같습니다.

### [OTA\\_Init](#)

시스템에서 OTA 에이전트 ("OTA 작업")를 시작하여 OTA 엔진을 초기화합니다. OTA 에이전트는 하나만 있을 수 있습니다.

#### **OTA\_Shutdown**

OTA 에이전트에 신호를 보내 종료합니다. OTA 에이전트는 선택적으로 모든 MQTT 작업 알림 주제의 구독을 취소하고 진행 중인 OTA 작업이 있는 경우 중지하고 모든 리소스를 지웁니다.

#### **OTA\_GetState**

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 중지합니다.

#### **OTA\_Resume**

OTA 에이전트 작업을 다시 시작합니다.

#### **OTA\_SignalEvent**

OTA 에이전트 작업에 이벤트를 알립니다.

#### **OTA\_EventProcessingTask**

OTA 에이전트 이벤트 처리 루프입니다.

#### **OTA\_GetStatistics**

수신, 대기, 처리 및 삭제된 패킷 수를 포함하는 OTA 메시지 패킷의 통계를 가져옵니다.

#### **OTA\_Err\_strerror**

OTA 오류에 대한 오류 코드를 문자열로 변환합니다.

#### **OTA\_JobParse\_strerror**

OTA Job 구문 분석 오류 코드를 문자열로 변환합니다.

#### **OTA\_PalStatus\_strerror**

OTA PAL 상태에 대한 상태 코드를 문자열로 변환합니다.

#### **OTA\_OsStatus\_strerror**

OTA OS 상태에 대한 상태 코드를 문자열로 변환합니다.

## **API 참조**

자세한 내용은 단원을 참조하십시오.[AWS IoTOTA \(무선\) 업데이트: 함수](#).

## 사용 예

MQTT 프로토콜을 사용하는 일반적인 OTA 지원 디바이스 애플리케이션은 다음과 같은 API 호출 시퀀스를 사용하여 OTA 에이전트를 구동합니다.

1. 에 ConnectAWS IoT 코럼 MQTT 에이전트. 자세한 정보는 [CoreMQTT 에이전트 라이브러리 \(p. 215\)](#)을 참조하십시오.
2. 을 호출하여 OTA 에이전트 초기화 OTA\_Init 버퍼, 필수 ota 인터페이스, 사물 이름 및 응용 프로그램 콜백을 포함합니다. 콜백은 OTA 업데이트 작업을 완료한 이후에 실행되는 애플리케이션별로 직을 구현합니다.
3. OTA 업데이트가 완료되면 FreeRTOS는 다음 이벤트 중 하나를 사용하여 작업 완료 콜백을 호출합니다. accepted, rejected 또는 self test.
4. 유효성 검사 오류 등으로 인해 새 펌웨어 이미지가 거부된 경우 애플리케이션에서는 일반적으로 알림을 무시하고 다음 업데이트를 대기할 수 있습니다.
5. 업데이트가 유효하고 수락됨으로 표시된 경우 OTA\_ActivateNewImage를 호출하여 디바이스를 재설정하고 새 펌웨어 이미지를 부팅합니다.

## 이식

플랫폼에 OTA 기능을 이식하는 방법에 대한 자세한 내용은 단원을 참조하십시오. [OTA 라이브러리 이식](#) FreeRTOS 포팅 안내서를 참조하십시오.

## Memory:

코드 크기 AWS IoTOTA (ARM Cortex-M용 GCC로 생성된 예제)		
ota.c	8.0K	7.2K
ota_인터페이스.c	0.1K	0.1K
오타_베이스64.c	0.6K	0.6K
오타_mqtt.c	2.3K	2.2K
오타_cbor.c	0.8K	0.6K
오타_HTTP.c	0.3K	0.3K
추정 합계	12.1K	11.0K

## 코어프KCS11 라이브러리

### 개요

공개 키 암호화 표준 #11은 암호화 토큰을 관리하고 사용하기 위한 플랫폼 독립적인 API를 정의합니다. PKCS #11은 표준 및 표준 자체에서 정의한 API를 참조합니다. PKCS #11 암호화 API는 키 스토리지, 암호화 객체에 대한 get/set 속성 및 세션 의미 체계를 추출합니다. 일반적인 암호화 객체를 조작하는 데 널리 사용되며, 응용 프로그램이 지정하는 기능을 통해 응용 프로그램 소프트웨어가 해당 객체를 응용 프로그램의 메모리에 노출시키지 않고 암호화 객체를 사용, 생성, 수정 및 삭제할 수 있기 때문에 중요합니다. 예를 들어, FreeRTOS AWS 참조 통합은 PKCS #11 API의 작은 하위 집합을 사용하여 인증되고 보호되는 네트워크 연결

을 만드는 데 필요한 비밀 (개인) 키에 액세스합니다.[TLS\(전송 계층 보안\)](#)응용 프로그램이 키를 '보지 않고' 프로토콜입니다.

CorePKCS11 라이브러리에는 Mbed TLS에서 제공하는 암호화 기능을 사용하는 PKCS #11 인터페이스 (API) 의 소프트웨어 기반 모의 구현이 포함되어 있습니다. 소프트웨어 모의 (mock) 을 사용하면 빠른 개발과 유연성이 가능하지만, 모의 객체를 프로덕션 디바이스에 사용되는 보안 키 스토리지와 관련된 구현으로 대체할 것으로 예상됩니다. 일반적으로 TPM (신뢰할 수 있는 플랫폼 모듈), HSM (하드웨어 보안 모듈), 보안 요소 또는 기타 보안 하드웨어 앤클레이브와 같은 보안 암호화된 프로세서의 공급업체는 하드웨어와 함께 PKCS #11 구현을 배포합니다. 따라서 CorePKCS11 소프트웨어 전용 모의 라이브러리의 목적은 프로덕션 장치에서 암호 프로세서 특정 PKCS #11 구현으로 전환하기 전에 신속한 프로토 타이핑 및 개발을 허용하는 비 하드웨어 특정 PKCS #11 구현을 제공하는 것입니다.

비대칭 키, 난수 생성 및 해싱과 관련된 작업에 중점을 둔 PKCS #11 표준의 하위 집합만 구현됩니다. 대상 사용 사례에는 TLS 인증을 위한 인증서 및 키 관리, 소형 임베디드 장치에서 코드 서명 서명 확인 등이 포함됩니다. 파일 보기 `pkcs11.h` FreeRTOS 소스 코드 리포지토리에서 OASIS 표준 기구에서 가져온 를 참조하십시오. 에서 [FreeRTOS 참조 구현](#) PKCS #11 API 호출은 TLS 헬퍼 인터페이스에서 중에 TLS 클라이언트 인증을 위해 호출합니다. `SOCKETS_Connect`. 1회 개발자 프로비저닝 워크플로우에서 인증을 위한 프라이빗 키와 TLS 클라이언트 인증서를 가져오기 위해서도 PKCS #11 API를 호출합니다. AWS IoT MQTT 브로커. 이 두 사용 사례 (프로비저닝 및 TLS 클라이언트 인증) 에서는 PKCS #11 인터페이스 표준의 작은 하위 집합만 구현해야 합니다.

## 기능

PKCS #11 하위 집합이 사용됩니다. 이 목록은 프로비저닝, TLS 클라이언트 인증 및 정리를 지원하기 위해 투린을 호출하는 순서와 거의 동일합니다. 함수에 대한 자세한 내용은 표준 위원회에서 제공하는 PKCS #11 문서를 참조하십시오.

## 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 NIST P-256 곡선을 가진 ECDSA 및 PKCS #11 2048비트 RSA (서명 전용)를 사용합니다. 다음 지침에서는 P-256 클라이언트 인증서를 기반으로 AWS IoT 사물을 생성하는 방법을 설명합니다.

다음 버전 이상의 AWS CLI 및 OpenSSL을 사용 중인지 확인하십시오.

```
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 을 (를) 사용한 것으로 가정합니다.`aws configure`를 구성하는 명령AWS CLI. 자세한 내용은 단원을 참조하십시오.[를 통한 빠른 구성aws configure의 AWS Command Line Interface 사용 설명서](#).

생성AWS IoT-P-256 클라이언트 인증서를 기반으로 사물

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 요청을 AWS IoT에 제출합니다.

```
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 허용적이므로, 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. 를 생략할 수 있습니다.  
`greengrass:*` 연결 및 검색을 위해 FreeRTOS 데모를 실행하지 않으려면 작업을 수행합니다.

```
{
    "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

플랫폼에 CorePKCS11 라이브러리를 이식하는 방법에 대한 자세한 내용은 단원을 참조하십시오. [코렐 KCS11 라이브러리 포팅](#) FreeRTOS 포팅 안내서를 참조하십시오.

## Memory 사용

코드 크기 (ARM Cortex-M용 GCC로 생성된 예제)

코어_pkcs11.c	0.8K	0.8K
코어_pki_utils.c	0.5K	0.3K
코어_pkcs11_엠베틀스	8.7K	7.3K
총 추정	10.0K	8.4K

# 보안 소켓 라이브러리

## 개요

FreeRTOS를 사용할 수 있습니다. [보안 소켓](#) 안전하게 통신하는 내장형 애플리케이션을 만드는 라이브러리입니다. 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 단원을 참조하십시오. `SOCKETS_SetSockOpt`의 [Secure Sockets API](#) 참조.

### Note

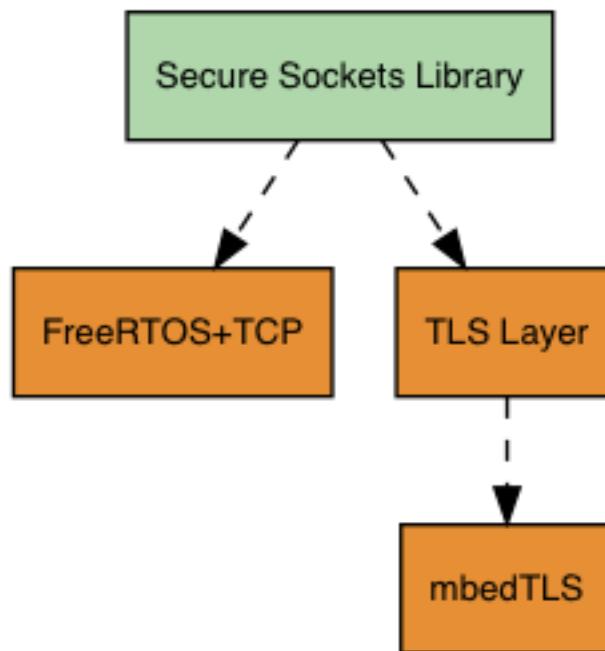
현재는 클라이언트 API만 있으며 [경량 IP \(LWIP\)](#) 서버 측 Bind API는 FreeRTOS 보안 소켓에 대해 지원됩니다.

## 종속성 및 요구 사항

FreeRTOS 보안 소켓 라이브러리는 TCP/IP 스택 및 TLS 구현에 종속됩니다. FreeRTOS용 포트는 다음 세 가지 방법 중 하나로 이러한 종속성을 충족합니다.

- TCP/IP 및 TLS의 사용자 지정 구현
- TCP/IP 및 FreeRTOS TLS 계층의 사용자 지정 구현 [mbedtls](#)
- [FreeRTOS+TCP](#) 그리고 FreeRTOS TLS 레이어 [mbedtls](#)

아래 종속성 다이어그램은 FreeRTOS 보안 소켓 라이브러리에 포함된 참조 구현을 보여 줍니다. 이 참조 구현은 FreeRTOS+TCP와 mbedtls을 종속성으로 하여 이더넷과 Wi-Fi를 통한 TLS 및 TCP/IP를 지원합니다. FreeRTOS TLS 계층에 대한 자세한 내용은 단원을 참조하십시오. [전송 계층 보안 \(p. 227\)](#).



## 기능

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 참조](#)의 보안 소켓 오류 코드를 참조하십시오.

#### Note

FreeRTOS 보안 소켓 API가 오류 코드를 반환하는 경우 [CoreMQTT 라이브러리](#) (p. 214) FreeRTOS 보안 소켓 라이브러리에 종속된 오류 코드를 반환합니다. AWS\_IOT\_MQTT\_SEND\_ERROR.

## 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 육텟으로 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 FreeRTOS 보안 소켓 라이브러리에서 지원되지 않습니다. [경량 IP \(LWIP\) 서버 층 Bind API](#), 클라우드 API가 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\\_IPInit](#)도 호출해야 합니다.

## API 참조

전체 API 참조는 [보안 소켓 API 참조](#)를 참조하십시오.

## 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.\n
Change this to the certificate of your choice. */
static const char ctlsECHO_SERVER_CERTIFICATE_PEM[ ] =
"-----BEGIN CERTIFICATE-----\n"
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owu2eiedgPySjAKBggqhkJOPQODAjA5\n"
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQDExBBWF6b24g\n"
"Um9vdCBDQSAzMb4XDTE1MDUyNjAwMDAwMFoXTDQwMDUyNjAwMDAwMFowOTELMakG\n"
"A1UEBhMCVVMxDzANBgNVBAoTBkFTYXpbjEZMBCGA1UEAxMQQW1hem9uIFJvb3Qg\n"
"Q0EgMzbZMBMGBYqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG81Kl\n"
"ui2yEujSLtf6ycXYqm0fc4E705hrOXwzpcVoho6AF2hiRVd9RFgdssf1ZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVROPAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDACKBggqhkJOPQODAgNJADBGAIEA4IWSoxe3jfkr\n"
"BqWTrBqYaGFy+uGh0PsceGCmQ5nFuM0CIQCcAu/xIJyz1vnrxir4tiz+OpAUfTeM\n"
"YyRIHN8wfdfVoOw==\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_SNDFTIMEO, &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 \(p. 312\)](#) 단원을 참조하십시오.

## 이식

FreeRTOS 보안 소켓은 TCP/IP 스택 및 TLS 구현에 종속됩니다. 스택에 따라 보안 소켓 라이브러리를 이식 하려면 다음 중 일부를 이식해야 할 수 있습니다.

- [FreeRTOS+TCP TCP/IP 스택](#)
- [The 코어프KCS11 라이브러리 \(p. 219\)](#)
- [The 전송 계층 보안 \(p. 227\)](#)

포팅에 대한 자세한 내용은 단원을 참조하십시오.[보안 소켓 라이브러리 이식](#)FreeRTOS 포팅 안내서를 참조 하십시오.

# AWS IoT 디바이스 샐도우 라이브러리 소개

이AWS IoT 현재 상태를 저장하고 검색하는 Device Shadow 라이브러리 (그림자) 등록된 모든 디바이스의 디바이스의 샐도우는 디바이스가 오프라인 상태인 경우에도 웹 애플리케이션에서 상호 작용할 수 있는 디바이스의 영구적인 가상 표현입니다. 장치 상태가 그림자로 캡처됩니다.[JSON](#)document. 명령을 에 보낼 수 있습니다.AWS IoT MQTT 또는 HTTP를 통한 Device Shadow 서비스를 통해 알려진 최신 디바이스 상태를 퀴리하거나 상태를 변경합니다. 각 디바이스 샐도우는 해당 디바이스 이름으로 고유하게 식별됩니다.의회에서

특정 디바이스 또는 논리적 엔터티의 표현 AWS 클라우드 자세한 내용은 단원을 참조하십시오. [에서 디바이스 관리 AWS IoT](#). 그림자에 대한 자세한 내용은 에서 확인하실 수 있습니다. [AWS IoT 선적 서류 비치](#).

이 AWS IoT Device Shadow 라이브러리에는 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없습니다. 또한 스레딩 또는 동기화와 같은 플랫폼 종속성이 없습니다. 모든 MQTT 라이브러리 및 모든 JSON 라이브러리와 함께 사용할 수 있습니다.

이 라이브러리는 자유롭게 사용할 수 있으며 [MIT 오픈 소스 라이선스](#).

코드 크기 AWS IoT Device Shadow (ARM Cortex-M용 GCC로 생성된 예제)		
그림자.c	1.2K	0.9K
총 추정	1.2K	0.9K

## AWS IoT 작업 라이브러리

### 소개

AWS IoT Jobs는 하나 이상의 연결된 장치에 보류 종임을 알리는 서비스입니다. 일. 작업을 사용하여 장치 집합을 관리하고 장치의 펌웨어 및 보안 인증서를 업데이트하거나 장치 재시작 및 진단 수행과 같은 관리 작업을 수행할 수 있습니다. 자세한 내용은 단원을 참조하십시오. [작업의 AWS IoT 개발자 안내서](#). 와의 상호 작용 AWS IoT 작업 서비스 [MQTT](#), 경량 게시-구독 프로토콜입니다. 이 라이브러리는 에서 사용하는 MQTT 주제 문자열을 작성하고 인식하는 API를 제공합니다. AWS IoT 작업 서비스

이 AWS IoT Jobs 라이브러리는 C로 작성되었으며 다음을 준수하도록 설계되었습니다. [이소 C90과 미스라 C:2012](#). 라이브러리는 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없습니다. 모든 MQTT 라이브러리 및 모든 JSON 라이브러리와 함께 사용할 수 있습니다. 라이브러리는 [교정쇄](#) 안전한 메모리 사용과 힙 할당이 없으므로 IoT 마이크로 컨트롤러에 적합하지만 다른 플랫폼에도 완벽하게 휴대 할 수 있습니다.

이 라이브러리는 자유롭게 사용할 수 있으며 [MIT 오픈 소스 라이선스](#).

코드 크기 AWS IoT 작업 (ARM Cortex-M용 GCC로 생성된 예제)		
직무.c	1.8K	1.5K
추정 합계	1.8K	1.5K

## 전송 계층 보안

FreeRTOS 전송 계층 보안 (TLS) 인터페이스는 암호화 구현 세부 정보를 수집하는 데 사용되는 선택적 씬 래퍼입니다. [Secure Sockets Layer](#) 프로토콜 스택에서 위에 있는 (SSL) 인터페이스입니다. 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. 단원을 참조하십시오. [iot\\_tls.h](#) FreeRTOS 소스 코드 리포지토리의

SSL에서 암호화 라이브러리로 직접 인터페이스 할 수 있으므로 TLS 인터페이스는 선택 사항입니다. 이 인터페이스는 네트워크 전송 및 TLS의 전체 스택 오프로드 구현을 포함하는 MCU 솔루션에는 사용되지 않습니다.

TLS 인터페이스 포팅에 대한 자세한 내용은 단원을 참조하십시오. [TLS 라이브러리 이식](#)의 FreeRTOS 이식 안내서.

# Wi-Fi 라이브러리

## 개요

더 FreeRTOS Wi-Fi 라이브러리는 포트별 Wi-Fi 구현을 공통 API로 추상화하여 Wi-Fi 기능을 갖춘 모든 FreeRTOS 검증 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 코어.

## 기능

Wi-Fi 라이브러리에는 다음 기능이 있습니다.

- WEP, WPA, WPA3 인증 Support
- 액세스 포인트 검색
- 전력 관리
- 네트워크 프로파일링

Wi-Fi 라이브러리의 기능에 대한 자세한 내용은 아래를 참조하십시오.

## Wi-Fi 모드

Wi-Fi 디바이스는 세 가지 모드 중 하나일 수 있습니다. 스테이션, 액세스 포인트 또는 P2P `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, 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\_ConnectAP를 호출하고 네트워크 자격 증명을 제공하여 네트워크에 연결할 수 있습니다. WIFI\_Disconnect를 호출하여 네트워크에서 Wi-Fi 디바이스를 연결 해제할 수 있습니다. 검색 및 연결에 대한 자세한 내용은 [사용 예 \(p. 230\)](#) 및 [API 참조 \(p. 230\)](#)를 참조하십시오.

## 전력 관리

애플리케이션과 사용 가능한 전원에 따라 Wi-Fi 디바이스마다 전력 요구 사항이 다릅니다. 디바이스를 항상 켜서 지역 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\_SetPMMMode 함수를 사용하여 디바이스에 대한 전력 모드를 설정합니다. WIFI\_GetPMMMode 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\_NetworkAdd는 네트워크 프로필을 추가하고, WIFI\_NetworkGet은 네트워크 프로필을 검색하고, WIFI\_NetworkDel은 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 참조 \(p. 230\)](#)를 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.c에서 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 \(p. 230\)](#) 및 [API 참조 \(p. 230\)](#)를 참조하세요.

## API 참조

전체 API 참조는 [Wi-Fi API 참조](#)를 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

```
WIFI_NetworkParams_t xNetworkParams;
WIFI_ReturnCode_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.h에 정의된 함수를 구현해야 합니다. 적어도 이 구현은 필수적이지 않거나 지원되지 않는 함수에 대해 eWiFiNotSupported를 반환해야 합니다.

Wi-Fi 라이브러리 이식에 대한 자세한 내용은 단원을 참조하십시오.[Wi-Fi 라이브러리 이식](#)FreeRTOS 이식  
안내서를 참조하십시오.

# FreeRTOS 데모

FreeRTOS는 demos는 주 FreeRTOS 디렉터리의 폴더에 있습니다. FreeRTOS에서 실행할 수 있는 모든 예제는 common폴더, 아래demos. 또한 FreeRTOS 인증 플랫폼별로 폴더가demosfolder. 이FreeRTOS 콘솔을 선택한 대상 플랫폼만 아래에 하위 디렉터리가 있습니다.demos.

데모 애플리케이션을 사용하기 전에 [FreeRTOS 시작하기 \(p. 14\)](#)에서 자습서를 완료하는 것이 좋습니다. CoreeRTT 에이전트 데모를 설정 및 실행하는 방법을 설명합니다.

## FreeRTOS 데모 실행

다음 주제에서는 FreeRTOS 데모를 설정하고 실행하는 방법을 설명합니다.

- Bluetooth Low Energy 데모 애플리케이션 (p. 233)
- Microchip Curiosity PIC32MZEF용 데모 부트로더 (p. 244)
- AWS IoT Device Defender 데모 (p. 249)
- AWS IoT Greengrass Discovery 데모 애플리케이션 (p. 253)
- CoreHttp 데모 (p. 256)
- AWS IoT 작업 라이브러리 (p. 264)
- CoremQTT 데모 (p. 265)
- OTA(Over-the-Air) 업데이트 데모 애플리케이션 (p. 271)
- 보안 소켓 에코 클라이언트 데모 (p. 312)
- AWS IoT 디바이스 색도우 데모 애플리케이션 (p. 309)

O|DEMO\_RUNNER\_RunDemos함수,freertos/demos/demo\_runner/iot\_demo\_runner.cfile는 단일 데모 애플리케이션이 실행되는 분리된 스레드를 초기화합니다. 기본적으로 DEMO\_RUNNER\_RunDemosCoremQTT 에이전트 데모를 호출하고 시작합니다. FreeRTOS를 다운로드할 때 선택한 구성에 따라 또는 FreeRTOS를 다운로드한 위치에 따라 다른 예제 실행 함수가 기본적으로 시작될 수 있습니다. 데모 응용 프로그램을 활성화하려면freertos/vendors/vendor/boards/board/aws\_demos/config\_files/aws\_demo\_config.h파일을 실행하고 실행할 데모를 정의합니다.

### Note

모든 예제가 함께 상호 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 개요

FreeRTOS Bluetooth Low Energy 데모 애플리케이션에는

- MQTT over Bluetooth Low Energy (p. 240) 데모

이 애플리케이션은 MQTT over Bluetooth Low Energy 서비스를 사용하는 방법을 보여 줍니다.

- Wi-Fi 프로비저닝 (p. 241) 데모

이 애플리케이션은 Bluetooth Low Energy Wi-Fi 프로비저닝 서비스를 사용하는 방법을 보여 줍니다.

- 일반 속성 서버 (p. 243) 데모

이 애플리케이션은 FreeRTOS Bluetooth Low Energy 미들웨어 API를 사용하여 간단한 GATT 서버를 생성하는 방법을 보여 줍니다.

Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오.[FreeRTOS 시작하기 \(p. 14\)](#).

## 사전 조건

이 데모를 따라 수행하려면 Bluetooth Low Energy 기능을 지원하는 마이크로 컨트롤러가 필요합니다. 또한 [FreeRTOS 디바이스용 iOS SDK \(p. 208\)](#) 또는 [FreeRTOS 블루투스 디바이스용 Android SDK \(p. 207\)](#)이 필요합니다.

## 설정 AWS IoT FreeRTOS 블루투스 저에너지용 Amazon Cognito

디바이스를 에 연결하려면 AWS IoT MQTT 전반에 대해 를 설정해야 합니다.AWS IoT 및 Amazon Cognito.

### AWS IoT 설정

- 를 설정합니다.AWS 계정 켜기|<https://aws.amazon.com/>.
- [AWS IoT 콘솔](#)을 열고 탐색 창에서 관리를 선택한 후 사물을 선택합니다.
- 생성을 선택한 후 단일 사물 생성을 선택합니다.
- 디바이스에 대해 기억하기 쉬운 이름을 입력한 후 다음을 선택합니다.
- 모바일 디바이스를 통해 클라우드에 마이크로 컨트롤러를 연결하는 경우 인증서없이 사물 생성을 선택합니다. 모바일 SDK는 디바이스 인증에 Amazon Cognito 를 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.

- 등록된 사물 목록에서 방금 생성한 사물을 선택한 다음 사물 페이지에서 상호 작용을 선택합니다. AWS IoT REST API 엔드포인트를 기록해 듭니다.

설정에 대한 자세한 내용은 [AWS IoT 시작하기](#)를 참조하십시오.

### Amazon Cognito 사용자 풀을 생성하려면

- Amazon Cognito 콘솔을 열고 를 선택합니다. 사용자 풀 관리.
- [Create a user pool]을 선택합니다.
- 사용자 풀에 이름을 지정한 다음 기본값 검토를 선택합니다.
- 왼쪽 탐색 창에서 앱 클라이언트를 선택한 다음 앱 클라이언트 추가를 선택합니다.
- 앱 클라이언트의 이름을 입력한 후 앱 클라이언트 생성을 선택합니다.
- 왼쪽 탐색 창에서 검토를 선택한 후 풀 생성을 선택합니다.

사용자 풀의 일반 설정 페이지에 나타나는 풀 ID를 기록해 듭니다.

- 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 콘솔을 열고 풀 선택합니다. 자격 증명 풀 관리.
2. 자격 증명 풀의 이름을 입력합니다.
3. 인증 공급자를 확장하고 Cognito 탭을 선택한 후 사용자 풀 ID와 앱 클라이언트 ID를 입력합니다.
4. [Create Pool]을 선택합니다.
5. 세부 정보 보기 확장하고 2개의 IAM 역할 이름을 적어 둡니다. 선택한 IAM 역할은 Amazon Cognito에 액세스하기 위해 인증된 자격 증명 및 인증되지 않은 자격 증명에 대한 IAM 역할을 생성하려면
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 단원을 참조하십시오.[Amazon Cognito 시작하기](#).

### 인증된 자격 증명에 IAM 정책을 생성하여 연결하려면

1. IAM 콘솔을 열고 탐색 창에서 풀 선택합니다. Roles.
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.

유지AWS IoT 및 Amazon Cognito 정보가 수중에 있습니다. AWS 클라우드를 사용하여 모바일 애플리케이션을 인증하려면 앤드포인트와 ID가 필요합니다.

## Bluetooth Low Energy용 FreeRTOS 환경 설정

환경을 설정하려면 다음을 사용하여 FreeRTOS를 다운로드해야 합니다.[Bluetooth Low Energy 라이브러리 \(p. 198\)](#)マイ크로컨트롤러에서 모바일 디바이스에 FreeRTOS Bluetooth 디바이스용 Mobile SDK를 다운로드하고 구성합니다.

FreeRTOS Bluetooth Low Energy로 마이크로 컨트롤러 환경을 설정하려면

1. FreeRTOS 다운로드 또는 복제[GitHub](#). 자세한 내용은 [README.md](#) 파일을 참조하십시오.

## 2. 마이크로컨트롤러에 FreeRTOS를 설정합니다.

Freertos 인증 마이크로 컨트롤러에서 FreeTOS를 시작하는 방법에 대한 자세한 내용은 가이드를 참조하십시오.[FreeRTOS 시작하기](#).

### Note

FreeRTOS 및 이식된 FreeRTOS 블루투스 Low Energy 라이브러리를 지원하는 Bluetooth Low Energy 사용 마이크로 컨트롤러에서 데모를 실행할 수 있습니다. 현재 FreeRTOS MQTT over Bluetooth Low Energy (p. 240) 데모 프로젝트는 다음 Bluetooth Low Energy 사용 디바이스에 완벽하게 이식됩니다.

- Espressif ESP32-DevKitC와 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, FreeRTOS 딜렉터리에 있습니다. `network_manager/aws_iot_network_manager.c`. 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 블루투스 Low Energy 모바일 SDK 데모 애플리케이션

FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션은 GitHub...에서 Bluetooth 디바이스용 [Android SDK](#) 아래에 `amazon-freertos-ble-android-sdk/app` 및 Bluetooth 디바이스용 iOS SDK 아래에 `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo`. 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 디바이스용 iOS SDK \(p. 208\)](#)이 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. 해당 리전에서 AWS IoT 정책을 생성합니다(아직 설정하지 않은 경우).

### Note

이 정책은 Amazon Cognito 인증 자격 증명에 대해 생성된 IAM 정책과 다릅니다.

- a. [AWS IoT 콘솔](#)을 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. Replace `aws-region`과 `aws-account`당신과 함께 AWS 리전 및 계정 ID입니다.

```
{
    "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)를 선택합니다.

5. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift`을 열고 다음 변수를 재정의합니다.
  - `region`: 해당 AWS 리전.
  - `iotPolicyName`: 해당 AWS IoT 정책 이름.
  - `mqttCustomTopic`: 게시하려는 MQTT 주제.
6. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json`을(를) 업니다.

CognitoIdentity 아래에서 다음 변수를 재정의합니다.

- PoolId: Amazon Cognito 자격 증명 풀 ID입니다.
- Region: 해당 AWS 리전.

CognitoUserPool 아래에서 다음 변수를 재정의합니다.

- PoolId: Amazon Cognito 사용자 풀 ID입니다.
- AppClientId: 해당 앱 클라이언트 ID.
- AppClientSecret: 해당 앱 클라이언트 암호.
- Region: 해당 AWS 리전.

### Android SDK 데모 애플리케이션을 구성하려면

구성 변수를 정의할 경우 구성 파일에 제공된 자리 표시자 값의 형식을 사용합니다.

1. [FreeRTOS 블루투스 디바이스용 Android SDK \(p. 207\)](#)이 설치되어 있는지 확인합니다.
2. 해당 리전에서 AWS IoT 정책을 생성합니다(아직 설정하지 않은 경우).

#### Note

이 정책은 Amazon Cognito 인증 자격 증명에 대해 생성된 IAM 정책과 다릅니다.

- a. [AWS IoT 콘솔](#)을 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. Replace **aws-region**과 **aws-account**당신과 함께 AWS 리전 및 계정 ID입니다.

```
{  
    "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)를 선택합니다.

3. Open<https://github.com/aws/amazon-freertos-ble- 안드로이드 SDK/blob/마스터/앱/SRC/메인/자바/소프트웨어/아마존/프리 토스/데모/DemoConstants.java>다음 변수를 재정의합니다.

- AWS\_IOT\_POLICY\_NAME: 해당 AWS IoT 정책 이름.
  - AWS\_IOT\_REGION: 해당 AWS 리전.
4. Open<https://github.com/aws/amazon-freertos-ble-> 안드로이드 SDK/blob/마스터/앱/SRC/메인/입술/생/aws구성.json.

CognitoIdentity 아래에서 다음 변수를 재정의합니다.

- PoolId: Amazon Cognito 자격 증명 풀 ID입니다.
- Region: 해당 AWS 리전.

CognitoUserPool 아래에서 다음 변수를 재정의합니다.

- PoolId: Amazon Cognito 사용자 풀 ID입니다.
- AppClientId: 해당 앱 클라이언트 ID.
- AppClientSecret: 해당 앱 클라이언트 암호.
- Region: 해당 AWS 리전.

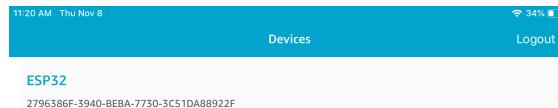
Bluetooth Low Energy를 통해 마이크로 컨트롤러와의 보안 연결을 찾아보고 설정하려면

1. 마이크로컨트롤러와 모바일 디바이스를 안전하게 페어링하려면 (6단계) 입력 및 출력 기능을 모두 갖춘 직렬 터미널 에뮬레이터 (예:TeraTerm). [터미널 에뮬레이터 설치 \(p. 22\)](#)의 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

FreeRTOS가 있는 모든 장치 및 기기 정보 서비스 ([freertos/.../device\\_information](#)) 범위 내에 있는 항목이 목록에 나타납니다.

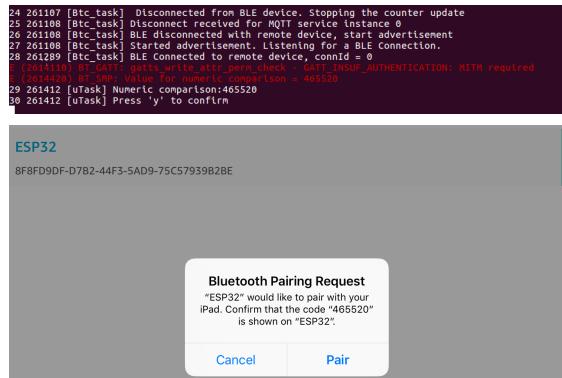
5. 디바이스 목록에서 마이크로 컨트롤러를 선택합니다. 애플리케이션에서 보드와의 연결을 설정하고 연결된 디바이스 옆에 녹색 선이 나타납니다.



선을 왼쪽으로 끌어서 마이크로컨트롤러에서 연결을 해제할 수 있습니다.



#### 6. 메시지가 표시되면 마이크로컨트롤러와 모바일 디바이스를 페어링합니다.



두 디바이스의 수 비교를 위한 코드가 동일한 경우 디바이스를 연결합니다.

#### Note

Bluetooth Low Energy Mobile SDK 데모 애플리케이션에서는 Amazon Cognito를 사용하여 사용자 인증을 Amazon Cognito 사용자 및 자격 증명 풀을 설정하고 IAM 정책을 인증된 자격 증명에 연결 했는지 확인합니다.

## MQTT over Bluetooth Low Energy

MQTT over Bluetooth Low Energy 데모에서 마이크로 컨트롤러는 메시지를 게시합니다.AWSMQTT 프록시를 통한 클라우드

데모 MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 []를 선택합니다. 테스트를 선택하고 MQTT 테스트 클라이언트 클라이언트를 눌러 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 Energy 데모를 활성화하려면

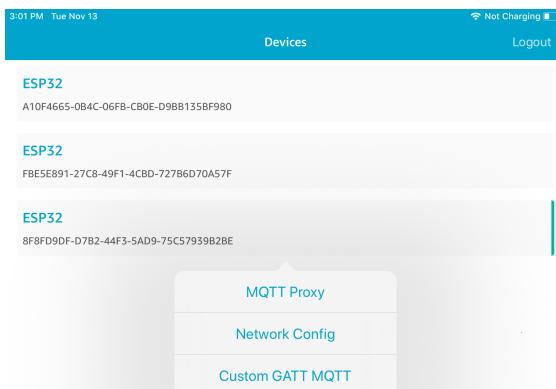
1. `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`를 열고 `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`를 정의합니다.
2. `Opendedemos/include/aws_clientcredential.h`, 및 구성 `clientcredentialMQTT_BROKER_ENDPOINT`와 함께 AWS IoT 브로커 앤드포인트. 구성 `clientcredentialIOT_THING_NAMEBLE` 마이크로 컨트롤러 장치의 이름이 AWS IoT 브로커 앤드포인트는 AWS IoT 콘솔 선택 설정 왼쪽 탐색 창에서 또는 명령을 실행하여 CLI를 통해 다음을 실행합니다.  
`aws iot describe-endpoint --endpoint-type=iot:Data-ATS.`

### Note

이AWS IoT브로커 엔드포인트와 사물 이름은 모두 Cognito ID와 사용자 풀이 구성된 동일한 리전에 있어야 합니다.

### 데모를 실행하려면

1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
2. [FreeRTOS 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 \(p. 236\)](#)을 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 마이크로 컨트롤러로 Wi-Fi 네트워크 자격 증명을 안전하게 전송할 수 있도록 해주는 FreeRTOS Bluetooth Low Energy 서비스입니다. Wi-Fi 프로비저닝 서비스를 위한 소스 코드는 [freertos/.../wifi\\_provisioning](#)에서 찾을 수 있습니다.

### Note

Wi-Fi 프로비저닝 데모는 현재 Espressif ESP32-에서 지원됩니다. DevKitC.

### 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

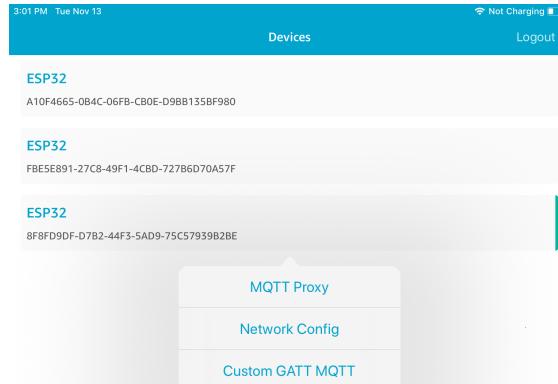
Wi-Fi 프로비저닝 서비스는 기본적으로 비활성화되어 있습니다.

2. Bluetooth Low Energy와 Wi-Fi를 모두 활성화하도록 [Network Manager \(p. 236\)](#)을 구성합니다.

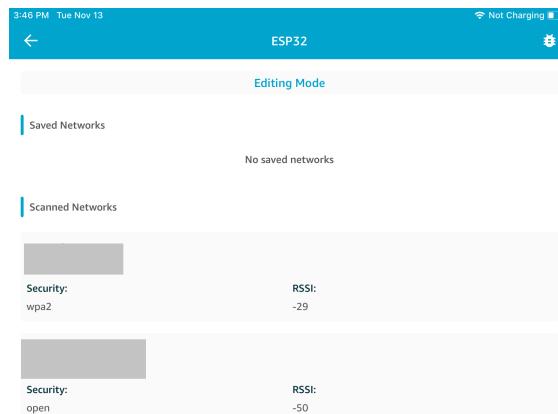
### 데모를 실행하려면

1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.

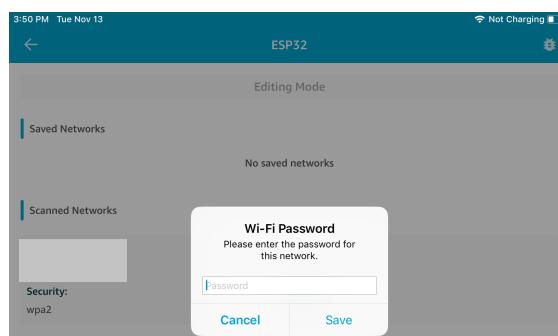
2. 를 사용하여 마이크로 컨트롤러와 모바일 디바이스를 연결했는지 확인합니다.[FreeRTOS 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 \(p. 236\)](#).
3. 데모 모바일 앱의 디바이스 목록에서 마이크로 컨트롤러를 선택한 다음 네트워크 구성을 선택하여 네트워크 구성 설정을 엽니다.



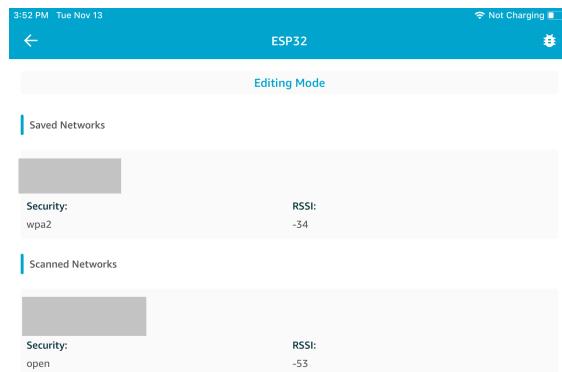
4. 보드에 대한 Network Config(네트워크 구성)를 선택하면 마이크로 컨트롤러는 근처에 있는 네트워크 목록을 모바일 디바이스로 전송합니다. 사용 가능한 Wi-Fi 네트워크가 Scanned Networks(스캔된 네트워크) 아래 목록에 표시됩니다.



Scanned Networks(스캔된 네트워크) 목록에서 네트워크를 선택한 후 필요한 경우 SSID 및 암호를 입력합니다.

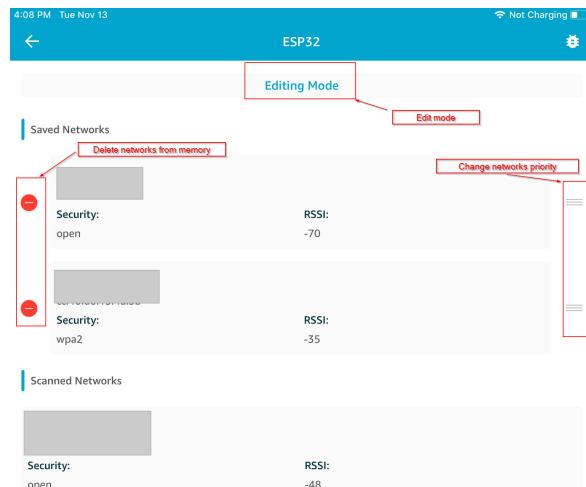


マイクロコントローラーが接続されたネットワークを登録します。登録されたネットワークが Saved Networks(登録済みネットワーク)の下に表示されます。



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 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 \(p. 236\)](#)에 전송합니다.

Bluetooth Low Energy Mobile SDK를 사용하여 마이크로 컨트롤러의 GATT 서버에 연결되고 데모 모바일 애플리케이션과 별별로 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.h`에서(여기서 `vendor`는 공급업체의 이름이고 `board`는 데모를 실행하는 데 사용하는 보드 이름임) `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )`를 정의문 목록에 추가합니다.

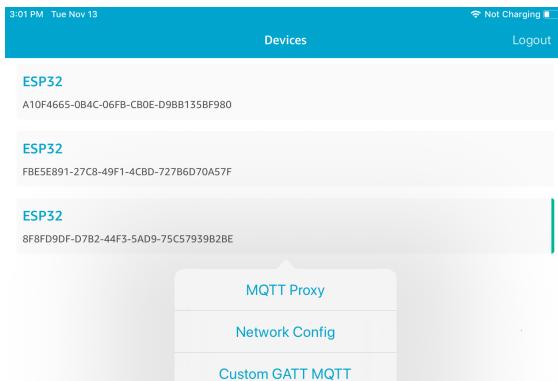
### Note

Bluetooth Low Energy GATT 데모는 기본적으로 비활성화되어 있습니다.

2. `freertos/vendors/vendor/boards/board/aws_demos/config_files/`  
`aws_demo_config.h`를 열고 `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`을 주석으로 처리한 다음 `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`을 정의합니다.

#### 데모를 실행하려면

1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
2. FreeRTOS 블루투스 Low Energy 모바일 SDK 데모 애플리케이션 (p. 236)을 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 종분 카운터 메시지가 `your-thing-name/example/topic` 주제에 나타납니다.

## Microchip Curiosity PIC32MZEF용 데모 부트로더

### Note

마이크로칩과 일치하여 FreeRTOS 레퍼런스 통합 리포지토리 메인 브랜치에서 Curiosity PIC32MZEF (DM320104) 를 제거하고 더 이상 새 릴리스에서 수행하지 않을 것입니다. 마이크로칩은 공식 공지새로운 설계에는 PIC32MZEF (DM320104) 가 더 이상 권장되지 않습니다. PIC32MZEF 프로젝트와 소스 코드는 이전 릴리스 태그를 통해 계속 액세스할 수 있습니다. Microchip은 고객이 호기심을 사용할 것을 권장합니다. [Pic32MZ-EF-2.0 개발 보드 \(DM320209\)](#) 새로운 디자인을 위해 Pic32mzv1 플랫폼은 여전히 에서 찾을 수 있습니다. [v202012.00](#)의 FreeRTOS 참조 통합 저장소를 참조하십시오. 하지만 이 플랫폼은 에서 더 이상 지원되지 않습니다. [v202107.00](#) FreeRTOS 레퍼런스의

이 데모 부트로더는 펌웨어 버전 확인, 암호화 서명 검증, 애플리케이션 셀프 테스트를 수행합니다. 이러한 기능은 FreeRTOS의 OTA (Over-the-Air) 펌웨어 업데이트를 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 유ти리티를 사용하여 디바이스에 플래시할 수 있는 서명된 애플리케이션을 생성할 수 있습니다.

부트로더는 OTA에 필요한 다음 기능을 지원합니다.

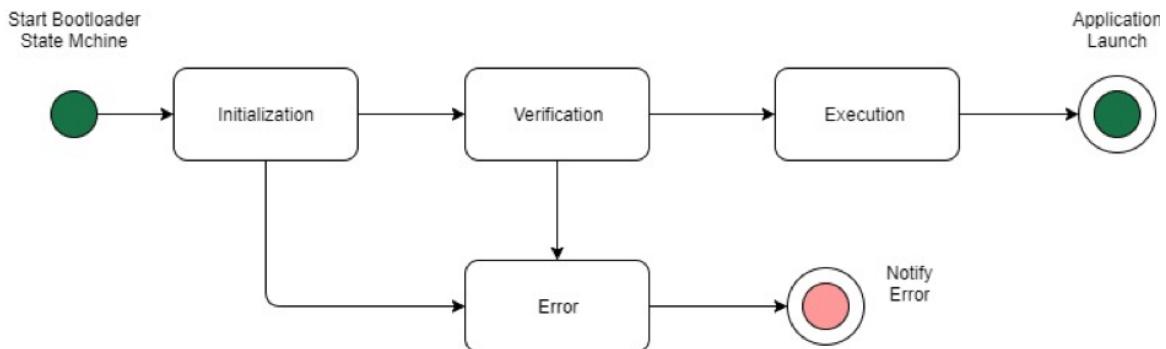
- 디바이스에 애플리케이션 이미지를 유지하여 이미지 간에 전환합니다.
- 수신된 OTA 이미지의 셀프 테스트 실행 및 실패 시 롤백을 허용합니다.
- OTA 업데이트 이미지의 서명과 버전을 확인합니다.

#### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오.[FreeRTOS 시작하기 \(p. 14\)](#).

## 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.
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

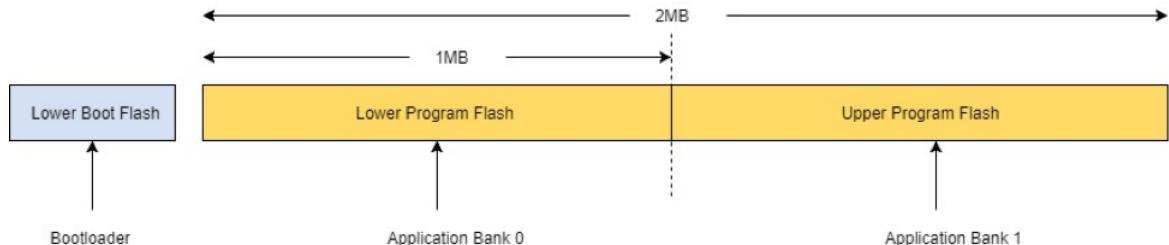
기본 이미지를 실행하는 구성 옵션이 활성화된 경우 부트로더는 기본 실행 주소에서 애플리케이션을 시작합니다. 이 옵션은 디버깅할 때를 제외하고는 비활성화해야 합니다.

#### 부트로더 오류 상태

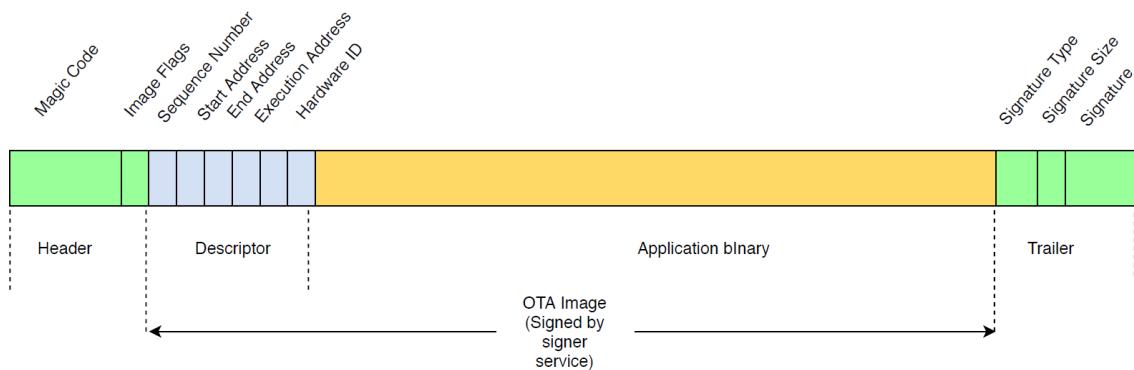
부트로더가 오류 상태이고, 디바이스에 유효한 이미지가 없습니다. 부트로더가 사용자에게 알려야 합니다. 기본 구현은 콘솔에 로그 메시지를 전송하고, 보드의 LED를 계속 빠르게 깜박이는 것입니다.

## 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 이미지를 삭제됩니다.

상태	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.
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 유ти리티를 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. 유효 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 플랫폼으로 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-ecdsa`입니다.

#### 서명 크기

암호화 서명의 크기(바이트)입니다.

#### 서명(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.h`에 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 비활성화 할 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 로드 가능한 프로젝트로 포함되어 있습니다.`aws_demos` 프로젝트 위치 [freertos/vendors/microchip/boards/curiosity\\_pic32mzef/aws\\_demos/mplab/FreeRTOS](#) 소스 코드 리포지토리의 `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` 유ти리티가 제공됩니다. 부트로더 유ти리티 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` 유ти리티 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

### 소개

이 데모는 AWS IoT 연결할 디바이스 디펜더 라이브러리 [AWS IoT Device Defender](#) 데모에서는 CoreMQTT 라이브러리를 사용하여 TLS (상호 인증)를 통해 MQTT 연결을 설정합니다. AWS IoT MQTT 브로커와 CoreJSON 라이브러리에서 수신된 응답을 검증하고 구문 분석합니다. AWS IoT Device Defender 서비스. 이 데모에서는 장치에서 수집된 지표를 사용하여 JSON 형식 보고서를 작성하는 방법과 구성된 보고서를 AWS IoT Device Defender 서비스. 데모에서는 CoreMqtt 라이브러리에 콜백 함수를 등록하여 AWS IoT Device Defender 보낸 보고서가 수락 또는 거부되었는지 여부를 확인하는 서비스입니다.

#### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오. [FreeRTOS 시작하기 \(p. 14\)](#).

## 기능

이 데모에서는 메트릭을 수집하고 JSON 형식으로 장치 수비수 보고서를 작성한 다음 AWS IoT Device Defender보안 MQTT 연결을 통한 서비스 AWS IoT MQTT 브로커. 데모에는 표준 네트워킹 지표와 사용자 지정 지표가 포함되어 있습니다. 사용자 지정 지표의 경우 데모에는 다음이 포함됩니다.

- 이름이 지정된 지표 "task\_numbers" FreeRTOS 작업 ID 목록입니다. 이 지표의 유형은 "숫자 목록"입니다.
- 이름이 지정된 지표 "stack\_high\_water\_mark" 데모 애플리케이션 작업의 스택 하이 워터마크입니다. 이 지표의 유형은 "숫자"입니다.

네트워킹 메트릭을 수집하는 방법은 사용 중인 TCP/IP 스택에 따라 다릅니다. FreeRTOS+TCP 및 지원되는 LWIP 구성의 경우, 디바이스에서 실제 메트릭을 수집하여 해당 메트릭에 제출하는 메트릭 수집 구현을 제공합니다. AWS IoT Device Defender보고서.에 대한 구현을 찾을 수 있습니다. [FreeRTOS+TCP과 LWIP\(GitHub에 있음\)](#).

다른 TCP/IP 스택을 사용하는 보드의 경우 모든 네트워킹 메트릭에 대해 0을 반환하는 메트릭 수집 함수에 대한 스텝 정의를 제공합니다. 예 함수를 구현합니다. [freertos/demos/device\\_defender\\_for\\_aws/metrics\\_collector/stub/metrics\\_collector.c](#) 네트워크 스택에서 실제 메트릭을 전송할 수 있습니다. 파일은 다음에서도 사용할 수 있습니다. [GitHub](#) 웹 사이트.

ESP32의 경우 기본 LWIP 구성은 코어 잠금을 사용하지 않으므로 데모에서는 스텝 메트릭을 사용합니다. 참조 LWIP 지표 수집 구현을 사용하려면 에서 다음 매크로를 정의하십시오. `lwiopts.h`:

```
#define LINK_SPEED_OF_YOUR_NETIF_IN_BPS 0
#define LWIP_TCPIP_CORE_LOCKING 1
#define LWIP_STATS 1
#define MIB2_STATS 1
```

다음은 데모를 실행할 때의 예제 출력입니다.

```
24 1682 [iot_thread] [INFO] MQTT connection successfully established with broker.

25 1682 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.

26 1682 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1682 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1722 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1722 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.

30 1722 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2322 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2322 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2382 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2382 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.

35 2382 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 2982 [iot_thread] [INFO] the published payload:{"ned": {"rid": 1109, "v": "1.0"}, "met": {"tp": {"pts": [{"pt": 33251}], "t": 1}, "up": [{"pts": [], "t": 0}], "ns": {"bi": 0, "bo": 0, "pi": 0, "po": 0}}, "tc": {"ec": {"cs": [{"ip": 33251, "rad": "44.236.152.27:888337 2"}]}}, "ns": {"bi": 0, "bo": 0, "pi": 0, "po": 0}}.37 2982 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.

38 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
39 3102 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
40 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
41 3102 [iot_thread] [INFO] PUBACK received for packet id 3.

42 3102 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.

43 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
44 3102 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSUCCESS.
45 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
46 3502 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.

47 3542 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
48 3542 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.

49 4142 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.

50 4202 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
51 4202 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.

52 4802 [iot_thread] [INFO] Disconnected from the broker.
53 4802 [iot_thread] [INFO] [DEMO][4802] memory_metrics::freertos_heap::before::bytes:::2088152
54 4802 [iot_thread] [INFO] [DEMO][4802] memory_metrics::freertos_heap::after::bytes:::1985556
55 4802 [iot_thread] [INFO] [DEMO][4802] memory_metrics::demo_task_stack::before::bytes:::1908
56 4802 [iot_thread] [INFO] [DEMO][4802] memory_metrics::demo_task_stack::after::bytes:::1908
57 5802 [iot_thread] [INFO] [DEMO][5802] Demo completed successfully.
58 5804 [iot_thread] [INFO] [INIT][5804] SDK cleanup done.
59 5804 [iot_thread] [INFO] [DEMO][5804] -----DEMO FINISHED-----
```

보드가 FreeRTOS+TCP 또는 지원되는 LWIP 구성을 사용하지 않는 경우 출력은 다음과 같습니다.

```

24 1716 [iot_thread] [INFO] MQTT connection successfully established with broker.

25 1716 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.

26 1716 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1716
[iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1756 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1756 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.

30 1756 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2356 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2356
[iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2436 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2436 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.

35 2436 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 3036 [iot_thread] [ERROR] Using stub definition of GetNetworkStats! Please implement for your network stack to get correct metrics.
37 3036 [iot_thread] [ERROR] Using stub definition of GetOpenTcpPorts! Please implement for your network stack to get correct metrics.
38 3036 [iot_thread] [ERROR] Using stub definition of GetOpenUdpPorts! Please implement for your network stack to get correct metrics.
39 3036 [iot_thread] [ERROR] Using stub definition of GetEstablishedConnections! Please implement for your network stack to get correct metrics.
40 3036 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1079,"u": "1.0"}, "met": {"tp": {"pts": [], "t": 0}, "up": {"pts": [], "t": 0}, "ns": {"bi": 0, "bo": 0, "po": 0}, "tc": {"ec": {"cs": [], "t": 0}}}, "cmet": {"stack_high_water_mark": [{"num": 1}], "stack_low_water_mark": [{"num": 1}], "stack_size": 1}, "state": "PUBLISHING", "topic": "$aws/things/DemoThing/defender/metrics/json", "qos": 1, "retain": false, "id": 3}.
41 3036 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.

42 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
43 3196 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
44 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
45 3196 [iot_thread] [INFO] PUBACK received for packet id 3.

46 3196 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.

47 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
48 3196 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
49 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
50 3596 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.

51 3656 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
52 3656 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.

53 4256 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.

54 4336 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
55 4336 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.

56 4936 [iot_thread] [INFO] Disconnected from the broker.
57 4936 [iot_thread] [INFO ] [DEMO][4936] memory_metrics::freertos_heap::before::bytes:::2088152
58 4936 [iot_thread] [INFO ] [DEMO][4936] memory_metrics::freertos_heap::after::bytes:::1985556
59 4936 [iot_thread] [INFO ] [DEMO][4936] memory_metrics::demo_task_stack::before::bytes:::1908
60 4936 [iot_thread] [INFO ] [DEMO][4936] memory_metrics::demo_task_stack::after::bytes:::1908
61 5936 [iot_thread] [INFO ] [DEMO][5936] Demo completed successfully.
62 5938 [iot_thread] [INFO ] [INIT][5938] SDK cleanup done.
63 5938 [iot_thread] [INFO ] [DEMO][5938] -----DEMO FINISHED-----
```

데모의 소스 코드는 다음 주소로 다운로드 중입니다.[freertos/demos/device\\_defender\\_for\\_aws/](https://github.com/FreeRTOS/demos/tree/main/device_defender_for_aws) 디렉토리 또는 [GitHub](https://github.com) 웹 사이트.

## 구독AWS IoT Device Defender 토픽

이 구독자 디펜더 토픽 함수는 게시된 Device Defender 보고서에 대한 응답을 수신할 MQTT 주제를 구독합니다. 매크로를 사용합니다. DEFENDER\_API\_JSON\_ACCEPTED 허용된 장치 수비수 보고서에 대한 응답이 수신되는 주제 문자열을 구성합니다. 매크로를 사용합니다. DEFENDER\_API\_JSON\_REJECTED 거부된 장치 수비수 보고서에 대한 응답을 수신할 주제 문자열을 구성합니다.

## 디바이스 지표 수집

이 [컬렉트장치메트릭](#) 함수는 예 정의된 함수를 사용하여 네트워킹 메트릭을 수집합니다.  
다.metrics\_collector.h. 수집된 메트릭은 송수신된 바이트 수와 패킷 수, 열린 TCP 포트, 열린 UDP 포트 및 설정된 TCP 연결입니다.

## 생성 AWS IoT Device Defender 보고서

이 [생성장치 분석보고서](#) 함수는 예 정의된 함수를 사용하여 장치 수비수 보고서를 생성합니다.  
다.report\_builder.h. 이 함수는 네트워킹 메트릭과 버퍼를 취하여 다음과 같은 형식으로 JSON 문서를 만듭니다. AWS IoT Device Defender 제공된 버퍼에 씁니다.에서 예상되는 JSON 문서의 형식입니다. AWS IoT Device Defender은 예 지정됩니다. [Device-side metrics](#)의 AWS IoT 개발자 안내서.

## 게시 AWS IoT Device Defender 보고서

이 AWS IoT Device Defender 보고서는 JSON 게시를 위한 MQTT 주제에 게시됩니다. AWS IoT Device Defender 보고서. 이 보고서는 매크로를 사용하여 구성됩니다. DEFENDER\_API\_JSON\_PUBLISH이 그림과 같아 [코드 조각](#) GitHub 웹 사이트에 있습니다.

## 응답 처리를 위한 콜백

이 [게시콜백](#) 함수는 들어오는 MQTT 게시 메시지를 처리합니다. 그것은 Defender\_MatchTopic의 API AWS IoT Device Defender 들어오는 MQTT 메시지가 AWS IoT Device Defender 서비스. 메시지가 AWS IoT Device Defender 서비스를 사용하면 수신된 JSON 응답을 구문 분석하고 응답에서 보고서 ID를 추출합니다. 그런 다음 보고서 ID가 보고서에 전송된 ID와 동일한 것으로 확인됩니다.

# AWS IoT Greengrass Discovery 데모 애플리케이션

실행하기 전에 AWS IoT Greengrass FreeRTOS를 위한 Discovery 데모를 설정해야 합니다. AWS IoT Greengrass, 및 AWS IoT. AWS를 설정하려면 [사용자 설정 AWS 계정 및 권한 \(p. 15\)](#)의 지침을 수행합니다. AWS IoT Greengrass를 설정하려면 Greengrass 그룹을 생성한 후 Greengrass 코어를 추가해야 합니다. AWS IoT Greengrass 설정에 대한 자세한 내용은 [AWS IoT Greengrass 시작하기](#)를 참조하십시오.

AWS 및 AWS IoT Greengrass를 설정한 후 AWS IoT Greengrass에 대해 몇 가지 추가 권한을 구성해야 합니다.

### AWS IoT Greengrass 권한을 설정하려면

- 해당 검색으로 이동합니다. [IAM 콘솔](#).
- 탐색 창에서 를 선택합니다. Roles 그런 다음 찾아서 선택합니다. 그린그래스\_ServiceRole.
- 선택정책 연결을 선택합니다. 아마존S3FullAccess과 AWSIoTFullAccess를 선택한 다음 를 선택합니다. 정책 연결.
- [AWS IoT 콘솔](#)로 이동합니다.
- 탐색 창에서 Greengrass를 선택하고 그룹을 선택한 후 앞에서 만든 Greengrass 그룹을 선택합니다.
- 설정을 선택한 후 역할 추가를 선택합니다.
- 선택그린그래스\_ServiceRole를 선택한 다음 를 선택합니다. Save.

이 빠른 연결워크플로우에서 [FreeRTOS 콘솔](#) 보드를 빠르게 연결할 수 있습니다. AWS IoT 데모를 실행하려면 다음을 수행합니다. 현재 다음 보드에서는 FreeRTOS 구성을 사용할 수 없습니다.

- Cypress CYW943907AEVAL1F 개발 키트
- Cypress CYW954907AEVAL1F 개발 키트
- Espressif ESP-WROVER-KIT

- Espressif ESP32-DevKitC
- Nordic nRF52840-DK

보드를 다음 위치에 연결할 수도 있습니다. AWS IoT FreeRTOS 데모를 수동으로 구성하십시오.

### 1. AWS IoT에 MCU 보드 등록 (p. 15)

보드를 등록한 후에는 새로운 Greengrass 정책을 생성하여 디바이스 인증서에 연결해야 합니다.

새로운 AWS IoT Greengrass 정책을 만들려면

1. [AWS IoT 콘솔](#)로 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 대한 AWS IoT Greengrass 권한을 부여합니다.

5. 생성(Create)를 선택합니다.

AWS IoT Greengrass 정책을 디바이스의 인증서에 연결하려면

1. [AWS IoT 콘솔](#)로 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 (p. 17)

#### Note

FreeRTOS 콘솔에서 FreeRTOS를 다운로드하는 경우에 연결 AWS IoT Greengrass-### 그 대신에 연결 AWS IoT-###.

### 3. FreeRTOS 데모 구성 (p. 18).

`freertos/vendors/vendor/boards/board/aws_demos/config_files/`  
aws\_demo\_config.h를 열고 #define CONFIG\_CORE\_MQTT\_MUTUAL\_AUTH\_DEMO\_ENABLED를 주석으로 처리한 다음 CONFIG\_GREENGRASS\_DISCOVERY\_DEMO\_ENABLED를 정의합니다.

설정 후 AWS IoT와 AWS IoT Greengrass 및 FreeRTOS를 다운로드하고 구성하면 디바이스에서 Greengrass 를 빌드, 플래시 및 실행할 수 있습니다. 보드의 하드웨어 및 소프트웨어 개발 환경을 설정하려면 [보드별 시작 안내서 \(p. 31\)](#)의 지침을 수행해야 합니다.

Greengrass 데모는 코어에 Greengrass 코어 및 AWS IoT MQTT 클라이언트에 일련의 메시지를 게시합니다. 에서 메시지를 보려면 AWS IoT MQTT 클라이언트, [AWS IoT 콘솔](#), 선택 테스트, 선택 MQTT 테스트 클라이언트 다음 구독을 다음에 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) 를 찾아 Amazon EC2 콘솔로 이동한 다음 왼쪽 탐색 패널에서 를 선택합니다. Instances. Amazon EC2 인스턴스를 선택한 다음 를 선택합니다. 설명패널. Public DNS (IPv4)에 대한 항목을 찾아 기록해 둡니다.
2. 다음에 대한 항목 찾기보안 그룹Amazon 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
사용자 지정 TCP	TCP	8883	::/0	MQTT 커뮤니케이션
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::/0	-
모든 ICMP - IPv4	ICMP	모두	0.0.0.0/0	-
모든 ICMP - IPv4	ICMP	모두	::/0	-

4. AWS IoT 콘솔에서 Greengrass를 선택하고 Groups을 선택한 다음 앞에서 만든 Greengrass 그룹을 선택합니다. [Settings]를 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 구독을 추가합니다. Greengrass 그룹페이지를 선택합니다. 구독그린 다음 선택하십시오구독 추가여기에 나온 것처럼 정보를 입력합니다.

## 구독

소스	대상	주제
TIGG1	IoT Cloud	freertos/demos/ggd

여기서 “Source”는 AWS IoT에서 만들어진 것 AWS IoT보드를 등록했을 때 콘솔 - 여기에 제공된 예제에서 “TIGG1”입니다.

9. AWS IoT Greengrass 그룹 배포를 시작하고 배포가 성공적인지 확인합니다. 이제 AWS IoT Greengrass 검색 데모를 성공적으로 실행할 수 있어야 합니다.

# CoreHttp 데모

이 데모는 CoreHTTP 라이브러리를 사용하는 방법에 대해 알아보는 데 유용하게 활용할 수 있습니다.

## 주제

- [CoreHTTP 상호 인증 데모 \(p. 256\)](#)
- [코어HTTP 기본 Amazon S3 업로드 데모 \(p. 257\)](#)
- [CoreHttp 기본 S3 다운로드 데모 \(p. 259\)](#)
- [코어HTTP 기본 멀티스레드 데모 \(p. 261\)](#)

## CoreHTTP 상호 인증 데모

### 소개

CoreHTTP (상호 인증) 데모 프로젝트는 클라이언트와 서버 간의 상호 인증을 통해 TLS를 사용하여 HTTP 서버에 연결하는 방법을 보여줍니다. 이 데모에서는 MBETTLS 기반 전송 인터페이스 구현을 사용하여 서버 및 클라이언트 인증된 TLS 연결을 설정하고 HTTP에서 요청 응답 워크플로를 보여줍니다.

#### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오.[FreeRTOS 시작하기 \(p. 14\)](#).

### 기능

이 데모에서는 다음을 완료하는 방법을 보여주는 예제를 사용하여 단일 응용 프로그램 작업을 만듭니다.

- 에서 HTTP 서버에 Connect.AWS IoT 앤드포인트.
- POST 요청을 전송합니다.
- 응답을 수신합니다.
- 서버에서 연결을 해제합니다.

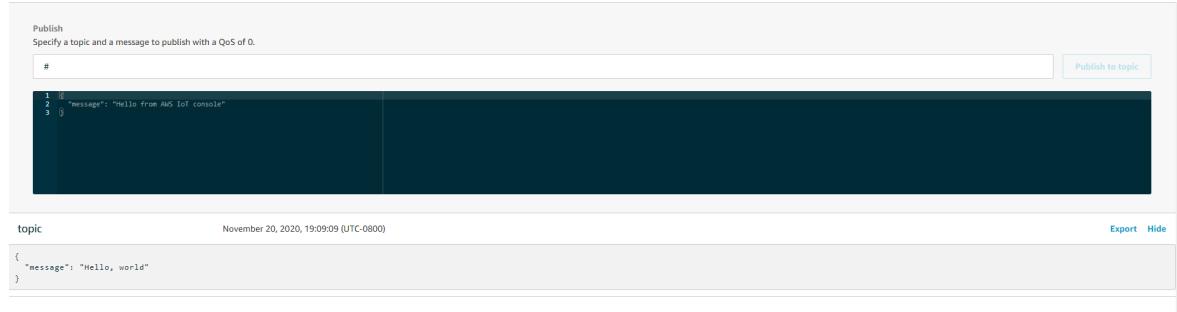
이 단계를 완료하면 데모에서는 다음 스크린샷과 비슷한 출력이 생성됩니다.

```

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:119] 13 1566 [iot_thread] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8443.14 1566 [iot_thread]
13 1622 [iot_thread] DNS[0x60F5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (35.166.102.88) will be stored
14 1622 [iot_thread] DNS[0x60F5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.238.184.132) will be stored
15 1622 [iot_thread] DNS[0x60F5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.238.184.133) will be stored
16 1622 [iot_thread] DNS[0x60F5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.238.184.134) will be stored
17 1622 [iot_thread] DNS[0x60F5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (52.32.144.134) will be stored
18 1622 [iot_thread] DNS[0x60F5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.238.184.135) will be stored
19 1622 [iot_thread] DNS[0x60F5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.238.184.136) will be stored
20 1622 [iot_thread] DNS[0x60F5]: 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] Demo completed successfully.29 2082 [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] [INFO ][HTTPDemo] [http_demo_mutual_auth.c:418] 27 2082 [iot_thread] [INFO ][HTTPDemo] [http_demo_mutual_auth.c:418] 28 2082 [iot_thread] Demo completed successfully.29 2082 [iot_thread]
28 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 2082 [iot_thread] [INFO ][DEMO][2082] Demo completed successfully.
35 3084 [iot_thread] [INFO ][INIT][3084] SDK cleanup done.
36 3084 [iot_thread] [INFO ][DEMO][3084] -----DEMO FINISHED-----

```

이 AWS IoTconsole 에서는 다음 스크린샷과 비슷한 출력을 생성합니다.



## 소스 코드 구성

데모 소스 파일의 이름이 지정되었습니다.`http_demo_mutual_auth.c`에서 찾을 수 있습니다.  
다.[freertos/demos/coreHTTP/디렉토리](#) 및 [GitHub](#) 웹 사이트.

## 에 연결 AWS IoTHTTP 서버

이 연결서버와 백오피 시도 함수가 상호 인증된 TLS 연결을 시도합니다. AWS IoTHTTP 서버. 연결이 실패하면 시간 초과 후에 다시 시도합니다. 제한 시간 값은 최대 시도 횟수에 도달하거나 최대 시간 초과 값에 도달할 때까지 기하급수적으로 증가합니다. 이 `RetryUtils_BackoffAndSleep` 함수는 기하 급수적으로 증가하는 시간 초과 값과 반환을 제공합니다. `RetryUtilsRetriesExhausted` 최대 시도 횟수에 도달한 경우입니다. `connectToServerWithBackoffRetries` 구성된 시도 횟수 후에 브로커에 대한 TLS 연결을 설정할 수 없는 경우 함수는 실패 상태를 반환합니다.

## HTTP 요청 전송 및 응답 수신

이 `PRVSENDHTT` 프리퀘스트 함수는 POST 요청을 AWS IoTHTTP 서버.에서 REST API에 요청하는 방법에 대한 자세한 내용은 AWS IoT를 참조하십시오. [디바이스 통신 프로토콜 - HTTPS](#). 동일한 CoreHTTP API 호출로 응답이 수신됩니다. `HTTPClient_Send`.

## 코어HTTP 기본 Amazon S3 업로드 데모

### 소개

이 예에서는 Amazon Simple Storage Service (Amazon S3) HTTP 서버에 PUT 요청을 보내고 작은 파일을 업로드하는 방법을 보여 줍니다. 업로드 후 파일의 크기를 확인하기 위한 GET 요청도 수행합니다. 이 예제에서는 를 사용합니다. 네트워크 전송 인터페이스 이는 mbedTLS 사용하여 CoreHTTP를 실행하는 IoT 디바이스 클라이언트와 Amazon S3 HTTP 서버 간에 상호 인증된 연결을 설정합니다.

#### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오. [FreeRTOS 시작하기 \(p. 14\)](#).

## 단일 스레드 대 다중 스레드

CoreHTTP 사용 모델은 두 가지가 있습니다. 단일 스레드과 멀티스레딩(멀티태스킹). 이 섹션의 데모에서는 스레드에서 HTTP 라이브러리를 실행하지만 실제로는 단일 스레드 환경에서 CoreHttp를 사용하는 방법을 보여줍니다. 이 데모에서는 하나의 작업만 HTTP API를 사용합니다. 단일 스레드 응용 프로그램이 HTTP 라이브러리를 반복적으로 호출해야 하지만, 멀티스레드 응용 프로그램은 대신 에이전트 (또는 데몬) 작업 내에서 백그라운드에서 HTTP 요청을 보낼 수 있습니다.

## 소스 코드 구성

데모 소스 파일의 이름이 지정되었습니다. `http_demo_s3_upload.c`에서 찾을 수 있습니다. [freertos/demos/coreHTTP/디렉토리](#) 및 [GitHub 웹 사이트](#).

## Amazon S3 HTTP 서버 연결 구성

이 데모에서는 미리 서명된 URL을 사용하여 Amazon S3 HTTP 서버에 연결하고 다운로드할 객체에 대한 액세스 권한을 부여합니다. Amazon S3 HTTP 서버의 TLS 연결은 서버 인증만 사용합니다. 응용 프로그램 수준에서 객체에 대한 액세스는 사전 서명된 URL 쿼리의 매개 변수를 사용하여 인증됩니다. 아래 단계에 따라 연결을 구성합니다. AWS.

1. 를 설정합니다. AWS 계정:
  - a. 아직 설정하지 않았다면 [AWS 계정 생성](#).
  - b. 계정 및 권한은 다음을 사용하여 설정합니다. AWS Identity and Access Management(스피). IAM을 사용하여 계정의 각 사용자에 대한 권한을 관리합니다. 기본적으로 사용자는 루트 소유자가 부여할 때까지 권한이 없습니다.
    - i. 에 IAM 사용자를 추가하려면 [AWS 계정 참조](#).
    - ii. 사용자에게 권한 부여 AWS FreeRTOS에 액세스할 수 있는 계정 AWS IoT 정책을 추가하여 다음을 수행합니다.
      - `AmazonS3FullAccess`
2. 다음 단계에 따라 Amazon S3 버킷을 생성합니다. [S3 버킷을 생성하려면 어떻게 해야 합니까?](#) 의 Amazon 심플 스토리지 서비스 사용 설명서.
3. 의 단계에 따라 Amazon S3 파일을 업로드합니다. [S3 버킷에 파일 및 폴더를 업로드하려면 어떻게 해야 합니까?](#).
4. 에 있는 [스크립트](#)를 사용하여 미리 서명된 URL을 생성합니다. `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 파일.

사용 지침은 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md` 파일.

## 기능

데모는 먼저 TLS 서버 인증을 사용하여 Amazon S3 HTTP 서버에 연결합니다. 그런 다음 에 지정된 데이터를 업로드하는 HTTP 요청을 만듭니다. `democonfigDEMO_HTTP_UPLOAD_DATA`. 파일을 업로드한 후 파일 크기를 요청하여 파일이 성공적으로 업로드되었는지 확인합니다. 데모의 소스 코드는 다음에서 확인할 수 있습니다. [GitHub 웹 사이트](#).

## Amazon S3 HTTP 서버에 연결

이 [연결서버와 백오프 시도](#) 함수가 HTTP 서버에 대한 TCP 연결을 시도합니다. 연결이 실패하면 시간 초과 후에 다시 시도합니다. 제한 시간 값은 최대 시도 횟수에 도달하거나 최대 시간 초과 값에 도달할 때까지 기하급수적으로 증가합니다. 이 `connectToServerWithBackoffRetries` 구성된 시도 횟수 후에 서버에 대한 TCP 연결을 설정할 수 없는 경우 함수는 실패 상태를 반환합니다.

0|prvConnectToServer함수는 서버 인증만 사용하여 Amazon S3 HTTP 서버에 대한 연결을 설정하는 방법을 보여 줍니다. MbedTLS 기반 전송 인터페이스를 사용합니다.[FreeRTOS-Plus/Source/Application-Protocols/network\\_transport/freertos\\_plus\\_tcp/using\\_mbedtls/using\\_mbedtls.c](#)파일. 정의 prvConnectToServer에서 찾을 수 있습니다.[GitHub](#) 웹 사이트.

## 데이터 업로드

0|prvUploadS3ObjectFile함수는 PUT 요청을 생성하고 업로드할 파일을 지정하는 방법을 보여 줍니다. 파일이 업로드되는 Amazon S3 버킷과 업로드할 파일의 이름은 미리 서명된 URL에 지정됩니다. 메모리를 절약하기 위해 요청 헤더와 응답을 수신하는 데 동일한 버퍼가 사용됩니다. 응답은 다음을 사용하여 동기적으로 수신됩니다.[HTTPClient\\_SendAPI](#) 함수입니다. A200 OK응답 상태 코드는 Amazon S3 HTTP 서버에서 예상됩니다. 다른 상태 코드는 오류입니다.

소스 코드입니다.[prvUploadS3ObjectFile\(\)](#)에서 찾을 수 있습니다.[GitHub](#) 웹 사이트.

## 업로드 확인

0|prvVerifyS3ObjectFileSize함수 호출[prvGetS3ObjectFileSize](#)S3 버킷에 있는 객체의 크기를 검색합니다. Amazon S3 HTTP 서버는 현재 미리 서명된 URL을 사용하는 HEAD 요청을 지원하지 않으므로 0번째 바이트가 요청됩니다. 파일의 크기가 응답에 포함되어 있습니다.[Content-Range](#) 헤더 필드. A206 Partial Content서버에서 응답이 예상됩니다. 다른 응답 상태 코드는 오류입니다.

소스 코드입니다.[prvGetS3ObjectFileSize\(\)](#)에서 찾을 수 있습니다.[GitHub](#) 웹 사이트.

## CoreHttp 기본 S3 다운로드 데모

### 소개

이 데모에서는 사용 방법을 보여 줍니다. 범위 요청Amazon S3 HTTP 서버에서 파일을 다운로드합니다. 범위 요청은 사용할 때 CoreHTTP API에서 기본적으로 지원됩니다.[HTTPClient\\_AddRangeHeader](#)HTTP 요청을 생성합니다. 마이크로컨트롤러 환경의 경우 범위 요청을 적극 권장합니다. 단일 요청 대신 별도의 범위로 큰 파일을 다운로드하면 네트워크 소켓을 차단하지 않고도 파일의 각 섹션을 처리할 수 있습니다. 범위 요청은 TCP 연결에서 재전송해야 하는 패킷이 떨어질 위험을 낮추어 장치의 전력 소비를 개선합니다.

이 예제에서는 를 사용합니다.[네트워크 전송 인터페이스](#)이는 mbedTLS 사용하여 CoreHTTP를 실행하는 IoT 디바이스 클라이언트와 Amazon S3 HTTP 서버 간에 상호 인증된 연결을 설정합니다.

#### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오.[FreeRTOS 시작하기 \(p. 14\)](#).

### 단일 스레드 대 다중 스레드

CoreHTTP 사용 모델은 두 가지가 있습니다. 단일 스레드과 멀티스레딩(멀티태스킹). 이 섹션의 데모는 스레드에서 HTTP 라이브러리를 실행하지만 실제로는 단일 스레드 환경에서 CoreHTTP를 사용하는 방법을 보여 줍니다 (데모에서 HTTP API를 사용하는 하나의 작업만 사용). 단일 스레드 응용 프로그램이 HTTP 라이브러리를 반복적으로 호출해야 하지만, 멀티스레드 응용 프로그램은 대신 에이전트 (또는 데몬) 작업 내에서 백그라운드에서 HTTP 요청을 보낼 수 있습니다.

## 소스 코드 구성

데모 프로젝트의 이름이 지정되었습니다.[http\\_demo\\_s3\\_download.c](#)에서 찾을 수 있습니다.[freertos/demos/coreHTTP/디렉토리](#) 및 [GitHub](#) 웹 사이트.

## Amazon S3 HTTP 서버 연결 구성

이 데모에서는 미리 서명된 URL을 사용하여 Amazon S3 HTTP 서버에 연결하고 다운로드할 객체에 대한 액세스 권한을 부여합니다. Amazon S3 HTTP 서버의 TLS 연결은 서버 인증만 사용합니다. 응용 프로그램 수

준에서 객체에 대한 액세스는 사전 서명된 URL 쿼리의 매개 변수를 사용하여 인증됩니다. 아래 단계에 따라 연결을 구성합니다.AWS.

1. 를 설정합니다.AWS계정:
  - a. 아직 설정하지 않았다면[생성 및 활성화AWS계정](#).
  - b. 계정 및 권한은 다음을 사용하여 설정됩니다.AWS Identity and Access Management(스피). IAM을 사용하면 계정의 각 사용자에 대한 권한을 관리할 수 있습니다. 기본적으로 사용자는 루트 소유자가 부여할 때까지 권한이 없습니다.
    - i. 에 IAM 사용자를 추가하려면[AWS계정](#), 참조[IAM 사용 설명서](#).
    - ii. 사용자에게 권한 부여AWSFreeRTOS에 액세스할 수 있는 계정AWS IoT다음 정책을 추가하여 다음을 수행합니다.
      - AmazonS3FullAccess
2. 다음 단계에 따라 S3에서 버킷을 생성합니다.[S3 버킷을 생성하려면 어떻게 해야 하나요?](#)의Amazon 심플 스토리지 서비스 콘솔 사용 설명서.
3. 다음 단계에 따라 S3에 파일을 업로드합니다.[S3 버킷에 파일 및 폴더를 업로드하려면 어떻게 해야 합니까?](#).
4. 에 있는 스크립트를 사용하여 미리 서명된 URL을 생성합니다.[FreeRTOS-Plus/Demo/coreHTTP\\_Windows\\_Simulator/Common/presigned\\_url\\_generator/presigned\\_urls\\_gen.py](#). 사용 지침은 단원을 참조하십시오.[FreeRTOS-Plus/Demo/coreHTTP\\_Windows\\_Simulator/Common/presigned\\_url\\_generator/README.md](#).

## 기능

데모에서는 먼저 파일의 크기를 검색합니다. 그런 다음 범위 크기를 사용하여 루프에서 각 바이트 범위를 순차적으로 요청합니다.`democonfigRANGE_REQUEST_LENGTH`.

데모 소스 코드는 다음 사이트에서 확인할 수 있습니다.[GitHub](#)웹 사이트.

## Amazon S3 HTTP 서버에 연결

함수[연결서버와 백오픈 시도 \(\)](#)HTTP 서버에 대한 TCP 연결을 시도합니다. 연결이 실패하면 시간 초과 후에 다시 시도합니다. 제한 시간 값은 최대 시도 횟수에 도달하거나 최대 시간 초과 값에 도달할 때까지 기하급수적으로 증가합니다.`connectToServerWithBackoffRetries()`구성된 시도 횟수 후에 서버에 대한 TCP 연결을 설정할 수 없는 경우 실패 상태를 반환합니다.

함수[prvConnectToServer\(\)](#)에서는 서버 인증만 사용하여 Amazon S3 HTTP 서버에 대한 연결을 설정하는 방법을 보여 줍니다. 파일에 구현된 MBEDTLS 기반 전송 인터페이스를 사용합니다.[프리어토스 플러스/소스/애플리케이션 프로토콜/네트워크\\_전송/프리어토스\\_플러스\\_TCP/사용\\_mbedtls/사용\\_MBEDTLS.C](#).

소스 코드입니다.`prvConnectToServer()`에서 찾아볼 수 있습니다.[GitHub](#).

## 범위 요청 생성

API 함수[HTTPClient\\_AddRangeHeader\(\)](#)에서는 HTTP 요청 헤더에 바이트 범위를 직렬화하여 범위 요청을 형성합니다. 범위 요청은 이 데모에서 파일 크기를 검색하고 파일의 각 섹션을 요청하는 데 사용됩니다.

함수[prvGetS3ObjectFileSize\(\)](#)는 S3 버킷에서 파일의 크기를 검색합니다. `Connection: keep-alive`응답이 전송된 후 연결을 열어 두기 위해 이 첫 번째 요청에 헤더가 Amazon S3 추가됩니다. S3 HTTP 서버는 현재 미리 서명된 URL을 사용하는 HEAD 요청을 지원하지 않으므로 0번째 바이트가 요청됩니다. 파일의 크기가 응답에 포함되어 있습니다.`Content-Range`헤더 필드. `A206 Partial Content`서버에서 응답이 필요합니다. 수신된 다른 응답 상태 코드는 오류입니다.

소스 코드입니다.`prvGetS3ObjectFileSize()`에서 찾아볼 수 있습니다.[GitHub](#).

파일 크기를 검색한 후 이 데모에서는 다운로드할 파일의 각 바이트 범위에 대해 새 범위 요청을 만듭니다. 이를 사용합니다. `HTTPClient_AddRangeHeader()` 파일의 각 섹션에 대해.

## 범위 요청 전송 및 응답 수신

함수 `prvDownloadS3ObjectFile()` 전체 파일이 다운로드될 때까지 범위 요청을 루프에 보냅니다. API 함수 `HTTPClient_Send()` 요청을 보내고 응답을 동기적으로 수신합니다. 함수가 반환되면 응답은 `xResponse`. 그런 다음 상태 코드가 다음과 같이 확인됩니다. 206 Partial Content. 지금까지 다운로드된 바이트 수는 `Content-Length` 헤더 값입니다.

소스 코드입니다. `prvDownloadS3ObjectFile()`에서 찾아볼 수 있습니다. [GitHub](#).

# 코어HTTP 기본 멀티스레드 데모

## 소개

이 데모에서는 FreeRTOS의 스레드 안전 대기열 처리되기를 기다리는 요청과 응답을 보유합니다. 이 데모에서는 세 가지 작업을 기록해 두어야 합니다.

- 기본 작업은 요청 대기열에 요청이 표시될 때까지 기다립니다. 네트워크를 통해 이러한 요청을 보낸 다음 응답을 응답 대기열에 배치합니다.
- 요청 작업은 HTTP 라이브러리 요청 객체를 만들어 서버로 전송하여 요청 대기열에 배치합니다. 각 요청 객체는 애플리케이션이 다운로드하도록 구성한 S3 파일의 바이트 범위를 지정합니다.
- 응답 작업은 응답이 응답 대기열에 표시될 때까지 기다립니다. 수신하는 모든 응답을 기록합니다.

이 기본 멀티스레드 데모는 서버 인증에서만 TLS 연결을 사용하도록 구성되며, 이는 Amazon S3 HTTP 서버에서 필요합니다. 응용 프로그램 계층 인증은 다음을 사용하여 수행합니다. 서명 버전 4의 매개 변수 [미리 서명된 URL 쿼리](#).

## 소스 코드 구성

데모 프로젝트의 이름이 지정되었습니다. `http_demo_s3_download_multithreaded.c`에서 찾을 수 있습니다. [freertos/demos/coreHTTP/](#) 디렉토리 및 [GitHub](#) 웹 사이트.

## 데모 프로젝트 빌드

데모 프로젝트는 [비주얼 스튜디오 무료 커뮤니티 에디션](#). 데모를 빌드하려면 다음을 수행합니다.

- 열기 `mqtt_multitask_demo.sln` 비주얼 스튜디오 IDE에서 제공하는 Visual Studio 솔루션 파일입니다.
- Select 솔루션 구축 IDE에서 빌드 메뉴.

### Note

Microsoft Visual Studio 2017 또는 이전 버전을 사용 중인 경우 플랫폼 툴셋 사용 중인 버전과 호환 가능: 프로젝트 -> RTOSDemos 속성 -> 플랫폼 도구 세트.

## 데모 프로젝트 구성

데모에서는 FreeRTOS+TCP TCP/IP 스택에 제공된 지침을 따릅시오. [TCP/IP 스타터 프로젝트 CNAME](#):

- 설치 사전 요구 사항 구성 요소(예: WinPCAP).
- 선택적으로 정적 또는 동적 IP 주소, 게이트웨이 주소 및 네트워크 설정.
- 선택적으로 MAC 주소 설정.

4. 이더넷 네트워크 인터페이스 선택 호스트 컴퓨터에서
5. 중요한 것은 네트워크 연결 테스트 HTTP 데모를 실행하기 전에

## Amazon S3 HTTP 서버 연결 구성

다음 안내를 따릅니다. [Amazon S3 HTTP 서버 연결 구성 \(p. 259\)](#)의 코어 HTTP 기본 다운로드 데모.

### 기능

데모에서는 총 세 가지 작업을 생성합니다.

- 네트워크를 통해 요청을 보내고 응답을 받는 것입니다.
- 전송 요청을 생성하는 것입니다.
- 수신된 응답을 처리하는 응답입니다.

이 데모에서 기본 작업은 다음과 같습니다.

1. 요청 및 응답 대기열을 생성합니다.
2. 서버에 대한 연결을 만듭니다.
3. 요청 및 응답 작업을 만듭니다.
4. 요청 대기열이 네트워크를 통해 요청을 보낼 때까지 기다립니다.
5. 네트워크를 통해 수신된 응답을 응답 대기열에 배치합니다.

요청 태스크:

1. 각 범위 요청을 작성합니다.

응답 태스크:

1. 수신된 각 응답을 처리합니다.

## 타입 디프

데모에서는 멀티스레딩을 지원하기 위해 다음과 같은 구조를 정의합니다.

요청 항목

다음 구조는 요청 대기열에 배치할 요청 항목을 정의합니다. 요청 작업이 HTTP 요청을 생성한 후 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 구조에서는 응답 대기열에 배치할 응답 항목을 정의합니다. 기본 HTTP 작업이 네트워크를 통해 응답을 수신한 후 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;
```

## 기본 HTTP 전송 태스크

주요 응용 프로그램 작업:

1. 호스트 주소에 대해 미리 서명된 URL을 구문 분석하여 Amazon S3 HTTP 서버와의 연결을 설정합니다.
2. S3 버킷에 있는 객체에 대한 경로에 대해 미리 서명된 URL을 구문 분석합니다.
3. 서버 인증을 통해 TLS를 사용하여 Amazon S3 HTTP 서버에 연결합니다.
4. 요청 및 응답 대기열을 생성합니다.
5. 요청 및 응답 작업을 만듭니다.

함수 `prvHTTPDemoTask()`이 설정을 수행하고 데모 상태를 제공합니다. 이 함수의 소스 코드는 에서 찾을 수 있습니다.[GitHub](#).

함수에서 수행 `prvDownloadLoop()` 기본 작업은 요청 대기열의 요청을 차단하고 대기합니다. 요청을 받으면 API 함수를 사용하여 요청을 보냅니다. `HTTPClient_Send()`. API 함수가 성공하면 응답을 응답 대기열에 배치합니다.

에 대한 소스 코드 `prvDownloadLoop()`에서 찾아볼 수 있습니다.[GitHub](#).

## HTTP 요청 작업

요청 작업은 함수에 지정되어 있습니다. `prvRequestTask`. 이 함수의 소스 코드는 에서 찾을 수 있습니다.[GitHub](#).

요청 작업은 Amazon S3 버킷에 있는 파일의 크기를 검색합니다. 이는 함수에서 수행됩니다. `prvGetS3ObjectFileSize`. “Connection: keep-alive” 헤더가 이 요청에 Amazon S3 추가되어 응답이 전송된 후 연결을 열어 둡니다. Amazon S3 HTTP 서버는 현재 미리 서명된 URL을 사용하는 HEAD 요청을 지원하지 않으므로 0번째 바이트가 요청됩니다. 파일의 크기는 응답에 포함되어 있습니다. `Content-Range` 헤더 필드. A206 Partial Content 서버에서 응답이 필요합니다. 수신된 다른 응답 상태 코드는 오류입니다.

소스 코드입니다. `prvGetS3ObjectFileSize`에서 찾아볼 수 있습니다.[GitHub](#).

파일 크기를 검색한 후 요청 작업은 파일의 각 범위를 계속 요청합니다. 각 범위 요청은 기본 작업이 보낼 요청 대기열에 배치됩니다. 파일 범위는 데모 사용자가 매크로에서 구성합니다. `democonfigRANGE_REQUEST_LENGTH`. 범위 요청은 함수를 사용하여 HTTP 클라이언트 라이브러리 API에서 기본적으로 지원됩니다. `HTTPClient_AddRangeHeader`. 함수 `prvRequestS3ObjectRange` 사용 방법을 설명합니다. `HTTPClient_AddRangeHeader()`.

함수의 소스 코드입니다.`prvRequestS3ObjectRange`에서 찾아볼 수 있습니다.[GitHub](#).

## HTTP 응답 작업

응답 작업은 네트워크를 통해 수신된 응답을 응답 대기열에서 대기합니다. 기본 작업은 HTTP 응답을 성공적으로 수신하면 응답 대기열을 채웁니다. 이 작업은 상태 코드, 헤더 및 본문을 로깅하여 응답을 처리합니다. 예를 들어, 실제 응용 프로그램은 플래시 메모리에 응답 본문을 작성하여 응답을 처리할 수 있습니다. 응답 상태 코드가 아닌 경우 `206 partial content`라면 작업이 기본 작업에 데모가 실패해야 함을 알립니다. 응답 태스크는 함수에 지정되어 있습니다.`prvResponseTask`. 이 함수의 소스 코드는 에서 찾을 수 있습니다.[GitHub](#).

# AWS IoT작업 라이브러리

## 소개

이 AWS IoTJobs 라이브러리 데모에 연결하는 방법을 보여줍니다. [AWS IoT작업 서비스 MQTT 연결을 통해](#)에서 작업 검색 AWS IoT를 (를) 디바이스에서 처리합니다. 이 AWS IoTJobs 데모 프로젝트는 [FreeRTOS Windows 포트](#) 따라서 다음을 사용하여 구축하고 평가할 수 있습니다. [Visual Studio Windows](#)에서 버전. 마이크로컨트롤러 하드웨어가 필요하지 않습니다. 데모는 다음과 같은 보안 연결을 설정합니다. AWS IoTMQTT 브로커와 동일한 방식으로 TLS를 사용합니다. [CoreMQTT 상호 인증 데모 \(p. 266\)](#).

### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오. [FreeRTOS 시작하기 \(p. 14\)](#).

## 소스 코드 구성

데모 코드는 `jobs_demo.c` 파일 및 에서 찾을 수 있습니다. [GitHub](#) 웹 사이트 또는 `freertos/demos/jobs_for_aws/` 디렉터리.

## 구성 AWS IoTMQTT 브로커 연결

이 데모에서는 MQTT 연결을 AWS IoTMQTT 브로커. 이 연결은 [CoreMQTT 상호 인증 데모 \(p. 266\)](#).

## 기능

데모에서는 작업을 수신하는 데 사용되는 워크플로우를 보여 줍니다. AWS IoT 장치에서 처리할 수 있습니다. 데모는 대화식이며 다음 중 하나를 사용하여 작업을 생성해야 합니다. AWS IoT 콘솔 또는 AWS Command Line Interface(AWS CLI). 작업 생성에 대한 자세한 내용은 단원을 참조하십시오. [작업 생성](#)의 AWS CLI 명령 참조. 데모에서는 작업 문서에 `action`으로 설정된 키 `print`를 사용하여 콘솔에 메시지를 인쇄합니다.

이 작업 문서에 대해서는 다음 형식을 참조하십시오.

```
{  
    "action": "print",  
    "message": "ADD_MESSAGE_HERE"  
}
```

이 AWS CLI를 사용하여 다음 예제 명령과 같이 작업을 생성하십시오.

```
aws iot create-job \  
  --job-id t12 \  
  --targets arn:aws:iot:region:123456789012:thing/device1 \  
  --
```

```
--document '{"action": "print", "message": "hello world!"}'
```

데모에서는 다음 작업 문서도 사용합니다. action로 설정된 키 publish를 사용하여 주제에 메시지를 다시 게시합니다. 이 작업 문서에 대해서는 다음 형식을 참조하십시오.

```
{  
    "action": "publish",  
    "message": "ADD_MESSAGE_HERE",  
    "topic": "topic/name/here"  
}
```

데모는 다음과 같은 작업 문서를 받을 때까지 반복됩니다. action로 설정된 키 exit를 사용하여 데모를 종료 합니다. 작업 문서의 형식은 다음과 같습니다.

```
{  
    "action": "exit"  
}
```

## Jobs 데모의 진입점

Jobs 데모 진입 점 함수의 소스 코드는 다음에서 확인할 수 있습니다. [GitHub](#). 이 함수는 다음 작업을 수행합니다.

1. 의 도우미 함수를 사용하여 MQTT 연결 설정 mqtt\_demo\_helpers.c.
2. MQTT 주제 구독 NextJobExecutionChangedAPI, 헬퍼 함수 사용 mqtt\_demo\_helpers.c. 주제 문자열은에서 정의한 매크로를 사용하여 이전에 어셈블됩니다. AWS IoT 작업 라이브러리
3. 에 대한 MQTT 주제에 게시합니다. StartNextPendingJobExecutionAPI, 헬퍼 함수 사용 mqtt\_demo\_helpers.c. 주제 문자열은에서 정의한 매크로를 사용하여 이전에 어셈블됩니다. AWS IoT 작업 라이브러리
4. 반복해서 전화 MQTT\_ProcessLoop 전달되는 수신 메시지를 수신하려면 prvEventCallback 처리를 위해.
5. 데모가 종료 작업을 수신한 후 MQTT 주제의 구독을 취소하고 의 도우미 함수를 사용하여 연결을 끊습니다. mqtt\_demo\_helpers.c 파일.

## 수신된 MQTT 메시지에 대한 콜백

이 PRVEVENT 콜백 함수 호출 Jobs\_MatchTopic( 사용) AWS IoT 들어오는 MQTT 메시지를 분류하는 작업 라이브러리입니다. 메시지 유형이 새 작업에 해당하는 경우 prvNextJobHandler()이 호출됩니다.

이 PRV 넥스트 작업 핸들러 함수 및 호출하는 함수는 JSON 형식 메시지에서 작업 문서를 구문 분석하고 작업에서 지정한 작업을 실행합니다. 특히 흥미로운 것은 prvSendUpdateForJob 함수.

## 실행 중인 작업에 대한 업데이트 보내기

함수 PRVSENDUPdate( ) 전화Jobs\_Update( ) Jobs 라이브러리에서 바로 뒤에 오는 MQTT 게시 작업에 사용된 주제 문자열을 채웁니다.

## CoremQTT 데모

이러한 데모는 CoreMqtt 라이브러리 사용 방법에 대해 알아보는 데 유용하게 활용할 수 있습니다.

주제

- CoremQTT 상호 인증 데모 (p. 266)
- 코렘QTT 에이전트 연결 공유 데모 (p. 269)

## CoremQTT 상호 인증 데모

### 소개

CoreMQTT 상호 인증 데모 프로젝트는 클라이언트와 서버 간의 상호 인증을 통해 TLS를 사용하여 MQTT 브로커에 연결하는 방법을 보여줍니다. 이 데모에서는 MBETTLS 기반 전송 인터페이스 구현을 사용하여 서버 및 클라이언트 인증된 TLS 연결을 설정하고 에서 MQTT의 구독-게시 워크플로우를 보여 줍니다. QoS 1수준. 주제 필터를 구독한 다음 필터와 일치하는 항목에 게시하고 QoS 1 수준에서 서버에서 해당 메시지를 다시 수신할 때까지 기다립니다. 브로커에 게시하고 브로커로부터 동일한 메시지를 수신하는 사이클은 무한정 반복됩니다. 이 데모의 메시지는 QoS 1에서 전송되며, 이는 MQTT 사양에 따라 하나 이상의 배달을 보장합니다.

#### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오.[FreeRTOS 시작하기 \(p. 14\)](#).

### 소스 코드

데모 소스 파일의 이름은 `mqtt_demo_mutual_auth.c`에서 찾을 수 있습니다.[freertos/demos/coreMQTT](#)/디렉터리 및 [GitHub](#) 웹 사이트.

### 기능

데모는 브로커에 연결하고, 브로커의 주제를 구독하고, 브로커의 주제에 게시한 다음 마지막으로 브로커와의 연결을 해제하는 방법을 보여주는 일련의 예제를 반복하는 단일 응용 프로그램 작업을 만들습니다. 데모 응용 프로그램은 모두 동일한 주제를 구독하고 게시합니다. 데모가 MQTT 브로커에게 메시지를 게시할 때마다 브로커는 동일한 메시지를 데모 애플리케이션으로 다시 보냅니다.

데모를 성공적으로 완료하면 다음 이미지와 유사한 출력이 생성됩니다.

```

59 1548 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1798] 40 1548 [iot_thread] MQTT connection established with the broker.41 1548 [iot_thread]
42 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:675] 43 1548 [iot_thread] An MQTT connection is established with a3c4bx1sn0lp8-ats.iot.us-west-2.amazonaws.com.44 1548 [iot_thread]
45 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:747] 46 1548 [iot_thread] Attempt to subscribe to the MQTT topic MyIOTThingTest5/example/topic.47
1548 [iot_thread]
48 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:761] 49 1548 [iot_thread] SUBSCRIBE sent for topic MyIOTThingTest5/example/topic to broker.50 154
9 [iot_thread]
51 1588 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 52 1588 [iot_thread] Packet received. ReceivedBytes=3.53 1588 [iot_thread]
54 1588 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:907] 55 1588 [iot_thread] Subscribed to topic MyIOTThingTest5/example/topic with maximum QoS
1.56 1588 [iot_thread]
57 2188 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:458] 58 2188 [iot_thread] Publish to the MQTT topic MyIOTThingTest5/example/topic.59 2188 [iot_th
read]
59 2188 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:465] 61 2188 [iot_thread] Attempt to receive publish message from broker.62 2188 [iot_thread]
53 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 64 2248 [iot_thread] Packet received. ReceivedBytes=2.65 2248 [iot_thread]
56 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] 67 2248 [iot_thread] Ack packet serialized with result: MQTTSuccess.68 2248 [iot_thread]
59 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] 70 2248 [iot_thread] State record updated. New state=MQTPublishDone.71 2248 [iot_thread]
72 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:888] 73 2248 [iot_thread] PUBACK received for packet Id 2.74 2248 [iot_thread]
75 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 76 2248 [iot_thread] Packet received. ReceivedBytes=45.77 2248 [iot_thread]
78 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] 79 2248 [iot_thread] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.80 2248 [iot_thread]
81 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] 82 2248 [iot_thread] State record updated. New state=MQTPubAckSend.83 2248 [iot_thread]
84 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:958] 85 2248 [iot_thread] Incoming QoS : 1
86 2248 [iot_thread]
97 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:969] 88 2248 [iot_thread] Incoming Publish Topic Name: MyIOTThingTest5/example/topic matches sub
scribed topic.Incoming Publish Message : Hello World!89 2248 [iot_thread]
90 2848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:478] 91 2848 [iot_thread] Keeping Connection Idle...92 2848 [iot_thread]
93 4848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:458] 94 4848 [iot_thread] Publish to the MQTT topic MyIOTThingTest5/example/topic.95 4848 [iot_th
read]
96 4848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:465] 97 4848 [iot_thread] Attempt to receive publish message from broker.98 4848 [iot_thread]
99 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 100 4888 [iot_thread] Packet received. ReceivedBytes=2.101 4888 [iot_thread]
102 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] 103 4888 [iot_thread] Ack packet serialized with result: MQTTSuccess.104 4888 [iot_thread]
105 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] 106 4888 [iot_thread] State record updated. New state=MQTPublishDone.107 4888 [iot_thread]
108 4888 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:888] 109 4888 [iot_thread] PUBACK received for packet Id 3.110 4888 [iot_thread]
111 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 112 4928 [iot_thread] Packet received. ReceivedBytes=45.113 4928 [iot_thread]
114 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] 115 4928 [iot_thread] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.116 4928 [iot_thread]
117 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] 118 4928 [iot_thread] State record updated. New state=MQTPubAckSend.119 4928 [iot_thread]
120 4928 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:958] 121 4928 [iot_thread] Incoming QoS : 1
122 4928 [iot_thread]
123 4928 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:969] 124 4928 [iot_thread] Incoming Publish Topic Name: MyIOTThingTest5/example/topic matches su
bscribed topic.Incoming Publish Message : Hello World!125 4928 [iot_thread]
126 5528 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:478] 127 5528 [iot_thread] Keeping Connection Idle...128 5528 [iot_thread]

```

O|AWS IoTconsole 은 다음 이미지와 유사한 출력을 생성합니다.

The screenshot shows the AWS IoT Core console interface. At the top, there's a 'Publish' section where a JSON message is being typed into a text input field:

```

1 "message": "Hello from AWS IoT console"
2
3

```

Below this, three published messages are listed in a table:

Topic	Timestamp	Actions
MyIOTThingTest5/example/topic	November 03, 2020, 13:03:57 (UTC-0800)	Export Hide
MyIOTThingTest5/example/topic	November 03, 2020, 13:03:52 (UTC-0800)	Export Hide
MyIOTThingTest5/example/topic	November 03, 2020, 13:03:47 (UTC-0800)	Export Hide

Each message row contains a green header stating 'We cannot display the message as JSON, and are instead displaying it as UTF-8 String.' followed by the message content 'Hello World!'. The last message row also includes a timestamp 'November 03, 2020, 13:03:47 (UTC-0800)'.

## 지수 백오프 및 지터로 로직 재시도

이[PRV백오프포재시도](#)함수는 TLS 연결 또는 MQTT 구독 요청과 같이 서버와의 네트워크 작업 실패가 지수 백오프 및 지터로 재시도될 수 있는 방법을 보여줍니다. 이 함수는 다음 재시도 시도의 백오프 기간을 계산하고 재시도 시도가 소진되지 않은 경우 백오프 지연을 수행합니다. 백오프 기간을 계산하려면 난수를 생성해야 하기 때문에 함수는 PKCS11 모듈을 사용하여 난수를 생성합니다. PKCS11 모듈을 사용하면 공급업체 플랫폼이 지원하는 경우 TRNG (트루 난수 생성기)에 액세스할 수 있습니다. 연결 재시도 중에 디바이스로부터 충돌할 확률을 완화할 수 있도록 장치별 엔트로피 소스를 사용하여 난수 생성기를 시드하는 것이 좋습니다.

## MQTT 브로커에 연결

이[PRV커넥트서버와백오프리시도](#)함수는 MQTT 브로커에 상호 인증된 TLS 연결을 시도합니다. 연결에 실패하면 백오프 기간이 지나면 다시 시도합니다. 백오프 기간은 최대 시도 횟수에 도달하거나 최대 백오프 기간에도 도달할 때까지 기하급수적으로 증가합니다. 이[BackoffAlgorithm\\_GetNextBackoff](#)함수는 기하 급수적으로 증가하는 백오프 값 및 리턴을 제공합니다[RetryUtilsRetriesExhausted](#)최대 횟수에 도달한 경우입니다. 이[prvConnectToServerWithBackoffRetries](#)구성된 시도 횟수 후에 브로커에 대한 TLS 연결을 설정할 수 없는 경우 함수는 실패 상태를 반환합니다.

이[PRV생성QT연결브로커 포함](#)함수는 클린 세션으로 MQTT 브로커에 대한 MQTT 연결을 설정하는 방법을 보여줍니다. TLS 전송 인터페이스를 사용합니다. 이 인터페이스는[FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls\\_freertos.c](#)파일. 브로커에 대한 연결 유지 초를 설정하고 있음을 명심하십시오.[xConnectInfo](#).

다음 함수는 다음을 사용하여 MQTT 컨텍스트에서 TLS 전송 인터페이스 및 시간 함수가 설정되는 방법을 보여줍니다.[MQTT\\_Init](#)함수. 또한 이벤트 콜백 함수 포인터 ([prvEventCallback](#))가 설정됩니다. 이 콜백은 수신 메시지를 보고하는 데 사용됩니다.

## MQTT 주제 구독

이[PRVMQTT구독/백오프리트리](#)함수는 MQTT 브로커에서 주제 필터를 구독하는 방법을 보여줍니다. 이 예제에서는 하나의 주제 필터를 구독하는 방법을 보여 주지만 동일한 구독 API 호출에서 주제 필터 목록을 전달

하여 둘 이상의 주제 필터를 구독할 수 있습니다. 또한 MQTT 브로커가 구독 요청을 거부하는 경우 구독은 자수 백오프와 함께 다시 시도합니다.RETRY\_MAX\_ATTEMPTS.

## 주제에 게시

이[PRVMQT퍼블리시토픽](#)함수는 MQTT 브로커의 주제에 게시하는 방법을 보여줍니다.

## 수신 메시지

응용 프로그램은 앞에서 설명한 대로 브로커에 연결하기 전에 이벤트 콜백 함수를 등록합니다.  
이[prvMQTTDemoTask](#)함수를 호출합니다.[MOTT\\_ProcessLoop](#)들어오는 메시지를 수신하는 함수. 들어오는 MQTT 메시지가 수신되면 응용 프로그램에서 등록한 이벤트 콜백 함수를 호출합니다. 이[PRVEVENT](#)콜백 함수는 이러한 이벤트 콜백 함수의 예입니다.[prvEventCallback](#)는 들어오는 패킷 유형을 검사하고 적절한 처리기를 호출합니다. 아래 예제에서는 함수를 호출합니다.[prvMQTTProcessIncomingPublish\(\)](#)수신 메시지 처리 또는[prvMQTTProcessResponse\(\)](#)승인(ACK)을 처리합니다.

## 들어오는 MQTT 게시 패킷 처리

이[PRVMQT프로세싱](#)함수는 MQTT 브로커에서 게시 패킷을 처리하는 방법을 보여줍니다.

## 주제의 구독 취소

워크플로의 마지막 단계는 브로커가 게시된 메시지를 보내지 않도록 주제 구독을 취소하는 것입니다.[mqtexampleTOPIC](#). 다음은 함수의 정의입니다.[PRVMQT구독 취소주제](#).

## 데모에 사용된 루트 CA 변경

기본적으로 FreeRTOS 데모는 아마존 루트 CA 1 인증서 (RSA 2048비트 키)를 사용하여 AWS IoT Core 서버. 기타를 사용할 수 있습니다.[서버 인증을 위한 CA 인증서](#)Amazon Root CA 3 인증서 (ECC 256비트 키)를 포함합니다. CoreMQTT 상호 인증 데모의 루트 CA를 변경하려면 다음과 같이 하십시오.

1. 텍스트 편집기에서 [freertos/vendors/vendor/boards/board/aws\\_demos/config\\_files/mqtt\\_demo\\_mutual\\_auth\\_config.h](#) 파일을 업니다.
2. 파일에서 다음 줄을 찾습니다.

```
* #define democonfigROOT_CA_PEM      "...insert here..."
```

이 줄의 주석을 제거하고 필요한 경우 주석 블록 끝을 지나서 이동합니다. \*/.

3. 사용할 CA 인증서를 복사한 다음"...insert here..." 텍스트. 결과는 다음 예제와 같아야 합니다.

```
#define democonfigROOT_CA_PEM      "-----BEGIN CERTIFICATE-----\n" \
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkJOPQDAAjA5\n" \
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n" \
"Um9vdCBQSAzMBA4XDTE1MDUyNjAwMDAwMFoXDT0wMDUyNjAwMDAwMFowOTELMAkG\n" \
"A1UEBhMCVVNmDzANBgNVBAotBkFtYXpvbjEZMBcGA1UEAxAMQQW1hem9uIFJvb3Qg\n" \
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG81K1\n" \
"ui2yEujsLtf6ycXYqm0fc4E705hrOXwzpcVOho6AF2hiRVd9RFgszf1ZwjrzT6j\n" \
"QjBAMA8GA1UDxEWEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UDDgQWBBSr\n" \
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBgqhkJOPQDAGNJADBGA1EA4IWSoxe3jfkr\n" \
"BqWTTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyz1vnrxir4tiz+OpAUFeM\n" \
"YyRIHN8wfdVoOw==\n" \
"-----END CERTIFICATE-----\n"
```

4. (선택 사항) 다른 데모의 루트 CA를 변경할 수 있습니다. 각각에 대해 1~3단계를 반복합니다.  
다.[freertos/vendors/vendor/boards/board/aws\\_demos/config\\_files/demo-name\\_config.h](#)파일.

# 코렘QTT 에이전트 연결 공유 데모

## 소개

CoreMqtt 연결 공유 데모 프로젝트는 멀티스레드 애플리케이션을 사용하여 AWS 클라이언트와 서버 간의 상호 인증과 함께 TLS를 사용하는 MQTT 브로커입니다. 이 데모에서는 MBETTLS 기반 전송 인터페이스 구현을 사용하여 서버 및 클라이언트 인증 TLS 연결을 설정하고 MQTT의 구독-게시 워크플로우를 QoS 1 수준. 데모는 주제 필터를 구독하고 필터와 일치하는 주제에 게시한 다음 QoS 1 수준의 서버에서 해당 메시지를 다시 수신할 때까지 기다립니다. 브로커에 게시하고 브로커에서 동일한 메시지를 수신하는 이러한 사이클은 생성된 각 작업에 대해 여러 번 반복됩니다. 이 데모의 메시지는 QoS 1에서 전송되며, 이는 MQTT 사양에 따라 하나 이상의 배달을 보장합니다.

### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오.[FreeRTOS 시작하기 \(p. 14\)](#).

이 데모에서는 스레드 안전 대기열을 사용하여 MQTT API와 상호 작용하는 명령을 보관합니다. 이 데모에서 두 가지 작업을 기록해 두가지가 있습니다.

- MQTT 에이전트 (기본) 작업은 명령 대기열에서 명령을 처리하고 다른 작업은 대기열에 넣습니다. 이 작업은 명령 큐에서 명령을 처리하는 동안 루프에 들어갑니다. 종료 명령이 수신되면 이 작업은 루프에서 벗어납니다.
- 데모 하위 펍 작업은 MQTT 주제에 대한 구독을 만든 다음 게시 작업을 만들어 명령 대기열로 푸시합니다. 그런 다음 이러한 게시 작업은 MQTT 에이전트 작업에 의해 실행됩니다. 데모 하위 펍 작업은 게시가 완료될 때까지 기다렸다가 명령 완료 콜백의 실행으로 표시된 다음 게시가 시작되기 전에 짧은 지연을 입력합니다. 이 작업은 애플리케이션 작업에서 CoreMqtt 에이전트 API를 사용하는 방법의 예를 보여줍니다.

들어오는 게시 메시지의 경우 CoreMqtt 에이전트는 단일 콜백 함수를 호출합니다. 이 데모에는 구독한 주제에 대해 들어오는 게시 메시지에 대해 호출할 콜백을 지정할 수 있는 구독 관리자도 포함되어 있습니다. 이 데모에서 에이전트의 수신 게시 콜백은 구독 관리자를 호출하여 구독을 등록한 모든 작업에 게시를 팬아웃합니다.

이 데모에서는 상호 인증과 함께 TLS 연결을 사용하여 AWS. 데모 중에 네트워크가 예기치 않게 연결이 끊어지면 클라이언트는 지수 백오프 로직을 사용하여 다시 연결을 시도합니다. 클라이언트가 성공적으로 다시 연결되었지만 브로커가 이전 세션을 재개할 수 없으면 클라이언트는 이전 세션과 동일한 주제를 다시 구독합니다.

## 단일 스레드 vs 멀티스레드

CoreMQTT 사용 모델에는 단일 스레드와 멀티스레드 (멀티태스킹) 의 두 가지가 있습니다. 단일 스레드 모델은 하나의 스레드에서만 CoreMQTT 라이브러리를 사용하므로 MQTT 라이브러리에서 반복적으로 명시적으로 호출해야 합니다. 멀티스레드 사용 사례는 여기에 설명된 데모와 같이 에이전트 (또는 데몬) 작업 내에서 백그라운드에서 MQTT 프로토콜을 실행할 수 있습니다. 에이전트 작업에서 MQTT 프로토콜을 실행할 때 명시적으로 MQTT 상태를 관리하거나 `MQTT_ProcessLoopAPI` 함수입니다. 또한 에이전트 작업을 사용하는 경우 뮤텍스와 같은 동기화 프리미티브 없이도 여러 애플리케이션 작업이 단일 MQTT 연결을 공유할 수 있습니다.

## 소스 코드

데모 소스 파일의 이름이 지정되었습니다.`mqtt_agent_task.c`과 `simple_sub_pub_demo.c`에서 찾을 수 있습니다.[freertos/demos/coreMQTT\\_Agent/](#)디렉터리 및 [GitHub](#) 웹 사이트.

## 기능

이 데모에서는 MQTT API 호출에 대한 요청을 처리하는 기본 작업과 이러한 요청을 생성하는 구성 가능한 하위 작업 수의 작업을 두 개 이상 만들습니다. 이 데모에서는 기본 작업이 하위 태스크를 만들고 처리 루프를 호

출한 다음 나중에 정리합니다. 기본 작업은 하위 작업 간에 공유되는 브로커에 대한 단일 MQTT 연결을 만듭니다. 하위 작업은 브로커와 함께 MQTT 구독을 만든 다음 메시지를 게시합니다. 각 하위 작업은 게시에 대해 고유한 주제를 사용합니다.

## 기본 과제

주요 응용 작업, [런코렐쿼터젠티데모](#), MQTT 세션을 설정하고, 하위 작업을 만들고, 처리 루프를 실행합니다. [MQTTagent\\_CommandLoop\(\)](#) 종료 명령이 수신될 때까지 네트워크가 예기치 않게 연결이 끊어지면 데모가 백그라운드에서 브로커에 다시 연결되고 브로커와의 구독을 다시 설정합니다. 처리 루프가 종료되면 브로커와의 연결이 끊어집니다.

### 명령

CoreMqtt 에이전트 API를 호출하면 에서 처리되는 에이전트 작업의 대기열로 전송되는 명령이 생성됩니다. [MQTTagent\\_CommandLoop\(\)](#). 명령이 생성될 때 선택적 완료 콜백 및 컨텍스트 매개 변수가 전달될 수 있습니다. 해당 명령이 완료되면 완료 콜백은 전달된 컨텍스트와 명령의 결과로 생성된 모든 반환 값으로 호출됩니다. 완료 콜백에 대한 서명은 다음과 같습니다.

```
typedef void (* MQTTagentCommandCallback_t )( void * pCmdCallbackContext,  
                                            MQTTagentReturnInfo_t * pReturnInfo );
```

명령 완성 컨텍스트는 사용자 정의입니다. 이 데모에서는 다음과 같습니다. [구조체 MQtAGENTCommandContext](#).

다음과 같은 경우 명령이 완료된 것으로 간주됩니다.

- QoS > 0으로 구독, 구독 취소 및 게시합니다. 해당 승인 패킷이 수신되면
- 기타 모든 작업: 해당 CoreMqtt API가 호출되면

게시 정보, 구독 정보 및 완료 컨텍스트를 포함하여 명령에서 사용하는 모든 구조는 명령이 완료될 때까지 범위 내에 있어야 합니다. 호출 태스크는 완료 콜백을 호출하기 전에 명령의 구조를 재사용해서는 안 됩니다. 완료 콜백은 MQTT 에이전트에 의해 호출되므로 명령을 만든 작업이 아니라 에이전트 작업의 스레드 컨텍스트와 함께 실행됩니다. 작업 알림 또는 대기열과 같은 프로세스 간 통신 메커니즘을 사용하여 명령 완료 호출 작업을 알릴 수 있습니다.

### 명령 루프 실행

명령은 에서 지속적으로 처리됩니다. [MQTTagent\\_CommandLoop\(\)](#). 처리할 명령이 없는 경우 루프는 최대 값을 기다립니다. [MQTT\\_AGENT\\_MAX\\_EVENT\\_QUEUE\\_WAIT\\_TIME](#) 대기열에 추가되고 명령이 추가되지 않은 경우 단일 이터레이션을 실행합니다. [MQTT\\_ProcessLoop\(\)](#). 이렇게 하면 MQTT Keep-Alive가 관리되고 대기열에 명령이 없는 경우에도 들어오는 게시가 모두 수신됩니다.

명령 루프 함수는 다음과 같은 이유로 반환됩니다.

- 명령은 그 외에 모든 상태 코드를 반환합니다. [MQTTSUCCESS](#). 오류 상태는 명령 루프에 의해 반환되므로 처리 방법을 결정할 수 있습니다. 이 데모에서는 TCP 연결이 다시 설정되고 다시 연결이 시도됩니다. 오류가 발생하면 MQTT를 사용하는 다른 작업의 개입 없이 백그라운드에서 다시 연결할 수 있습니다.
- 연결 해제 명령 (부터) [MQTTagent\\_Disconnect](#)) 가 처리됩니다. 명령 루프가 종료되어 TCP의 연결을 끊을 수 있습니다.
- 종료 명령 (보낸 사람) [MQTTagent\\_Terminate](#)) 가 처리됩니다. 또한 이 명령은 대기열에 있는 명령이나 승인 패킷을 대기 중인 명령을 오류로 표시하며 반환 코드는 [MQTTRecvFailed](#).

### 구독 관리자

데모에서는 여러 주제를 사용하기 때문에 구독 관리자는 구독한 주제를 고유한 콜백 또는 작업과 연결하는 편리한 방법입니다. 이 데모의 구독 관리자는 단일 스레드이므로 여러 작업에서 동시에 사용해서는 안됩니다.

다. 이 데모에서는 구독 관리자 함수는 MQTT 에이전트로 전달되는 콜백 함수에서만 호출되며 에이전트 작업의 스레드 컨텍스트에서만 실행됩니다.

## 간단한 구독-게시 작업

각 인스턴스 PRV심플서스크리브퍼블리쉬태스크에서 MQTT 주제에 대한 구독을 만들고 해당 주제에 대한 게시 작업을 만듭니다. 여러 게시 유형을 시연하기 위해 짹수 번호가 매겨진 작업은 QoS 0 (게시 패킷이 전송되면 완료) 을 사용하고 홀수 작업은 QoS 1 (PUBACK 패킷 수신 시 완료) 을 사용합니다.

# OTA(Over-the-Air) 업데이트 데모 애플리케이션

FreeRTOS에는 무선 (OTA) 라이브러리의 기능을 시연하는 데모 애플리케이션이 포함되어 있습니다. OTA 데모 애플리케이션은 [freertos/demos/ota/ota\\_demo\\_core\\_mqtt/ota\\_demo\\_core\\_mqtt.c](#) 또는 [freertos/demos/ota/ota\\_demo\\_core\\_http/ota\\_demo\\_core\\_http.c](#) 파일입니다.

OTA 데모 애플리케이션은 다음을 수행합니다.

1. FreeRTOS 네트워크 스택과 MQTT 버퍼 풀을 초기화합니다.
2. 다음을 사용하여 OTA 라이브러리를 연습하기 위한 작업을 생성합니다.`vRunOTAUpdateDemo()`.
3. `_establishMqttConnection()`를 사용하여 MQTT 클라이언트를 생성합니다.
4. AWS IoT MQTT 브로커 사용 `IotMqtt_Connect()` MQTT 연결 해제 콜백을 등록합니다.`.prvNetworkDisconnectCallback`.
5. `OTA_AgentInit()`를 호출하여 OTA 작업을 생성하고, OTA 작업이 완료될 때 사용할 콜백을 등록합니다.
6. MQTT 연결을 재사용합니다.`xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. MQTT 연결이 끊어지면 애플리케이션은 OTA 에이전트를 일시 종지하고 지터가 있는 지수 지연을 사용하여 다시 연결을 시도한 다음 OTA 에이전트를 다시 시작합니다.

OTA 업데이트를 사용하려면 먼저 의 모든 사전 조건을 완료합니다.[FreeRTOS 무선 업데이트 \(p. 140\)](#)

OTA 업데이트 설정을 완료한 후 OTA 기능을 지원하는 플랫폼에서 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행 디바이스별 데모 지침은 다음 FreerTos 적격 디바이스에서 사용할 수 있습니다.

- Texas Instruments CC3220SF-LAUNCHXL (p. 274)
- Microchip Curiosity PIC32MZEF (p. 276)
- Espressif ESP32 (p. 280)
- Renesas RX65N에서 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행 (p. 281)

디바이스에서 OTA 데모 애플리케이션을 빌드, 플래시 및 실행한 후에는 AWS IoT 콘솔 또는 AWS CLI를 사용하여 OTA 업데이트 작업을 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.

```
249 21207 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT] Received: 0 Queued: 0
Processed: 0 Dropped: 0
250 21247 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
ReceivedBytes=601.
251 21247 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
PUBLISH packet: DeserializerResult=MQTTSuccess.
252 21248 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
state=MQTTPubAckSend.
```

```

253 21249 [MQTT Agent Task] [ota_demo_core_mqtt.c:976] [INFO] [MQTT] Received job
message callback, size 548.
254 21252 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
255 21253 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
256 21255 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
257 21256 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[filepath: aws_demos.bin]
258 21257 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
value]=[filesize: 1164016]
259 21258 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
value]=[fileid: 0]
260 21259 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[certfile: ecdsa-sha256-signer.crt.pem]
261 21260 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-sha256-
ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
262 21261 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
value]=[fileType: 0]
263 21262 [OTA Agent Task] [ota.c:2199] [INFO] [OTA] Job document was accepted.
Attempting to begin the update.
264 21263 [OTA Agent Task] [ota.c:2323] [INFO] [OTA] Job parsing success:
OtaJobParseErr_t=OtaJobParseErrNone, Job name=AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb
265 21318 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT] Received: 0 Queued:
0 Processed: 0 Dropped: 0
266 21418 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT] Received: 0 Queued:
0 Processed: 0 Dropped: 0
267 21469 [OTA Agent Task] [ota.c:938] [INFO] [OTA] Setting OTA data interface.
268 21470 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current State=[CreatingFile],
Event=[ReceivedJobDocument], New state=[CreatingFile]
269 21482 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
ReceivedBytes=3.
270 21483 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED
to topic $aws/things/_test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/data/cbor to bro
271 21484 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
State=[RequestingFileBlock], Event=[CreateFile], New state=[RequestingFileBlock]
272 21518 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT] Received: 0 Queued:
0 Processed: 0 Dropped: 0
273 21532 [MQTT Agent Task] [core_mqtt_agent_command_functions.c:76] [INFO] [MQTT]
Publishing message to $aws/things/_test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/
274 21534 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
packet to broker $aws/things/_test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/get/cbor
275 21534 [OTA Agent Task] [ota_mqtt.c:1112] [INFO] [OTA] Published to MQTT topic to
request the next block: topic=$aws/things/_test_infra_thing71/streams/AFR_OTA-945d320b-
a18b-441b-b435-4a1
276 21537 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
State=[WaitingForFileBlock], Event=[RequestFileBlock], New state=[WaitingForFileBlock]
277 21558 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
ReceivedBytes=4217.
278 21559 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
PUBLISH packet: DeserializerResult=MQTTSuccess.
279 21560 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
state=MQTTPublishDone.
280 21561 [MQTT Agent Task] [ota_demo_core_mqtt.c:1026] [INFO] [MQTT] Received data
message callback, size 4120.
281 21563 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block: Block
index=0, Size=4096
282 21566 [OTA Agent Task] [ota.c:2683] [INFO] [OTA] Number of blocks remaining: 284
...
// Output removed for brevity

```

```

3672 42745 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block: Block
index=284, Size=752
3673 42747 [OTA Agent Task] [ota.c:2633] [INFO] [OTA] Received final block of the
update.
(428298) ota_pal: No such certificate file: ecdsa-sha256-signer.crt.pem. Using
certificate in ota_demo_config.h.
3674 42818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT] Received: 285
Queued: 285 Processed: 284 Dropped: 0
3675 42918 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT] Received: 285
Queued: 285 Processed: 284 Dropped: 0

... // Output removed for brevity

3678 43197 [OTA Agent Task] [ota.c:2654] [INFO] [OTA] Received entire update and
validated the signature.
3685 43215 [OTA Agent Task] [ota_demo_core_mqtt.c:862] [INFO] [MQTT] Received
OtaJobEventActivate callback from OTA Agent.

... // Output removed for brevity

2 39 [iot_thread] [INFO ][DEMO][390] -----STARTING DEMO-----

[0;32mI (3633) WIFI: WIFI_EVENT_STA_CONNECTED
[0;32mI (4373) WIFI: SYSTEM_EVENT_STA_GOT_IP

... // Output removed for brevity

4 351 [sys_evt] [INFO ][DEMO][3510] Connected to WiFi access point, ip address:
255.255.255.0.
5 351 [iot_thread] [INFO ][DEMO][3510] Successfully initialized the demo. Network type
for the demo: 1
6 351 [iot_thread] [ota_demo_core_mqtt.c:1902] [INFO] [MQTT] OTA over MQTT demo,
Application version 0.9.1
7 351 [iot_thread] [ota_demo_core_mqtt.c:1323] [INFO] [MQTT] Creating a TLS connection
to <endpoint>-ats.iot.us-west-2.amazonaws.com:8883.
9 718 [iot_thread] [core_mqtt.c:886] [INFO] [MQTT] Packet received. ReceivedBytes=2.
10 718 [iot_thread] [core_mqtt_serializer.c:970] [INFO] [MQTT] CONNACK session present
bit not set.
11 718 [iot_thread] [core_mqtt_serializer.c:912] [INFO] [MQTT] Connection accepted.

... // Output removed for brevity

17 736 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED to topic
$aws/things/_test_infra_thing71/jobs/notify-next to broker.
18 737 [OTA Agent Task] [ota_mqtt.c:381] [INFO] [OTA] Subscribed to MQTT topic: $aws/
things/_test_infra_thing71/jobs/notify-next
30 818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT] Received: 0 Queued: 0
Processed: 0 Dropped: 0
31 819 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
32 820 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
value]=[execution.statusDetails.updatedBy: 589824]
33 822 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
34 823 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
35 824 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[filepath: aws_demos.bin]
36 825 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
value]=[filesize: 1164016]
37 826 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
value]=[fileid: 0]
38 827 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
value]=[certfile: ecdsa-sha256-signer.crt.pem]

```

```
39 828 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-sha256-
ecdsa: MEQCIEISFkIHHiZAvkPpu6McJtx7SYOD... ]
40 829 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
value]=[fileType: 0]
41 830 [OTA Agent Task] [ota.c:2102] [INFO] [OTA] In self test mode.
42 830 [OTA Agent Task] [ota.c:1936] [INFO] [OTA] New image has a higher version number
than the current image: New image version=0.9.1, Previous image version=0.9.0
43 832 [OTA Agent Task] [ota.c:2120] [INFO] [OTA] Image version is valid: Begin testing
file: File ID=0
53 896 [OTA Agent Task] [ota.c:794] [INFO] [OTA] Beginning self-test.
62 971 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
packet to broker $aws/things/_test_infra_thing71/jobs/AFR_OTA-9702f1a3-b747-4c3e-a0eb-
a3b0cf83ddbb/update to br63 971 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-
serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
65 973 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
state=MQTTPublishDone.
64 973 [OTA Agent Task] [ota_demo_core_mqtt.c:902] [INFO] [MQTT] Successfully updated
with the new image.
```

## OTA(무선) 데모 구성

OTA 데모 구성은 에서 제공하는 데모별 구성 옵션입니다.`aws_iot_ota_update_demo.c`. 이러한 구성은 OTA 라이브러리 구성 파일에 제공된 OTA 라이브러리 구성과 다릅니다.

### OTA\_데모\_유지\_알리브\_초

MQTT 클라이언트의 경우 이 구성은 하나의 제어 패킷 전송을 완료하고 다음 제어 패킷의 전송을 시작하는 사이에 경과할 수 있는 최대 시간 간격입니다. 제어 패킷이 없으면 PINGREQ가 전송됩니다. 브로커는 이 유지 간격의 1.5 배에 메시지 또는 PINGREQ 패킷을 보내지 않는 클라이언트의 연결을 끊어야 합니다. 이 구성은 애플리케이션의 요구 사항에 따라 조정해야 합니다.

### OTA\_데모\_콘\_리트리\_베이스\_인터발\_초

네트워크 연결을 재시도하기 전의 기본 간격(초)입니다. OTA 데모에서는 이 기본 시간 간격 후에 다시 연결을 시도합니다. 실패한 시도마다 간격이 두 배가 됩니다. 이 기본 지연의 최대값까지 임의의 지연도 간격에 추가됩니다.

### OTA\_데모\_콘\_재시도\_맥스\_인터발\_초

네트워크 연결을 다시 시도하기 전의 최대 간격(초)입니다. 다시 연결 지연은 실패한 모든 시도에서 두 배가 되지만 이 최대값과 동일한 간격의 지터까지만 이동할 수 있습니다.

## Texas Instruments CC3220SF-LAUNCHXL 에 대해 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

### FreeRTOS 및 OTA 데모 코드를 다운로드하려면

1. AWS IoT 콘솔로 이동하여 탐색 창에서 소프트웨어를 선택합니다.
2. UNDFreeRTOS 디바이스 소프트웨어, 선택다운로드 구성.
3. 소프트웨어 구성 목록에서 AWS IoT에 연결 - TI를 선택합니다. 구성 이름(다운로드 링크 아님)을 선택합니다.
4. 라이브러리에서 다른 라이브러리 추가를 선택한 후 OTA Updates(OTA 업데이트)를 선택합니다.
5. 데모 프로젝트에서 OTA Updates(OTA 업데이트)를 선택합니다.
6. 이름 필수 아래에 **Connect-to-IoT-OTA-TI**를 입력한 다음 생성 후 다운로드를 선택합니다.

FreeRTOS 및 OTA 데모 코드가 포함된 zip 파일을 컴퓨터에 저장합니다.

### 데모 애플리케이션을 빌드하려면

1. .zip 파일의 압축을 풁니다.
2. FreeRTOS 시작하기 (p. 14)의 지침에 따라 aws\_demos 프로젝트를 Code Composer Studio로 가져오고 AWS IoT 엔드포인트, Wi-Fi SSID 및 암호, 보드에 대한 프라이빗 키 및 인증서를 구성합니다.
3. Open `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`를 주석으로 처리합니다.  
`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`를 열고 를 정의합니다.  
다.`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` 또는 `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
4. 솔루션을 빌드하고 오류 없이 빌드되는지 확인합니다.
5. 터미널 에뮬레이터를 시작하고 다음 설정을 사용하여 보드에 연결합니다.
  - : 전송 속도 115200
  - 데이터 비트: 8
  - 패리티: None
  - 정지 비트: 1
6. 보드에서 프로젝트를 실행하여 Wi-Fi 및 AWS IoT MQTT 메시지 브로커에 연결할 수 있는지 확인합니다.

실행되면 터미널 에뮬레이터에 다음과 같은 텍스트가 표시됩니다.

```
0 1000 [Tmr Svc] Simple Link task created
Device came up in Station mode
1 2534 [Tmr Svc] Write certificate...
2 5486 [Tmr Svc] [ERROR] Failed to destroy object. PKCS11_PAL_DestroyObject failed.
3 5486 [Tmr Svc] Write certificate...
4 5776 [Tmr Svc] Security alert threshold = 15
5 5776 [Tmr Svc] Current number of alerts = 1
6 5778 [Tmr Svc] Running Demos.
7 5779 [iot_thread] [INFO] [DEMO][5779] -----STARTING DEMO-----
8 5779 [iot_thread] [INFO][INIT][5779] SDK successfully initialized.
Device came up in Station mode
[WLAN EVENT] STA Connected to the AP: afrlab-pepper , BSSID: 74:83:c2:b4:46:27
[NETAPP EVENT] IP acquired by the device
Device has connected to afrlab-pepper
Device IP Address is 192.168.36.176
9 8283 [iot_thread] [INFO ][DEMO][8282] Successfully initialized the demo. Network type
for the demo: 1
10 8283 [iot_thread] [INFO] OTA over MQTT demo, Application version 0.9.0
11 8283 [iot_thread] [INFO] Creating a TLS connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com:8883.
12 8852 [iot_thread] [INFO] Creating an MQTT connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com.
13 8914 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
14 8914 [iot_thread] [INFO] CONNACK session present bit not set.
15 8914 [iot_thread] [INFO] Connection accepted.
16 8914 [iot_thread] [INFO] Received MQTT CONNACK successfully from broker.
17 8914 [iot_thread] [INFO] MQTT connection established with the broker.
18 8915 [iot_thread] [INFO] Received: 0 Queued: 0 Processed: 0 Dropped: 0
19 8953 [OTA Agent T] [INFO] Current State=[RequestingJob], Event=[Start], New
state=[RequestingJob]
20 9008 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=3.
21 9015 [OTA Agent T] [INFO] SUBSCRIBED to topic $aws/things/__test_infra_thing73/jobs/
notify-next to broker.
22 9015 [OTA Agent T] [INFO] Subscribed to MQTT topic: $aws/things/
__test_infra_thing73/jobs/notify-next
23 9504 [MQTT Agent ] [INFO] Publishing message to $aws/things/__test_infra_thing73/
jobs/$next/get.
24 9535 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=2.
```

```

25 9535 [MQTT Agent ] [INFO] Ack packet deserialized with result: MQTTSuccess.
26 9536 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
27 9537 [OTA Agent T] [INFO] Sent PUBLISH packet to broker $aws/things/
__test_infra_thing73/jobs/$next/get to broker.
28 9537 [OTA Agent T] [WARN] OTA Timer handle NULL for Timerid=0, can't stop.
29 9537 [OTA Agent T] [INFO] Current State=[WaitingForJob], Event=[RequestJobDocument],
New state=[WaitingForJob]
30 9539 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=120.
31 9539 [MQTT Agent ] [INFO] De-serialized incoming PUBLISH packet:
DeserializedResult=MQTTSuccess.
32 9540 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
33 9540 [MQTT Agent ] [INFO] Received job message callback, size 62.
34 9616 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=execution
35 9616 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=execution.jobId
36 9617 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=execution.jobDocument
37 9617 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=execution.jobDocument.afr_ota
38 9617 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=execution.jobDocument.afr_ota.protocols
39 9618 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=execution.jobDocument.afr_ota.files
40 9618 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=filesize
41 9618 [OTA Agent T] [INFO] Failed job document content check: Required job document
parameter was not extracted: parameter=fileid
42 9619 [OTA Agent T] [INFO] Failed to parse JSON document as AFR_OTA job:
DocParseErr_t=7
43 9619 [OTA Agent T] [INFO] No active job available in received job document:
OtaJobParseErr_t=OtaJobParseErrNoActiveJobs
44 9619 [OTA Agent T] [ERROR] Failed to execute state transition handler: Handler
returned error: OtaErr_t=OtaErrJobParserError
45 9620 [OTA Agent T] [INFO] Current State=[WaitingForJob],
Event=[ReceivedJobDocument], New state=[CreatingFile]
46 9915 [iot_thread] [INFO] Received: 0 Queued: 0 Processed: 0 Dropped: 0
47 10915 [iot_thread] [INFO] Received: 0 Queued: 0 Processed: 0 Dropped: 0
48 11915 [iot_thread] [INFO] Received: 0 Queued: 0 Processed: 0 Dropped: 0
49 12915 [iot_thread] [INFO] Received: 0 Queued: 0 Processed: 0 Dropped: 0
50 13915 [iot_thread] [INFO] Received: 0 Queued: 0 Processed: 0 Dropped: 0
51 14915 [iot_thread] [INFO] Received: 0 Queued: 0 Processed: 0 Dropped: 0

```

## Microchip Curiosity PIC32MZEF 에 대해 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

### Note

마이크로칩과 일치하여 FreeRTOS 레퍼런스 통합 리포지토리 메인 브랜치에서 Curiosity PIC32MZEF (DM320104) 를 제거하고 더 이상 새 릴리스에서 수행하지 않을 것입니다. 마이크로칩은 [공식 공지](#)새로운 설계에는 PIC32MZEF (DM320104) 가 더 이상 권장되지 않습니다. PIC32MZEF 프로젝트와 소스 코드는 이전 릴리스 태그를 통해 계속 액세스할 수 있습니다. Microchip은 고객이 호기심을 사용할 것을 권장합니다.[Pic32MZ-EF-2.0 개발 보드 \(DM320209\)](#)새로운 디자인을 위해 Pic32mzv1 플랫폼은 여전히에서 찾을 수 있습니다.[v202012.00](#)의 FreeRTOS 참조 통합 저장소를 참조하십시오. 그러나 플랫폼은 더 이상에서 지원되지 않습니다.[v202107.00](#)FreeRTOS 레퍼런스의

### FreeRTOS OTA 데모 코드를 다운로드하려면

1. AWS IoT 콘솔로 이동하여 탐색 창에서 소프트웨어를 선택합니다.
2. UNDFreeRTOS 디바이스 소프트웨어, 선택다운로드 구성.

3. 소프트웨어 구성 목록에서 AWS IoT에 연결 - Microchip을 선택합니다. 구성 이름(다운로드 링크 아님)을 선택합니다.
4. 라이브러리에서 다른 라이브러리 추가를 선택한 후 OTA Updates(OTA 업데이트)를 선택합니다.
5. 디모 프로젝트에서 OTA Update(OTA 업데이트)를 선택합니다.
6. UND 이름 필수 사용자 지정 FreeRTOS 소프트웨어 구성에 대한 이름을 입력합니다.
7. 생성 후 다운로드를 선택합니다.

OTA 업데이트 디모 애플리케이션을 빌드하려면

1. 방금 다운로드한 .zip 파일의 압축을 풉니다.
2. [FreeRTOS 시작하기 \(p. 14\)](#)의 지침에 따라 aws\_demos 프로젝트를 MPLAB X IDE로 가져오고, AWS IoT 엔드포인트, Wi-Fi SSID 및 암호, 보드에 대한 프라이빗 키 및 인증서를 구성합니다.
3. 열기 `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` 파일을 입력하고 인증서를 입력합니다.

```
[ ] = "your-certificate-key";
```

4. 코드 서명 인증서의 내용을 여기에 붙여 넣습니다.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [ ] = "your-certificate-key";
```

와 동일한 형식을 따릅니다.`aws_clientcredential_keys.h`— 각 줄은 줄 바꿈 문자 ('\n')로 끝나고 인용 부호로 둑여 있어야 합니다.

예를 들어, 인증서는 다음과 비슷합니다.

```
"-----BEGIN CERTIFICATE-----\n"
"MIIBXTCCAOgAwIBAgIJAM4DeybZcTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n"
"FnRlc3Rf621nbmVyQGFtYXpvbi5jb20wHhcNMTcxMTkxODM1WhcNMTgxMTAz\n"
"MTkxODM2WjAhMR8wHQYDVQBZZZ0ZXN0X3NpZ25lckBhbWF6b24uY29tMFkwEwYH\n"
"KoZIZj0CAQYIKoZIZj0DAQcDQgAERavZfvwL1X+E4dIF7dbkVMu4IrJ1CAsFkc8\n"
"gZxPzn683H40XMK1tDZPEwr9ng78w9+QYQg7ygnr2stz8yhh06MkMCiWcYDVR0P\n"
"BAQDAgeAMBGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIF0R\n"
"r5cb7rEUNtWoVgd05MacrgOABfSoVYvBOK9fp63WAqt5h3BaS123coKSGg84twlq\n"
"TkO/pV/xEmyZmZdV+HxV/OM=\n"
"-----END CERTIFICATE-----\n";
```

5. [Python 3](#) 이상을 설치합니다.
6. `pip install pyopenssl`을 실행하여 `pyOpenSSL`을 설치합니다.
7. 코드 서명 인증서를 `demos/ota/bootloader/utility/codesigner_cert_utility/` 경로에 .pem 형식으로 복사합니다. 인증서 파일의 이름을 `aws_ota_codesigner_certificate.pem`으로 바꿉니다.
8. Open `freertos/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h`를 주석으로 처리합니다.`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`를 열고 를 정의합니다.`.CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` 또는 `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
9. 솔루션을 빌드하고 오류 없이 빌드되는지 확인합니다.
10. 터미널에 블레이터를 시작하고 다음 설정을 사용하여 보드에 연결합니다.
  - : 전송 속도 115200
  - 데이터 비트: 8
  - 패리티: None
  - 정지 비트: 1

11. 보드에서 디버거를 분리하고 보드에 대한 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] vDHCPPProcess: offer ac140a0eip
                                         1 36297 [IP-task] vDHCPPProcess: 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] vDHCPPProcess: offer c0a804beip
3 7230 [IP-task] vDHCPPProcess: 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 AWS IoT에 연결되고 OTA 업데이트에 필요한 MQTT 주제를 구독할 수 있음을 보여줍니다. 대기 중인 OTA 업데이트 작업이 없으므로 Missing job parameter 메시지가 필요합니다.

## Espressif ESP32 에 대해 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

1. FreeRTOS 소스 다운로드[GitHub](#). 자세한 내용은 [README.md](#) 파일을 참조하십시오. 모든 필요한 소스와 라이브러리가 포함된 IDE에서 프로젝트를 생성합니다.
2. [Espressif 시작하기](#)의 지침에 따라 필요한 GCC 기반 도구 체인을 설정합니다.
3. Open `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`를 주석으로 처리합니다.`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`를 열고 를 정의합니다.  
`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` 또는 `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
4. `vendors/espressif/boards/esp32/aws_demos` 디렉터리에서 `make`를 실행하여 데모 프로젝트를 빌드합니다. 데모 프로그램을 플래시하고 실행하여 출력을 확인할 수 있습니다.`make flash monitor`의 설명에 따라 [에스프레시프 시작하기](#).

5. OTA 업데이트 데모를 실행하기 전에:

- Open `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`를 주석으로 처리합니다.  
다.`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`를  
열고 를 정의합니다.`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED`또  
는`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
- Open `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` 및  
SHA-256/ECDSA 코드 서명 인증서를 다음 위치에 복사합니다.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

## Renesas RX65N에서 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

이 장에서는 Renesas RX65N에서 FreeRTOS OTA 데모 애플리케이션을 다운로드, 빌드, 플래시 및 실행하는 방법을 보여줍니다.

### 주제

- 운영 환경 설정 (p. 281)
- 사용자 설정AWS자원 (p. 281)
- 헤더 파일 가져오기, 구성, aws\_demos 및 boot\_loader 빌드 (p. 283)

## 운영 환경 설정

이 섹션의 절차에서는 다음 환경을 사용합니다.

- IDE: e<sup>2</sup>스튜디오 7.8.0, 이자형<sup>2</sup>2020-07
- 툴체인: CCRX 컴파일러 v3.0.1
- 대상 디바이스: RSKRX65N-2MB
- Debugger: E<sup>2</sup>, E<sup>2</sup>라이트 에뮬레이터
- 소프트웨어: 르네사스 플래시 프로그래머, 르네사스 시큐어 플래시 Programmer.exe, 테라 용어

### 하드웨어를 설정하려면

1. E Connect<sup>2</sup>RX65N 보드 및 PC에 대한 라이트 에뮬레이터 및 USB 직렬 포트.
2. 전원을 RX65N에 Connect.

## 사용자 설정AWS자원

1. FreeRTOS 데모를 실행하려면 AWS 계정이 있는 경우 AWS Identity and Access Management(IAM) 액세스 권한이 있는 사용자 AWS IoT 서비스. 아직 의 단계를 따르지 않았다면 [사용자 설정AWS계정 및 권한 \(p. 15\)](#).
2. OTA 업데이트를 설정하려면 다음 단계를 따르십시오. [OTA 업데이트 사전 조건 \(p. 140\)](#). 특히 다음 단계를 따르세요. [MQTT를 사용한 OTA 업데이트 사전 조건 \(p. 151\)](#).
3. [AWS IoT 콘솔](#)을 엽니다.
4. 왼쪽 탐색 창에서  관리를 선택한 다음 를 선택합니다. 사물 사물을 생성하려면

사물이란에 디바이스 또는 논리적 엔터티의 표현입니다. AWS IoT. 사물은 물리적 디바이스 또는 센서일 수 있습니다(예: 전구 또는 벽면 스위치). 또한 예연결하지 않는 애플리케이션 또는 물리적 엔터티 인스턴스와 같은 논리적 엔터티 인스턴스와 같은 논리적 엔터티 인스턴스와 같은 논리적 엔터티 AWS IoT를 (를) 사용하는 장치(예: 엔진 센서 또는 제어 패널이 장착된 차량) 와 관련이 있습니다. AWS IoT은 (는) 사물을 관리하는 데 도움이 되는 사물 래지스트리를 제공합니다.

- a. 선택생성을 선택한 다음 를 선택합니다 단일 사물 생성.
- b. 를 입력합니다. 이름사물, 그런 다음다음.
- c. [Create certificate]를 선택합니다.
- d. 생성된 세 개의 파일을 다운로드한 다음Activate.
- e. Attach a policy(정책 연결)를 선택합니다.

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	9dba40d984.cert.pem	<a href="#">Download</a>
A public key	9dba40d984.public.key	<a href="#">Download</a>
A private key	9dba40d984.private.key	<a href="#">Download</a>

You also need to download a root CA for AWS IoT:

A root CA for AWS IoT [Download](#)

[Activate](#)

[Cancel](#)

[Done](#)

[Attach a policy](#)

- f. 에서 만든 정책을 선택합니다. [디바이스 정책 \(p. 151\)](#).

MQTT를 사용하여 OTA 업데이트를 수신하는 각 장치는에 항목으로 등록되어야 합니다. AWS IoT 그리고 여기에는 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 리소스](#)에서 확인할 수 있습니다.

## 참고

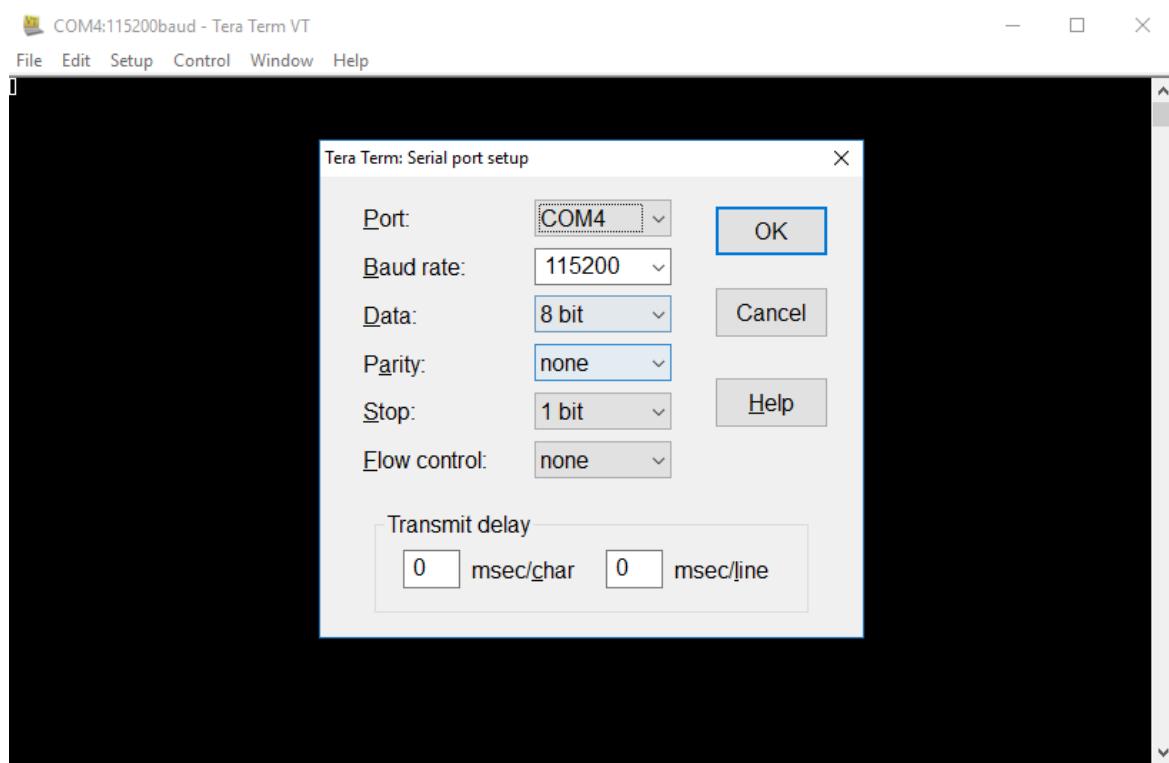
- `iot:Connect` 권한을 사용하면 MQTT를 통해 디바이스를 AWS IoT에 연결할 수 있습니다.
- AWS IoT 작업(.../jobs/\*) 항목에 대한 `iot:Subscribe` 및 `iot:Publish` 권한을 사용하면 연결된 디바이스에서 작업 알림과 작업 문서를 수신하고 작업 실행 완료 상태를 게시할 수 있습니다.
- AWS IoT OTA 스트림(.../streams/\*) 항목에 대한 `iot:Subscribe` 및 `iot:Publish` 권한을 사용하면 연결된 디바이스에서 AWS IoT로부터 OTA 업데이트 데이터를 가져올 수 있습니다. MQTT를 통해 펌웨어 업데이트를 수행하려면 이러한 권한이 필요합니다.
- `iot:Receive` 권한을 사용하면 AWS IoT Core에서 해당 주제에 대한 메시지를 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.

5. 코드 서명 프로파일을 생성하고 에 코드 서명 인증서를 등록하려면 AWS.

- a. 키 및 인증을 생성하려면 의 7.3 “OpenSSL 사용하여 ECDSA-SHA256 키 페어 생성”을 참조하십시오. [르네사스 MCU 펌웨어 업데이트 설계 정책](#).

- b. AWS IoT 콘솔을 엽니다. 왼쪽 탐색 창에서 관리그런 다음작업. Select작업 만들기그때OTA 업데이트 Job 생성.
- c. Under업데이트할 장치 선택고르다Select그런 다음 이전에 만든 항목을 선택합니다. 다음을 선택합니다.
- d. 온FreeRTOS OTA 업데이트 작업 생성페이지에서 다음을 수행합니다.
  - i. 용펌웨어 이미지 전송을 위한 프로토콜 선택, 선택MQTT.
  - ii. 용펌웨어 이미지 선택 및 서명, 선택새 펌웨어 이미지에 서명합니다.
  - iii. 용코드 서명 프로필, 선택생성.
  - iv. 에서코드 서명 프로필 생성창, a를 입력합니다.프로필 이름. 이디바이스 하드웨어 플랫폼을 선택합니다Windows 시뮬레이터. 이코드 서명 인증서고르다가져오기.
  - v. 인증서를 찾아서 선택합니다 (secp256r1.crt), 인증서 개인 키 (secp256r1.key) 및 인증서 체인 (ca.crt).
  - vi. 를 입력합니다.디바이스의 코드 서명 인증서의 경로명. 그런 다음 생성(Create)을 선택합니다.
6. 에 대한 코드 서명에 대한 액세스 권한을 부여하려면 AWS IoT를 따르세요. [Code Signing for AWS IoT에 대한 액세스 권한 부여 \(p. 149\)](#).

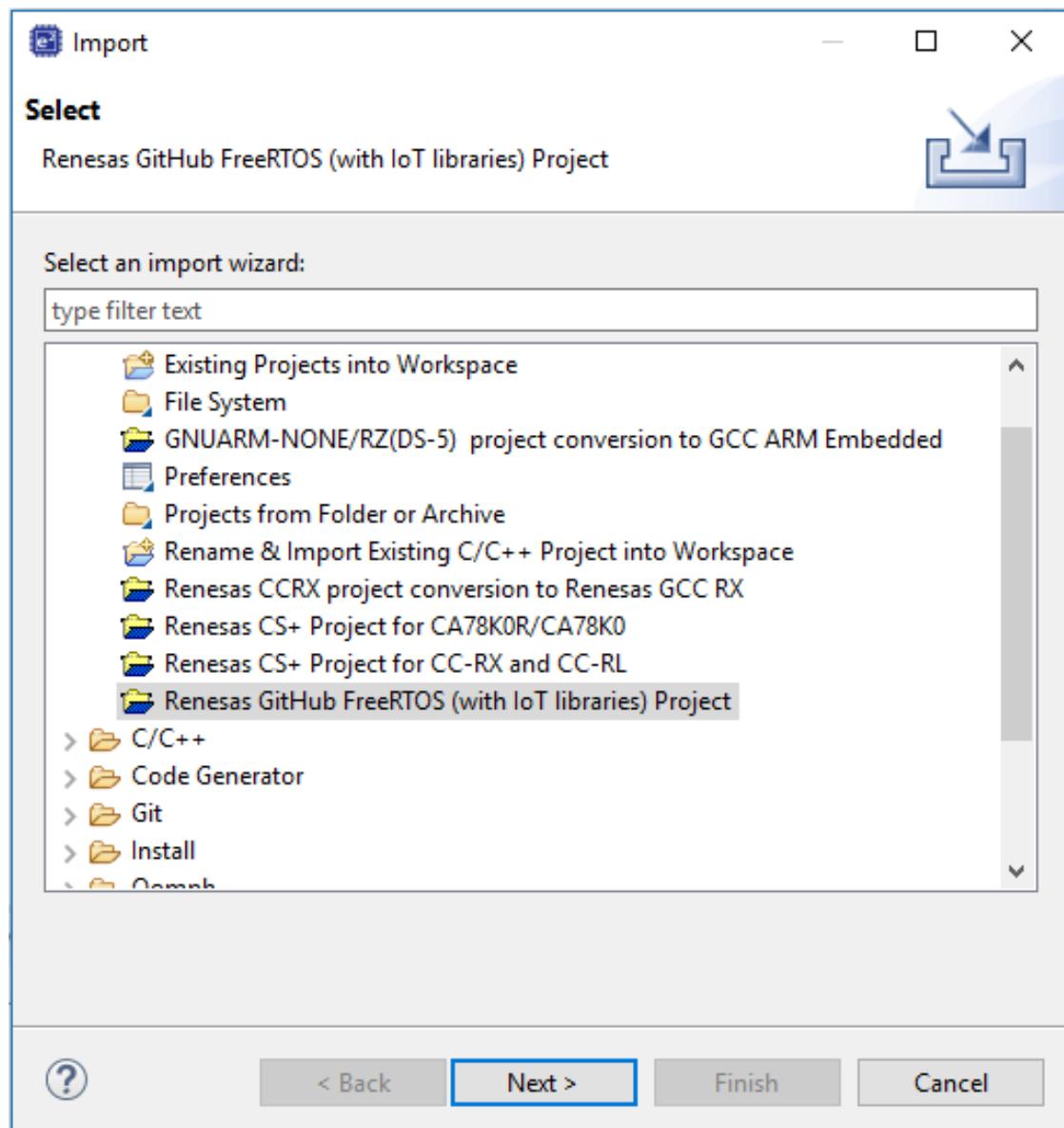
PC에 Tera Tera Term이 설치되어 있지 않은 경우에서 다운로드할 수 있습니다. <https://ttssh2.osdn.jp/index.html.en> 다음과 같이 설정합니다. 디바이스에서 PC로 USB 직렬 포트를 연결했는지 확인합니다.



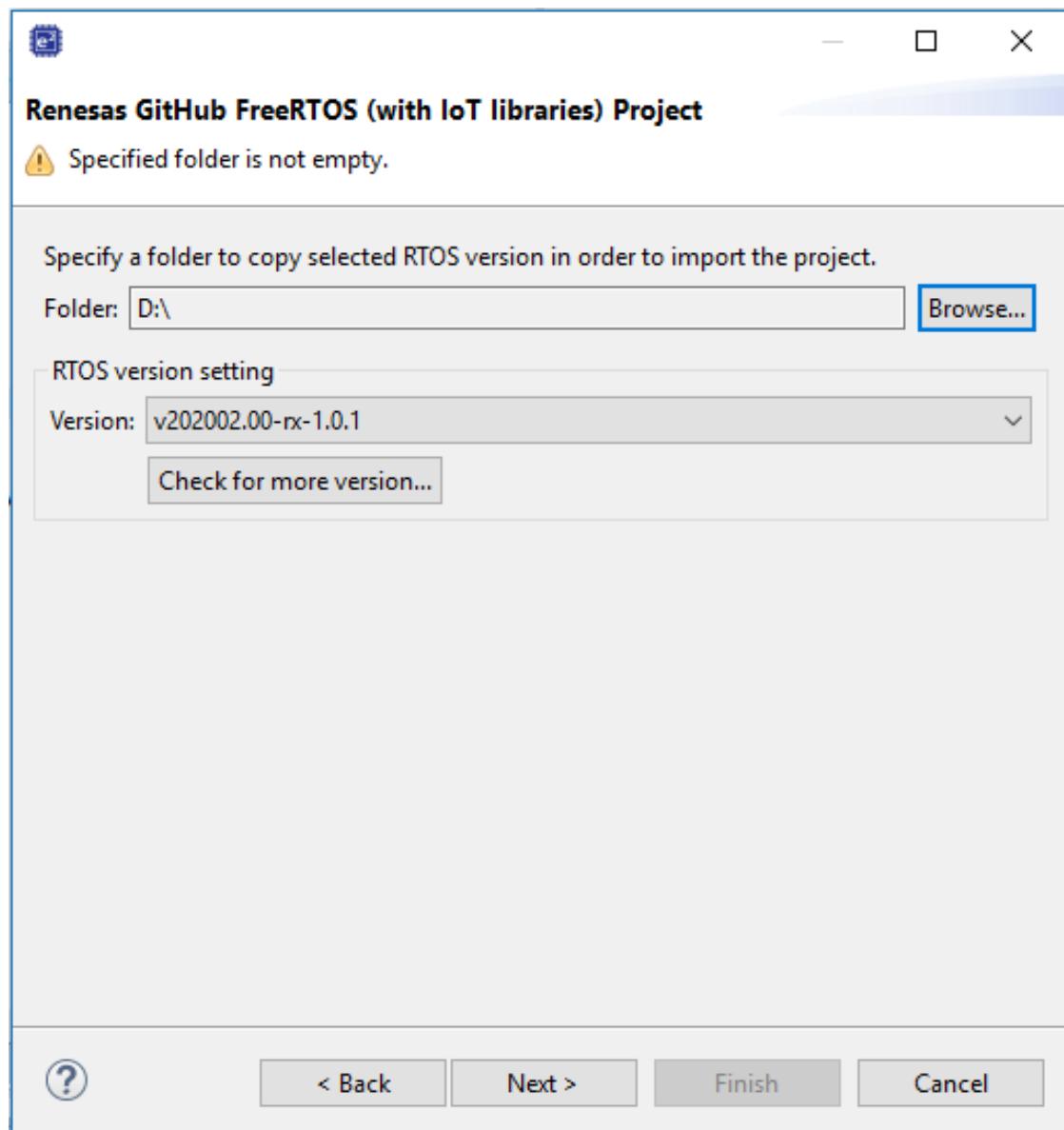
## 헤더 파일 가져오기, 구성, aws\_demos 및 boot\_loader 빌드

시작하려면 FreeRTOS 패키지의 최신 버전을 선택하면 GitHub에서 다운로드되고 자동으로 프로젝트로 가져옵니다. 이렇게 하면 FreeRTOS 구성 및 애플리케이션 코드 작성에 집중할 수 있습니다.

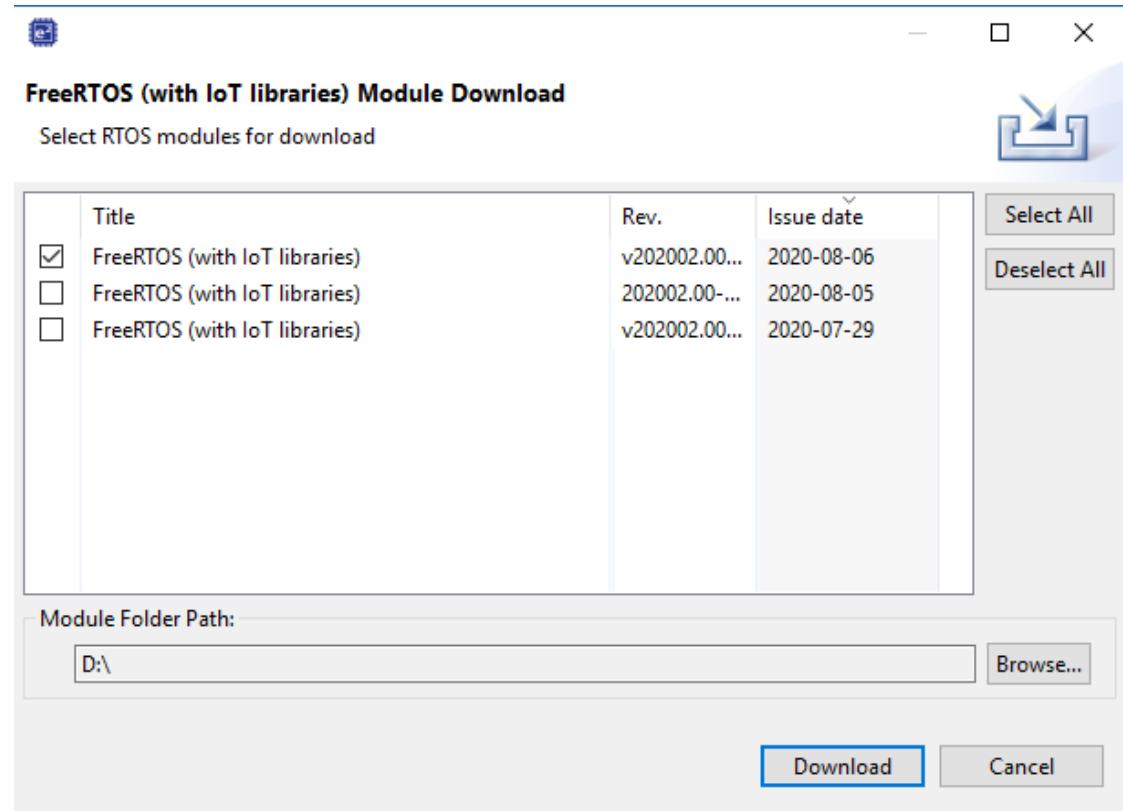
1. 실행 e<sup>2</sup>스튜디오.
2. 선택File를 선택한 다음 를 선택합니다. 가져오기....
3. Select르네사스 GitHub FreeRTOS (IoT 라이브러리 포함) 프로젝트.



- 선택된 많은 버전 확인...다운로드 대화 상자를 표시합니다.



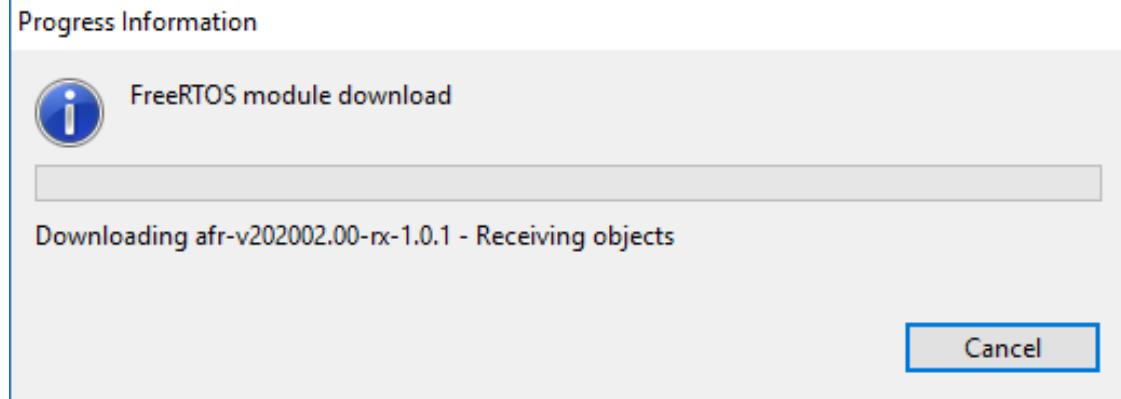
5. 최신 패키지를 선택합니다.



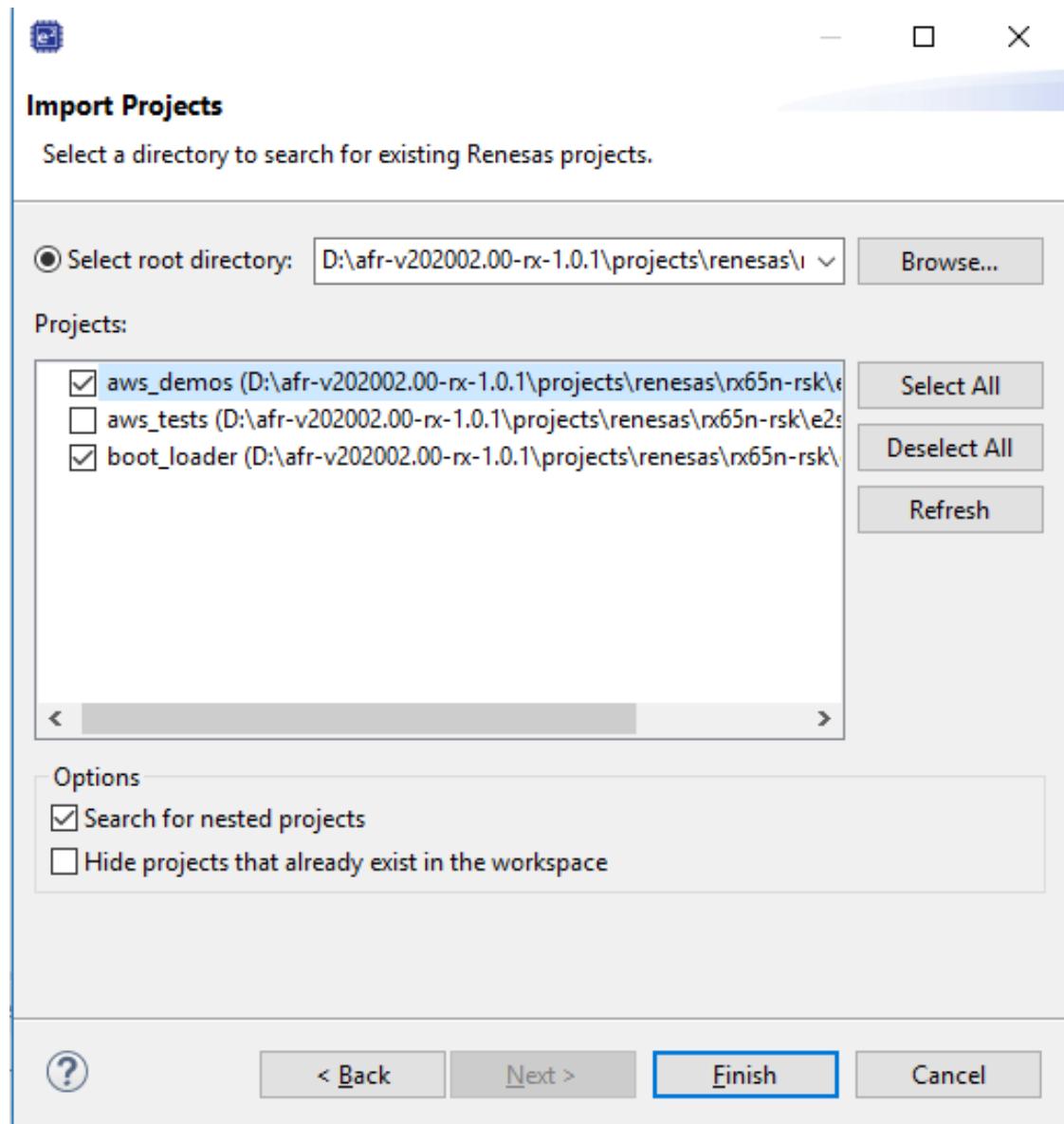
6. 선택동의최종 사용자 라이선스 계약에 동의합니다.



7. 다운로드가 완료될 때까지 기다립니다.

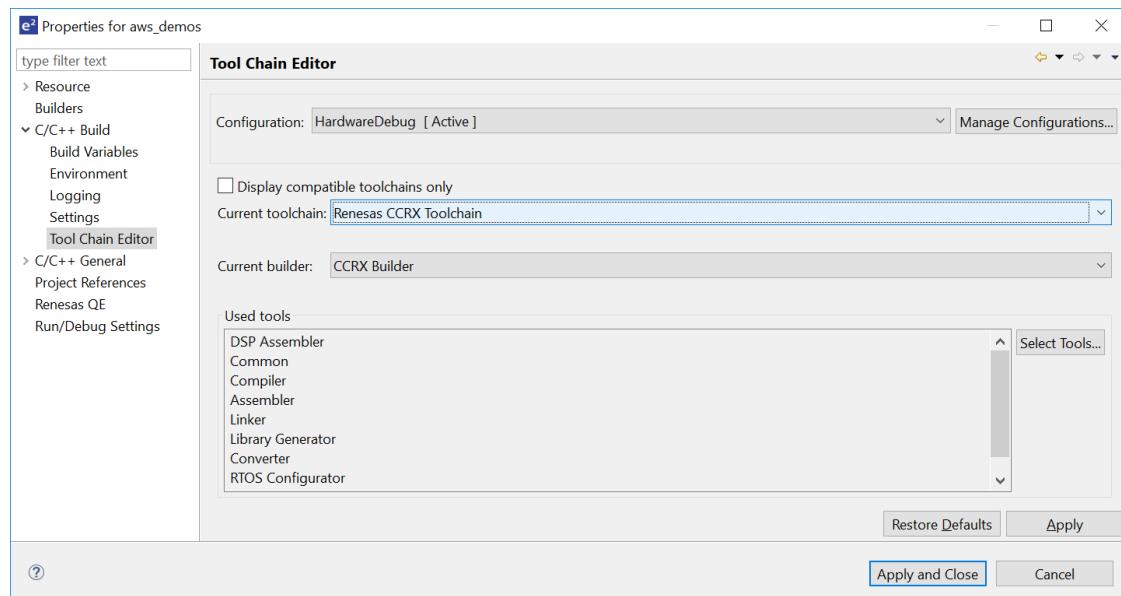


8. Selectaws\_demos과부트\_로더프로젝트를 선택한 다음Finish를 사용하여 가져올 수 있습니다.

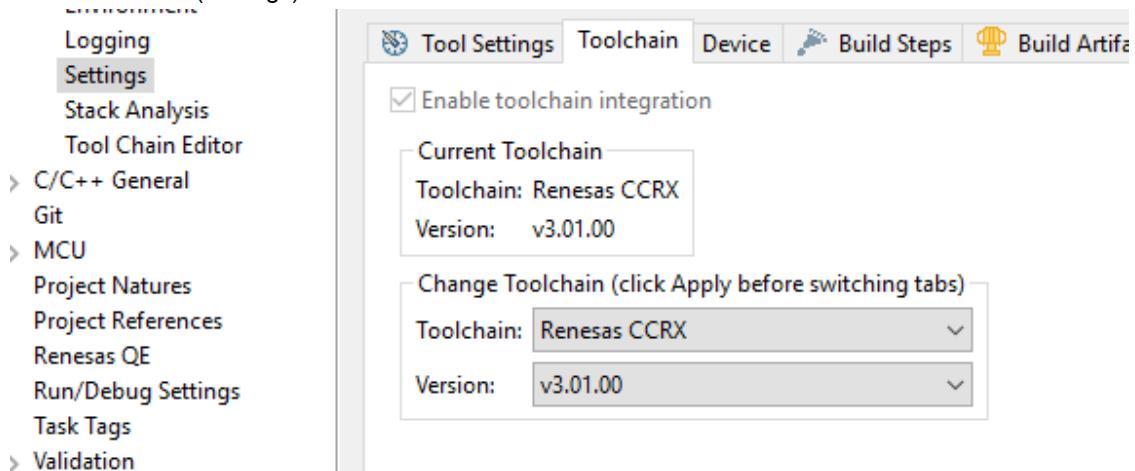


9. 두 프로젝트 모두 프로젝트 속성을 엽니다. 탐색 창에서 []를 체인 편집기.

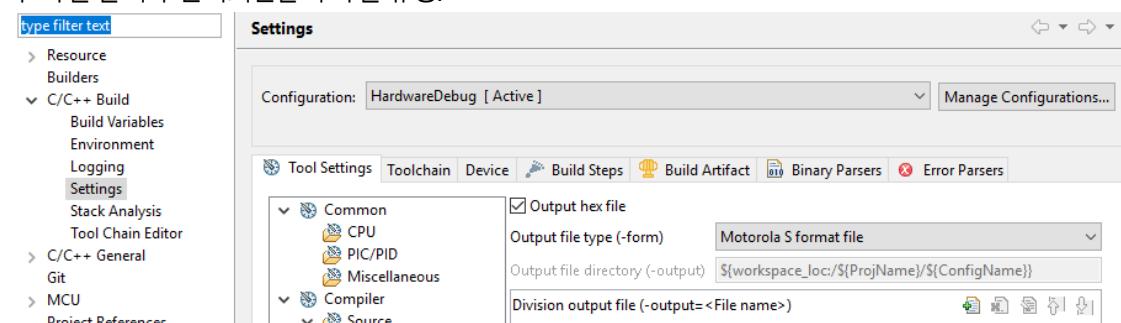
- Select 현재 도구 체인.
- Select 현재 빌더.

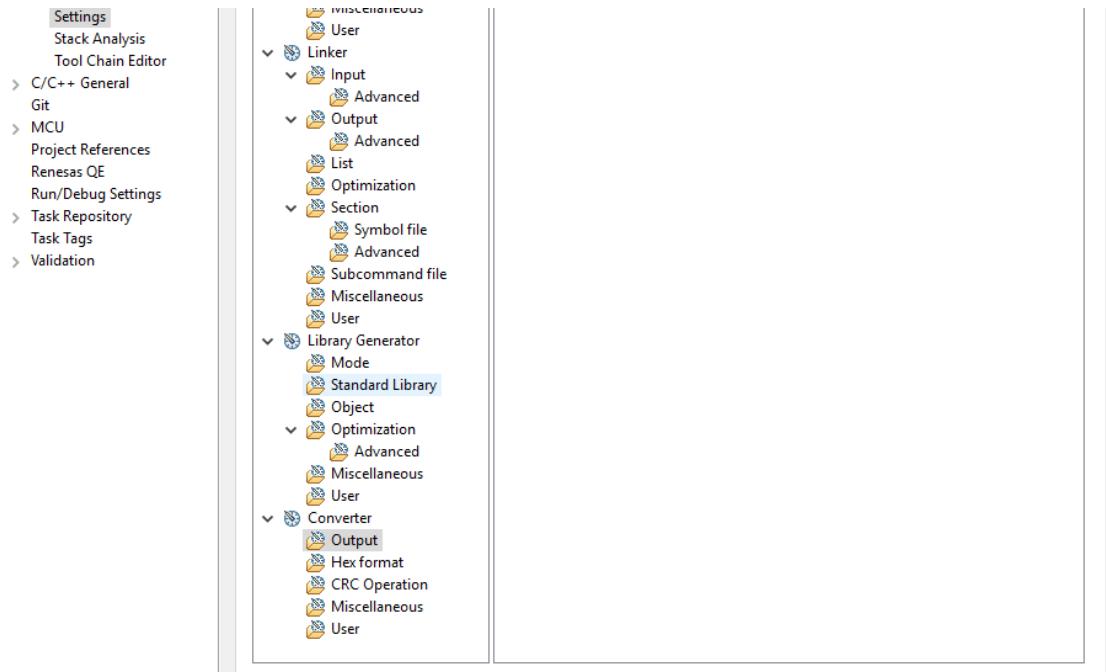


10. 탐색 창에서 설정(Settings)을 선택합니다. Select 툴체인탭을 클릭한 다음 툴체인을 선택합니다. Version.



Select 도구 설정탭, Export 변환기 그린 다음 선택을 선택합니다. 출력. 기본 창에서 확인하십시오. 16진수 파일 출력이 선택되면 출력 파일 유형.





11. 부트로더 프로젝트에서 `projects\renesas\rx65n-rsk\boot_loader\src\key\code_signer_public_key.h` 퍼블릭 키를 입력합니다. 퍼블릭 키를 생성하는 방법에 대한 자세한 내용은 단원을 참조하십시오. [RX65N 기반 Amazon Web Services 사용하여 FreeRTOS OTA를 구현하는 방법 및 섹션 7.3 “OpenSSL 사용하여 ECDSA-SHA256 키 페어 생성”](#) 르네사스 MCU 펌웨어 업데이트 설계 정책.

```

2  /* DISCLAIMER */
3  /* File Name : code_signer_public_key.h */
4  /* History : DD.MM.YYYY Version Description */
5
6  #ifndef CODE_SIGNER_PUBLIC_KEY_H_
7  #define CODE_SIGNER_PUBLIC_KEY_H_
8
9  /*
10   * PEM-encoded code signer public key.
11   *
12   * Must include the PEM header and footer:
13   * -----BEGIN CERTIFICATE-----\n"
14   * "...base64 data...\n"
15   * -----END CERTIFICATE-----"
16   */
17  // #define CODE_SIGNER_PUBLIC_KEY_PEM "Paste code signer public key here."
18  #define CODE_SIGNER_PUBLIC_KEY_PEM \
19      -----BEGIN PUBLIC KEY-----\n" \
20      "MFkwEwYHKoZIzj0DQoCDoGAEhVxq1tUJZ5LXmrur1mTTQzijLtQ"\n" \
21      "sz9cj31BZl89nyhmB13Ukaolv4/aWaa6fTuBPVeaiyEwJeQ7YBpYGC9iA=="\n" \
22      -----END PUBLIC KEY-----"
23
24  extern const uint8_t code_signer_public_key[];
25  extern const uint32_t code_signer_public_key_length;
26
27  #endif /* CODE_SIGNER_PUBLIC_KEY_H_ */

```

그런 다음 만들 프로젝트를 빌드합니다. `boot_loader.mot`.

12. 열기 `aws_demos` 프로젝트.
- AWS IoT 콘솔을 엽니다.
  - 왼쪽 탐색 창에서 설정을 선택합니다. 예 사용자 지정 앤드포인트를 기록해 둡니다 디바이스 데이터 앤드포인트 텍스트 상자.
  - 선택 관리를 선택한 다음 를 선택합니다. 사물. 를 기록해 둡니다. AWS IoT 보드의 사물 이름입니다.
  - 에서 `aws_demos` 프로젝트, 오픈 `demos/include/aws_clientcredential.h` 를 클릭하고 다음 값을 지정합니다.

```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```

```
28
29
30     */
31     * @brief MQTT Broker endpoint.
32     *
33     * @todo Set this to the fully-qualified DNS name of your MQTT broker.
34     */
35     #define clientcredentialMQTT_BROKER_ENDPOINT      "xxxxx-ats.iot.ap-northeast-1.amazonaws.com"
36
37     */
38     * @brief Host name.
39     *
40     * @todo Set this to the unique name of your IoT Thing.
41     */
42     #define clientcredentialIOT_THING_NAME          "thingname"
```

- e. tools/certificate\_configuration/CertificateConfigurator.html 파일을 엽니다.
- f. 이전에 다운로드한 인증서 PEM 파일과 프라이빗 키 PEM 파일을 가져옵니다.
- g. 선택aws\_clientcredential\_keys.h 생성 및 저장에서 이 파일을 대체합니다.demos/include/디렉터리.

## Certificate Configuration Tool

FreeRTOS Developer Demos

Provide client certificate and private key PEM files downloaded from the AWS IoT Console.

**Certificate PEM file:**

No file chosen

**Private Key PEM file:**

No file chosen

**⚠ Save the generated header file to the demos/common/include folder of the demo project.**

Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

- h. 열기|vendors/renesas/boards/rx65n-rsk/aws\_demos/config\_files/ota\_demo\_config.h파일을 선택하고 이러한 값을 지정합니다.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

어디에 #####-#파일의 값입니다.secp256r1.crt. 인증의 각 줄 뒤에 "\"를 추가해야 합니다. 생성에 대한 자세한 내용은 secp256r1.crt 파일, 참조RX65N 기반 Amazon Web Services 사용하여 FreeRTOS OTA를 구현하는 방법 및 섹션 7.3 “OpenSSL 사용하여 ECDSA-SHA256 키 페어 생성”르네사스 MCU 펌웨어 업데이트 설계 정책.

```

Project Explorer
aws_demos [HardwareDebug]
    Includes
    application_code
    config_files
        demos
            defender
            demo_runner
            dev_mode_key_provisioning
            greengrass_connectivity
            https
        include
            aws_application_version.h
            aws_clientcredential_keys.h
            aws_clientcredential.h
            aws_demo.h
            aws_iot_demo_network.h
            aws_ota_codesigner_certificate.h
            iot_config_common.h
            iot_demo_logging.h
            iot_demo_runner.h
            mqtt
            network_manager
            ota
            shadow
            tcp
aws_ota_codesigner_certificate.h
2
3 * FreeRTOS V202002.00
4
5 #ifndef __AWS_CODESIGN_KEYS_H__
6 #define __AWS_CODESIGN_KEYS_H__
7
8 /*
9  * PEM-encoded code signer certificate
10 *
11  * Must include the PEM header and footer:
12  * "-----BEGIN CERTIFICATE-----\n"
13  * "...base64 data...\n"
14  * "-----END CERTIFICATE-----\n";
15  */
16 static const char signingcredentialsIGNING_CERTIFICATE_PEM[] =
17 "-----BEGIN CERTIFICATE-----\n"
18 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
19 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
20 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
21 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
22 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
23 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
24 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
25 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
26 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
27 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
28 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
29 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
30 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
31 "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
32 "-----END CERTIFICATE-----\n";
33 #endif

```

### 13. 작업 A: 초기 펌웨어 버전 설치

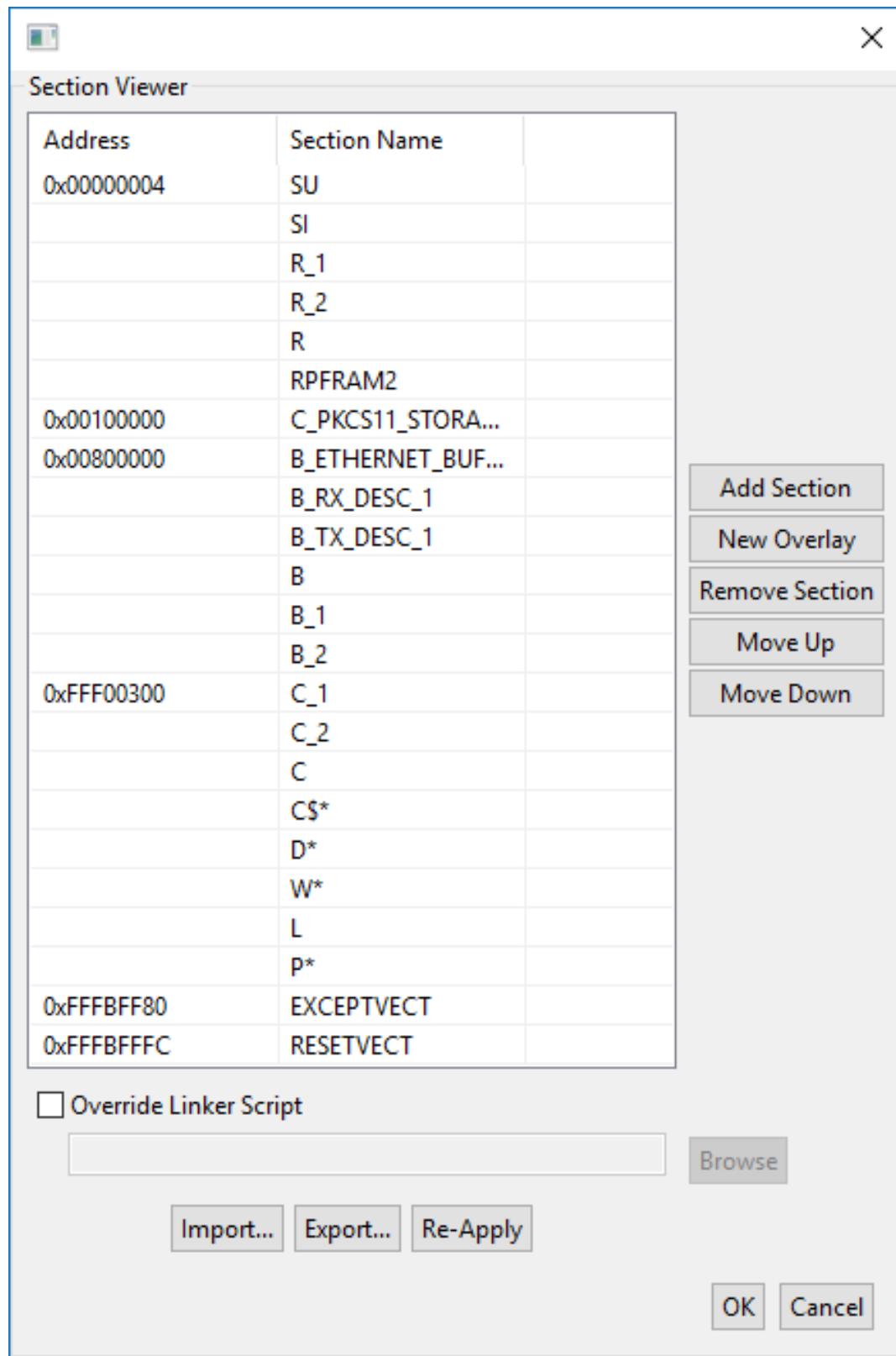
- 열기|vendors/renesas/boards/board/aws\_demos/config\_files/aws\_demo\_config.h 파일, 주석으로 처리합니다.#define CONFIG\_CORE\_MQTT\_MUTUAL\_AUTH\_DEMO\_ENABLED 다음 줄 하나를 정의합니다.CONFIG\_OTA\_MQTT\_UPDATE\_DEMO\_ENABLED 또는 CONFIG\_OTA\_HTTP\_UPDATE\_DEMO\_ENABLED.
- 열기|demos/include/ aws\_application\_version.h 파일을 지정하고 펌웨어의 초기 버전을 다음과으로 설정합니다.0.9.2.

```

25
26 #ifndef __AWS_APPLICATION_VERSION_H__
27 #define __AWS_APPLICATION_VERSION_H__
28
29 #include "iot_appversion32.h"
30 extern const AppVersion32_t xAppFirmwareVersion;
31
32 #define APP_VERSION_MAJOR      0
33 #define APP_VERSION_MINOR      9
34 #define APP_VERSION_BUILD      2
35
36 #endif
37

```

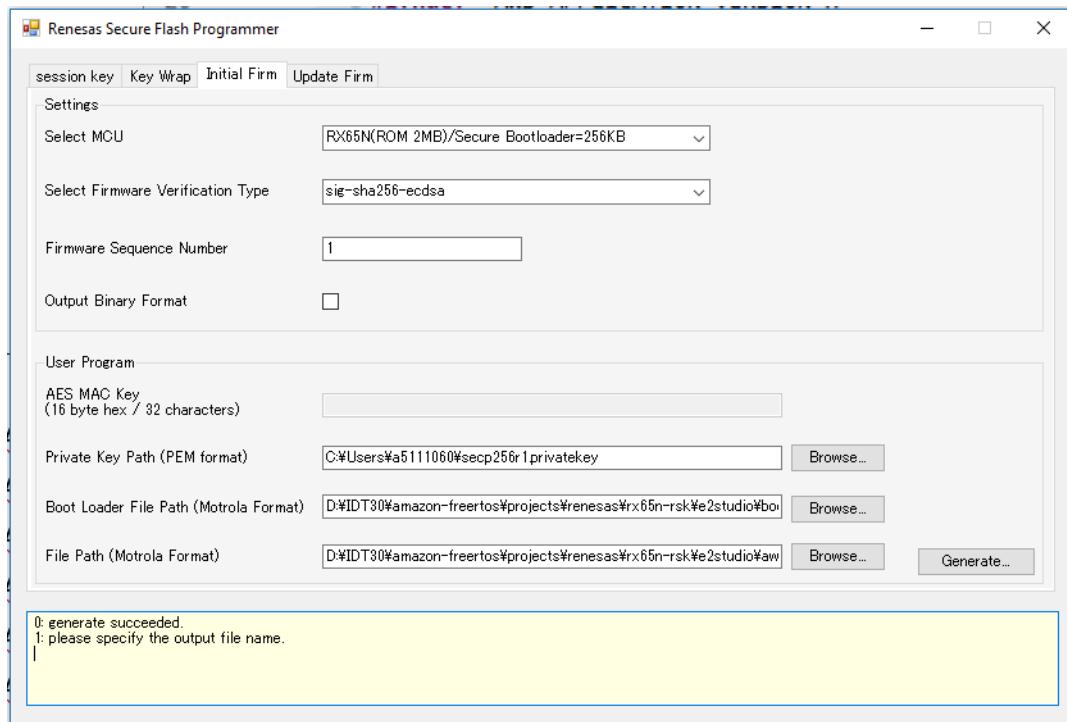
- 에서 다음 설정을 변경합니다.섹션 최종 사용자.



d. 선택빌드를 생성하려면aws\_demos.mot파일.

14. 파일 생성 userprog.mot 르네사스 시큐어 플래시 프로그래머와 함께 userprog.mot는 다음과 같은 조합입니다. aws\_demos.mot 과 boot\_loader.mot. 이 파일을 RX65N-RSK 로 플래시하여 초기 펌웨어를 설치할 수 있습니다.

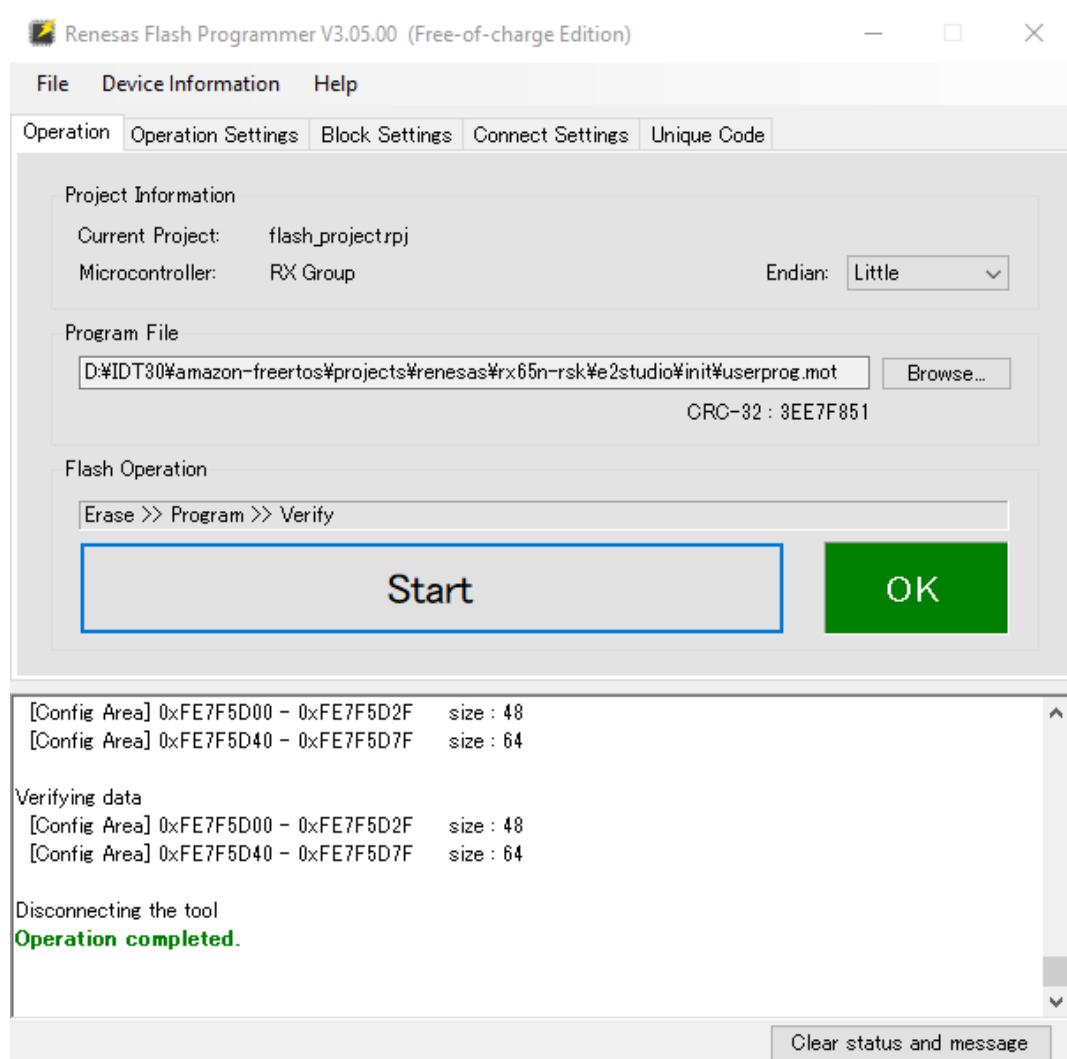
- 다운로드 <https://github.com/renesas/Amazon-FreeRTOS-Tools>를 열려면 Renesas Secure Flash Programmer.exe.
- Select 최초 회사를 탭한 후 다음 파라미터를 설정합니다.
  - 프라이빗 키 경로—의 위치입니다. secp256r1.privatekey.
  - 부트로더 파일 경로—의 위치입니다. boot\_loader.mot(projects\renesas\rx65n-rsk\ e2studio\boot\_loader\HardwareDebug).
  - 파일 경로—의 위치입니다. aws\_demos.mot(projects\renesas\rx65n-rsk\ e2studio\aws\_demos\HardwareDebug).



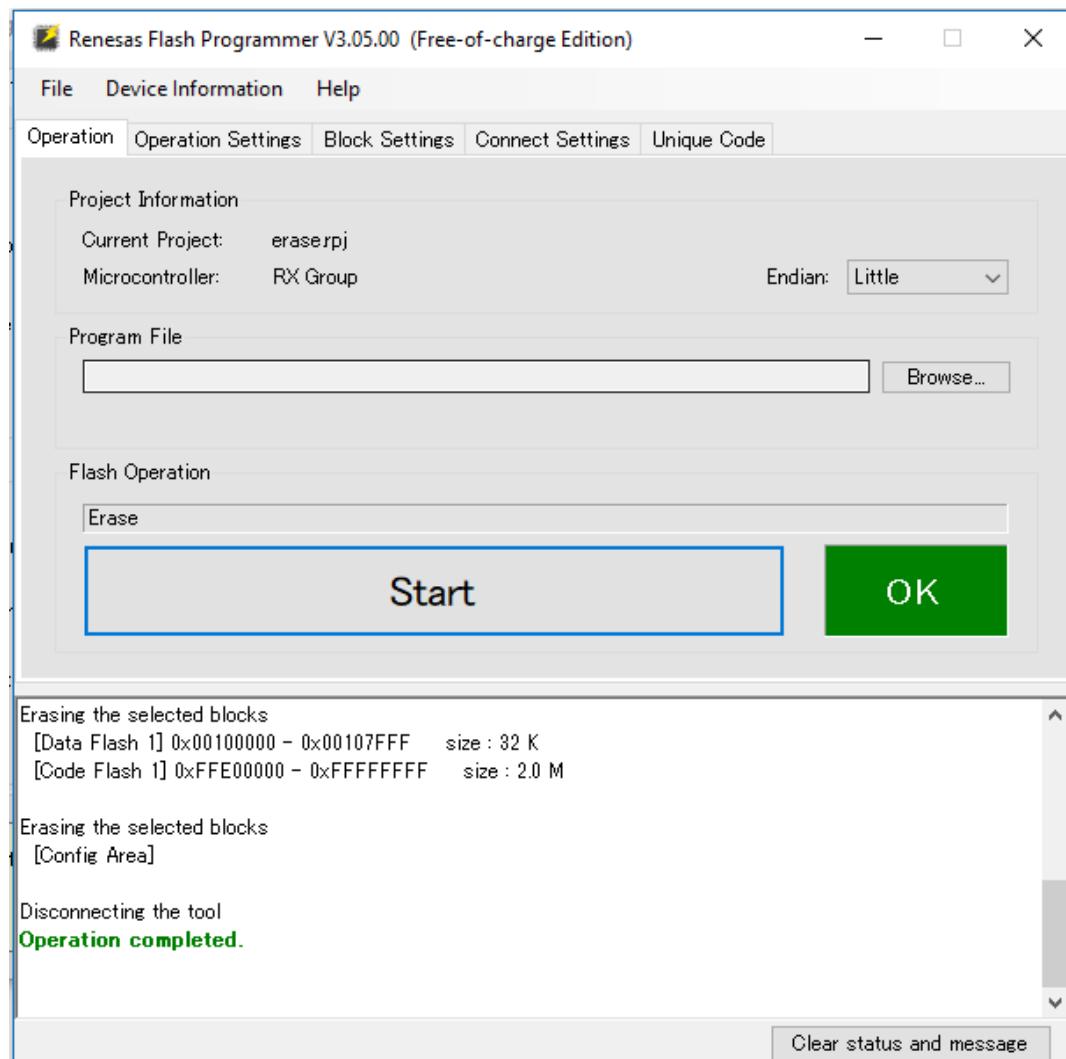
- 이름이 인 디렉터리를 생성합니다. init\_firmware를 생성하려면 userprog.mot 을 (를) init\_firmware 디렉터리. 생성이 성공했는지 확인합니다.

15. RX65N-RSK에서 초기 펌웨어를 플래시합니다.

- 최신 버전의 Renesas Flash 프로그래머 (프로그래밍 GUI)를 다운로드하십시오 <https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html>.
- 열기|vendors\renesas\rx\_mcu\_boards\boards\rx65n-rsk\aws\_demos\flash\_project\erase\_from\_bank\ erase.rpj 파일을 사용하여 응행에서 데이터를 지웁니다.
- 선택 Start 을 행을 지울 수 있습니다.



- d. 플래시하려면 userprog.mot, 선택찾아보기...를 탐색하고 로 이동합니다.init\_firmware디렉터리에서 userprog.mot파일 및 선택Start.



16. RX65N-RSK에 펌웨어의 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] vDHCPPProcess: offer c0a80a09ip
14 3597 [ETHER_RECEI] Heap: current 233200 lowest 233000
15 3597 [IP-task] vDHCPPProcess: 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.

```

```

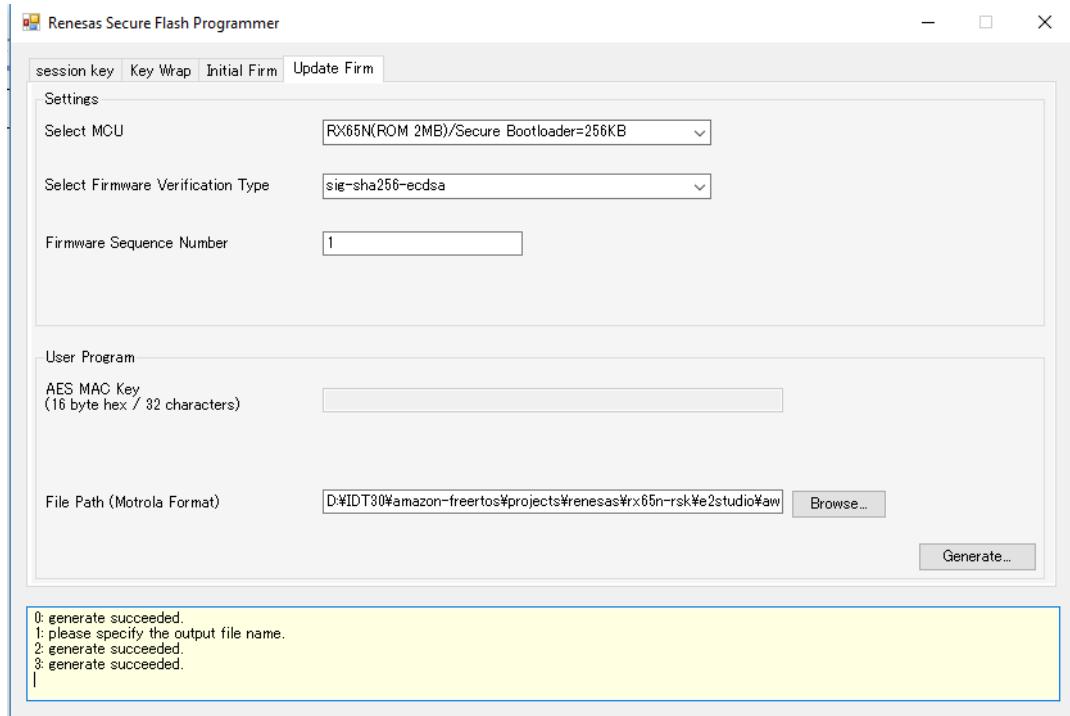
67 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8, SUBSCRIBE operation
818c48) Waiting for operation completion.
68 7436 [ETHER_RECEI] Heap: current 128248 lowest 127992
69 7480 [OTA Agent T] [INFO ][MQTT][7480] (MQTT connection 81cfc8, SUBSCRIBE operation
818c48) Wait complete with result 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. 열기 `demos/include/aws_application_version.h` 파일 및 종분 APP\_VERSION\_BUILD 토큰 값을 다음으로 지정합니다. 0.9.3.
- b. 프로젝트를 다시 빌드합니다.
- 생성 `userprog.rsu` 펌웨어 버전을 업데이트 하려면 Renesas Secure Flash 프로그래머와 함께 파일을 사용하십시오.
  - a. Amazon-FreeRTOS-Tools\Renesis Secure Flash Programmer.exe 파일을 엽니다.
  - b. Select 회사 업데이트(를) 탭하고 다음 파라미터를 설정합니다.

- 파일 경로의 위치입니다.`.aws_demos.mot` 파일 (`projects\renesas\rx65n-rsk\ e2studio\aws_demos\HardwareDebug`).
- c. `update_firmware`이라는 디렉터리를 생성합니다. 생성 `userprog.rsu` 그리고 그것을 `update_firmware` 디렉터리에 생성이 성공했는지 확인합니다.



19. 펌웨어 업데이트를 업로드합니다.`.userproj.rsu`에 설명된 대로 Amazon S3 버킷에 연결합니다.[업데이트를 저장할 Amazon S3 버킷 생성 \(p. 141\)](#).

Name	Last modified	Size
SignedImages	--	--
userprog.rsu	May 18, 2020 2:00:37 PM GMT+0900	767.5 KB

20. RX65N-RSK에서 펌웨어를 업데이트하는 작업을 생성합니다.

AWS IoTJobs는 하나 이상의 연결된 장치에 보류 중임을 알리는 서비스입니다.[작업](#). 작업을 사용하여 장치 집합을 관리하고 장치의 펌웨어 및 보안 인증서를 업데이트하거나 장치 재시작 및 진단 수행과 같은 관리 작업을 수행할 수 있습니다.

- [AWS IoT 콘솔](#)에 로그인합니다. 탐색 창에서  관리를 선택하고 작업.
- 선택 작업 만들기를 선택한 다음 를 선택합니다 OTA 업데이트 작업 생성. 사물을 선택한 다음 다음.
- 다음과 같이 FreeRTOS OTA 업데이트 작업을 만듭니다.

- 선택MQTT.
  - 이전 단원에서 만든 코드 서명 프로필을 선택합니다.
  - Amazon S3 버킷에 업로드한 펌웨어 이미지를 선택합니다.
  - 용디바이스에 있는 펌웨어 이미지의 경로명을 입력합니다.**test**.
  - 이전 단원에서 생성한 IAM 역할을 선택합니다.
- d. 다음(Next)을 선택합니다.

MQTT

Select and sign your firmware image

Code signing ensures that devices only run code published by trusted authors and that the code has not been altered or corrupted since it was signed. You have three options for code signing. [Learn more](#)

Sign a new firmware image for me

Select a previously signed firmware image

Use my custom signed firmware image

Code signing profile [Learn more](#)

ota_signing	SHA256	ECDSA	aaaaaaaa	<a href="#">Clear</a> <a href="#">Change</a>
-------------	--------	-------	----------	--

Select your firmware image in S3 or upload it

userprog.rsu	<a href="#">Change</a>
--------------	------------------------

Pathname of firmware image on device [Learn more](#)

test
------

IAM role for OTA update job

Choose a role which grants AWS IoT access to the S3, AWS IoT jobs and AWS Code signing resources to create an OTA update job. [Learn more](#)

Role (requires S3 access)

ota_test_beginner	<a href="#">Select</a>
-------------------	------------------------

[Cancel](#) [Back](#) [Next](#)

e. ID를 입력한 다음생성.

21. Tera Term을 다시 열어 펌웨어가 OTA 데모 버전 0.9.3으로 성공적으로 업데이트되었는지 확인합니다.

```

11 10710 [Thru Svc] The network is up and running
22 10710 [Thru Svc] Write certificate...
23 10752 [ETHER RECEI] Heap: current 232336 lowest 232136
24 11652 [ETHER RECEI] Heap: current 226352 lowest 225952
25 12405 [iot_thread] [INFO ][DEMO][12405] -----STARTING DEMO-----

26 12405 [iot_thread] [INFO ][INIT][12405] SDK successfully initialized.
27 12405 [iot_thread] [INFO ][DEMO][12405] Successfully initialized the demo. Network type for the demo: 4
28 12405 [iot_thread] [INFO ][MQTT][12405] MQTT library successfully initialized.
29 12405 [iot_thread] [INFO ][DEMO][12405] OTA demo version 0.9.3

30 12405 [iot_thread] [INFO ][DEMO][12405] Connecting to broker...

31 12405 [iot_thread] [INFO ][DEMO][12405] MQTT demo client identifier is rx65n-gr-rose (length 13).

```

22. 온AWS IoT콘솔, 작업 상태가 인지 확인합니다.성공.

The screenshot shows the AWS IoTOTA (OTA) Jobs page. At the top, it displays the job name "AFR\_OTA-demo\_test" and the status "COMPLETED". Below this, there are two sections: "Overview" and "Details". The "Overview" section includes a table with metrics: Queued (0), In progress (0), Timed out (0), Failed (0), Succeeded (1), Rejected (0), Canceled (0), and Removed (0). The "Details" section shows a table with a single row: "rx65n-gr-rose" with a status of "Succeeded" and last updated at "Jun 3, 2020 4:48:33 PM +0900". There are also "All Statuses" and "Refresh" buttons.

## 자습서: FreeRTOS 블루투스 저에너지를 사용하여 Espressif ESP32에서 OTA 업데이트 수행

이 자습서에서는 Android 기기에서 MQTT 블루투스 저에너지 프록시에 연결된 Espressif ESP32 마이크로 컨트롤러를 업데이트하는 방법을 보여줍니다. 다음을 사용하여 장치를 업데이트합니다. AWS IoTOTA (OTA) 업데이트 작업. 장치가 다음에 연결됩니다. AWS IoTAndroid 데모 앱에 입력된 Amazon Cognito 자격 증명 사용 공인 운영자가 클라우드에서 OTA 업데이트를 시작합니다. Android 데모 앱을 통해 기기가 연결되면 OTA 업데이트가 시작되고 장치에서 펌웨어가 업데이트됩니다.

FreeRTOS 버전 2019.06.00 주요 이상에는 Wi-Fi 프로비저닝 및 보안 연결에 사용할 수 있는 블루투스 저에너지 MQTT 프록시 지원이 포함되어 있습니다. AWS IoT 서비스. Bluetooth 저에너지 기능을 사용하면 Wi-Fi 없이도 모바일 장치에 페어링할 수 있는 저전력 장치를 구축할 수 있습니다. 기기는 GAP (일반 액세스 프로파일) 및 GATT (일반 속성) 프로파일을 사용하는 Android 또는 iOS 블루투스 저에너지 SDK를 통해 연결하여 MQTT를 사용하여 통신할 수 있습니다.

다음은 블루투스 저에너지에 대한 OTA 업데이트를 허용하기 위해 따라야 할 단계입니다.

- 스토리지 구성: Amazon S3 버킷 및 정책을 생성하고 업데이트를 수행할 수 있는 IAM 사용자를 구성합니다.
- 코드 서명 인증서를 생성합니다. 서명 인증서를 만들고 IAM 사용자가 펌웨어 업데이트에 서명하도록 허용합니다.
- Amazon Cognito 인증을 구성합니다. 사용자 풀에 대한 자격 증명 공급자, 사용자 풀 및 응용 프로그램 액세스를 만듭니다.
- FreeRTOS 구성: Bluetooth 저에너지, 클라이언트 자격 증명 및 코드 서명 공개 인증서를 설정합니다.
- Android 앱을 구성합니다. 자격 증명 공급자, 사용자 풀을 설정하고 애플리케이션을 Android 기기에 배포합니다.
- OTA 업데이트 스크립트를 실행합니다. OTA 업데이트를 시작하려면 OTA 업데이트 스크립트를 사용합니다.

업데이트 작동 방식에 대한 자세한 내용은 단원을 참조하십시오. [FreeRTOS 무선 업데이트 \(p. 140\)](#). 블루투스 저에너지 MQTT 프록시 기능을 설정하는 방법에 대한 자세한 내용은 게시물을 참조하십시오. [에프레시프 ESP32에서 FreeRTOS와 함께 블루투스 저에너지 사용](#) 리처드 강.

## 사전 조건

이 자습서의 단계를 수행하려면 다음 리소스가 필요합니다.

- ESP32 개발 보드.
- 마이크로USB 대 USB A 케이블.
- 원래 요청 ping에 대한 AWS 계정 (프리 티어로 충분함)
- 안드로이드 v 6.0 이상과 블루투스 버전 4.2 이상인 안드로이드 폰입니다.

개발 컴퓨터에는 다음이 필요합니다.

- Xtensa 툴체인과 FreeRTOS 소스 코드 및 예제를 위한 충분한 디스크 공간 (~500Mb)
- Android Studio를 설치합니다.
- AWS CLI를 설치합니다.
- Python3이 설치되었습니다.
- boto3와 Python용 소프트웨어 개발자 키트 (SDK).

이 자습서의 단계에서는 Xtensa 툴체인, ESP-IDF 및 FreeRTOS 코드가 /esp 험 디렉터리에 있는 디렉터리입니다. 를 추가해야 합니다. ~/esp/xtensa-esp32-elf/bin\$PATH 변수.

## 1단계: 스토리지 구성

1. 업데이트를 저장할 Amazon S3 버킷 생성 (p. 141) 펌웨어 이미지를 보관할 수 있도록 버전 관리를 활성화합니다.
2. OTA 업데이트 서비스 역할 생성 (p. 141) 다음 관리형 정책을 역할에 추가합니다.
  - AWSLambdaFunctionV2
  - AWSLambdaRole
  - AWSLambdaVPCAccessExecutionRole
  - AWSLambdaBasicExecutionRole
3. IAM 사용자 생성 OTA 업데이트를 수행할 수 있습니다. 이 사용자는 계정의 IoT 장치에 펌웨어 업데이트를 서명하고 배포할 수 있으며 모든 장치에서 OTA 업데이트를 수행할 수 있습니다. 액세스는 신뢰할 수 있는 엔터티로 제한되어야 합니다.
4. 다음 단계에 따라 OTA 사용자 정책 생성 (p. 143) 이를 IAM 사용자와 연결합니다.

## 2단계: 코드 서명 인증서 생성

1. 버전 관리가 활성화된 Amazon S3 버킷을 생성하여 펌웨어 이미지를 보관합니다.
2. 펌웨어에 서명하는데 사용할 수 있는 코드 서명 인증서를 생성합니다. 인증서를 가져올 때 Amazon 리소스 이름 (ARN)을 적어둡니다.

```
aws acm import-certificate --profile=ota-update-user --certificate file://  
ecdsasigner.crt --private-key file://ecdsasigner.key
```

출력 예:

```
{  
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"  
}
```

나중에 ARN을 사용하여 서명 프로필을 만듭니다. 원활 경우 다음 명령을 사용하여 프로파일을 생성할 수 있습니다.

```
aws signer put-signing-profile --profile=ota-update-user --profile-name esp32Profile  
--signing-material certificateArn=arn:aws:acm:us-east-1:account:certificate/certid --  
platform AmazonFreeRTOS-Default --signing-parameters certname=/cert.pem
```

출력 예:

```
{  
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"  
}
```

## 3단계: Amazon Cognito 인증 구성

### AWS IoT 정책 생성

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 콘솔의 오른쪽 상단 모서리에서 **[]**를 선택합니다. 내 계정. UND계정 설정을 눌러 12자리 계정 ID를 기록해둡니다.
3. 왼쪽 탐색 창에서 설정을 선택합니다. In디바이스 데이터 엔드포인트엔드포인트 값을 기록해둡니다. 엔드포인트는 다음과 같은 형식입니다.xxxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com. 이 예제에서는 AWS 지역은 "us-west-2"입니다.
4. 왼쪽 탐색 창에서 **[]**를 선택합니다. Secure, 선택정책을 선택한 다음 **[선택]**를 선택합니다. 생성. 계정에 정책이 없는 경우 "아직 정책이 없습니다"라는 메시지가 표시되고 선택할 수 있습니다. 정책 만들기.
5. 정책의 이름을 입력합니다 (예: "esp32\_mqtt\_proxy\_iot\_iot\_policy").
6. Add statements(설명문 추가) 섹션에서 Advanced mode(고급 모드)를 선택합니다. 다음 JSON을 복사하여 정책 편집기 창에 붙여 넣습니다. Replace aws-account-id를 계정 ID와 함께 aws-region 리전 (예: "us-west-2")과 함께.

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

7. 생성(Create)을 선택합니다.

#### 생성AWS IoT의회

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 왼쪽 탐색 창에서 **[ ]**를 선택합니다. 관리를 선택한 다음 **[ ]**를 선택합니다. 사물.
3. 오른쪽 상단 모서리에서 **[ ]**를 선택합니다. 생성. 계정에 등록된 사물이 없는 경우 “아직 사물이 없습니다”라는 메시지가 표시되고 선택할 수 있습니다. 사물 등록.
4. AWS IoT IoT 사물 생성 페이지에서 단일 사물 생성을 선택합니다.
5. 온사물 레지스트리에 디바이스 추가페이지에서 사물 이름을 입력합니다 (예: “esp32-ble”). 영숫자, 하이픈 (-), 밀줄 (\_) 문자만 사용할 수 있습니다. 다음(Next)을 선택합니다.
6. 온항목에 인증서를 추가하십시오페이지, 아래인증서 생략 및 사물 만들기, 선택인증서 없이 사물 만들기. 인증 및 권한 부여에 Amazon Cognito 자격 증명을 사용하는 BLE 프록시 모바일 앱을 사용하기 때문에 디바이스 인증서가 필요하지 않습니다.

#### Amazon Cognito App 클라이언트 생성

1. [Amazon Cognito 콘솔](#)에 로그인합니다.
2. 오른쪽 상단 탐색 배너에서 사용자 풀 생성.
3. 풀 이름을 입력합니다 (예: “esp32\_mqtt\_proxy\_사용자\_풀”).
4. [Review defaults]를 선택합니다.
5. InApp 클라이언트, 선택App 클라이언트 추가를 선택한 다음 **[ ]**를 선택합니다. 앱 클라이언트 추가.
6. 앱 클라이언트 이름 (예: “mqtt\_app\_client”)을 입력합니다.
7. 다음을 확인하십시오. 클라이언트 시크릿 생성가 선택됩니다.
8. [Create app client]를 선택합니다.
9. 풀 세부 정보로 돌아가기를 선택합니다.
10. 온검토사용자 풀의 페이지에서 풀 만들기. “사용자 풀이 성공적으로 생성되었습니다.”라는 메시지가 표시됩니다. 풀 ID를 기록해둡니다.
11. 탐색 창에서 **[ ]**를 선택합니다. 앱 클라이언트.
12. 선택세부 정보 표시. 앱 클라이언트 ID와 앱 클라이언트 암호를 기록해둡니다.

#### Amazon Cognito 자격 증명 풀 생성

1. [Amazon Cognito 콘솔](#)에 로그인합니다.
2. [Create new identity pool]을 선택합니다.
3. ID 풀의 이름을 입력합니다 (예: “mqtt\_proxy\_정체성\_풀”).
4. 확장인증 공급자.
5. 를 선택합니다. Cognito탭.
6. 이전 단계에서 기록한 사용자 풀 ID 및 앱 클라이언트 ID를 입력합니다.
7. [Create Pool]을 선택합니다.
8. 다음 페이지에서 인증된 자격 증명 및 인증되지 않은 자격 증명에 대해 새 역할을 생성하려면 허용.
9. 자격 증명 풀 ID를 적어 두십시오. 이 ID는 형식입니다.us-east-1:xxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx.

인증된 자격 증명에 IAM 정책을 연결합니다.

1. [Amazon Cognito 콘솔](#)을 엽니다.
2. 방금 만든 ID 풀을 선택합니다 (예: “mqtt\_proxy\_정체성\_풀”).

3. 자격 증명 풀 편집을 선택합니다.
4. 인증된 역할에 할당된 IAM 역할 (예: “코그너\_MQTT\_PROXY\_정체성\_풀Auth\_역할”) 을 기록해 둡니다.
5. **IAM 콘솔**을 엽니다.
6. 탐색 창에서 Roles(역할)를 선택합니다.
7. 할당된 역할 (예: “Cognito\_MQTT\_PROXY\_정체성\_풀Auth\_역할”) 을 검색한 다음 선택합니다.
8. 선택인라인 정책 추가를 선택한 다음 를 선택합니다.JSON.
9. 다음 정책을 입력합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:AttachPolicy",  
                "iot:AttachPrincipalPolicy",  
                "iot:Connect",  
                "iot:Publish",  
                "iot:Subscribe"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

10. 정책 검토를 선택합니다.
11. 정책 이름 (예: “mQTProxyCognitoPolicy”) 을 입력합니다.
12. 정책 생성(Create policy)을 선택합니다.

## 4단계: Amazon FreeRTOS 구성

1. Amazon FreeRTOS 코드를 최신 버전으로 다운로드합니다.[FreeRTOS 콘솔](#) 또는 [FreeRTOS GitHub의 저장소](#).
2. OTA 업데이트 데모를 활성화 하려면 다음 단계를 따르십시오.[Espressif ESP32 시작하기 DevKitC와 ESP-WROVER-KIT \(p. 46\)](#).
3. 다음 파일에서 이러한 추가 수정 작업을 수행합니다.
  - a. Openvendors/espressif/boards/esp32/aws\_demos/config\_files/ aws\_demo\_config.h를 열고 를 CONFIG\_OTA\_UPDATE\_DEMO\_ENABLED.
  - b. Openvendors/espressif/boards/esp32/aws\_demos/common/config\_files/ aws\_demo\_config.h를 변경democonfigNETWORK\_TYPES에 AWSIOT\_NETWORK\_TYPE\_BLE.
  - c. Opendemos/include/aws\_clientcredential.h다음에 대한 엔드포인트 URL을 입력합니다.clientcredentialMQTT\_BROKER\_ENDPOINT.

사물 이름을 입력합니다.clientcredentialIOT\_THING\_NAME(예: “esp32-ble”). Amazon Cognito 자격 증명을 사용할 때 인증서를 추가할 필요가 없습니다.

  - d. Openvendors/espressif/boards/esp32/aws\_demos/ config\_files/aws\_iot\_network\_config.h를 변경configSUPPORTED\_NETWORKS과 configENABLED\_NETWORKS를 포함하도록 제한AWSIOT\_NETWORK\_TYPE\_BLE.
  - e. 열기vendors/**vendor**/boards/**board**/aws\_demos/config\_files/ ota\_demo\_config.h파일을 입력하고 인증서를 입력합니다.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

응용 프로그램이 시작되어 데모 버전을 인쇄해야합니다.

```
11 13498 [iot_thread] [INFO ][DEMO][134980] Successfully initialized the demo. Network
type for the demo: 2
12 13498 [iot_thread] [INFO ][MQTT][134980] MQTT library successfully initialized.
13 13498 [iot_thread] OTA demo version 0.9.20
14 13498 [iot_thread] Creating MQTT Client...
```

## 5단계: Android 앱 구성

- 안드로이드 블루투스 저에너지 SDK와 샘플 앱을 다운로드 [아마존 프리어토스-블-안드로이드-sdk](#) GitHub의 저장소.
- 파일 열기 `app/src/main/res/raw/awsconfiguration.json` 다음 JSON 샘플의 지침을 사용하여 풀 ID, 리전, AppClientID 및 AppClientSecret를 입력합니다.

```
{
    "UserAgent": "MobileHub/1.0",
    "Version": "1.0",
    "CredentialsProvider": {
        "CognitoIdentity": {
            "Default": {
                "PoolId": "Cognito->Manage Identity Pools->Federated Identities-
>mqtt_proxy_identity_pool->Edit Identity Pool->Identity Pool ID",
                "Region": "Your region (for example us-east-1)"
            }
        },
        "IdentityManager": {
            "Default": {}
        },
        "CognitoUserPool": {
            "Default": {
                "PoolId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool -> General
Settings -> PoolId",
                "AppClientId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
General Settings -> App clients ->Show Details",
                "AppClientSecret": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
General Settings -> App clients ->Show Details",
                "Region": "Your region (for example us-east-1)"
            }
        }
    }
}
```

- Open `app/src/main/java/software/amazon/freertos/DemoConstants.java` 앞에서 생성한 정책 이름을 입력합니다 (예: `esp32_mqtt_###_###_##`) 및 리전 (예: `us-east-1`).
- 데모 앱을 빌드하고 설치합니다.
  - Android Studio를 선택합니다. 빌드 그런 다음 모듈 만들기 앱.
  - 선택 실행 그런 다음 앱 실행. Android 스튜디오의 logcat 창으로 이동하여 로그 메시지를 모니터링할 수 있습니다.
  - Android 기기의 로그인 화면에서 계정을 만듭니다.
  - 사용자를 생성합니다. 사용자가 이미 있는 경우 자격 증명을 입력합니다.
  - Amazon FreeRTOS 데모가 디바이스의 위치에 액세스할 수 있도록 허용합니다.
  - Bluetooth Low Energy 장치를 스캔합니다

- g. 찾은 장치의 슬라이더를 다음으로 이동합니다.:
- h. 뉴트로NRF52840에 대한 직렬 포트 디버그 콘솔에 있습니다.
- i. 선택페이지 & Connect.
5. 이 추가 정보...연결이 설정되면 링크가 활성화됩니다. 연결이 완료되면 Android 기기 logcat에서 연결 상태가 "BLE\_CONNECTED"로 변경되어야 합니다.

```
2019-06-06 20:11:32.160 23484-23497/software.amazon.freertos.demo I/FRD: BLE connection state changed: 0; new state: BLE_CONNECTED
```

6. 메시지를 전송하기 전에 Amazon FreeRTOS 디바이스와 Android 디바이스가 MTU를 협상합니다. logcat에 다음 결과가 표시됩니다.

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD: onMTUChanged : 512 status: Success
```

7. 기기가 앱에 연결되고 MQTT 프록시를 사용하여 MQTT 메시지를 보내기 시작합니다. 장치가 통신할 수 있는지 확인하려면 MQTT\_CONTROL 특성 데이터 값이 01로 변경되었는지 확인합니다.

```
2019-06-06 20:12:28.752 23484-23496/software.amazon.freertos.demo D/FRD: <-<-<- Writing to characteristic: MQTT_CONTROL with data: 01
2019-06-06 20:12:28.839 23484-23496/software.amazon.freertos.demo D/FRD: onCharacteristicWrite for: MQTT_CONTROL; status: Success; value: 01
```

8. 장치가 페어링되면 ESP32 콘솔에 프롬프트가 표시됩니다. BLE를 활성화하려면. 이 단계를 수행하기 전에는 데모가 작동하지 않습니다.

```
E (135538) BT_GATT: GATT_INSUF_AUTHENTICATION: MITM Required
W (135638) BT_L2CAP: l2cble_start_conn_update, the last connection update command still pending.
E (135908) BT_SMP: Value for numeric comparison = 391840
15 13588 [InputTask] Numeric comparison:391840
16 13589 [InputTask] Press 'y' to confirm
17 14078 [InputTask] Key accepted
W (146348) BT_SMP: FOR LE SC LTK IS USED INSTEAD OF STK
18 16298 [iot_thread] Connecting to broker...
19 16298 [iot_thread] [INFO ][MQTT][162980] Establishing new MQTT connection.
20 16298 [iot_thread] [INFO ][MQTT][162980] (MQTT connection 0x3ffd5754, CONNECT operation 0x3ffd586c) Waiting for operation completion.
21 16446 [iot_thread] [INFO ][MQTT][164450] (MQTT connection 0x3ffd5754, CONNECT operation 0x3ffd586c) Wait complete with result SUCCESS.
22 16446 [iot_thread] [INFO ][MQTT][164460] New MQTT connection 0x3ffc0ccc established.
23 16446 [iot_thread] Connected to broker.
```

## 6단계: OTA 업데이트 스크립트 실행

1. 필수 구성 요소를 설치하려면 다음 명령을 실행합니다.

```
pip3 install boto3
```

```
pip3 install pathlib
```

2. 에서 FreeRTOS 애플리케이션 버전 증가 `demos/include/aws_application_version.h`.
3. 새.bin 파일을 빌드합니다.
4. Python 스크립트 다운로드 [start\\_ota.py](#). 스크립트에 대한 도움말 내용을 보려면 터미널 창에서 다음 명령을 실행합니다.

```
python3 start_ota.py -h
```

다음과 같은 내용이 표시되어야 합니다.

```
usage: start_ota.py [-h] --profile PROFILE [--region REGION]
                     [--account ACCOUNT] [--devicetype DEVICETYPE] --name NAME
                     --role ROLE --s3bucket S3BUCKET --otasingningprofile
                     OTASIGNINGPROFILE --signingcertificateid
                     SIGNINGCERTIFICATEID [--codelocation CODELOCATION]
Script to start OTA update
optional arguments:
-h, --help            show this help message and exit
--profile PROFILE     Profile name created using aws configure
--region REGION       Region
--account ACCOUNT     Account ID
--devicetype DEVICETYPE thing|group
--name NAME           Name of thing/group
--role ROLE           Role for OTA updates
--s3bucket S3BUCKET   S3 bucket to store firmware updates
--otasingningprofile OTASIGNINGPROFILE
                     Signing profile to be created or used
--signingcertificateid SIGNINGCERTIFICATEID
                     certificate id (not arn) to be used
--codelocation CODELOCATION
                     base folder location (can be relative)
```

- 제공된 항목을 사용한 경우 AWS CloudFormation 리소스를 생성하려면 다음 명령을 실행합니다.

```
python3 start_ota_stream.py --profile otausercf --name esp32-ble --role
                           ota_ble_iot_role-sample --s3bucket afr-ble-ota-update-bucket-sample --
                           otasingningprofile abcd --signingcertificateid certificateid
```

ESP32 디버그 콘솔에서 업데이트 시작을 확인할 수 있습니다.

```
38 2462 [OTA Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
---
49 2867 [OTA Task] [prvIngestDataBlock] Received file block 1, size 1024
50 2867 [OTA Task] [prvIngestDataBlock] Remaining: 1290
51 2894 [OTA Task] [prvIngestDataBlock] Received file block 2, size 1024
52 2894 [OTA Task] [prvIngestDataBlock] Remaining: 1289
53 2921 [OTA Task] [prvIngestDataBlock] Received file block 3, size 1024
54 2921 [OTA Task] [prvIngestDataBlock] Remaining: 1288
55 2952 [OTA Task] [prvIngestDataBlock] Received file block 4, size 1024
56 2953 [OTA Task] [prvIngestDataBlock] Remaining: 1287
57 2959 [iot_thread] State: Active Received: 5 Queued: 5 Processed: 5 Dropped: 0
```

- OTA 업데이트가 완료되면 OTA 업데이트 프로세스에서 필요에 따라 장치가 다시 시작됩니다. 그런 다음 업데이트된 펌웨어를 사용하여 연결을 시도합니다. 업그레이드가 성공하면 업데이트된 펌웨어가 활성 상태로 표시되고 콘솔에 업데이트된 버전이 표시됩니다.

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

# AWS IoT 디바이스 샘플 디바이스 애플리케이션

## 소개

이 데모는 사용 방법을 보여줍니다. AWS IoT에 연결할 Device Shadow 라이브러리 [AWS 디바이스 샘플 서비스](#). 그것은 CoreMQTT 라이브러리 (p. 214) TLS (상호 인증) 와의 MQTT 연결을 설정합니다. AWS IoT MQTT 브로커와 CoreJSON 라이브러리 파서를 사용하여 AWS 샘플 서비스. 데모에서는 그림자 문서를 업데이트하는 방법 및 샘플 문서를 삭제하는 방법과 같은 기본적인 그림자 작업을 보여 줍니다. 또한 데모에서는 CoreMqtt 라이브러리에 콜백 함수를 등록하여 그림자와 같은 메시지를 처리하는 방법을 보여줍니다. ./update과 /update/delta에서 보낸 메시지 AWS IoT Device Shadow 서비스.

이 데모는 샘플 문서 (상태) 업데이트 요청과 업데이트 응답이 동일한 응용 프로그램에서 수행되기 때문에 학습 연습으로만 사용됩니다. 실제 프로덕션 시나리오에서 외부 애플리케이션은 장치가 현재 연결되어 있지 않더라도 원격으로 디바이스 상태 업데이트를 요청합니다. 장치가 연결되면 업데이트 요청을 승인합니다.

### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오. [FreeRTOS 시작하기 \(p. 14\)](#).

## 기능

데모에서는 그림자를 보여주는 일련의 예제를 반복하는 단일 응용 프로그램 작업을 만듭니다. ./update과 /update/delta 원격 장치의 상태 토글을 시뮬레이션하는 콜백입니다. 새와 함께 샘플 문서를 업데이트를 보냅니다. desired 상태 및 장치가 변경될 때까지 기다립니다. reported 새로운 것에 대한 응답으로 상태 desired 상태입니다. 또한, 그림자 /update 콜백은 변화하는 그림자 상태를 인쇄하는 데 사용됩니다. 이 데모에서는 보안 MQTT 연결을 사용합니다. AWS IoT MQTT 브로커가 있으며 powerOn 상태 샘플 문서입니다.

데모에서는 다음 작업을 수행합니다.

1. 의 도우미 함수를 사용하여 MQTT 연결을 설정합니다. `shadow_demo_helpers.c`.
2. 에 의해 정의된 매크로를 사용하여 디바이스 샘플 작업을 위한 MQTT 주제 문자열 어셈블 AWS IoT Device Shadow 라이브러리.
3. 기존 디바이스 샘플 문서를 삭제하기 위해 디바이스 샘플 문서를 삭제하는 데 사용되는 MQTT 주제에 게시합니다.
4. MQTT 주제를 구독합니다. ./update/delta, ./update/accepted과 ./update/rejected에서 도우미 함수 사용 `shadow_demo_helpers.c`.
5. 원하는 상태를 게시합니다. powerOn에서 도우미 함수 사용 `shadow_demo_helpers.c`. 이로 인해 ./update/delta 디바이스로 보낼 메시지입니다.
6. 들어오는 MQTT 메시지 처리 `prvEventCallback`를 클릭하고 에 의해 정의된 함수를 사용하여 메시지가 장치 샘플 문서와 관련되어 있는지 확인합니다. AWS IoT Device Shadow 라이브러리 (`Shadow_MatchTopic`). 메시지가 장치 샘플 문서인 경우 ./update/delta 메시지를 표시하면 메인 데모 함수 두 번째 메시지를 게시하여 보고 된 상태를 다음과 같이 업데이트합니다. powerOn. If an ./update/accepted 메시지가 수신되었습니다. 메시지가 동일한지 확인합니다. clientToken 업데이트 메시지에 이전에 게시된 것과 같습니다. 그러면 데모의 끝이 표시됩니다.

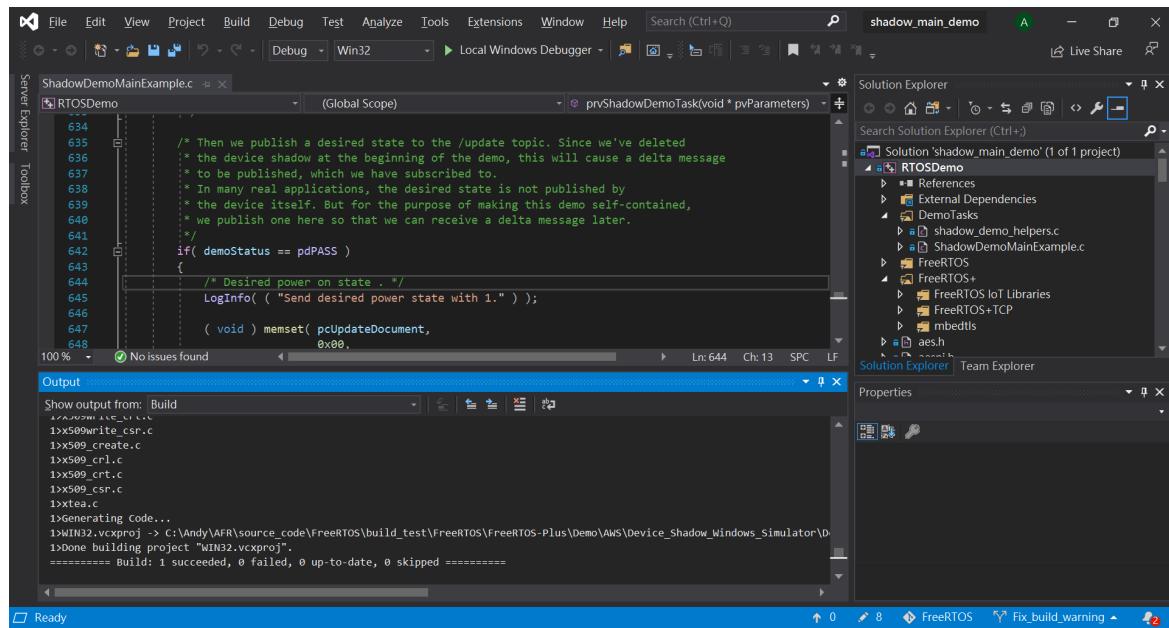
```

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:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [ShadowDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1}}
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"},90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$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":{"desired":{"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:678] 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 clientToken=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 10236 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 10236 [ShadowDemo] Demo completed successfully.138 10236 [ShadowDemo]
139 10236 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:750] 140 10236 [ShadowDemo] Deleting Shadow Demo Task.141 10236 [ShadowDemo]

```

데모는 파일에서 찾을 수 있습니다.[freertos/demos/device\\_shadow\\_for\\_aws/shadow\\_demo\\_main.c](#) 또는 커기[GitHub](#).

다음 스크린샷은 데모가 성공할 때 예상되는 결과를 보여줍니다.



## 에 연결하려면AWS IoT MQTT 브로커

에 연결하려면AWS IoT MQTT 브로커, 우리는 다음과 같은 방법을 사용합니다.[MQTT\\_Connect\(\)](#)의[CoreMQTT 상호 인증 데모 \(p. 266\)](#).

## 섀도우 문서 삭제

섀도우 문서를 삭제하려면 `xPublishToTopic` 빈 메시지와 함께, 예 의해 정의된 매크로 사용 AWS IoT Device Shadow 라이브러리. 이 작업은 `MOTT_Publish`에 게시하려면/`delete` 주제. 다음 코드 섹션에서는 함수에서 이 작업이 어떻게 이루어지는지 보여줍니다.`.prvShadowDemoTask`.

```
/* First of all, try to delete any Shadow document in the cloud. */
returnStatus = PublishToTopic( SHADOW_TOPIC_STRING_DELETE( THING_NAME ),
                               SHADOW_TOPIC_LENGTH_DELETE( THING_NAME_LENGTH ),
                               pcUpdateDocument,
                               OU );
```

## 섀도우 주제 구독

Device Shadow 주제를 구독하여 AWS IoT 그림자 변경에 대한 브로커 Device Shadow 항목은 Device Shadow 섀도우 라이브러리에 정의된 매크로에 의해 어셈블됩니다. 다음 코드 섹션에서는 이 작업이 어떻게 이루어지는지 보여줍니다.`prvShadowDemoTask` 함수.

```
/* 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 보내려면 데모가 호출합니다. `xPublishToTopic` Device Shadow 라이브러리에서 정의한 매크로를 사용하여 JSON 형식의 메시지를 표시합니다. 이 작업은 `MQTT_Publish`에 게시하려면/`delete` 주제. 다음 코드 섹션에서는 이 작업이 어떻게 이루어지는지 보여줍니다. `prvShadowDemoTask` 함수.

```
#define SHADOW_REPORTED_JSON      \
    "{"                                \
    "\"state\":{\"\"                                \
    "\"reported\":{\"\"                                \
    "\"powerOn\":%01d\"                                \
    \"}\"                                \
    \"},\"                                \
    "\"clientToken\":\"%06lu\" \"                                \
    \"}\"\n\n"
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 섀도 업데이트 메시지 처리

에 등록된 사용자 콜백 함수입니다. [코렐QTT 클라이언트 라이브러리](#)을 사용하여 MQTT\_Initfunction, 수신 패킷 이벤트에 대해 알려드립니다. 콜백 함수 보기 [PRVEVENT 콜백](#)(GitHub에 있음).

콜백 함수는 들어오는 패킷이 유형인지 확인합니다. MQTT\_PACKET\_TYPE\_PUBLISH Device Shadow 라이브러리 API 사용 [Shadow\\_MatchTopic](#) 들어오는 메시지가 섀도 메시지인지 확인합니다.

들어오는 메시지가 유형의 그림자 메시지인 경우 ShadowMessageTypeUpdateDelta 그런 다음 전화합니다. [PRVUPDATE 델타 핸들러](#)이 메시지를 처리합니다. 핸들러 prvUpdateDeltaHandlerCoreJSON 라이브러리를 사용하여 메시지를 구문 분석하여 powerOnstate 및 이 값을 로컬로 유지 관리되는 현재 장치 상태와 비교합니다. 서로 다른 경우 로컬 장치 상태가 새 값을 반영하도록 업데이트됩니다. powerOn 섀도 문서의 state입니다.

들어오는 메시지가 유형의 그림자 메시지인 경우 ShadowMessageTypeUpdateAccepted 그런 다음 전화합니다. [PRVUPDATE 수락 처리기](#)이 메시지를 처리합니다. 핸들러 prvUpdateAcceptedHandlerCoreJSON 라이브러리를 사용하여 메시지를 구문 분석하여 clientToken 메시지에서. 이 핸들러 함수는 JSON 메시지의 클라이언트 토큰이 응용 프로그램에서 사용하는 클라이언트 토큰과 일치하는지 확인합니다. 일치하지 않으면 함수가 경고 메시지를 기록합니다.

## 보안 소켓 에코 클라이언트 데모

다음 예제에서는 단일 RTOS 작업을 사용합니다. 이 예제의 소스 코드는 `demos/tcp/aws_tcp_echo_client_single_task.c`에서 찾을 수 있습니다.

시작하기 전에 마이크로컨트롤러에 FreeRTOS를 다운로드하고 FreeRTOS 데모 프로젝트를 빌드 및 실행했는지 확인합니다. FreeRTOS를 복제하거나 다운로드할 수 있습니다. [GitHub](#). 자세한 내용은 [README.md](#) 파일을 참조하십시오.

데모를 실행하려면

### Note

FreeRTOS 데모를 설정하고 실행하려면 다음 단계를 따르십시오. [FreeRTOS 시작하기 \(p. 14\)](#). TCP 서버와 클라이언트 데모는 현재 Cypress CYW943907AEVAL1F 및 CYW954907AEVAL1F 개발 키트에서 지원되지 않습니다.

1. 의 지침을 따르십시오. [TLS 에코 서버 설정](#) FreeRTOS 포팅 안내서를 참조하십시오.

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 업니다.

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 9000으로 설정합니다.
6. configTCP\_ECHO\_TASKS\_SINGLE\_TASK\_TLS\_ENABLED 변수를 1로 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\_PEM에 복사합니다.
8. Open `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`를 주석으로 처리합니다.  
#define CONFIG\_CORE\_MQTT\_MUTUAL\_AUTH\_DEMO\_ENABLED를 정의합니다.  
또는 CONFIG\_OTA\_MQTT\_UPDATE\_DEMO\_ENABLED 또는 CONFIG\_OTA\_HTTP\_UPDATE\_DEMO\_ENABLED.

マイクロコントローラ와 TLS Echo Server는 동일한 네트워크에 있어야 합니다. 데모가 (main.c)를 시작하면 Received correct string from echo server라는 로그 메시지가 표시됩니다.

# AWS IoT Device TesterFreeRTOS

AWS IoT Device Tester(IDT)는 IoT 디바이스를 검증할 수 있는 다운로드 가능한 테스트 프레임워크입니다. 사용할 수 있습니다. AWS IoT Device TesterFreeRTOS에서 FreeRTOS 인증 제품군을 실행하고 디바이스에 대한 사용자 지정 테스트 스위트를 생성 및 실행할 수 있습니다.

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.

## FreeRTOS 자격 제품군

FreeRTOS 자격에 IDT를 사용하여 해당 디바이스에서 FreeRTOS 운영 체제가 로컬로 작동하고 AWS IoT. 특히 FreeRTOS 라이브러리의 이식 계층 인터페이스가 올바르게 구현되었는지 확인합니다. 또한 수행합니다. end-to-end 다음을 수행하는 테스트 AWS IoT Core. 예를 들어 보드에서 MQTT 메시지를 송수신하고 올바르게 처리할 수 있는지 확인합니다. FreeRTOS용 IDT에서 실행하는 테스트는 [FreeRTOS GitHub 저장소](#).

FreeRTOS용 IDT는 예제를 제출할 수 있는 테스트 보고서를 생성합니다. AWS IoT 하드웨어를 AWS 파트너 장치 카탈로그. 자세한 내용은 [AWS 디바이스 검증 프로그램](#)을 참조하십시오.

다음 다이어그램은 FreeRTOS 자격에 대한 테스트 인프라 설정을 보여 줍니다.



FreeRTOS IDT는 테스트 제품군과 테스트 그룹으로 테스트 리소스를 구성합니다.

- 테스트 제품군은 FreeRTOS 특정 버전의 FreeRTOS에서 디바이스가 작동하고 있는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 BLE 및 MQTT 메시징과 같은 특정 기능과 관련된 개별 테스트 집합입니다.

자세한 정보는 [AWS IoT Device TesterFreeRTOS 테스트 제품군 버전 \(p. 355\)](#)을 참조하십시오.

## 맞춤형 테스트 제품군

IDT v4.0.0부터 FreeRTOS용 IDT는 표준화된 구성 설정 및 결과 형식을 테스트 세트 환경과 결합하여 장치 및 장치 소프트웨어에 대한 사용자 지정 테스트 스위트를 개발할 수 있습니다. 자체 내부 검증을 위한 사용자 지정 테스트를 추가하거나 기기 확인을 위해 고객에게 제공할 수 있습니다.

사용자 지정 테스트 스위트를 구성하는 방법에 따라 사용자 지정 테스트 세트를 실행하기 위해 사용자에게 제공해야 하는 설정 구성이 결정됩니다. 자세한 정보는 [IDT를 사용하여 자체 테스트 스위트를 개발하고 실행 하십시오. \(p. 359\)](#)을 참조하십시오.

## AWS IoT Device Tester용 FreeRTOS 지원되는 버전

이 주제에서는 FreeRTOS용 IDT의 지원되는 버전을 나열합니다. 가장 좋은 방법은 FreeRTOS용 IDT의 대상 버전을 지원하는 최신 버전의 FreeRTOS를 사용하는 것입니다. FreeRTOS용 IDT의 각 버전에는 해당 버전의 FreeRTOS가 하나 이상 있습니다. FreeRTOS의 새 릴리스를 사용하려면 새 버전의 FreeRTOS용 IDT를 다운로드해야 합니다.

소프트웨어를 다운로드하면 용 IDT 라이선스 계약에 동의하는 것입니다.

### 의 최신 버전 AWS IoT Device Tester용 FreeRTOS

다음 링크를 사용하여 FreeRTOS용 IDT의 최신 버전을 다운로드합니다.

FreeRTOS 202107.00용 IDT v4.5.2 및 테스트 스위트 버전 1.6.2 (FreeRTOS 202107.00 LTS 라이브러리 사용)

- IDT v4.5.2에 대한 테스트 세트 FRQ\_1.6.2가 있는 경우[리눅스](#)
- IDT v4.5.2에 대한 테스트 세트 FRQ\_1.6.2가 있는 경우[macOS](#)
- IDT v4.5.2에 대한 테스트 세트 FRQ\_1.6.2가 있는 경우[윈도우](#)

#### 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

- FreeRTOS 202107.00 LTS 라이브러리를 사용하는 FreeRTOS 202107.00을 지원합니다. FreeRTOS 202107.00 릴리스에 포함된 항목에 대한 자세한 내용은[변경 로그.md](#)GitHub에 있는 파일을 선택합니다.
- 사용자 지정 테스트 스위트를 구성하기 위한 새로운 IDT 테스트 오케스트레이터를 구현합니다. 자세한 내용은 단원을 참조하십시오.[IDT 테스트 오케스트레이터 구성 \(p. 375\)](#)
- 경미한 버그 수정 및 개선 사항.

#### 테스트 제품군 버전

- FRQ\_1.6.2
  - 2022.01.25 출시

## FreeRTOS용 IDT의 이전 버전

FreeRTOS용 IDT의 이전 버전도 지원됩니다.

FreeRTOS 202107.00용 IDT v4.3.0 및 테스트 스위트 버전 1.6.1 (FreeRTOS 202107.00 LTS 라이브러리 사용)

- 용 테스트 제품군 FRQ\_1.6.1이 있는 IDT v4.3.0[리눅스](#)
- 용 테스트 제품군 FRQ\_1.6.1이 있는 IDT v4.3.0[macOS](#)

- 용 테스트 제품군 FRQ\_1.6.1이 있는 IDT v4.3.0 [윈도우](#)

#### 릴리스 정보

- FreeRTOS 202107.00 LTS 라이브러리를 사용하는 FreeRTOS 202107.00을 지원합니다. FreeRTOS 202107.00 릴리스에 포함된 항목에 대한 자세한 내용은 [변경 로그.md](#) GitHub에 있는 파일을 선택합니다.
- 구성 및 실행 기능 추가 AWS IoT Device Tester 웹 기반 사용자 인터페이스를 통해 FreeRTOS를 사용할 수 있습니다. 단원을 참조하십시오. [FreeRTOS용 IDT 사용자 인터페이스를 사용하여 FreeRTOS 인증 제품군을 실행합니다.](#) (p. 341) 시작하기.

#### 테스트 제품군 버전

- FRQ\_1.6.1
  - 2021.07.26 출시

#### FreeRTOS 202107.00용 IDT v4.1.0 및 테스트 스위트 버전 1.6.0

- 용 테스트 제품군 FRQ\_1.6.0이 있는 IDT v4.1.0 [리눅스](#)
- 용 테스트 제품군 FRQ\_1.6.0이 있는 IDT v4.1.0 [macOS](#)
- 용 테스트 제품군 FRQ\_1.6.0이 있는 IDT v4.1.0 [윈도우](#)

#### 릴리스 정보

- FreeRTOS 202107.00을 지원합니다. FreeRTOS 202107.00 릴리스에 포함된 항목에 대한 자세한 내용은 [변경 로그.md](#) GitHub에 있는 파일을 선택합니다.
- OTA 자격에서 다음 테스트 케이스를 제거합니다.
  - OTA 에이전트
  - OTA 누락 파일 이름
  - OTA 최대 구성 블록 수
- OTA 데이터플레인 제거 Both OTA 자격의 테스트 그룹.에서 [device.json 파일](#) (p. 329), [OTADataPlaneProtocol](#) 이제 구성이 수락됩니다. HTTP 또는 MQTT 지원되는 값입니다.
- 다음 변경 사항을 구현합니다. [freertosFileConfiguration](#)에서 구성은 선택합니다. [userdata.json 파일](#) (p. 332) FreeRTOS 소스 코드 변경 사항은 다음과 같습니다.
  - 에 대해 지정된 파일 이름을 변경합니다. [otaAgentTestsConfig](#)과 [otaAgentDemosConfig](#)...에서 [aws\\_ota\\_agent\\_config.h](#)에 [ota\\_config.h](#).
  - 새 항목을 추가합니다. [otaDemosConfig](#) 새 파일의 파일 경로를 지정하는 선택적 구성 [ota\\_demo\\_config.h](#) 파일.
  - 새 필드를 추가합니다. [testStartDelayms](#)에 [userdata.json](#) FreeRTOS 테스트 그룹을 실행하기 위해 디바이스가 플래시되는 시간과 테스트 실행을 시작하는 시간 사이의 지연을 지정합니다. 값은 밀리초 단위로 지정해야 합니다. 이 지연은 테스트 출력이 누락되지 않도록 IDT에 연결할 수 있는 기회를 제공하는데 사용할 수 있습니다.

#### 테스트 제품군 버전

- FRQ\_1.6.0
  - 2021.07.21 출시

#### IDT v4.0.3 및 테스트 스위트 버전 1.5.1 (FreeRTOS 202012.00 LTS 라이브러리 사용)

- IDT v4.0.3 테스트 세트 FRQ\_1.5.1을 위한 [리눅스](#)

- IDT v4.0.3 테스트 세트 FRQ\_1.5.1을 위한 macOS
- IDT v4.0.3 테스트 세트 FRQ\_1.5.1을 위한 윈도우

#### 릴리스 정보

- 하드웨어 보안 모듈에서 잠긴 자격 증명이 있는 장치의 자격 증명을 Support.
- 경미한 버그 수정 및 개선 사항.

#### 테스트 제품군 버전

- FRQ\_1.5.1
  - 2021.07.30을 출시했습니다.

자세한 내용은 [용에 대한 Support 정책AWS IoT Device TesterFreeRTOS \(p. 426\)](#) 단원을 참조하세요.

## FreeRTOS용 IDT의 지원되지 않는 버전

이 단원에서는 FreeRTOS용 IDT의 지원되지 않는 버전을 나열합니다. 지원되지 않는 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다. 자세한 정보는 [용에 대한 Support 정책AWS IoT Device TesterFreeRTOS \(p. 426\)](#)을 참조하십시오.

IDT-FreeRTOS의 다음 버전은 더 이상 지원되지 않습니다.

IDT v4.0.1 및 테스트 스위트 버전 1.4.1 FreeRTOS 202012.00용

#### 릴리스 정보

- FreeRTOS LTS 라이브러리를 사용하는 FreeRTOS 202012.00을 지원합니다. FreeRTOS 202012.00 릴리스에 포함된 항목에 대한 자세한 내용은 [변경 로그.md](#) GitHub에 있는 파일입니다.
- 추가 OTA (무선) E2E (엔드 투 엔드) 테스트 케이스를 소개합니다.
- FreeRTOS LTS 라이브러리를 사용하는 FreeRTOS 202012.00을 실행하는 개발 보드의 자격을 지원합니다.
- 셀룰러 연결을 사용하여 FreeRTOS 개발 보드의 자격 요건에 대한 지원을 추가합니다.
- echo 서버 구성의 버그를 수정했습니다.
- 다음을 사용하여 사용자 정의 테스트 스위트를 개발하고 실행할 수 있습니다. AWS IoT Device TesterFreeRTOS 용 자세한 정보는 [IDT를 사용하여 자체 테스트 스위트를 개발하고 실행하십시오. \(p. 359\)](#)을 참조하십시오.
- 코드 서명된 IDT 응용 프로그램을 제공하므로 Windows 또는 macOS에서 실행할 때 권한을 부여할 필요가 없습니다.
- BLE 테스트 결과 파싱 로직을 개선했습니다.

#### 테스트 제품군 버전

- FRQ\_1.4.1
  - 2021.01.19를 출시했습니다.

FreeRTOS 202011.01을 위한 IDT v3.4.0 및 테스트 스위트 버전 1.3.0

## 릴리스 정보

- FreeRTOS 202011.01을 지원합니다. FreeRTOS 202011.01 릴리스에 포함된 항목에 대한 자세한 내용은 [변경 로그.md](#) GitHub에 있는 파일입니다.
- 'RSA'가 유효한 PKCS11 구성 옵션이 아닌 버그가 수정되었습니다.
- OTA 테스트 후 Amazon S3 버킷이 올바르게 정리되지 않는 버그가 수정되었습니다.
- FullMQTT 테스트 그룹 내의 새로운 테스트 케이스를 지원하기 위한 업데이트입니다.

## 테스트 제품군 버전

- FRQ\_1.3.0
  - 2020.11.05를 출시했습니다.

FreeRTOS 202007.00용 IDT v3.3.0 및 테스트 스위트 버전 1.2.0

## 릴리스 정보

- FreeRTOS 202007.00을 지원합니다. FreeRTOS 202007.00 릴리스에 포함된 항목에 대한 자세한 내용은 [변경 로그.md](#) GitHub에 있는 파일입니다.
- Over The Air (OTA) 업데이트 일시 중단 및 재개 기능을 검증하기 위한 새로운 엔드 투 엔드 테스트
- eu-central-1 지역의 사용자가 OTA 테스트에 대한 구성 유효성 검사를 통과할 수 없던 버그를 수정했습니다.
- 추가됨--update-idt 매개 변수에 대한 매개 변수 run-suite 명령. 이 옵션을 사용하여 IDT 업데이트 프롬프트에 대한 응답을 설정할 수 있습니다.
- 추가됨--update-managed-policy 매개 변수에 대한 매개 변수 run-suite 명령. 이 옵션을 사용하여 관리형 정책 업데이트 프롬프트에 대한 응답을 설정할 수 있습니다.
- 다음을 포함한 내부 개선 및 버그 수정 사항:
  - 자동 테스트 세트 업데이트의 경우 구성 파일 업그레이드가 개선되었습니다.

## 테스트 제품군 버전

- FRQ\_1.2.0
  - 2020.09.17을 출시했습니다.
  - 다음을 포함한 내부 개선 및 버그 수정 사항:
    - OTA의 경우 MQTT가 구성된 경우 연결 끊기 및 다시 시작 테스트에서 MQTT 데이터플레인을 올바르게 사용합니다.
    - BLE의 경우 이제 라즈베리 파이 이미지의 암호를 변경할 수 있습니다.

용 IDT v3.0.2용 FreeRTOS

## 릴리스 정보

- FreeRTOS 202002.00을 지원합니다. FreeRTOS 202002.00 릴리스에 포함된 항목에 대한 자세한 내용은 [변경 로그.md](#) GitHub에 있는 파일입니다.
- IDT 내에서 테스트 제품군의 자동 업데이트를 추가합니다. IDT는 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 항목을 사용하십시오.`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).

- 을 추가합니다.`list-supported-products`명령 — 설치된 IDT 버전에서 지원하는 FreeRTOS 및 테스트 제품군 버전을 나열합니다.
- 추가`list-test-cases`명령 — 테스트 그룹에서 사용할 수 있는 테스트 케이스를 나열합니다.
- 을 추가합니다.`test-id`에 대한 옵션`run-suite`명령 — 이 옵션을 사용하여 테스트 그룹에서 개별 테스트 케이스를 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 [변경 로그.md](#)GitHub에 있는 파일입니다.
- 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 잘 못 구성된 경우 향상된 오류 메시지와 함께 테스트가 빠르게 실패합니다.
- 을 추가했습니다.[AWS 관리형 정책 AWSIoTDeviceTesterForFreeRTOSFullAccess](#)실행에 필요한 권한AWS IoT Device Tester. 새 릴리스에 추가 권한이 필요한 경우 IAM 권한을 업데이트 할 필요가 없도록 이 관리형 정책에 추가합니다.
- 결과 디렉터리에 있는 `AFO_Report.xml` 파일이 이제 `FRO_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.
- 테스트 시 AWS IoT ATS 앤드포인트를 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 앤드포인트를 지원합니다.

201908.00용 IDT v1.4.1

#### 릴리스 정보

- 새 PKCS11 라이브러리와 테스트 사례 업데이트에 대한 지원을 추가했습니다.
- 실행 가능한 오류 코드를 도입했습니다. 자세한 내용은 단원을 참조하십시오.[IDT 오류 코드 \(p. 420\)](#)
- IDT를 실행하는 데 사용된 IAM 정책이 업데이트되었습니다.

201906.00 메이저용 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 지원이 추가되었습니다.
  - CMAKE 빌드 시스템으로 FreeRTOS 디바이스를 테스트할 수 있도록 지원이 추가됨

#### 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

## 용 IDT 다운로드

이 주제에서는 FreeRTOS용 IDT를 다운로드하는 옵션을 설명합니다. 다음 소프트웨어 다운로드 링크 중 하나를 사용하거나 지침에 따라 프로그래밍 방식으로 IDT를 다운로드할 수 있습니다.

#### 주제

- [IDT를 수동으로 다운로드 \(p. 321\)](#)
- [IDT를 프로그래밍 방식으로 다운로드 \(p. 321\)](#)

소프트웨어를 다운로드하면[AWS IoT Device Tester라이선스 계약](#).

#### 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.

## IDT를 수동으로 다운로드

이 주제에서는 FreeRTOS용 IDT의 지원되는 버전을 나열합니다. 모범 사례로, 최신 버전의 를 사용하는 것이 좋습니다. AWS IoT Device Tester대상 버전의 FreeRTOS를 지원합니다. FreeRTOS 새 릴리스를 사용하려면 새 버전의 의 새 버전을 다운로드해야 합니다. AWS IoT Device Tester. 다음과 같이 테스트 실행을 시작할 때 알림이 제공됩니다. AWS IoT Device Tester 사용 중인 FreeRTOS 버전과 호환되지 않습니다.

지원되는 버전AWS IoT Device Tester용 FreeRTOS (p. 315) 섹션을 참조하세요.

## IDT를 프로그래밍 방식으로 다운로드

IDT는 프로그래밍 방식으로 IDT를 다운로드할 수 있는 URL을 검색하는 데 사용할 수 있는 API 작업을 제공합니다. 이 API 작업을 사용하여 최신 버전의 IDT가 있는지 확인할 수도 있습니다. 이 API 작업에는 다음과 같은 엔드포인트가 있습니다.

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

이 API 작업을 호출하려면 를 수행할 수 있는 권한이 있어야 합니다. **iot-device-tester:LatestIdtAction**. 다음을 포함하십시오 AWS 서명, 포함 **iot-device-tester** 서비스 이름으로

## API 요청

HostOs— 호스트 시스템의 운영 체제입니다. 다음 옵션 중 하나를 선택합니다.

- mac
- linux
- windows

TestSuiteType — 테스트 제품군의 유형입니다. 다음 옵션을 선택합니다.

FR— 용 IDT FreeRTOS

ProductVersion

(선택 사항) FreeRTOS 버전입니다. 이 서비스는 해당 버전의 FreeRTOS에 대한 최신 호환 버전의 IDT를 반환합니다. 이 옵션을 지정하지 않으면 서비스는 최신 버전의 IDT를 반환합니다.

## API 응답

API 응답의 형식은 다음과 같습니다. 이 `DownloadURL` zip 파일이 포함되어 있습니다.

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

## 예제

다음 예제를 참조하여 프로그래밍 방식으로 IDT를 다운로드할 수 있습니다. 이 예제에서는 예 저장된 자격 증명을 사용합니다.`AWS_ACCESS_KEY_ID`과 `AWS_SECRET_ACCESS_KEY`환경 변수. 모범 보안 사례를 따르려면 코드에 자격 증명을 저장하지 마십시오.

Example 예: cURL 버전 7.75.0 이상을 사용하여 다운로드 (맥 및 리눅스)

버전 cURL 7.75.0 이상을 사용하는 경우aws-sigv4플래그 - API 요청에 서명합니다. 이 예제에서는 을 사용 합니다jq응답에서 다운로드 URL을 구문 분석합니다.

- Replace`us-west-2`당신과 함께AWS 리전. 리전 코드 목록은 단원을 참조하십시오.[리전 엔드포인트](#).
- Replace`Linux`호스트 컴퓨터의 운영 체제와 함께 사용할 수 있습니다.
- Replace`202107.00`FreeRTOS 버전을 사용하십시오.

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

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

Example 예: 이전 버전의 cURL (Mac 및 Linux) 를 사용하여 다운로드

다음과 같은 cURL 명령을 사용할 수 있습니다.AWS서명하고 계산하는 서명입니다. 의 서명 및 계산 방법에 대한 자세한 내용은AWS서명, 참조[서명AWSAPI 요청](#).

- Replace`Linux`호스트 컴퓨터의 운영 체제와 함께 사용할 수 있습니다.
- Replace####날짜 및 시간 (예:`20220210T004606Z`).
- Replace##날짜 (등)`20220210`.
- Replace`AWSRegion`당신과 함께AWS 리전. 리전 코드 목록은 단원을 참조하십시오.[리전 엔드포인트](#).
- Replace`AWSSignature`와 함께[AWS서명](#)당신이 생성하는 것입니다.

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

Example 예: Python 스크립트를 사용하여 다운로드

이 예제에서는 파이썬을 사용합니다[요청라이브러리](#) 이 예제는 Python 예제에서는 다음과 같이 수정된 것입니다.[SignAWSAPI 요청](#)의AWS일반 참조.

- Replace`us-west-2`해당 리전을 사용합니다. 리전 코드 목록은 단원을 참조하십시오.[리전 엔드포인트](#).
- Replace`Linux`호스트 컴퓨터의 운영 체제와 함께 사용할 수 있습니다.

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
```

```
# This file is licensed under the Apache License, Version 2.0 (the "License").  
# You may not use this file except in compliance with the License. A copy of the  
# License is located at  
#  
# http://aws.amazon.com/apache2.0/  
#  
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS  
# OF ANY KIND, either express or implied. See the License for the specific  
# language governing permissions and limitations under the License.  
  
# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html  
# This version makes a GET request and passes the signature  
# in the Authorization header.  
import sys, os, base64, datetime, hashlib, hmac  
import requests # pip install requests  
# ***** REQUEST VALUES *****  
method = 'GET'  
service = 'iot-device-tester'  
host = 'download.devcicetester.iotdevicesecosystem.amazonaws.com'  
region = 'us-west-2'  
endpoint = 'https://download.devcicetester.iotdevicesecosystem.amazonaws.com/latestidt'  
request_parameters = 'HostOs=linux&TestSuiteType=FR'  
  
# Key derivation functions. See:  
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-  
examples-python  
def sign(key, msg):  
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()  
  
def getSignatureKey(key, dateStamp, regionName, serviceName):  
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)  
    kRegion = sign(kDate, regionName)  
    kService = sign(kRegion, serviceName)  
    kSigning = sign(kService, 'aws4_request')  
    return kSigning  
  
# Read AWS access key from env. variables or configuration file. Best practice is NOT  
# to embed credentials in code.  
access_key = os.environ.get('AWS_ACCESS_KEY_ID')  
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')  
if access_key is None or secret_key is None:  
    print('No access key is available.')  
    sys.exit()  
  
# Create a date for headers and the credential string  
t = datetime.datetime.utcnow()  
amzdate = t.strftime('%Y%m%dT%H%M%SZ')  
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope  
  
# ***** TASK 1: CREATE A CANONICAL REQUEST *****  
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html  
# Step 1 is to define the verb (GET, POST, etc.)--already done.  
# Step 2: Create canonical URI--the part of the URI from domain to query  
# string (use '/' if no path)  
canonical_uri = '/latestidt'  
# Step 3: Create the canonical query string. In this example (a GET request),  
# request parameters are in the query string. Query string values must  
# be URL-encoded (space=%20). The parameters must be sorted by name.  
# For this example, the query string is pre-formatted in the request_parameters variable.  
canonical_querystring = request_parameters  
# Step 4: Create the canonical headers and signed headers. Header names  
# must be trimmed and lowercase, and sorted in code point order from  
# low to high. Note that there is a trailing '\n'.  
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'  
# Step 5: Create the list of signed headers. This lists the headers  
# in the canonical_headers list, delimited with ";" and in alpha order.
```

```

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

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

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

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

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

```

## FreeRTOS 자격 제품군 (FRQ) 과 함께 IDT 사용

FreeRTOS 자격 IDT를 사용하여 FreeRTOS 운영 체제가 로컬로 작동하고 와 통신할 수 있는지 확인할 수 있습니다. AWS IoT. 특히, FreeRTOS 라이브러리의 이식 계층 인터페이스가 올바르게 구현되었는지 확인합니다. 또한 이 IDT는 AWS IoT Core와의 앤드 투 앤드 테스트를 수행합니다. 예를 들어 보드에서 MQTT 메시지를 송수신하고 올바르게 처리할 수 있는지 확인합니다. 용 IDT에서 FreeRTOS에서 실행하는 테스트는 [FreeRTOS GitHub 리포지토리](#).

테스트는 내장형 애플리케이션으로 실행되어 보드에 플래시됩니다. 애플리케이션 바이너리 이미지에는 FreeRTOS, 반도체 공급업체의 이식된 FreeRTOS 인터페이스, 보드 디바이스 드라이버가 포함되어 있습니다.

다. 테스트의 용도는 이식된 FreeRTOS 인터페이스가 디바이스 드라이버 위에서 올바르게 작동하는지 확인하는 것입니다.

용 IDT는 에 제출할 수 있는 테스트 보고서를 생성합니다.AWS IoT하드웨어를AWS파트너 장치 카탈로그. 자세한 내용은 [AWS 디바이스 검증 프로그램](#)을 참조하십시오.

FreeRTOS용 IDT는 테스트 할 보드에 연결된 호스트 컴퓨터 (Windows, macOS 또는 Linux) 에서 실행됩니다. IDT는 테스트 사례를 실행하고 결과를 집계합니다. 또한 테스트 실행을 관리하기 위한 명령줄 인터페이스를 제공합니다.

용 IDT는 디바이스 테스트 외에 FreeRTOS에서 리소스를 생성합니다.AWS IoT검증 프로세스를 쉽게 진행하기 위한 사물, FreeRTOS 그룹, Lambda 함수 등 이러한 리소스를 만들기 위해 FreeRTOS 용 IDT는AWS에 구성된 자격 증명config.json를 대신하여 API 호출을 수행합니다. 이러한 리소스는 테스트 중 다양한 시점에서 프로비저닝됩니다.

용 FreeRTOS를 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.

#### 주제

- [사전 조건 \(p. 325\)](#)
- [처음으로 마이크로 컨트롤러 보드의 테스트 준비 \(p. 328\)](#)
- [FreeRTOS용 IDT 사용자 인터페이스를 사용하여 FreeRTOS 인증 제품군을 실행합니다. \(p. 341\)](#)
- [Bluetooth Low Energy 테스트 실행 \(p. 348\)](#)
- [FreeRTOS 검증 제품군 실행 \(p. 351\)](#)
- [AWS IoT Device TesterFreeRTOS 테스트 제품군 버전 \(p. 355\)](#)
- [결과 및 로그 이해 \(p. 356\)](#)

## 사전 조건

이 단원에서는 다음을 사용하여 마이크로 컨트롤러를 테스트하기 위한 사전 조건을 설명합니다.AWS IoT Device Tester.

## FreeRTOS 다운로드

FreeRTOS 릴리스는 다음에서 다운로드할 수 있습니다.[GitHub](#)다음 명령을 실행합니다.

```
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 에 나열된 IDT 버전에 해당하는 FreeRTOS 버전 (예: 202007.00)입니다.[지원되는 버전AWS IoT Device Tester용 FreeRTOS \(p. 315\)](#). 이렇게 하면 하위 모듈을 포함한 전체 소스 코드가 있고 FreeRTOS 버전에 올바른 IDT 버전을 사용하고, 그 반대의 경우도 마찬가지입니다.

Windows의 경우 260자의 경로 길이 제한이 있습니다. FreeRTOS의 경로 구조는 여러 수준으로 깊게 되어 있기 때문에 Windows를 사용하는 경우 파일 경로를 260자 제한 미만으로 유지하십시오. 예를 들어

FreeRTOS 복제c:\FreeRTOS보다는c:\users\username\programs\projects\myproj\FreeRTOS\  
\.

### LTS 자격에 대한 고려 사항 (LTS 라이브러리를 사용하는 FreeRTOS 자격)

- マイクロコントローラが長期間支援(LTS)するバージョンのFreeRTOSをサポートする場合、AWSパートナー デバイス カタログからマニフェスト ファイルを提供する必要があります。詳細な内容は、FreeRTOS ダウンロード チェックリストを参照してください。
- マイクロコントローラが LTS バージョンの FreeRTOS をサポートするか確認する場合は、AWS IoT Device Tester(IDT) FreeRTOS 対応 (FRQ) テスト スイート バージョン v1.4.x が含まれています。
- LTS バージョンの FreeRTOS のサポートは 2020.12.xx バージョンの FreeRTOS で終了します。

### 용 IDT FreeRTOS 다운로드

FreeRTOS의 모든 버전에는 검증 테스트를 수행하기 위한 해당 버전의 용 IDT가 있습니다.에서 적절한 버전의 FreeRTOS 용 IDT를 다운로드합니다.[지원되는 버전AWS IoT Device Tester용 FreeRTOS \(p. 315\)](#).

읽기 및 쓰기 권한이 FreeRTOS 파일 시스템의 위치에 용 IDT의 압축을 풉니다. Microsoft Windows에는 경로 길이에 대한 문자 제한이 있으므로 FreeRTOS의 IDT를 다음과 같은 루트 디렉터리에 압축을 풉니다.c:  
\또는d:\.

#### 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.

### AWS 계정 생성 및 구성

다음 단계에 따라 AWS 계정을 생성하고 구성합니다. AWS 계정, IAM 사용자 및 IAM 정책, 테스트를 실행하는 동안 사용자를 대신하여 리소스에 액세스할 수 있는 권한을 FreeRTOS 용 IDT에 부여하는 IAM 정책.

- 이미 AWS 계정이 있다면 다음 단계로 건너뛰십시오. [AWS 계정](#)을 생성합니다.
- 특정 권한으로 서비스 역할을 생성할 수 있는 IAM 권한을 FreeRTOS 용 IDT에 부여하는 IAM 정책을 생성합니다.
  - IAM 콘솔에 로그인합니다.
  - 탐색 창에서 정책(Policies)을 선택합니다.
  - 콘텐츠 창에서 정책 생성을 선택합니다.
  - JSON 탭을 선택하고 다음 권한을 JSON 텍스트 상자에 복사합니다.

#### Important

다음 정책 템플릿은 역할을 생성하고, 정책을 생성하고, 역할에 정책을 연결할 수 있는 IDT 권한을 부여합니다. 용 FreeRTOS의 IDT는 역할을 생성하는 테스트에 이러한 권한을 사용합니다. 정책 템플릿은 사용자에게 관리자 권한을 제공하지 않지만, AWS 계정에 대한 관리자 액세스 권한을 부여하기 위해 이러한 권한을 잠재적으로 사용할 수 있습니다.

#### Most Regions

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
"Effect": "Allow",
"Action": [
    "iam:CreatePolicy",
    "iam:DetachRolePolicy",
    "iam>DeleteRolePolicy",
    "iam>DeletePolicy",
    "iam>CreateRole",
    "iam>DeleteRole",
    "iam:AttachRolePolicy"
],
"Resource": [
    "arn:aws:iam::*:policy/idt*",
    "arn:aws:iam::*:role/idt*"
]
}
]
```

### Beijing and Ningxia Regions

다음 정책 템플릿은 베이징 및 낭샤 리전에서 사용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam>DeleteRolePolicy",
                "iam>DeletePolicy",
                "iam>CreateRole",
                "iam>DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws-cn:iam::*:policy/idt*",
                "arn:aws-cn:iam::*:role/idt*"
            ]
        }
    ]
}
```

- e. 작업이 완료되면 정책 검토를 선택합니다.
  - f. 검토 페이지에서 정책 이름에 IDTFreeRTOSIAMPermissions를 입력합니다. 정책 요약을 검토하여 정책에 의해 부여된 권한을 확인합니다.
  - g. 정책 생성(Create policy)을 선택합니다.
3. 를 실행하는 데 필요한 권한을 갖는 IAM 사용자를 생성합니다.AWS IoT Device Tester.
- a. 의 1~5단계를 수행합니다.[IAM 사용자 생성 \(콘솔\)](#).
  - b. IAM 사용자에게 필요한 권한을 연결하려면
    - i. Set permissions(권한 설정) 페이지에서 Attach existing policies to user directly(사용자에게 직접 기준 정책 연결)를 선택합니다.
    - ii. 2단계에서 만든 IDTFreeRTOSIAMPermissions 정책을 검색합니다. 확인란을 선택합니다.
    - iii. AWSIoTDeviceTesterForFreeRTOSFullAccess 정책을 검색합니다. 확인란을 선택합니다.
  - c. [다음: 권한(Next: Tags)]를 선택합니다.
  - d. [다음: 권한(Next: 검토를 클릭하여 선택 사항의 요약을 봅니다.
  - e. 사용자 생성을 선택합니다.

- f. 사용자의 액세스 키(액세스 키 ID와 보안 액세스 키)를 보려면 각 암호와 액세스 키 옆에 있는 표시를 선택한 다음 Download.csv를 선택합니다. 파일을 안전한 위치에 저장합니다.

## AWS IoT Device Tester 관리형 정책

AWSIoTDeviceTesterForFreeRTOSFullAccess 관리형 정책에는 디바이스 테스터가 지표를 실행하고 수집할 수 있는 다음 권한이 포함되어 있습니다.

- iot-device-tester:SupportedVersion

용 D에서 사용할 수 있도록 IDT에서 지원하는 FreeRTOS 버전 및 테스트 제품군 버전의 목록을 가져올 수 있는 권한을 부여합니다.AWS CLI.

- 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 수 있는 권한을 AWS IoT Device Tester에 부여합니다.

- iot-device-tester:SendMetrics

AWS IoT Device Tester 사용 지표 데이터를 게시할 권한을 부여합니다.

## (선택 사항) AWS Command Line Interface를 설치합니다.

몇 가지 작업은 AWS CLI를 사용하여 수행할 수도 있습니다. AWS CLI가 설치되지 않은 경우 [AWS CLI 설치](#)의 지침을 따릅니다.

구성AWS CLI의 경우AWS를 실행하여 사용하려는 리전aws configure명령줄에서 를 실행합니다. 에 대한 자세한 내용은AWSFreeRTOS용 IDT를 지원하는 리전은 다음을 참조하십시오.[AWS리전 및 엔드포인트](#). 에 대한 자세한 내용aws configure만나다를 통한 빠른 구성aws configure.

## 처음으로 마이크로 컨트롤러 보드의 테스트 준비

FreeRTOS 인터페이스를 이식할 때 용 IDT를 사용하여 테스트할 수 있습니다. 보드의 디바이스 드라이버에 대한 FreeRTOS 인터페이스를 이식한 후 AWS IoT Device Tester마이크로 컨트롤러 보드에서 검증 테스트를 실행합니다.

### 라이브러리 이식 계층 추가

를 디바이스에 이식하려면[FreeRTOS 이식 안내서](#).

### AWS 자격 증명 구성

다음을 구성해야 합니다.AWS용 자격 증명AWS IoT Device Tester와 통신하기AWS클라우드. 자세한 내용은 단원을 참조하십시오.[설정AWS개발을 위한 자격 증명 및 리전](#). 유효함AWS자격 증명이 에 지정되어 있어야 합니다.[devicetester\\_extract\\_location](#)/[devicetester\\_afreertos\\_\[win/mac/linux\]](#)/configs/config.json구성 파일입니다.

## 용 IDT에서 FreeRTOS 디바이스 폴 만들기

테스트할 디바이스는 디바이스 폴에서 구성됩니다. 각 디바이스 폴은 하나 이상의 동일한 디바이스로 구성됩니다. 폴에서 단일 디바이스를 테스트하거나 폴에서 여러 디바이스를 테스트하도록 FreeRTOS를 구성할 수 있습니다. 자격 프로세스를 가속화하기 위해 용 IDT는 동일한 사양이 있는 디바이스를 병렬로 테스트할 수 있습니다. 라운드 로빈 방법을 사용하여 디바이스 폴에 있는 각 디바이스에 대해 다른 테스트 그룹을 실행합니다.

configs 폴더에서 device.json 템플릿의 devices 섹션을 편집하여 디바이스 폴에 하나 이상의 디바이스를 추가할 수 있습니다.

### Note

동일한 폴에 있는 모든 디바이스는 기술 사양과 SKU가 동일해야 합니다.

다른 테스트 그룹에 대한 소스 코드의 병렬 빌드를 활성화하기 위해 용 IDT는 FreeRTOS 앱축이 풀린 폴더 내부의 결과 폴더에 소스 코드를 복사합니다. 빌드 또는 플래시 명령에 있는 소스 코드 경로는 다음 중 하나를 사용하여 참조해야 합니다.`testdata.sourcePath` 또는 `sdkPath` 변수. 용 FreeRTOS는 이 변수를 복사된 소스 코드의 임시 경로로 바꿉니다. 자세한 내용은 [용 IDT FreeRTOS 변수 \(p. 340\)](#) 단원을 참조하십시오.

다음은 여러 디바이스가 있는 디바이스 폴을 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"  
          }  
        ]  
      },  
      {  
        "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 폴을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 폴에 속한 디바이스는 동일한 유형이어야 합니다. 테스트 제품군을 실행할 때 폴에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다.

#### **sku**

테스트하는 보드를 고유하게 식별하는 영숫자 값입니다. SKU는 정규화된 보드를 추적하는 데 사용됩니다.

#### **Note**

보드를 리스트팅하려는 경우 AWS Partner Device Catalog의 경우 여기서 지정하는 SKU가 목록 등록 프로세스에 사용하는 SKU와 일치해야 합니다.

#### **features**

디바이스의 지원되는 기능이 포함된 배열입니다. AWS IoT Device Tester에서는 이 정보를 사용하여 실행할 자격 테스트를 선택합니다.

지원되는 값은 다음과 같습니다.

#### **TCP/IP**

보드가 TCP/IP 스택을 지원하는지 여부 및 보드가 온칩에서 지원되는지(MCU) 또는 다른 모듈로 오 프로드되는지를 나타냅니다. 검증에는 TCP/IP가 필요합니다.

#### **WIFI**

보드에 Wi-Fi 기능이 있는지 여부를 나타냅니다. 로 설정해야 합니다. No 만약 Cellular 다음의 경우 이로 설정됩니다. Yes.

### Cellular

보드에 셀룰러 기능이 있는지 여부를 나타냅니다. 로 설정해야 합니다. No 만약 WIFI 다음의 경우 이로 설정됩니다. Yes. 이 기능이 로 설정된 경우 Yes, 전체 보안 소켓 테스트는 다음을 사용하여 실행됩니다. AWS IoT micro EC2 인스턴스로 인해 계정에 추가 비용이 발생할 수 있습니다. 자세한 내용은 [Amazon EC2 요금](#)을 참조하세요.

### TLS

보드가 TLS를 지원하는지 여부를 나타냅니다. 검증에는 TLS가 필요합니다.

### PKCS#11

보드가 지원하는 퍼블릭 키 암호화 알고리즘을 나타냅니다. 검증에는 PKCS#11이 필요합니다. 지원되는 값은 ECC, RSA, Both, No입니다. Both는 ECC 및 RSA 알고리즘을 모두 지원하는 보드를 가리킵니다.

### Key Provisioning

신뢰할 수 있는 X.509 클라이언트 인증서를 보드에 작성하는 방법을 나타냅니다. 유효 값은 Import, Onboard 및 No입니다. 검증에는 키 프로비저닝이 필요합니다.

- 보드에서 프라이빗 키 가져오기를 허용하는 경우에는 Import를 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 Onboard를 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.

### Note

IDT v4.1.0부터 시작합니다. OtaDataPlaneProtocol만 수락합니다. MQTT과 HTTP 지원도는 값입니다.

### BLE

보드가 Bluetooth Low Energy(BLE)를 지원하는지 여부를 나타냅니다.

### devices.id

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

### devices.connectivity.protocol

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 지원되는 값: uart.

### devices.connectivity.serialPort

테스트 할 디바이스에 연결하는 데 사용되는 호스트 컴퓨터의 직렬 포트입니다.

### devices.secureElementConfig.PublicKeyAsciiHexFilePath

온보드 프라이빗 키에서 추출한 16진수 바이트 퍼블릭 키를 포함하는 파일에 대한 절대 경로입니다.

형식 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  
dcbb 4e8d 75b3 2586 e2cc 0c
```

공개 키가.der 형식인 경우 공개 키를 직접 16 진수로 인코딩하여 16 진수 파일을 생성할 수 있습니다.

16진수 파일을 생성하는.der 공개 키에 대한 예제 명령:

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

공개 키가.pem 형식이면 base64로 인코딩된 부분을 추출하여 바이너리 형식으로 디코딩한 다음 16진수 인코딩하여 16 진수 파일을 생성할 수 있습니다.

예를 들어 다음 명령을 사용하여.pem 공개 키에 대한 16진수 파일을 생성합니다.

1. key의 base64로 인코딩 된 부분을 꺼내 (머리글과 바닥글 제거) 파일에 저장합니다 (예: 이름 지정)base64key를 실행하여.der 형식으로 변환하려면 다음 명령을 실행하십시오.

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

2. 실행xxd명령을 사용하여 16 진수 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 잡긴 자격 증명이 있는 사전 프로비저닝된 보안 요소가 있고 객체를 가져오거나 만들거나 제거할 수 없는 경우 “예”로 설정합니다. 이 구성은 경우에만 적용됩니다.features가지고 있습니다KeyProvisioning다음과 함께 “온보드”로 설정PKCS11“ECC”로 설정합니다.

#### **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**

용 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 로드하여 Echo Server 구성을 지원합니다.customPath. 자세한 내용은 단원을 참조하십시오.[에코 서버 설정](#)의FreeRTOS 이식 안내서. 다음 JSON 예제에서는 여러 디바이스를 테스트하기 위해 FreeRTOS를 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": [
        "{{config.idtRootPath}}/relative-path-to/build-parallel.sh
{{testData.sourcePath}} {{enableTests}}"
    ]
},
"flashTool": {
    "name": "your-flash-tool-name",
    "version": "your-flash-tool-version",
    "command": [
        "/{{config.idtRootPath}}/relative-path-to/flash-parallel.sh
{{testData.sourcePath}} {{device.connectivity.serialPort}} {{buildImageName}}"
    ],
    "buildImageInfo" : {
        "testsImageName": "tests-image-name",
        "demosImageName": "demos-image-name"
    }
},
"testStartDelayms": 0,
"clientWifiConfig": {
    "wifiSSID": "ssid",
    " wifiPassword": "password",
    "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
},
"testWifiConfig": {
    "wifiSSID": "ssid",
    " wifiPassword": "password",
    "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
},
//*****
//This section is used to start echo server based on server certificate generation
method,
//When certificateGenerationMethod is set as Automatic specify the eccCurveFormat to
generate certifcate and key based on curve format,
//When certificateGenerationMethod is set as Custom specify the certificatePath and
PrivateKeyPath to be used to start echo server
//*****
"echoServerCertificateConfiguration": {
    "certificateGenerationMethod": "Automatic | Custom",
    "customPath": {
        "clientCertificatePath": "/path/to/clientCertificate",
        "clientPrivateKeyPath": "/path/to/clientPrivateKey",
        "serverCertificatePath": "/path/to/serverCertificate",
        "serverPrivateKeyPath": "/path/to/serverPrivateKey"
    },
    "eccCurveFormat": "P224 | P256 | P384 | P521"
},
"echoServerConfiguration": {
    "securePortForSecureSocket": 33333, // Secure tcp port used by SecureSocket test.
Default value is 33333. Ensure that the port configured isn't blocked by the firewall or
your corporate network
}

```

```

        "insecurePortForSecureSocket": 33334, // Insecure tcp port used by SecureSocket
        test. Default value is 33334. Ensure that the port configured isn't blocked by the
        firewall or your corporate network
        "insecurePortForWiFi": 33335 // Insecure tcp port used by Wi-Fi test. Default value
        is 33335. Ensure that the port configured isn't blocked by the firewall or your corporate
        network
    },
    "otaConfiguration": {
        "otaFirmwareFilePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
        "deviceFirmwareFileName": "ota-image-name-on-device",
        "otaDemoConfigFilePath": "{{testData.sourcePath}}/relative-path-to/ota-demo-config-
header-file",
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
            "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": boolean,
            // *****Use signerPlatform if you choose aws for
            signingMethod*****
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            // *****Use signCommand if you choose custom for
            signingMethod*****
            "signCommand": [
                "/absolute-path-to/sign.sh {{inputImagePath}}
                {{outputSignatureFilePath}}"
            ]
        }
    },
    // *****Remove the section below if you're not configuring CMake*****
    "cmakeConfiguration": {
        "boardName": "board-name",
        "vendorName": "vendor-name",
        "compilerName": "compiler-name",
        "frToolchainPath": "/path/to/freertos/toolchain",
        "cmakeToolchainPath": "/path/to/cmake/toolchain"
    },
    "freertosFileConfiguration": {
        "required": [
            {
                "configName": "pkcs11Config",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-name/
aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-name/
aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
            {
                "configName": "otaAgentTestsConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-name/
aws_tests/config_files/ota_config.h"
            },
            {
                "configName": "otaAgentDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-name/
aws_demos/config_files/ota_config.h"
            }
        ],
    }
}

```

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

다음 목록에서는 `userdata.json`에서 사용되는 속성을 나열합니다.

#### **sourcePath**

이식된 FreeRTOS 소스 코드의 루트에 대한 경로입니다. SDK를 사용한 병렬 테스트의 경우 `sourcePath` 다음을 사용하여 설정할 수 있습니다. `userData.sdkConfiguration.path` 자리 표시자. 예제:

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

#### **vendorPath**

공급업체별 FreeRTOS 코드에 대한 경로입니다. 직렬 테스트의 경우 `vendorPath`을 절대 경로로 설정 할 수 있습니다. 예제:

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

병렬 테스트의 경우 `vendorPath`을 `{{testData.sourcePath}}` 자리 표시자를 사용하여 설정할 수 있습니다. 예제:

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

이 `vendorPath` 변수는 SDK 없이 실행할 때만 필요하며, 그렇지 않으면 제거할 수 있습니다.

#### Note

SDK 없이 병렬로 테스트를 실행할 때 `{{testData.sourcePath}}` 다음 위치에 자리 표시자를 사용해야 합니다. `vendorPath, buildTool, flashTool` 필드. 단일 디바이스로 테스트를 실행하는 경우 `vendorPath, buildTool, flashTool` 필드에 절대 경로를 사용해야 합니다. SDK를 사용하여 실행하는 경우 `{{sdkPath}}` 다음 위치에 자리 표시자를 사용해야 합니다. `sourcePath, buildTool, 및 flashTool` 명령.

#### **sdkConfiguration**

포팅에 필요한 것 이상으로 파일 및 폴더 구조를 수정한 FreeRTOS를 검증하는 경우 이 블록에서 SDK 정보를 구성해야 합니다. SDK 내에서 포팅된 FreeRTOS를 사용할 수 없는 경우 이 블록을 완전히 생략해야 합니다.

##### **sdkConfiguration.name**

FreeRTOS에서 사용하고 있는 SDK의 이름입니다. SDK를 사용하지 않는 경우 전체 `sdkConfiguration` 블록을 생략해야 합니다.

##### **sdkConfiguration.version**

FreeRTOS에서 사용하고 있는 SDK의 버전입니다. SDK를 사용하지 않는 경우 전체 `sdkConfiguration` 블록을 생략해야 합니다.

##### **sdkConfiguration.path**

FreeRTOS 코드가 포함된 SDK 딜렉토리의 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 명령에 있는 소스 코드 경로에 대한 모든 참조는 AWS IoT Device Tester변하기 쉬운{{testdata.sourcePath}}SDK 경로에 대한 참조는 다음과 같이 바꿔야 합니다.{{sdkPath}}. 사용{{config.idtRootPath}}절대 또는 상대 IDT 경로를 참조하는 자리 표시자입니다.

#### **testStartDelayms**

FreeRTOS 테스트 러너가 테스트 실행을 시작하기 전에 대기할 시간 (밀리초)을 지정합니다. IDT가 네트워크 또는 기타 대기 시간으로 인해 연결하고 로깅을 시작하기 전에 테스트 중인 장치가 중요한 테스트 정보를 출력하기 시작하는 경우에 유용할 수 있습니다. 허용되는 최대 값은 30000밀리초 (30초)입니다. 이 값은 FreeRTOS 테스트 그룹에만 적용되며 OTA 테스트와 같이 FreeRTOS 테스트 러너를 사용하지 않는 다른 테스트 그룹에는 적용되지 않습니다.

#### **flashTool**

디바이스에 대한 플래시 명령이 포함된 플래시 스크립트(.sh 또는 .bat)의 전체 경로입니다. 플래시 명령에 있는 소스 코드 경로에 대한 모든 참조는 FreeRTOS 변수로 바꿔야 합니다.{{testdata.sourcePath}}SDK경로에 대한 모든 참조는 용 IDT 변수로 바꿔야 합니다.{{sdkPath}}.Use{{config.idtRootPath}}절대 또는 상대 IDT 경로를 참조하는 자리 표시자입니다.

#### **buildImageInfo**

##### **testsImageName**

에서 테스트를 빌드할 때 빌드 명령으로 인해 생성되는 파일의 이름입니다.*freertos-source/testsfolder*.

##### **demosImageName**

에서 테스트를 빌드할 때 빌드 명령으로 인해 생성되는 파일의 이름입니다.*freertos-source/demosfolder*.

#### **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 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있는지 확인하십시오.

##### **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 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있는지 확인하십시오.

##### **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 또는 수동으로 제공할지 여부를 지정합니다.

##### **customPath**

다음의 경우, certificateGenerationMethod “사용자 지정”입니다.  
.certificatePath과 privateKeyPath은 필수입니다.

##### **certificatePath**

서버 인증서의 파일 경로를 지정합니다.

##### **privateKeyPath**

프라이빗 키의 파일 경로를 지정합니다.

##### **eccCurveFormat**

보드에서 지원하는 곡선 형식을 지정합니다. 필요한 경우 PKCS11에서 “ecc”로 설정됩니다.  
.device.json. 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. 예: {{testData.sourcePath}}/relative-path/to/ota/image/from/source/root.

#### **deviceFirmwareFileName**

MCU 디바이스에 OTA 펌웨어가 위치하는 전체 파일 경로입니다. 일부 디바이스는 이 필드를 사용하지 않지만 그래도 값을 입력해야 합니다.

#### **otaDemoConfigFilePath**

aws\_demo\_config.h에 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 Custom입니다.

##### Note

베이징 및 낭샤 지역의 경우Custom.AWS이러한 리전에서는 코드 서명이 지원되지 않습니다.

#### **signerHashingAlgorithm**

디바이스에서 지원되는 해싱 알고리즘입니다. 가능한 값은 SHA1 또는 SHA256입니다.

#### **signerSigningAlgorithm**

디바이스에서 지원되는 서명 알고리즘입니다. 가능한 값은 RSA 또는 ECDSA입니다.

#### **signerCertificate**

OTA에 사용되는 신뢰할 수 있는 인증서입니다.

AWS 코드 서명 방법의 경우 AWS Certificate Manager에 업로드된 신뢰할 수 있는 인증서에 Amazon 리소스 이름(ARN)을 사용합니다.

사용자 정의 코드 서명 방법의 경우 서명자 인증서 파일의 절대 경로를 사용합니다.

신뢰할 수 있는 인증서 생성에 대한 자세한 내용은 [코드 서명 인증서 생성 \(p. 144\)](#) 단원을 참조하십시오.

#### **signerCertificateFileName**

디바이스에 있는 코드 서명 인증서의 파일 이름입니다. 이 값은 실행 시 제공한 파일 이름과 일치해야 합니다.`.aws acm import-certificate` 명령.

자세한 정보는 [코드 서명 인증서 생성 \(p. 144\)](#)을 참조하십시오.

#### **compileSignerCertificate**

코드 서명자 서명 확인 인증서를 프로비저닝하거나 플래시하지 않는 경우 `true`로 설정합니다. 따라서 이 항목을 프로젝트로 컴파일해야 합니다. AWS IoT Device Tester는 신뢰할 수 있는 인증서를 가져와서 `aws_codesigner_certificate.h`에 컴파일합니다.

#### **untrustedSignerCertificate**

일부 OTA 테스트에서 신뢰할 수 없는 인증서로 사용되는 두 번째 인증서의 ARN 또는 파일 경로입니다. 인증서 생성에 대한 자세한 내용은 단원을 참조하십시오. [코드 서명 인증서 생성](#).

#### **signerPlatform**

AWS Code Signer에서 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`을 구성합니다

#### **signCommand**

사용자 정의 코드 서명을 수행하는 데 사용되는 명령입니다. `/configs/script_templates` 디렉터리에서 템플릿을 찾을 수 있습니다.

명령에 두 자리 표시자 `{{inputImagePath}}` 및 `{{outputSignatureFilePath}}`가 필요합니다. `{{inputImagePath}}`는 서명하기 위해 IDT에서 만든 이미지의 파일 경로입니다. `{{outputSignatureFilePath}}`는 스크립트에서 생성하는 서명의 파일 경로입니다.

#### **cmakeConfiguration**

CMake 구성[선택 사항]

##### **Note**

CMake 테스트 사례를 실행하려면 보드 이름, 공급업체 이름 및 `frToolchainPath` 또는 `compilerName`을 제공해야 합니다. 또한 다음을 제공할 수 있습니다. `cmakeToolchainPath` CMake 도구 체인의 사용자 지정 경로가 있는 경우

#### **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**

구성 중인 테스트의 이름입니다.

**filePath**

에 있는 구성 파일에 대한 절대 경로입니다.[freertosrepo](#)입니다. 사용 {{testData.sourcePath}} 변수를 사용하여 경로를 정의합니다.

**optional**

이 섹션에서는 구성 파일을 이동한 선택적 테스트 (예: OTA, WiFi 등) 를 지정합니다.

**configName**

구성 중인 테스트의 이름입니다.

**filePath**

에 있는 구성 파일에 대한 절대 경로입니다.[freertosrepo](#)입니다. 사용 {{testData.sourcePath}} 변수를 사용하여 경로를 정의합니다.

**Note**

CMake 테스트 사례를 실행하려면 보드 이름, 공급업체 이름 및 afrToolchainPath 또는 compilerName을 제공해야 합니다. CMake 도구 체인에 대한 사용자 지정 경로가 있다면 cmakeToolchainPath를 제공할 수도 있습니다.

## 용 IDT FreeRTOS 변수

코드를 빌드하고 디바이스를 플래시하는 명령을 성공적으로 실행하려면 연결 및 디바이스에 대한 기타 정보가 필요할 수 있습니다.AWS IoT Device Tester를 사용하여 플래시 및 빌드 명령에 있는 디바이스 정보를 참조할 수 있습니다.[JsonPath](#). 단순 JsonPath 표현식을 사용하여 device.json 파일에 지정된 필수 정보를 가져올 수 있습니다.

### 경로 변수

FreeRTOS 용 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}}**

당신의 가치로 확장됩니다.userData.sdkConfiguration.path 빌드 및 플래시 명령에 사용되는 경우

**{{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.

```
 {{config.idtRootPath}}
```

로 확장합니다.AWS IoT Device Tester루트 경로. 이 변수는 빌드 및 플래시 명령에서 사용할 때 IDT의 절대 경로를 대체합니다.

## FreeRTOS용 IDT 사용자 인터페이스를 사용하여 FreeRTOS 인증 제품군을 실행합니다.

IDT v4.3.0부터 시작하여 AWS IoT Device Testerfor FreeRTOS (IDT-Freertos)에는 IDT 명령줄 실행 파일 및 관련 구성 파일과 상호 작용할 수 있는 웹 기반 사용자 인터페이스가 포함되어 있습니다. IDT-Freertos UI를 사용하여 새 구성을 생성하여 IDT 테스트를 실행하거나 기존 구성을 수정할 수 있습니다. UI를 사용하여 IDT 실행 파일을 호출하고 테스트를 실행할 수도 있습니다.

IDT-Freertos UI는 다음과 같은 기능을 제공합니다.

- IDT-Freertos 테스트를 위한 구성 파일 설정을 간소화합니다.
- IDT-Freertos를 사용하여 검증 테스트를 간단하게 실행할 수 있습니다.

명령줄을 사용하여 검증 테스트 실행에 대한 자세한 내용은 단원을 참조하십시오.[처음으로 마이크로 컨트롤러 보드의 테스트 준비 \(p. 328\)](#).

이 단원에서는 IDT-Freertos UI를 사용하기 위한 사전 요구 사항에 대해 설명하고 UI에서 자격 테스트 실행을 시작하는 방법을 설명합니다.

### 주제

- [사전 조건 \(p. 341\)](#)
- [IDT FreeRTOS UI 시작하기 \(p. 345\)](#)

## 사전 조건

이 단원에서는 다음을 사용하여 마이크로 컨트롤러를 테스트하기 위한 사전 조건을 설명합니다.AWS IoT Device Tester.

### 주제

- [지원되는 웹 브라우저 사용 \(p. 342\)](#)
- [FreeRTOS 다운로드 \(p. 342\)](#)
- [FreeRTOS 용 IDT 다운로드 \(p. 342\)](#)
- [AWS 계정 생성 및 구성 \(p. 343\)](#)

- AWS IoT Device Tester 관리형 정책 (p. 344)

## 지원되는 웹 브라우저 사용

IDT-Freertos UI는 다음 웹 브라우저를 지원합니다.

브라우저	버전
Google Chrome	최신 3개 주요 버전
Mozilla Firefox	최신 3개 주요 버전
Microsoft Edge	최신 3개 주요 버전
macOS용 Apple Safari	최신 3개 주요 버전

더 나은 경험을 위해 구글 크롬이나 모질라 파이어폭스를 사용하는 것이 좋습니다.

### Note

IDT-프리어토스 UI는 마이크로소프트 인터넷 익스플로러를 지원하지 않습니다.

## FreeRTOS 다운로드

FreeRTOS 릴리스는 다음에서 다운로드할 수 있습니다.[GitHub](#) 다음 명령을 사용합니다.

```
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 예나열된 IDT 버전에 해당하는 FreeRTOS 버전 (예: 202007.00)입니다. [지원되는 버전AWS IoT Device Tester용 FreeRTOS \(p. 315\)](#). 이렇게 하면 하위 모듈을 포함한 전체 소스 코드가 있고 FreeRTOS 버전에 올바른 IDT 버전을 사용하고, 그 반대의 경우도 마찬가지입니다.

Windows의 경우 260자의 경로 길이 제한이 있습니다. FreeRTOS의 경로 구조는 여러 수준으로 깊게 되어 있기 때문에 Windows를 사용하는 경우 파일 경로를 260자 제한 미만으로 유지하십시오. 예를 들어 FreeRTOS 복제C:\FreeRTOS보다는C:\Users\username\programs\projects\myproj\FreeRTOS\.

### LTS 자격에 대한 고려 사항 (LTS 라이브러리를 사용하는 FreeRTOS 자격)

- 마이크로컨트롤러가 장기 지원 (LTS) 기반 버전의 FreeRTOS를 지원하는 것으로 지정되도록 하려면 AWS 파트너 디바이스 카탈로그의 경우 매니페스트 파일을 제공해야 합니다. 자세한 내용은 단원을 참조하십시오. [FreeRTOS 검증 체크리스트](#)와 FreeRTOS 검증 안내서.
- 마이크로컨트롤러가 LTS 기반 버전의 FreeRTOS를 지원하는지 검증하고 AWS 파트너 장치 카탈로그, 다음을 사용해야 합니다. AWS IoT Device Tester(IDT) FreeRTOS 인증 (FRQ) 테스트 스위트 버전 v1.4.x가 포함되어 있습니다.
- LTS 기반 버전의 FreeRTOS 지원은 202012.xx 버전의 FreeRTOS로 제한됩니다.

## FreeRTOS 용 IDT 다운로드

모든 버전의 FreeRTOS에는 검증 테스트를 수행하기 위한 해당 버전의 용 IDT가 있습니다. 에서 적절한 버전의 FreeRT를 다운로드합니다. [지원되는 버전AWS IoT Device Tester용 FreeRTOS \(p. 315\)](#).

읽기 및 쓰기 권한이 FreeRTOS 파일 시스템의 위치에 용 IDT의 압축을 풉니다. Microsoft Windows에는 경로 길이에 대한 문자 제한이 있으므로 다음과 같은 루트 디렉터리에 용 IDT의 압축을 풉니다.c:\ 또는 d:\.

Note

로컬 드라이브에 IDT 패키지의 압축을 풉니다. 여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하도록 허용하면 시스템이 응답하지 않거나 데이터가 손상될 수 있습니다.

## AWS 계정 생성 및 구성

다음 단계에 따라 AWS 계정, IAM 사용자 및 테스트를 실행하는 동안 사용자를 대신하여 리소스에 액세스할 수 있는 권한을 용 IDT에 용 FreeRTOS 용 IDT에 부여하는 IAM 정책

1. 이미 AWS 계정이 있다면 다음 단계로 건너뛰십시오. [AWS 계정](#)을 생성합니다.
2. 특정 권한으로 서비스 역할을 생성할 수 있는 IAM 권한을 용 IDT에 부여하는 IAM 정책을 생성합니다.
  - a. [IAM 콘솔](#)에 로그인합니다.
  - b. 탐색 창에서 정책(Policies)을 선택합니다.
  - c. 콘텐츠 창에서 정책 생성을 선택합니다.
  - d. Choose the JSON 탭을 클릭하고 다음 권한을 JSON 텍스트 상자.

Important

다음 정책 템플릿은 역할을 생성하고, 정책을 생성하고, 역할에 정책을 연결할 수 있는 IDT 권한을 부여합니다. 용 IDT는 역할을 생성하는 테스트에 이러한 권한을 사용합니다. 정책 템플릿은 사용자에게 관리자 권한을 제공하지 않지만, 사용자에게 관리자 액세스 권한을 부여하기 위해 이러한 권한을 잠재적으로 사용할 수 있습니다. AWS 계정.

### Most Regions

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:CreatePolicy",  
                "iam:DetachRolePolicy",  
                "iam:DeleteRolePolicy",  
                "iam:DeletePolicy",  
                "iam:CreateRole",  
                "iam:DeleteRole",  
                "iam:AttachRolePolicy"  
            ],  
            "Resource": [  
                "arn:aws:iam::*:policy/idt*",  
                "arn:aws:iam::*:role/idt*"  
            ]  
        }  
    ]  
}
```

### Beijing and Ningxia Regions

다음 정책 템플릿은 베이징 및 낭샤 리전에서 사용할 수 있습니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreatePolicy",
        "iam:DetachRolePolicy",
        "iam:DeleteRolePolicy",
        "iam:DeletePolicy",
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:AttachRolePolicy"
    ],
    "Resource": [
        "arn:aws-cn:iam::*:policy/idt*",
        "arn:aws-cn:iam::*:role/idt*"
    ]
}
}
```

- e. 작업이 완료되면 정책 검토를 선택합니다.
- f. 검토 페이지에서 정책 이름에 **IDTFreeRTOSIAMPermissions**을 입력합니다. 정책 요약을 검토하여 정책에 의해 부여된 권한을 확인합니다.
- g. 정책 생성(Create policy)을 선택합니다.
3. 를 실행하는 데 필요한 권한이 있는 IAM 사용자를 생성합니다.AWS IoT Device Tester.
  - a. 의 1~5단계를 수행합니다.[IAM 사용자 생성 \(콘솔\)](#).
  - b. IAM 사용자에게 필요한 권한을 연결하려면
    - i. Set permissions(권한 설정) 페이지에서 Attach existing policies to user directly(사용자에게 직접 기준 정책 연결)를 선택합니다.
    - ii. 2단계에서 만든 IDTFreeRTOSIAMPermissions 정책을 검색합니다. 확인란을 선택합니다.
    - iii. AWSIoTDeviceTesterForFreeRTOSFullAccess 정책을 검색합니다. 확인란을 선택합니다.
  - c. [다음: 권한(Next: Tags)]를 선택합니다.
  - d. 선택 사항의 요약을 보려면 다음: Review]를 선택합니다.
  - e. 사용자 생성을 선택합니다.
  - f. 사용자의 액세스 키(액세스 키 ID와 보안 액세스 키)를 보려면 각 암호와 액세스 키 옆에 있는 표시를 선택한 다음 Download.csv를 선택합니다. 파일을 안전한 위치에 저장합니다.

## AWS IoT Device Tester 관리형 정책

장치 테스터를 실행하고 메트릭을 수집할 수 있도록 하려면AWSIoTDeviceTesterForFreeRTOSFullAccess 관리형 정책에는 다음 권한이 포함됩니다.

- **iot-device-tester:SupportedVersion**

에서 사용할 수 있도록 IDT에서 지원하는 FreeRTOS 버전 및 테스트 제품군 버전의 목록을 가져올 수 있는 권한을 부여합니다.AWS CLI.

- **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 수 있는 권한을 AWS IoT Device Tester에 부여합니다.

- iot-device-tester:SendMetrics

AWS IoT Device Tester 사용 지표 데이터를 게시할 권한을 부여합니다.

## IDT FreeRTOS UI 시작하기

이 섹션에서는 IDT-Freertos UI를 사용하여 구성 생성 또는 수정하는 방법을 보여 주는 다음 테스트 실행 방법을 보여 줍니다.

### 주제

- [구성 AWS 신입장 \(p. 345\)](#)
- [IDT 프리어토스 UI 열기 \(p. 346\)](#)
- [새 구성 생성합니다. \(p. 346\)](#)
- [기존 구성 수정 \(p. 347\)](#)
- [검증 테스트 실행 \(p. 348\)](#)

## 구성 AWS 신입장

에 대해 IAM 사용자 자격 증명을 구성해야 합니다. AWS에서 만든 사용자 [AWS 계정 생성 및 구성 \(p. 343\)](#). 두 가지 방법 중 하나로 자격 증명을 지정할 수 있습니다.

- 자격 증명 파일
- 환경 변수로

### 구성 AWS 자격 증명 파일이 있는 자격 증명

IDT는 AWS CLI와 동일한 자격 증명 파일을 사용합니다. 자세한 내용은 [구성 및 자격 증명 파일을 참조하십시오.](#)

자격 증명 파일의 위치는 사용 중인 운영 체제에 따라 달라집니다.

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

사용자 추가 AWS에 대한 자격 증명 `credentials` 다음 형식의 파일:

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

### Note

를 사용하지 않는 경우 `default` AWS 프로필, IDT-Freertos UI에서 프로필 이름을 지정해야 합니다.  
프로파일에 대한 자세한 내용은 단원을 참조하십시오. [명명된 프로파일](#).

### 구성 AWS 환경 변수가 있는 자격 증명

환경 변수는 운영 체제에서 유지 관리하고 시스템 명령에서 사용하는 변수입니다. SSH 세션을 닫으면 이러한 항목이 저장되지 않습니다. IDT 프리어토스 UI는 `AWS_ACCESS_KEY_ID`과 `AWS_SECRET_ACCESS_KEY` 저장할 환경 변수 AWS 자격 증명.

Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 `export`를 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>
```

## IDT 프리어토스 UI 열기

IDT 프리어토스 UI를 열려면

1. 지원되는 IDT-Freertos 버전을 다운로드하고 읽기 및 쓰기 권한이 있는 파일 시스템의 위치에 다운로드 한 아카이브의 압축을 풉니다.
2. 다음 명령을 실행하여 IDT-Freertos 설치 디렉터리로 이동합니다.

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

3. 다음 명령을 실행하여 IDT-Freertos UI를 엽니다.

Linux

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

Windows

```
./devicetestergui_win_x64-64
```

macOS

```
./devicetestergui_mac_x86-64
```

### Note

Mac의 경우 시스템에서 UI를 실행할 수 있도록 하려면 시스템 환경 설정 -> 보안 및 개인 정보. 테스트를 실행할 때 이 작업을 세 번 더 수행해야 할 수 있습니다.

기본 브라우저에서 IDT-FreeRTOS UI가 열립니다. 지원되는 브라우저에 대한 자세한 내용은 단원을 참조하십시오.[지원되는 웹 브라우저 사용 \(p. 342\)](#).

## 새 구성을 생성합니다.

처음 사용하는 사용자인 경우 IDT-Freertos에서 테스트를 실행하는 데 필요한 JSON 구성 파일을 설정하려면 새 구성을 만들어야 합니다. 그런 다음 테스트를 실행하거나 생성된 구성을 수정할 수 있습니다.

다음 예제를 보려면 config.json, device.json, 및 userdata.json 파일, 을 참조하십시오.[처음으로 마이크로 컨트롤러 보드의 테스트 준비 \(p. 328\)](#). 예를 들어 resource.json Bluetooth Low Energy (BLE) 테스트를 실행하는 데에만 사용되는 파일은 [Bluetooth Low Energy 테스트 실행 \(p. 348\)](#).

### 새 구성 생성

1. IDT-Freertos UI에서 탐색 메뉴를 연 다음 새 구성 생성.

### Important

You must configure your AWSUI를 열기 전에 자격 증명입니다. 자격 증명을 구성하지 않은 경우 IDT-Freertos UI 브라우저 창을 닫고 다음 단계를 수행합니다.[구성AWS신입장 \(p. 345\)](#) IDT-Freertos UI를 다시 엽니다.

2. 구성 마법사에 따라 검증 테스트를 실행하는 데 사용되는 IDT 구성 설정을 입력합니다. 마법사는 에 있는 JSON 구성 파일에서 다음 설정을 구성합니다.[\*devicetester-extract-location/config\*](#) 디렉터리.

- AWS설정—AWS 계정IDT-프리어토스가 만드는데 사용하는 정보 AWS 테스트 실행 중 리소스입니다. 이러한 설정은 에서 구성됩니다.*config.json* 파일.
- FreeRTOS 저장소—FreeRTOS 저장소 및 포팅된 코드에 대한 절대 경로와 수행할 자격 유형입니다. 이러한 설정은 에서 구성됩니다.*userdata.json* 파일.

자격 테스트를 실행하려면 먼저 디바이스에 대해 FreeRTOS를 포팅해야 합니다. 자세한 내용은 단원을 참조하십시오.[FreeRTOS 이식](#)

- 빌드 및 플래시—IDT에서 자동으로 테스트를 빌드하고 플래시하는 데 사용할 수 있는 하드웨어에 대한 빌드 및 플래시 명령입니다. 이러한 설정은 에서 구성됩니다.*userdata.json* 파일.
- Devices—테스트할 디바이스에 대한 디바이스 풀 설정입니다. 이러한 설정은 에서 구성됩니다.*.id* 과 *sku* 필드와 *devices*에 있는 장치 풀에 대한 블록 *device.json* 파일.
- 네트워킹—장치에 대한 네트워크 통신 지원을 테스트하기 위한 설정입니다. 이러한 설정은 에서 구성됩니다.*features*의 블록은 *device.json* 파일 및 *clientWifiConfig* 과 *testWifiConfig*의 블록은 *userdata.json* 파일.
- 에코 서버—보안 소켓 테스트를 위한 echo 서버 구성 설정입니다. 이러한 설정은 에서 구성됩니다.*userdata.json* 파일.

echo 서버 구성에 대한 자세한 내용은 단원을 참조하십시오.<https://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html>.

- CMake—(선택 사항) CMake 빌드 기능 테스트를 실행하기 위한 설정입니다. 이 구성은 빌드 시스템으로 CMake를 사용하는 경우에만 필요합니다. 이러한 설정은 에서 구성됩니다.*userdata.json* 파일.
- BLE—블루투스 저에너지 기능 테스트를 실행하기 위한 설정입니다. 이러한 설정은 에서 구성됩니다.*features*의 블록은 *device.json* 파일 및 *resource.json* 파일.
- OTA—OTA 기능 테스트를 실행하기 위한 설정입니다. 이러한 설정은 에서 구성됩니다.*features*의 블록은 *device.json* 파일 및 *userdata.json* 파일.

3. 온검토페이지에서 구성 정보를 확인합니다.

구성 검토를 마친 후 검증 테스트를 실행하려면 테스트 실행.

### 기존 구성 수정

IDT에 대한 구성 파일을 이미 설정한 경우 IDT-Freertos UI를 사용하여 기존 구성 설정을 수정할 수 있습니다. 에서 기존 구성 파일을 사용할 수 있는지 확인하십시오.[\*devicetester-extract-location/config\*](#) 디렉터리.

#### 새 구성 설정하려면

1. IDT-Freertos UI에서 탐색 메뉴를 연 다음 기존 구성 편집.

구성 대시보드에는 기존 구성 설정에 대한 정보가 표시됩니다. 구성이 올바르지 않거나 사용할 수 없는 경우 해당 구성의 상태는 다음과 같습니다.*Error validating configuration*.

2. 기존 구성 설정을 수정하려면 다음 단계를 완료하십시오.
  - a. 구성 설정의 이름을 선택하여 설정 페이지를 엽니다.

- b. 설정을 수정한 후 Save를 눌러 해당 구성 파일을 재생성합니다.

구성 수정을 마친 후 모든 구성 설정이 유효성 검사를 통과하는지 확인합니다. 각 구성 설정의 상태가 다음과 같은 경우 Valid, 이 구성은 사용하여 자격 테스트를 실행할 수 있습니다.

## 검증 테스트 실행

IDT-Freertos에 대한 구성은 생성한 후 자격 테스트를 실행할 수 있습니다.

검증 테스트를 실행하려면

1. 구성을 확인합니다.
2. 탐색 메뉴에서 테스트 실행.
3. 테스트 실행을 시작하려면 테스트 시작.

IDT-Freertos는 검증 테스트를 실행하고 테스트 실행 요약과 오류를 표시합니다. 테스트 콘솔. 테스트 실행이 완료되면 다음 위치에서 테스트 결과와 로그를 볼 수 있습니다.

- 테스트 결과는 다음 위치에 있습니다. `devicetester-extract-location/results/execution-id` 디렉터리.
- 테스트 로그는 `devicetester-extract-location/results/execution-id/logs` 디렉터리.

테스트 결과 및 로그에 대한 자세한 내용은 단원을 참조하십시오. [결과 및 로그 이해 \(p. 356\)](#).

## Bluetooth Low Energy 테스트 실행

이 단원에서는 를 사용하여 Bluetooth 테스트를 설정하고 실행하는 방법을 설명합니다. AWS IoT Device Tester용 FreeRTOS. 코어 검증에는 Bluetooth 테스트가 필요하지 않습니다. FreeRTOS Bluetooth를 지원하는 디바이스를 테스트하지 않으려는 경우 이 설정을 건너뛸 수 있으며 device.json의 BLE 기능을 로 설정된 상태로 그대로 두어야 합니다. No.

## 사전 조건

- 처음으로 마이크로 컨트롤러 보드의 테스트 준비 ([p. 328](#))의 지침을 따르세요.
- Raspberry Pi 4B 또는 3B+. (Raspberry Pi BLE 컴파니언 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 4B 이미지
  - Raspberry Pi 3B+ 이미지

### Note

Yocto 이미지는 테스트 용도로만 사용해야 합니다. AWS IoT Device Tester FreeRTOS 용도이며 다른 목적이 아닙니다.

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 key pair 페어가 있는 경우, 이를 허용하기 위해 새 키 페어를 생성하는 것이 가장 좋습니다. AWS IoT Device Tester FreeRTOS가 Raspberry Pi에 로그인할 수 있습니다.

### Note

Windows에는 SSH 클라이언트가 설치되어 있지 않습니다. Windows에 SSH 클라이언트를 설치하는 방법에 대한 자세한 내용은 [SSH 소프트웨어 다운로드](#)를 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 *Enter*를 눌러 계속합니다.

- iv. 라즈베리 파이에 SSH 키를 추가하려면 AWS IoT Device Tester FreeRTOS가 장치에 로그인할 수 있는 경우 `ssh-copy-id` 호스트 컴퓨터에서 명령을 실행합니다. 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.pub`라는 이름이 지정된다고 가정합니다. macOS와 Linux에서 기본 위치는 `~/.ssh/`입니다. Windows에서 기본 위치는 `C:\Users\user-name\.ssh`입니다. 퍼블릭 키의 이름을 다르게 지정하거나 다른 위치에 저장한 경우, `ssh-copy-id`에 `-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-ID](#)를 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 `no`로 설정합니다.
- C. `sshd_config` 파일을 저장하고 닫습니다.
- D. `/etc/init.d/sshd reload`를 실행하여 SSH 서버를 다시 로드합니다.

- g. `resource.json` 파일을 생성합니다.

- i. AWS IoT Device Tester를 추출한 디렉터리에서 `resource.json`이라는 이름의 파일을 생성합니다.
- 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. SSH에 퍼블릭 키 인증을 사용하도록 선택하지 않은 경우 `connectivity`의 단원 `resource.json` 파일.

```
"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": "pkı",
        "credentials": {
            "user": "root",
            "privKeyPath": "location-of-private-key"
        }
    }
}
```

## FreeRTOS 디바이스 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

에서 BLE 기능을 활성화한 후 `device.json` BLE 테스트는 실행할 때 실행됩니다. `devicetester_[linux | mac | win_x86-64]` run-suite 그룹 ID를 지정하지 않고.

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 준비 (p. 328)의 단계를 따랐는지 확인합니다. BLE 이외의 다른 테스트가 실패하면 문제가 Bluetooth 구성으로 인한 것이 아닐 가능성성이 높습니다.

## FreeRTOS 검증 제품군 실행

이 AWS IoT Device Tester용 FreeRTOS 실행 파일을 사용하여 IDT와 상호 작용할 수 있습니다. 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-adt y/n \
--update-managed-policy y/n \
--userdata userdata.json
```

디바이스의 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 파일.

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-id는 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.

### 용 IDT FreeRTOS 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.json에서 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 정보는 [테스트 제품군 버전 \(p. 355\)](#)을 참조하십시오.

#### test-id

(선택 사항) 쉼표로 구분된 목록으로 실행할 테스트입니다. 지정된 경우 group-id은 단일 그룹을 지정해야 합니다.

#### Example

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

#### 업데이트-idt

(선택 사항) 이 매개 변수가 설정되지 않고 최신 IDT 버전을 사용할 수 있는 경우 IDT를 업데이트하라는 메시지가 표시됩니다. 파라미터가 로 설정된 경우y, IDT는 최신 버전을 사용할 수 있음을 감지하면 테스트 실행을 중지합니다. 파라미터가 로 설정된 경우n, IDT는 테스트 실행을 계속합니다.

#### 업데이트 관리 정책

(선택 사항) 이 매개 변수를 사용하지 않고 IDT에서 관리형 정책이 최신 상태가 아님을 감지하면 관리형 정책을 업데이트하라는 메시지가 표시됩니다. 파라미터가 로 설정된 경우y, IDT는 관리형 정책이 최신 상태가 아니라는 것을 감지하면 테스트 실행을 중지합니다. 파라미터가 로 설정된 경우n, 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 n을 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 파일.  
다음 명령을 사용하여 특정 테스트 그룹을 실행합니다.

```
devicetester_[linux | mac | win] run-suite \
--suite-id FRO_1 --group-id group-id \
--pool-id pool-id \
--userdata userdata.json
```

단일 디바이스 폴에 대해 단일 테스트 세트를 실행하는 경우(즉, device.json 파일에 정의된 디바이스 폴이 하나만 있는 경우) suite-id 및 pool-id는 선택 사항입니다.

### 용 IDT FreeRTOS 명령줄 옵션

group-id

(선택 사항) 테스트 그룹을 지정합니다.

pool-id

테스트할 디바이스 폴을 지정합니다. 디바이스 폴이 하나만 있는 경우 이 옵션을 생략할 수 있습니다.

suite-id

(선택 사항) 실행할 테스트 제품군을 지정합니다.

## 용 IDT FreeRTOS 명령

용 IDT 명령은 FreeRTOS 연산을 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**

디바이스의 폴에 대해 테스트 제품군을 실행합니다.

--suite-id 옵션을 사용하여 테스트 제품군 버전을 지정하거나, 시스템에서 최신 버전을 사용하도록 이를 생략합니다.

--test-id를 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 제품군 실행 \(p. 351\)](#) 단원을 참조하십시오.

#### Note

IDT v3.0.0부터 IDT는 최신 테스트 제품군을 온라인으로 확인합니다. 자세한 정보는 [테스트 제품군 버전 \(p. 355\)](#)을 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**

디바이스의 풀에 대해 테스트 제품군을 실행합니다.

## 재검증을 위한 테스트

FreeRTOS 자격 테스트용 IDT의 새로운 버전이 릴리스되거나 보드별 패키지 또는 디바이스 드라이버를 업데이트하면 용 IDT를 사용하여 마이크로 컨트롤러 보드를 테스트할 수 있습니다. 이후 자격에 대해서는 최신 버전의 FreeRTOS 및 FreeRTOS가 있는지 확인하고 자격 테스트를 다시 실행해야 합니다.

## AWS IoT Device TesterFreeRTOS 테스트 제품군 버전

용 IDT는 테스트 리소스를 테스트 제품군 및 테스트 그룹으로 구성합니다.

- 테스트 제품군은 디바이스가 특정 버전의 FreeRTOS에서 작동하는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 BLE 및 MQTT 메시징과 같은 특정 기능과 관련된 개별 테스트 집합입니다.

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 정보는 [용에 대한 Support 정책AWS IoT Device TesterFreeRTOS \(p. 426\)](#)을 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.

또한 를 실행할 수 있습니다.`list-supported-versions`명령을 사용하여 현재 버전의 IDT에서 지원하는 FreeRTOS 및 테스트 제품군 버전을 나열합니다.

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

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.xml`에 제출하는 자격 테스트 보고서입니다.AWS을 (를) 에 기기를 나열하려면AWS파트너 장치 카탈로그. 보고서에는 다음 요소가 포함됩니다.

- 용 IDT FreeRTOS 버전.
- 테스트한 FreeRTOS 버전
- 통과를 기반으로 디바이스에서 지원하는 FreeRTOS의 기능입니다.
- `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 자격이 LTS 라이브러리를 사용하는 202012.00 버전인지 여부입니다.

`FRO_Report.xml`은 표준 [JUnit XML 형식](#)의 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

### 용 IDT FreeRTOS 결과 해석

`awsiotdevicetester_report.xml` 또는 `FRO_Report.xml`의 보고서 섹션에는 실행된 테스트 결과가 나열됩니다.

첫 번째 XML 태그 `<testsuites>`에는 테스트 실행의 전체 요약이 포함됩니다. 예제:

```
<testsuites name="FRO 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**

용 IDT가 FreeRTOS를 실행할 수 없는 테스트 사례의 수입니다.

**disabled**

이 속성은 사용되지 않으므로 무시해도 좋습니다.

테스트 사례 실패 또는 오류가 없는 경우 디바이스는 FreeRTOS 를 실행하기 위한 기술 요구 사항을 충족하며 AWS IoT 서비스, 장치에 장치를 나열하도록 선택한 경우 AWS Partner Device Catalog에서 이 보고서를 자격 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. 예제:

```
<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**

SDK로 IDT를 실행한 경우 이 블록에는 SDK의 이름과 버전이 포함됩니다. SDK로 IDT를 실행하지 않은 경우 이 블록에는 다음이 포함됩니다.

```
<sdk>
  <name>N/A</name>
  <version>N/A</version>
</sdk>
```

**features**

확인된 기능입니다. `required`로 표시된 기능은 자격에 대한 보드를 제출하는 데 필요합니다. 다음 코드 조각은 `awsiotdevicetester_report.xml` 파일입니다.

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

optional로 표시된 기능은 자격에 필수 기능이 아닙니다. 다음 코드 조각은 선택적 기능을 보여 줍니다.

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

필수 기능에 대한 테스트 실패 또는 오류가 없는 경우 디바이스는 FreeRTOS를 실행하기 위한 기술 요구 사항을 충족하며 AWS IoT 서비스. 장치에 장치를 나열하려는 경우 [AWS파트너 디바이스 카탈로그](#)에서 이 보고서를 자격 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. 예제:

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

형식은 다음과 비슷합니다. <testsuites> 태그, 하지만 skipped 사용되지 않으므로 무시해도 좋습니다. 각 <testsuite> XML 태그 내부에는 테스트 그룹에 실행된 각 테스트에 대한 <testcase> 태그가 있습니다. 예제:

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

#### lts

LTS 라이브러리를 사용하는 FreeRTOS 버전을 사용할 수 있는 경우 true이고 그렇지 않으면 false입니다.

#### <testcase> 태그에 사용되는 속성

##### name

테스트 사례의 이름입니다.

##### attempts

FreeRTOS에 대한 IDT가 테스트 사례를 실행한 횟수입니다.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예제:

```
<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 정보는 [문제 해결 \(p. 419\)](#)을 참조하십시오.

#### 로그 보기

용 IDT가 테스트 실행에서 생성하는 로그는 에서 확인할 수 있습니다. [devicetester-extract-location/results/execution-id/logs](#). 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 스위트를 개발하고 실행하십시오.

IDT v4.0.0부터 FreeRTOS용 IDT는 표준화된 구성 설정 및 결과 형식을 테스트 세트 환경과 결합하여 장치 및 장치 소프트웨어에 대한 사용자 지정 테스트 스위트를 개발할 수 있습니다. 자체 내부 검증을 위한 사용자 지정 테스트를 추가하거나 기기 확인을 위해 고객에게 제공할 수 있습니다.

IDT를 사용하여 다음과 같이 사용자 지정 테스트 스위트를 개발하고 실행합니다.

사용자 지정 테스트 스위트를 개발하려면

- 테스트하려는 디바이스에 대한 사용자 지정 테스트 로직을 사용하여 테스트 스위트를 만듭니다.
- 테스트 러너를 위한 맞춤형 테스트 스위트를 IDT에 제공합니다. 테스트 스위트의 특정 설정 구성에 대한 정보를 포함합니다.

사용자 지정 테스트 스위트를 실행하려면

- 테스트하려는 디바이스를 설정합니다.
- 사용하려는 테스트 스위트에서 필요에 따라 설정 구성을 구현합니다.
- IDT를 사용하여 사용자 지정 테스트 스위트를 실행합니다.
- IDT가 실행하는 테스트에 대한 테스트 결과 및 실행 로그를 봅니다.

## AWS IoT용 Device Tester FreeRTOS의 최신 버전 다운로드

를 다운로드합니다.[최신 버전 \(p. 315\)](#) IDT의 압축을 풀고 읽기 및 쓰기 권한이 있는 파일 시스템의 위치에 소프트웨어의 압축을 끕니다.

### 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 사용 중인 경우 경로를 260자 제한 아래로 유지하도록 IDT 압축을 C:\ 또는 D:\ 같은 루트 디렉터리에 끕니다.

## 테스트 제품군 생성 워크플로우

테스트 제품군은 세 유형의 파일로 구성됩니다.

- IDT에 테스트 스위트를 실행하는 방법에 대한 정보를 제공하는 구성 파일입니다.
- IDT가 테스트 케이스를 실행하는 데 사용하는 실행 파일을 테스트합니다.
- 테스트를 실행하는 데 필요한 추가 파일입니다.

사용자 지정 IDT 테스트를 생성하려면 다음 기본 단계를 완료하십시오.

1. [구성 파일 생성 \(p. 370\)](#) 테스트 스위트에 적합합니다.

2. 테스트 케이스 실행 파일 만들기 (p. 395) 테스트 스위트에 대한 테스트 로직을 포함합니다.
3. 확인 및 문서화 테스트 러너에 필요한 구성 정보 (p. 402) 테스트 제품군을 실행합니다.
4. IDT가 테스트 스위트를 실행하고 생성할 수 있는지 확인합니다. 테스트 결과 (p. 409) 예상대로.

샘플 사용자 지정 제품군을 빠르게 빌드하고 실행하려면 의 지침을 따르십시오. 자습서: 샘플 IDT 테스트 제품군 빌드 및 실행 (p. 360).

Python에서 사용자 정의 테스트 스위트를 만들려면 다음을 참조하십시오. 자습서: 간단한 IDT 테스트 제품군 개발 (p. 363).

## 자습서: 샘플 IDT 테스트 제품군 빌드 및 실행

이 AWS IoT Device Tester 다운로드에는 샘플 테스트 스위트의 소스 코드가 포함되어 있습니다. 이 자습서를 완료하여 샘플 테스트 스위트를 빌드하고 실행하여 사용 방법을 이해할 수 있습니다. AWS IoT Device Tester FreeRTOS를 사용하여 사용자 지정 테스트 스위트를 실행합니다. 이 자습서에서는 SSH를 사용하지만 사용 방법을 배우는 것이 유용합니다. AWS IoT Device Tester FreeRTOS 디바이스 사용.

이 자습서에서는 다음 단계를 완료합니다.

1. 샘플 테스트 스위트 구축 (p. 362)
2. IDT를 사용하여 샘플 테스트 세트 실행 (p. 363)

## 사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - AWS IoT Device Tester의 최신 버전
  - 파이썬 3.7 이상

컴퓨터에 설치된 Python의 버전을 확인하려면 다음 명령을 실행합니다.

```
python3 --version
```

Windows에서 이 명령을 사용하면 오류가 반환되는 경우 `python --version` 그 대신 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python3`에 대한 별칭으로 `python` 명령.

```
Set-Alias -Name "python3" -Value "python"
```

버전 정보가 반환되지 않거나 버전 번호가 3.7보다 작은 경우 의 지침을 따르십시오. Python 다운로드 Python 3.7+를 설치하려면. 자세한 내용은 단원을 참조하십시오. Python 문서.

- `urllib3`

다음을 확인하려면 `urllib3`이 제대로 설치되면 다음 명령을 실행합니다.

```
python3 -c 'import urllib3'
```

다음의 경우, `urllib3`이 설치되지 않은 경우에는 다음 명령을 실행하여 설치합니다.

```
python3 -m pip install urllib3
```

- 디바이스 요구 사항

- Linux 운영 체제가 있고 호스트 컴퓨터와 동일한 네트워크에 네트워크 연결이 있는 장치입니다.

를 사용할 것을 권장합니다.[Raspberry Pi](#)Raspberry Pi OS. 설정하도록 합니다.[SSH](#)라즈베리 파이에서 원격으로 연결합니다.

## IDT에 대한 장치 정보 구성

테스트를 실행하도록 IDT에 대한 장치 정보를 구성합니다. 을 업데이트해야 합니다.`device.json`에 있는 템플릿`<device-tester-extract-location>/configs` 다음 정보가 포함된 폴더입니다.

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

에서`devices` 객체, 다음 정보를 입력합니다.

### **id**

디바이스의 고유한 사용자 정의 식별자입니다.

### **connectivity.ip**

디바이스의 IP입니다.

### **connectivity.port**

선택 사항. 장치에 대한 SSH 연결에 사용할 포트 번호입니다.

### **connectivity.auth**

연결에 대한 인증 정보입니다.

이 속성은`connectivity.protocol`이 ssh로 설정된 경우에만 적용됩니다.

### **connectivity.auth.method**

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki

- password

#### **connectivity.auth.credentials**

인증에 사용되는 자격 증명입니다.

#### **connectivity.auth.credentials.user**

디바이스에 로그인하는 데 사용되는 사용자 이름입니다.

#### **connectivity.auth.credentials.privKeyPath**

디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 connectivity.auth.method가 pki로 설정된 경우에만 적용됩니다.

#### **devices.connectivity.auth.credentials.password**

디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 connectivity.auth.method가 password로 설정된 경우에만 적용됩니다.

#### Note

method가 pki로 설정된 경우에만 privKeyPath를 지정합니다.

method가 password로 설정된 경우에만 password를 지정합니다.

## 샘플 테스트 스위트 구축

이 `<device-tester-extract-location>/samples/python` 폴더에는 제공된 빌드 스크립트를 사용하여 테스트 스위트로 결합할 수 있는 샘플 구성 파일, 소스 코드 및 IDT Client SDK가 들어 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 스위트를 빌드하려면 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
```

이렇게 하면 샘플 테스트 스위트가 `IDTSampleSuitePython_1.0.0`의 폴더 `<device-tester-extract-location>/testsfolder`에 있는 파일 검토 `IDTSampleSuitePython_1.0.0` 폴더 - 샘플 테스트 스위트가 어떻게 구성되는지 이해하고 테스트 케이스 실행 파일 및 테스트 구성 파일의 다양한 예제를 볼 수 있습니다.

다음 단계: IDT를 사용하여 [샘플 테스트 스위트 실행 \(p. 363\)](#) 당신이 만든 것입니다.

## IDT를 사용하여 샘플 테스트 세트 실행

샘플 테스트 스위트를 실행하려면 호스트 컴퓨터에서 다음 명령을 실행합니다.

```
cd <device-tester-extract-location>/bin  
. /devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT는 샘플 테스트 스위트를 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 자습서를 완료할 때 발생하는 모든 문제를 해결하는 데 도움이 됩니다.

테스트 케이스가 성공적으로 실행되지 않음

테스트가 성공적으로 실행되지 않으면 IDT는 테스트 실행 문제를 해결하는 데 도움이 되는 오류 로그를 콘솔로 스트리밍합니다. 모든 것을 충족하는지 확인합니다. [전체 조건 \(p. 360\)](#)이 자습서에서는.

테스트 대상 디바이스에서 연결할 수 없습니다.

다음을 확인합니다.

- 귀하 `device.json` 파일에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.
- 호스트 컴퓨터에서 SSH를 통해 디바이스에 연결할 수 있습니다.

## 자습서: 간단한 IDT 테스트 제품군 개발

테스트 스위트는 다음을 결합합니다.

- 테스트 로직을 포함하는 테스트 실행 파일
- 테스트 스위트를 설명하는 구성 파일

이 자습서에서는 FreeRTOS용 IDT를 사용하여 단일 테스트 케이스를 포함하는 Python 테스트 스위트를 개발하는 방법을 보여줍니다. 이 자습서에서는 SSH를 사용하지만 사용 방법을 배우는 것이 유용합니다. AWS IoT Device TesterFreeRTOS 디바이스를 사용합니다.

이 자습서에서는 다음 단계를 완료합니다.

1. 테스트 스위트 디렉터리 만들기 (p. 364)
2. 구성 파일 생성 (p. 365)
3. 테스트 케이스 실행 파일 만들기 (p. 366)
4. 테스트 제품군 실행 (p. 369)

## 사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항

- AWS IoT Device Tester의 최신 버전
- 파이썬3.7 이상

컴퓨터에 설치된 Python의 버전을 확인하려면 다음 명령을 실행합니다.

```
python3 --version
```

Windows에서 이 명령을 사용하면 오류가 반환되는 경우python --version그 대신 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여python3에 대한 별칭으로python명령.

```
Set-Alias -Name "python3" -Value "python"
```

버전 정보가 반환되지 않거나 버전 번호가 3.7보다 작은 경우 의 지침을 따르십시오.[Python 다운로드](#)Python 3.7+를 설치하려면. 자세한 내용은 단원을 참조하십시오.[Python 문서](#).

- [urllib3](#)

다음을 확인하려면[urllib3](#)이 제대로 설치되면 다음 명령을 실행합니다.

```
python3 -c 'import urllib3'
```

다음의 경우,[urllib3](#)이 설치되지 않은 경우에는 다음 명령을 실행하여 설치합니다.

```
python3 -m pip install urllib3
```

- 디바이스 요구 사항

- Linux 운영 체제가 있고 호스트 컴퓨터와 동일한 네트워크에 네트워크 연결이 있는 장치입니다.

를 사용할 것을 권장합니다.[Raspberry Pi](#)Raspberry Pi OS. 설정하도록 합니다.[SSH](#)라즈베리 파이에서 원격으로 연결합니다.

## 테스트 스위트 디렉터리 만들기

IDT는 테스트 케이스를 각 테스트 세트 내의 테스트 그룹으로 논리적으로 분리합니다. 각 테스트 사례는 테스트 그룹 내에 있어야 합니다. 이 자습서에서는 라는 폴더를 만듭니다.[MyTestSuite\\_1.0.0](#)이 폴더 내에 다음 디렉터리 트리를 만듭니다.

```
MyTestSuite_1.0.0
### suite
### myTestGroup
### myTestCase
```

## 구성 파일 생성

테스트 스위트에는 다음 필수 항목이 포함되어야 합니다.[구성 파일 \(p. 370\)](#):

필수 파일

**suite.json**

테스트 제품군에 대한 정보가 포함됩니다. [Suite.json 구성 \(p. 370\)](#) 섹션을 참조하세요.

**group.json**

테스트 그룹에 대한 정보가 포함됩니다. 을 생성해야 합니다. [group.json 테스트 스위트 \(p. 370\)](#) 섹션을 참조하세요.

**test.json**

테스트 사례에 대한 정보가 포함됩니다. 을 생성해야 합니다. [test.json 테스트 스위트 \(p. 372\)](#) 섹션을 참조하세요.

1. 에서 `MyTestSuite_1.0.0/suite` 폴더, 만들기 `suite.json` 다음 구조의 파일을 생성합니다.

```
{
  "id": "MyTestSuite_1.0.0",
  "title": "My Test Suite",
  "details": "This is my test suite.",
  "userDataRequired": false
}
```

2. 에서 `MyTestSuite_1.0.0/myTestGroup` 폴더, 만들기 `group.json` 다음 구조의 파일을 생성합니다.

```
{
  "id": "MyTestGroup",
  "title": "My Test Group",
  "details": "This is my test group.",
  "optional": false
}
```

3. 에서 `MyTestSuite_1.0.0/myTestGroup/myTestCase` 폴더, 만들기 `test.json` 다음 구조의 파일을 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"
      ]
    }
  }
}
```

```
        "myTestCase.py"
    ],
},
"win": {
    "cmd": "python3",
    "args": [
        "myTestCase.py"
    ]
}
}
```

귀하의 디렉터리 트리 MyTestSuite\_1.0.0 폴더는 이제 다음과 같이 보여야 합니다.

```
MyTestSuite_1.0.0
### suite
### suite.json
### myTestGroup
### group.json
### myTestCase
### test.json
```

## IDT 클라이언트 SDK 받기

이 [IDT 클라이언트 SDK \(p. 395\)](#) IDT가 테스트 중인 장치와 상호 작용하고 테스트 결과를 보고할 수 있도록 합니다. 이 자습서에서는 Python 버전의 SDK를 사용합니다.

에서 <[device-tester-extract-location](#)>/sdks/python/폴더, 복사 idt\_client 폴더를 사용자 이름으로 MyTestSuite\_1.0.0/suite/myTestGroup/myTestCase 폴더로 이동합니다.

SDK가 성공적으로 복사되었는지 확인하려면 다음 명령을 실행합니다.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

## 테스트 케이스 실행 파일 만들기

테스트 사례 실행 파일에는 실행하려는 테스트 논리가 포함되어 있습니다. 테스트 스위트에는 여러 테스트 케이스 실행 파일이 포함될 수 있습니다. 이 자습서에서는 테스트 사례 실행 파일을 하나만 만듭니다.

1. 테스트 스위트 파일을 만듭니다.

에서 MyTestSuite\_1.0.0/suite/myTestGroup/myTestCase 폴더, 만들기 testCase.py 다음 콘텐츠가 포함된 파일을 생성합니다.

```
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 다음 테스트 로직을 testCase.py 파일:

- a. 테스트 대상 디바이스에서 SSH 명령을 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 보냅니다.

```
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()
```

## IDT에 대한 장치 정보 구성

테스트를 실행하도록 IDT에 대한 장치 정보를 구성합니다. 을 업데이트해야 합니다.device.json에 있는 템플릿<*device-tester-extract-location*>/configs 다음 정보가 포함된 폴더입니다.

```
[  
  {  
    "id": "pool",  
    "sku": "N/A",  
    "devices": [  
      {  
        "id": "<device-id>",  
        "connectivity": {  
          "protocol": "ssh",  
          "ip": "<ip-address>",  
          "port": "<port>",  
          "auth": {  
            "method": "pki | password",  
            "credentials": {  
              "username": "root",  
              "password": "password"  
            }  
          }  
        }  
      }  
    ]  
  }  
]
```

```
        "user": "<user-name>",
        "privKeyPath": "/path/to/private/key",
        "password": "<password>"
    }
}
]
]
```

에서 **devices** 객체, 다음 정보를 입력합니다.

**id**

디바이스의 고유한 사용자 정의 식별자입니다.

**connectivity.ip**

디바이스의 IP입니다.

**connectivity.port**

선택 사항. 장치에 대한 SSH 연결에 사용할 포트 번호입니다.

**connectivity.auth**

연결에 대한 인증 정보입니다.

이 속성은 **connectivity.protocol**이 ssh로 설정된 경우에만 적용됩니다.

**connectivity.auth.method**

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki
- password

**connectivity.auth.credentials**

인증에 사용되는 자격 증명입니다.

**connectivity.auth.credentials.user**

디바이스에 로그인하는 데 사용되는 사용자 이름입니다.

**connectivity.auth.credentials.privKeyPath**

디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 **connectivity.auth.method**가 pki로 설정된 경우에만 적용됩니다.

**devices.connectivity.auth.credentials.password**

디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 **connectivity.auth.method**가 password로 설정된 경우에만 적용됩니다.

**Note**

**method**가 pki로 설정된 경우에만 **privKeyPath**를 지정합니다.

**method**가 password로 설정된 경우에만 **password**를 지정합니다.

## 테스트 제품군 실행

테스트 스위트를 생성한 후에는 테스트가 예상대로 작동하는지 확인해야 합니다. 다음 단계를 완료하면 기존 디바이스 폴과 함께 테스트 제품군의 실행을 수행하십시오.

1. 복사하기|MyTestSuite\_1.0.0 폴더에 폴더를 <*device-tester-extract-location*>/tests.
2. 다음 명령을 실행합니다:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT는 테스트 스위트를 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 자습서를 완료할 때 발생하는 모든 문제를 해결하는데 도움이 됩니다.

### 테스트 케이스가 성공적으로 실행되지 않음

테스트가 성공적으로 실행되지 않으면 IDT는 테스트 실행 문제를 해결하는데 도움이 되는 오류 로그를 콘솔로 스트리밍합니다. 오류 로그를 확인하기 전에 다음을 확인하십시오.

- IDT 클라이언트 SDK가 예 설명된 대로 올바른 폴더에 있습니다.[이 단계 \(p. 366\)](#).
- 당신은 모든 것을 만난다. [전체 조건 \(p. 364\)](#)이 자습서에서는.

테스트 대상 디바이스에서 연결할 수 없습니다.

다음을 확인합니다.

- 귀하 device.json file에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.

- 호스트 컴퓨터에서 SSH를 통해 디바이스에 연결할 수 있습니다.

## IDT 테스트 세트 구성 파일 만들기

이 섹션에서는 사용자 지정 테스트 스위트를 작성할 때 포함하는 구성 파일을 만드는 형식에 대해 설명합니다.

### 필수 구성 파일

#### **suite.json**

테스트 제품군에 대한 정보가 포함됩니다. [Suite.json 구성 \(p. 370\)](#) 섹션을 참조하세요.

#### **group.json**

테스트 그룹에 대한 정보가 포함됩니다. 을 생성해야 합니다. [group.json 테스트 스위트의 각 테스트 그룹에 대한 파일 .json 그룹화 \(p. 371\)](#) 섹션을 참조하세요.

#### **test.json**

테스트 사례에 대한 정보가 포함됩니다. 을 생성해야 합니다. [test.json 테스트 스위트의 각 테스트 케이스에 대한 파일 test.json 구성 \(p. 372\)](#) 섹션을 참조하세요.

### 구성 파일 옵션

#### **test\_orchestrator.yaml 또는 state\_machine.json**

IDT가 테스트 스위트를 실행할 때 테스트가 실행되는 방법을 정의합니다. SSE[테스트\\_오케스트라.yaml 구성 \(p. 374\)](#).

#### Note

IDT v4.5.2부터는 `test_orchestrator.yaml` 파일을 사용하여 테스트 워크플로를 정의합니다. IDT의 이전 버전에서는 `state_machine.json` 파일. 상태 머신에 대한 자세한 내용은 단원 [을 참조하십시오. IDT 상태 머신 구성 \(p. 379\)](#).

#### **userdata\_schema.json**

에 대한 스키마를 정의합니다. [userdata.json 파일 \(p. 405\)](#) 테스트 러너는 설정 구성에 포함할 수 있습니다. 이 `userdata.json` 파일은 테스트를 실행하는 데 필요하지만 `device.json` 파일. [사용자 데이터 스키마.json 구성 \(p. 374\)](#) 섹션을 참조하세요.

구성 파일은 다음 위치에 배치됩니다. `<custom-test-suite-folder>` 다음과 같습니다.

```
<custom-test-suite-folder>
### suite
### suite.json
### test_orchestrator.yaml
### userdata_schema.json
### <test-group-folder>
### group.json
### <test-case-folder>
### test.json
```

## Suite.json 구성을 구성

이 `suite.json` 파일은 환경 변수를 설정하고 테스트 스위트를 실행하는 데 사용자 데이터가 필요한지 여부를 결정합니다. 다음 템플릿을 사용하여 구성 요소를 구성합니다. `<custom-test-suite-folder>/suite/suite.json` 파일:

```
{  
    "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 스위트에 대한 고유한 사용자 정의 ID입니다. 의 가치id는 테스트 제품군의 이름과 일치해야 합니다.suite.json파일이 위치합니다. 제품군의 이름 및 제품군의 버전도 다음 요구 사항을 충족해야 합니다.

- <suite-name> 밑줄을 포함할 수 없습니다.
- <suite-version>는 다음과 같이 표시됩니다.x.x.x 여기서x는 숫자입니다.

IDT 생성 테스트 보고서에 ID가 표시됩니다.

#### **title**

이 테스트 스위트에서 테스트 중인 제품 또는 기능에 대한 사용자 정의 이름입니다. 테스트 러너의 IDT CLI에 이름이 표시됩니다.

#### **details**

테스트 제품군의 용도에 대한 간략한 설명입니다.

#### **userDataRequired**

테스트 러너가 에 사용자 지정 정보를 포함해야 하는지 여부를 정의합니다\_userdata.json파일. 이 값을로 설정한 경우true를 포함해야 합니다.[userdata\\_schema.json파일 \(p. 374\)](#) 테스트 스위트 폴더에 있습니다.

#### **environmentVariables**

선택 사항. 이 테스트 제품군의 환경 변수입니다.

##### **environmentVariables.key**

환경 변수의 이름입니다.

##### **environmentVariables.value**

환경 변수의 값입니다.

## **.json 그룹화**

0group.jsonfile은 테스트 그룹의 필수 또는 선택 사항인지 여부를 정의합니다. 다음 템플릿을 사용하여 구성 요소를 구성합니다.<custom-test-suite-folder>/suite/<test-group>/group.json파일:

```
{
```

```
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

#### id

테스트 그룹에 대한 고유한 사용자 정의 ID입니다. 이 가치 id는 테스트 그룹 폴더의 이름과 일치해야 합니다. group.json 파일이 있으며 밑줄이 없어야 합니다 (\_). ID는 IDT 생성 테스트 보고서에 사용됩니다.

#### title

테스트 그룹에 대한 설명이 포함된 이름입니다. 테스트 러너의 IDT CLI에 이름이 표시됩니다.

#### details

테스트 그룹의 용도에 대한 간략한 설명입니다.

#### optional

선택 사항으로 설정 true로 IDT가 필수 테스트 실행을 마친 후 이 테스트 그룹을 선택적 그룹으로 표시합니다. 기본값은 false입니다.

## test.json 구성

0|test.json은 테스트 케이스 실행 파일과 테스트 케이스에서 사용하는 환경 변수를 결정합니다. 테스트 사례 실행 파일 생성에 대한 자세한 내용은 단원을 참조하십시오. [IDT 테스트 케이스 실행 파일 만들기 \(p. 395\)](#).

다음 템플릿을 사용하여 구성 요소를 구성합니다. <custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json 파일:

```
{
    "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>"
                ],
                "windows": {
                    "cmd": "/path/to/executable",
                    "args": [
                        "<argument>"
                    ]
                }
            }
        }
    }
}
```

```
        "args": [
            "<argument>"
        ],
    },
    "win": {
        "cmd": "/path/to/executable",
        "args": [
            "<argument>"
        ]
    }
},
"environmentVariables": [
{
    "key": "<name>",
    "value": "<value>",
}
]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

#### **id**

테스트 케이스에 대한 고유한 사용자 정의 ID입니다. 의 가치id는 테스트 사례 폴더의 이름과 일치해야 합니다.test.json파일이 있으며 밑줄이 없어야 합니다 (\_). ID는 IDT 생성 테스트 보고서에 사용됩니다.

#### **title**

테스트 사례에 대한 설명이 포함된 이름입니다. 테스트 러너의 IDT CLI에 이름이 표시됩니다.

#### **details**

테스트 사례의 용도에 대한 간략한 설명입니다.

#### **requireDUT**

선택 사항. 으로 설정true이 테스트를 실행하는 데 장치가 필요한 경우, 그렇지 않으면 로 설정합니다.false. 기본값은 true입니다. 테스트 러너는 테스트를 실행하는 데 사용할 장치를 구성합니다.device.json파일.

#### **requiredResources**

선택 사항. 이 테스트를 실행하는 데 필요한 리소스 디바이스에 대한 정보를 제공하는 배열입니다.

##### **requiredResources.name**

이 테스트가 실행 중일 때 리소스 디바이스를 제공하는 고유 이름입니다.

##### **requiredResources.features**

사용자 정의 리소스 디바이스 기능의 배열입니다.

##### **requiredResources.features.name**

기능 이름입니다. 이 장치를 사용할 장치 기능입니다. 이 이름은 테스트 러너에서 제공한 기능 이름과 일치합니다.resource.json파일.

##### **requiredResources.features.version**

선택 사항. 기능 버전입니다. 이 값은 테스트 러너가 제공하는 기능 버전과 일치합니다.resource.json파일. 버전이 제공되지 않으면 기능이 선택되지 않습니다. 기능에 버전 번호가 필요하지 않은 경우 이 필드를 비워 둡니다.

##### **requiredResources.features.jobSlots**

선택 사항. 이 기능이 지원할 수 있는 동시 테스트의 수입니다. 기본값은 1입니다. IDT가 개별 기능에 대해 별개의 장치를 사용하도록 하려면 이 값을 1로 설정하는 것이 좋습니다.

#### **execution.timeout**

IDT가 테스트 실행을 완료할 때까지 기다리는 시간 (밀리초)입니다. 이 값 설정에 대한 자세한 내용은 단원을 참조하십시오.[IDT 테스트 케이스 실행 파일 만들기 \(p. 395\)](#).

#### **execution.os**

IDT를 실행하는 호스트 컴퓨터의 운영 체제를 기반으로 실행할 테스트 케이스 실행 파일입니다. 지원되는 값은 linux, mac 및 win입니다.

#### **execution.os.cmd**

지정된 운영 체제에 대해 실행할 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 사례에 대해 설정된 환경 변수입니다.

#### **environmentVariables.key**

환경 변수의 이름입니다.

#### **environmentVariables.value**

환경 변수의 값입니다.

#### **Note**

에서 동일한 환경 변수를 지정하는 경우 test.json 파일 및 suite.json 파일, 의 값 test.json 파일이 우선합니다.

## **테스트\_오케스트라.yaml 구성**

테스트 오케스트레이터는 테스트 세트 실행 흐름을 제어하는 구조입니다. 테스트 스위트의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 종료 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 세트에 사용자 정의 테스트 오케스트레이터가 포함되어 있지 않은 경우 IDT는 테스트 오케스트레이터를 생성합니다.

기본 테스트 오케스트레이터는 다음 기능을 수행합니다.

- 테스트 러너에게 전체 테스트 세트 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않으면 에서는 테스트 스위트의 모든 테스트 그룹을 임의의 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례에 대한 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 테스트 오케스트레이터의 기능에 대한 자세한 내용은 단원을 참조하십시오.[IDT 테스트 오케스트레이터 구성 \(p. 375\)](#).

## **사용자 데이터\_스키마.json 구성**

userdata\_schema.json 은 테스트 러너가 사용자 데이터를 제공하는 스키마를 결정합니다. 테스트 세트에 없는 정보가 필요한 경우 사용자 데이터가 필요합니다.device.json 파일. 예를 들어 테스트에 Wi-Fi 네트워크 자격 증명, 특정 열린 포트 또는 사용자가 제공해야 하는 인증서가 필요할 수 있습니다. 이 정보는

IDT에 라는 입력 매개 변수로 제공할 수 있습니다.`userdata`, 값이 a인 경우`userdata.json`파일, 사용자가`<device-tester-extract-location>/configfolder`. 형식은 다음과 같습니다.`userdata.json`파일은 다음을 기반으로 합니다.`userdata_schema.json`테스트 제품군에 포함하는 파일입니다.

테스트 러너가 a를 제공해야 함을 나타내려면`userdata.json`파일:

1. 에서`suite.json`파일, 세트`userDataRequired`에`true`.
2. 당신 안에`<custom-test-suite-folder>`를 생성하려면`userdata_schema.json`파일.
3. 편집`userdata_schema.json`유효한 파일을 만들 수 있습니다.[IETF 초안 v4 JSON 스키마](#).

IDT가 테스트 스위트를 실행하면 자동으로 스키마를 읽고 이를 사용하여`userdata.json`테스트 러너가 제공하는 파일입니다. 유효한 경우`userdata.json`두 가지 모두에서 파일을 사용할 수 있습니다.[IDT 컨텍스트 \(p. 399\)](#) 그리고 [오케스트레이터 컨텍스트 테스트 \(p. 379\)](#).

## IDT 테스트 오케스트레이터 구성

IDT v4.5.2부터 IDT에는 새로운 기능이 포함되어 있습니다. 테스트 오케스트레이터 구성 요소. 테스트 오케스트레이터는 테스트 세트 실행 흐름을 제어하고 IDT가 모든 테스트 실행을 마친 후 테스트 보고서를 생성하는 IDT 구성 요소입니다. 테스트 오케스트레이터는 테스트 선택 및 사용자 정의 규칙에 따라 테스트가 실행되는 순서를 결정합니다.

테스트 세트에 사용자 정의 테스트 오케스트레이터가 포함되어 있지 않은 경우 IDT는 테스트 오케스트레이터를 생성합니다.

기본 테스트 오케스트레이터는 다음 기능을 수행합니다.

- 테스트 러너에게 전체 테스트 세트 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않으면 에서는 테스트 스위트의 모든 테스트 그룹을 임의의 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례에 대한 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

테스트 오케스트레이터가 IDT 상태 머신을 대체합니다. IDT 상태 머신 대신 테스트 제품군의 개발을 위해 테스트 제품군의 개발을 적극 권장합니다. 테스트 오케스트레이터는 다음과 같은 향상된 기능을 제공합니다.

- IDT 상태 머신에서 사용하는 명령형 형식과 비교하여 선언적 형식을 사용합니다. 이를 통해 지정하다 어떤 테스트를 실행하고 싶은지 언제 실행하고 싶습니다..
- 특정 그룹 처리, 보고서 생성, 오류 처리 및 결과 추적 관리 따라서 필요하지 않습니다.를 사용하여 이러한 작업을 수동으로 관리할 수 있습니다.
- 기본적으로 주석을 지원하는 YAML 형식을 사용합니다.
- 요구 80% 동일한 워크플로를 정의하는 테스트 오케스트레이터보다 디스크 공간이 적습니다.
- 사전 테스트 유효성 검사를 추가하여 워크플로 정의에 잘못된 테스트 ID 또는 순환 종속성이 포함되어 있지 않은지 확인합니다.

## 테스트 오케스트레이터 형식

다음 템플릿을 사용하여 고유한 템플릿을 구성할 수 있습니다.`<custom-test-suite-folder>/suite/test_orchestrator.yaml`파일:

```
Aliases:  
  string: context-expression
```

```
ConditionalTests:
```

```
- Condition: context-expression
Tests:
- test-descriptor

Order:
- - group-descriptor
- group-descriptor

Features:
- Name: feature-name
Value: support-description
Condition: context-expression
Tests:
- test-descriptor
OneOfTests:
- test-descriptor
IsRequired: boolean
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

#### Aliases

선택 사항. 컨텍스트 표현식에 매핑되는 사용자 정의 문자열입니다. 별칭을 사용하면 다음과 같은 이름을 생성할 수 있습니다. 테스트 오케스트레이터 구성에서 컨텍스트 표현식을 식별합니다. 이 기능은 여러 위치에서 사용하는 복잡한 컨텍스트 표현식이나 표현식을 생성하는 경우에 특히 유용합니다.

컨텍스트 표현식을 사용하여 다른 IDT 구성의 데이터에 액세스할 수 있는 컨텍스트 쿼리를 저장할 수 있습니다. 자세한 정보는 [컨텍스트에서 데이터 액세스 \(p. 401\)](#)을 참조하십시오.

#### Example 예

```
Aliases:
FizzChosen: "'${$pool.features[?(@.name == 'Fizz')].value[0]}' == 'yes'"
BuzzChosen: "'${$pool.features[?(@.name == 'Buzz')].value[0]}' == 'yes'"
FizzBuzzChosen: "'${$aliases.FizzChosen}' && '${$aliases.BuzzChosen}'"
```

#### ConditionalTests

선택 사항. 조건 목록 및 각 조건이 충족될 때 실행되는 해당 테스트 케이스입니다. 각 조건에는 여러 테스트 사례가 있을 수 있지만, 주어진 테스트 케이스를 하나의 조건에만 할당할 수 있습니다.

기본적으로 IDT는 이 목록의 조건에 할당되지 않은 모든 테스트 케이스를 실행합니다. 이 섹션을 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.

의 각 항목 ConditionalTests list에는 다음 파라미터가 포함되어 있습니다.

##### Condition

로 평가되는 컨텍스트 표현식입니다. 부울 USD 상당. 평가된 값이 true이면 IDT는 에 지정된 테스트 케이스를 실행합니다. Tests 파라미터.

##### Tests

테스트 설명자 목록입니다.

각 테스트 설명자는 테스트 그룹 ID와 하나 이상의 테스트 사례 ID를 사용하여 특정 테스트 그룹에서 실행할 개별 테스트를 식별합니다. 테스트 설명자는 다음 형식을 사용합니다.

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

### Example 예

다음 예제에서는 다음과 같이 정의할 수 있는 제네릭 컨텍스트 표현식을 사용합니다.`Aliases`.

```
ConditionalTests:  
  - Condition: "{$aliases.Condition1}"  
    Tests:  
      - GroupId: A  
      - GroupId: B  
  - Condition: "{$aliases.Condition2}"  
    Tests:  
      - GroupId: D  
  - Condition: "{$aliases.Condition1} || {$aliases.Condition2}"  
    Tests:  
      - GroupId: C
```

정의된 조건에 따라 IDT는 다음과 같이 테스트 그룹을 선택합니다.

- 다음의 경우, `Condition1` true, IDT는 테스트 그룹 A, B 및 C에서 테스트를 실행합니다.
- 다음의 경우, `Condition2` true, IDT는 테스트 그룹 C와 D에서 테스트를 실행합니다.

### Order

선택 사항. 테스트를 실행할 순서입니다. 테스트 그룹 수준에서 테스트 순서를 지정합니다. 이 섹션을 지정하지 않으면 IDT는 모든 적용 가능한 테스트 그룹을 임의의 순서로 실행합니다. 의 가치 `order`은 (는) 그룹 설명자 목록 목록입니다. 리스트팅하지 않은 모든 테스트 그룹 `order`는 나열된 다른 테스트 그룹과 병렬로 실행할 수 있습니다.

각 그룹 설명자 목록에는 그룹 설명자 중 하나가 더 포함되어 있으며 각 설명자에 지정된 그룹을 실행하는 순서를 식별합니다. 다음 형식을 사용하여 개별 그룹 설명자를 정의할 수 있습니다.

- `group-id`—기존 테스트 그룹의 그룹 ID입니다.
- [`group-id`, `group-id`] 서로에 대해 임의의 순서로 실행할 수 있는 테스트 그룹 목록입니다.
- "\*"—와일드카드. 이는 현재 그룹 설명자 목록에 아직 지정되지 않은 모든 테스트 그룹 목록과 동일합니다.

에 대한 가치 `order` 다음 요구 사항도 충족해야 합니다.

- 그룹 설명자에서 지정한 테스트 그룹 ID는 테스트 스위트에 있어야 합니다.
- 각 그룹 설명자 목록은 하나 이상의 테스트 그룹을 포함해야 합니다.
- 각 그룹 설명자 목록에는 고유한 그룹 ID가 포함되어야 합니다. 개별 그룹 설명자 내에서는 테스트 그룹 ID를 반복할 수 없습니다.
- 그룹 설명자 목록은 하나 이상의 와일드카드 그룹 설명자를 가질 수 있습니다. 와일드카드 그룹 설명자는 목록의 첫 번째 항목이나 마지막 항목이어야 합니다.

### Example 예제

테스트 그룹 A, B, C, D 및 E를 포함하는 테스트 스위트의 경우 다음 예제 목록은 IDT가 먼저 테스트 그룹 A를 실행한 다음 테스트 그룹 B를 실행한 다음 테스트 그룹 C, D 및 E를 임의의 순서로 실행하도록 지정하는 다양한 방법을 보여줍니다.

- `Order:`
  - A
  - B
  - [C, D, E]
- `Order:`
  - - A
  - B

- "\*"
- Order:
  - - A
  - B
  - - B
  - C
  - - B
  - D
  - - B
  - E

### Features

선택 사항. IDT에 추가하려는 제품 기능 목록 `awsiotdevicetester_report.xml` 파일. 이 섹션을 지정하지 않으면 IDT는 보고서에 제품 기능을 추가하지 않습니다.

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. `awsiotdevicetester_report.xml` 제품 기능은 다음과 같이 설정됩니다. `supported`, `not-supported` 또는 지정된 테스트가 통과되었는지 여부에 따라 사용자 정의 값입니다.

의 각 항목 `Featureslist`는 다음 파라미터로 구성됩니다.

#### Name

기능 이름입니다.

#### Value

선택 사항. 보고서에 사용할 사용자 지정 값 대신 `supported`. 이 값을 지정하지 않으면 기본 IDT는 피쳐 값을 로 설정합니다. `supported` 또는 `not-supported` 테스트 결과를 기반으로 합니다. 동일한 피처를 서로 다른 조건으로 테스트하는 경우, 에서 해당 피처의 각 인스턴스에 대해 사용자 정의 값을 사용할 수 있습니다. `Featureslist` 및 IDT는 지원되는 조건에 대한 기능 값을 연결합니다. 자세한 내용은 단원을 참조하십시오.

#### Condition

로 평가되는 컨텍스트 표현식입니다. 부울 USD 상당. 평가된 값이 `true`인 경우 IDT는 테스트 세트 실행을 마친 후 테스트 보고서에 기능을 추가합니다. 평가된 값이 `false`이면 테스트가 보고서에 포함되지 않습니다.

#### Tests

선택 사항. 테스트 설명자 목록입니다. 기능을 지원하려면 이 목록에 지정된 모든 테스트를 통과해야 합니다.

이 목록의 각 테스트 설명자는 테스트 그룹 ID와 하나 이상의 테스트 사례 ID를 사용하여 특정 테스트 그룹에서 실행할 개별 테스트를 식별합니다. 테스트 설명자는 다음 형식을 사용합니다.

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

를 지정해야 합니다. `Tests` 또는 `OneOfTests`의 각 기능에 대해 `Features` 나열.

#### OneOfTests

선택 사항. 테스트 설명자 목록입니다. 기능이 지원되려면 이 목록에 지정된 테스트 중 하나 이상이 통과해야 합니다.

이 목록의 각 테스트 설명자는 테스트 그룹 ID와 하나 이상의 테스트 사례 ID를 사용하여 특정 테스트 그룹에서 실행할 개별 테스트를 식별합니다. 테스트 설명자는 다음 형식을 사용합니다.

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

를 지정해야 합니다.`Tests` 또는 `OneOfTests`의 각 기능에 대해 `Features`나 열.

#### IsRequired

테스트 보고서에 기능이 필요한지 여부를 정의하는 부울 값입니다. 기본값은 `false`입니다.

## okaneスト레이터 컨텍스트 테스트

테스트 오케스트레이터 컨텍스트는 실행 중에 테스트 오케스트레이터가 사용할 수 있는 데이터를 포함하는 읽기 전용 JSON 문서입니다. 테스트 오케스트레이터 컨텍스트는 테스트 오케스트레이터에서만 액세스 할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어, 테스트 러너에 의해 구성된 정보를 `userdata.json` 파일을 사용하여 특정 테스트를 실행해야 하는지 여부를 확인합니다.

테스트 오케스트레이터 컨텍스트는 다음 형식을 사용합니다.

```
{
  "pool": {
    <device-json-pool-element>
  },
  "userData": {
    <userdata-json-content>
  },
  "config": {
    <config-json-content>
  }
}
```

#### pool

테스트 실행을 위해 선택한 장치 폴에 대한 정보입니다. 선택한 디바이스 폴의 경우 이 정보는 예 정의된 해당 최상위 디바이스 폴 배열 요소에서 검색됩니다. `device.json` 파일.

#### userData

의 정보 `userdata.json` 파일.

#### config

의 정보 `config.json` 파일.

JsonPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의에서 JSONPath 쿼리의 구문은 다음과 같습니다. `{}{{query}}`. 테스트 오케스트레이터 컨텍스트에서 데이터에 액세스할 때 각 값이 문자열, 숫자 또는 부울.

JSONPath 표기법을 사용하여 컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 단원을 참조하십시오. [IDT 컨텍스트 사용 \(p. 399\)](#).

## IDT 상태 머신 구성

#### Important

IDT v4.5.2부터 이 상태 머신은 더 이상 사용되지 않습니다. 새로운 테스트 오케스트레이터를 사용하는 것이 좋습니다. 자세한 정보는 [IDT 테스트 오케스트레이터 구성 \(p. 375\)](#)을 참조하십시오.

상태 머신은 테스트 세트 실행 흐름을 제어하는 구조입니다. 테스트 스위트의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 종료 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 세트에 사용자 정의 상태 머신이 포함되어 있지 않으면 IDT가 자동으로 상태 머신을 생성합니다. 기본 상태 머신은 다음 기능을 수행합니다.

- 테스트 런너에게 전체 테스트 세트 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않으면 에서는 테스트 스위트의 모든 테스트 그룹을 임의의 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례에 대한 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 테스트 제품군의 상태 머신은 다음 기준을 충족해야 합니다.

- 각 상태는 테스트 그룹이나 제품 보고서 파일을 실행하는 등 IDT가 수행할 작업에 해당합니다.
- 상태로 전환하면 상태와 연관된 작업이 실행됩니다.
- 각 상태는 다음 상태에 대한 전환 규칙을 정의합니다.
- 종료 상태는 다음 중 하나여야 합니다.`Succeed` 또는 `Fail`.

## 상태 머신 형식

다음 템플릿을 사용하여 고유한 템플릿을 구성할 수 있습니다.<`custom-test-suite-folder`>/suite/`state_machine.json` 파일:

```
{  
    "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 시스템에 대한 설명입니다.

### StartAt

IDT가 테스트 제품군의 실행을 시작하는 상태의 이름. 의 가치`StartAt`에 나열된 상태 중 하나로 설정해야 합니다.`States` 객체입니다.

### States

사용자 정의 상태 이름을 유효한 IDT 상태로 매핑하는 객체입니다. 각 주.## ## 객체에 매핑된 유효한 상태의 정의가 포함되어 있습니다.## ##.

0|states 객체는 다음을 포함해야 합니다.Succeed과 Fail상태입니다. 유효한 상태에 대한 자세한 내용은 단원을 참조하십시오.[유효한 상태 및 상태 정의 \(p. 381\)](#).

## 유효한 상태 및 상태 정의

이 섹션에서는 IDT 상태 시스템에서 사용할 수 있는 모든 유효한 상태의 상태 정의에 대해 설명합니다. 다음 상태 중 일부는 테스트 케이스 수준의 구성을 지원합니다. 그러나 절대적으로 필요한 경우가 아니면 테스트 케이스 수준 대신 테스트 그룹 수준에서 상태 전환 규칙을 구성하는 것이 좋습니다.

### 주 정의

- [RunTask \(p. 381\)](#)
- [Choice \(p. 382\)](#)
- [Parallel \(p. 383\)](#)
- [AddProduct기능 \(p. 384\)](#)
- [보고서 \(p. 386\)](#)
- [LogMessage \(p. 386\)](#)
- [SelectGroup \(p. 387\)](#)
- [Fail \(p. 387\)](#)
- [Succeed \(p. 387\)](#)

### RunTask

0|RunTaskstate 는 테스트 스위트에 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 전환될 상태의 이름.

#### TestGroup

선택 사항. 실행할 테스트 그룹의 ID입니다. 이 값을 지정하지 않으면 IDT는 테스트 러너가 선택하는 테스트 그룹을 실행합니다.

#### TestCases

선택 사항. 예 지정된 그룹의 테스트 케이스 ID로 구성된 배열TestGroup. 의 가치에 따라TestGroup과TestCases, IDT는 다음과 같이 테스트 실행 동작을 결정합니다.

- 이 두 경우TestGroup과TestCases를 지정하면 IDT는 테스트 그룹에서 지정된 테스트 케이스를 실행합니다.
- 일시TestCases이 지정되지만TestGroup가 지정되지 않은 경우 IDT는 지정된 테스트 케이스를 실행합니다.
- 일시TestGroup이 지정되지만TestCases지정하지 않으면 IDT는 지정된 테스트 그룹 내에서 모든 테스트 사례를 실행합니다.

- 둘 다TestGroup또는TestCases를 지정하면 IDT는 테스트 러너가 IDT CLI에서 선택하는 테스트 그룹에서 모든 테스트 케이스를 실행합니다. 테스트 러너에 대해 그룹 선택을 활성화하려면 두 가지를 모두 포함해야 합니다.RunTask과Choice구하의 상태statemachine.json파일. 이 기능의 작동 방식에 대한 예는 단원을 참조하십시오.상태 머신 예제: 사용자가 선택한 테스트 그룹 실행 (p. 390).

테스트 러너에 IDT CLI 명령을 활성화하는 방법에 대한 자세한 내용은 단원을 참조하십시오.the section called “IDT CLI 명령 활성화” (p. 397).

#### ResultVar

테스트 실행 결과와 함께 설정할 컨텍스트 변수의 이름입니다. 예 대한 값을 지정하지 않은 경우에는 이 값을 지정하지 마십시오.TestGroup. IDT는 사용자가 정의한 변수의 값을 설정합니다.ResultVar에true또는false다음을 기반으로 합니다.

- 변수 이름이 양식인 경우text\_text\_passed를 설정하면 첫 번째 테스트 그룹의 모든 테스트가 통과되었는지 또는 건너뛰었는지 여부에 따라 값이 설정됩니다.
- 다른 모든 경우에는 값이 모든 테스트 그룹의 모든 테스트가 통과되었는지 또는 건너뛰었는지 여부로 설정됩니다.

일반적으로 를 사용합니다.RunTaskstate 는 개별 테스트 케이스 ID를 지정하지 않고 테스트 그룹 ID를 지정하여 IDT가 지정된 테스트 그룹의 모든 테스트 케이스를 실행하도록 합니다. 이 상태에서 실행되는 모든 테스트 케이스는 무작위로 병렬로 실행됩니다. 그러나 모든 테스트 케이스에서 장치를 실행해야 하고 단일 장치만 사용할 수 있는 경우 테스트 케이스가 순차적으로 실행됩니다.

#### 오류 처리

지정된 테스트 그룹 또는 테스트 케이스 ID가 유효하지 않은 경우 이 상태는RunTaskError실행 오류. 상태에 실행 오류가 발생하면hasExecutionError상태 머신 컨텍스트의 변수true.

#### Choice

0|Choicestate를 사용하면 사용자 정의 조건에 따라 변환할 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

에 정의된 표현식이 없는 경우 전환될 기본 상태choices다음과 같이 평가할 수 있습니다.true.

#### FallthroughOnError

선택 사항. 표현식을 평가할 때 상태가 오류가 발생하는 경우의 동작을 지정합니다. 으로 설정true평가에서 오류가 발생하는 경우 표현식을 건너뛰려는 경우 일치하는 표현식이 없으면 상태 머신 0|Default상태입니다. 만약FallthroughOnError값이 지정되지 않은 경우 기본값은false.

#### Choices

현재 상태에서 작업을 실행한 후 전환할 상태를 결정하는 표현식 및 상태의 배열입니다.

### Choices.Expression

부울 값으로 평가되는 표현식 문자열입니다. 표현식이 다음과 같이 평가되는 경우 true 선택한 후 상태 머신이 예 정의된 상태로 전환됩니다. Choices.Next. 표현식 문자열은 상태 머신 컨텍스트에서 값을 검색한 다음 해당 값에 대한 연산을 수행하여 부울 값에 도달합니다. 상태 시스템 컨텍스트에 액세스하는 방법에 대한 자세한 내용은 단원을 참조하십시오. [상태 머신 컨텍스트 \(p. 387\)](#).

### Choices.Next

표현식이 예 정의된 경우 전환될 상태의 이름 Choices.Expression을 평가합니다. true.

### 오류 처리

0|Choicestate에서는 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.

을 사용할 수 없습니다. Catch이 상태의 오류를 처리하기 위해 block을 수행합니다. 오류가 발생했을 때 상태 머신의 실행을 중지하려면 다음을 설정해야 합니다. FallthroughOnError에 false. 그러나 설정하는 것이 좋습니다. FallthroughOnError에 true 사용 사례에 따라 다음 중 하나를 수행합니다.

- 액세스 중인 변수가 어떤 경우에는 존재하지 않을 것으로 예상되는 경우 Default 및 추가 Choices 다음 상태를 지정할 블록입니다.
- 액세스하고 있는 변수가 항상 존재해야 하는 경우 Default 상태입니다 Fail.

### Parallel

0|Parallelstate를 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 전환될 상태의 이름.

### Branches

실행할 상태 머신 정의의 배열입니다. 각 상태 시스템 정의는 자체적인 것을 포함해야 합니다 StartAt, Succeed, 및 Fail 상태입니다. 이 배열의 상태 머신 정의는 자체 정의 외부의 상태를 참조 할 수 없습니다.

### Note

각 브랜치 상태 머신은 동일한 상태 머신 컨텍스트를 공유하므로 한 브랜치에서 변수를 설정한 다음 다른 분기에서 해당 변수를 읽으면 예기치 않은 동작이 발생할 수 있습니다.

0|Parallelstate는 모든 브랜치 상태 머신을 실행한 후에만 다음 상태로 이동합니다. 디바이스가 필요한 각 상태는 디바이스를 사용할 수 있을 때까지 실행을 기다립니다. 여러 장치를 사용할 수 있는 경우 이 상태는

여러 그룹의 테스트 케이스를 병렬로 실행합니다. 사용 가능한 장치가 충분하지 않으면 테스트 케이스가 순차적으로 실행됩니다. 테스트 케이스는 병렬로 실행될 때 임의의 순서로 실행되기 때문에 다른 장치를 사용하여 동일한 테스트 그룹에서 테스트를 실행할 수 있습니다.

#### 오류 처리

브랜치 상태 머신과 상위 상태 머신이 모두 fail 실행 오류를 처리하기 위한 state입니다.

분기 상태 머신은 상위 상태 머신에 실행 오류를 전송하지 않으므로 Catch블록 - 브랜치 상태 머신에서 실행 오류를 처리합니다. 대신 `hasExecutionErrors` 공유 상태 머신 컨텍스트의 값입니다. 이 기능의 작동 방식에 대한 예는 단원을 참조하십시오. [상태 머신 예제: 두 개의 테스트 그룹을 병렬로 실행 \(p. 393\)](#).

#### AddProduct기능

이 `AddProductFeaturesstate`를 사용하면 제품 기능을 `awsiotdevicetester_report.xml` IDT에 의해 생성된 파일입니다.

제품 기능은 장치가 충족할 수 있는 특정 기준에 대한 사용자 정의 정보입니다. 예: MQTT 제품 기능은 장치가 MQTT 메시지를 올바르게 게시하도록 지정할 수 있습니다. 보고서에서 제품 기능은 다음과 같이 설정됩니다. `supported`, `not-supported` 또는 지정된 테스트가 통과되었는지 여부에 따라 사용자 지정 값

#### Note

이 `AddProductFeaturesstate`는 자체적으로 보고서를 생성하지 않습니다. 이 상태는 다음 단계로 전환해야 합니다. [Reportstate \(p. 386\)](#) 보고서를 생성합니다.

```
{  
    "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 전환될 상태의 이름.

#### Features

에 표시할 다양한 제품 기능 `awsiotdevicetester_report.xml` 파일.

#### Feature

기능 이름입니다.

#### FeatureValue

선택 사항. 보고서에 대신 사용할 사용자 지정 값입니다.`supported`. 이 값을 지정하지 않으면 테스트 결과에 따라 피쳐 값이로 설정됩니다.`supported` 또는`not-supported`.

에 대해 사용자 지정 값을 사용할 경우`FeatureValue`를 사용하여 서로 다른 조건을 사용하여 동일한 피쳐를 테스트할 수 있으며 IDT는 지원되는 조건에 대한 피쳐 값을 연결합니다. 예를 들어, 다음 블록문은`MyFeature` 두 개의 개별 피쳐 값이 있는 피쳐:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

두 테스트 그룹 모두 통과하면 기능 값이로 설정됩니다.`first-feature-supported`, `second-feature-supported`.

#### Groups

선택 사항. 테스트 그룹 ID의 어레이입니다. 기능이 지원되려면 지정된 각 테스트 그룹 내의 모든 테스트를 통과해야 합니다.

#### OneOfGroups

선택 사항. 테스트 그룹 ID의 어레이입니다. 이 기능을 지원하려면 지정된 테스트 그룹 중 하나 이상의 테스트를 통과해야 합니다.

#### TestCases

선택 사항. 테스트 사례 ID의 어레이입니다. 이 값을 지정하면 다음 사항이 적용됩니다.

- 이 기능을 지원하려면 지정된 모든 테스트 케이스가 통과해야 합니다.
- Groups 테스트 그룹 ID는 하나만 포함해야 합니다.
- OneOfGroups 지정되지 않아야 합니다.

#### IsRequired

선택 사항. 으로 설정`false`보고서에서 이 기능을 선택적 기능으로 표시합니다. 기본값은`true`입니다.

#### ExecutionMethods

선택 사항. 다음과 일치하는 실행 메서드의 배열`protocol`에 지정된 값`device.json`파일. 이 값을 지정하면 테스트 러너는 다음을 지정해야 합니다.`protocol`보고서에 피쳐를 포함하기 위해 이 배열의 값 중 하나와 일치하는 값입니다. 이 값을 지정하지 않으면 피쳐가 항상 보고서에 포함됩니다.

`O|AddProductFeatures` 상태, 값을 설정해야 합니다.`ResultVar`의`RunTask` 다음 값 중 하나를 지정합니다.

- 개별 테스트 사례 ID를 지정한 경우`ResultVar`에`group-id_test-id_passed`.
- 개별 테스트 사례 ID를 지정하지 않은 경우`ResultVar`에`group-id_passed`.

`O|AddProductFeatures` 상태는 다음과 같은 방식으로 테스트 결과를 확인합니다.

- 테스트 케이스 ID를 지정하지 않은 경우 각 테스트 그룹의 결과는 `group-id_passed` 상태 머신 컨텍스트의 변수입니다.
- 테스트 사례 ID를 지정한 경우 각 테스트의 결과는 `group-id_test-id_passed` 상태 머신 컨텍스트의 변수입니다.

#### 오류 처리

이 상태에서 제공된 그룹 ID가 유효한 그룹 ID가 아닌 경우 이 상태는 `AddProductFeaturesError` 실행 오류. 상태에 실행 오류가 발생하면 `hasExecutionErrors` 상태 머신 컨텍스트의 변수 `true`.

#### 보고서

0|Reportstate가 생성됨 `suite-name_Report.xml`과 `awsiotdevicetester_report.xml` 파일을 생성합니다. 이 상태는 보고서를 콘솔로 스트리밍합니다.

```
{  
    "Type": "Report",  
    "Next": "<state-name>"  
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

#### Next

현재 상태에서 작업을 실행한 후 전환될 상태의 이름.

항상 다음으로 전환해야 합니다. Report 테스트 실행 흐름의 끝을 향하여 테스트 러너가 테스트 결과를 볼 수 있도록 합니다. 일반적으로 이 상태 이후의 다음 상태는 다음과 같습니다. Succeed.

#### 오류 처리

이 상태에서 보고서 생성에 문제가 발생하면 `ReportError` 실행 오류.

#### LogMessage

0|LogMessagestate가 생성됨 `test_manager.log` 로그 메시지를 콘솔로 전송하고 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 전환될 상태의 이름.

#### Level

로그 메시지를 작성할 오류 수준입니다. 유효하지 않은 레벨을 지정하면 이 상태에서 오류 메시지가 생성되고 무시됩니다.

#### Message

기록할 메시지입니다.

## SelectGroup

O|SelectGroupstate 는 상태 머신 컨텍스트를 업데이트하여 선택된 그룹을 나타냅니다. 이 상태로 설정된 값은 이후의 모든 경우에 사용됩니다.Choice상태입니다.

```
{  
    "Type": "SelectGroup",  
    "Next": "<state-name>"  
    "TestGroups": [  
        <group-id>  
    ]  
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

### Next

현재 상태에서 작업을 실행한 후 전환될 상태의 이름.

### TestGroups

선택된 것으로 표시될 테스트 그룹 배열입니다. 이 배열의 각 테스트 그룹 ID에 대해 **group-id\_selected** 변수가로 설정되어 있습니다.true컨텍스트에서. IDT가 지정된 그룹이 있는지 여부를 검증하지 않으므로 유효한 테스트 그룹 ID를 제공해야 합니다.

## Fail

O|Failstate 는 상태 머신이 올바르게 실행되지 않았음을 나타냅니다. 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{  
    "Type": "Fail"  
}
```

## Succeed

O|Succeedstate는 상태 머신이 올바르게 실행되었음을 나타냅니다. 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{  
    "Type": "Succeed"  
}
```

## 상태 머신 컨텍스트

상태 머신 컨텍스트는 실행 중에 상태 머신에서 사용할 수 있는 데이터를 포함하는 읽기 전용 JSON 문서입니다. 상태 머신 컨텍스트는 상태 머신에서만 액세스할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어, 테스트 러너에 의해 구성된 정보를 **userdata.json** 파일을 사용하여 특정 테스트를 실행해야 하는지 여부를 확인합니다.

상태 시스템 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 폴에 대한 정보입니다. 선택한 디바이스 폴의 경우 이 정보는 예 정의된 해당 최상위 디바이스 폴 배열 요소에서 검색됩니다.`device.json` 파일.

#### userData

의 정보 `userdata.json` 파일.

#### config

정보 핀 `config.json` 파일.

#### suiteFailed

값은 로 설정됩니다.`false` 상태 시스템이 시작될 때. 에서 테스트 그룹이 실패하는 경우 `RunTaskstate` 이면 이 값이 로 설정됩니다.`true` 상태 시스템 실행의 나머지 기간 동안입니다.

#### specificTestGroups

테스트 러너가 전체 테스트 세트 대신 실행할 특정 테스트 그룹을 선택하면 이 키가 만들어지고 특정 테스트 그룹 ID 목록이 포함됩니다.

#### specificTestCases

테스트 러너가 전체 테스트 세트 대신 실행할 특정 테스트 케이스를 선택하면 이 키가 만들어지고 특정 테스트 사례 ID 목록이 포함됩니다.

#### hasExecutionErrors

상태 머신이 시작되면 종료되지 않습니다. 실행 오류가 발생하는 상태가 있으면 이 변수가 만들어지고 로 설정됩니다.`true` 상태 시스템 실행의 나머지 기간 동안입니다.

JsonPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의에서 JSONPath 쿼리의 구문은 다음과 같습니다.`{}{$.query}`. 일부 상태 내에서 JSONPath 쿼리를 자리 표시자 문자열로 사용할 수 있습니다. IDT는 자리 표시자 문자열을 컨텍스트에서 평가된 JSONPath 쿼리 값으로 바꿉니다. 다음 값에 자리 표시자 를 사용할 수 있습니다.

- 이 `|TestCases` 가치 `RunTask` 상태입니다.
- 이 `|Expression` 값 `Choice` 상태입니다.

상태 머신 컨텍스트에서 데이터에 액세스할 때 다음 조건이 충족되는지 확인하세요.

- JSON 경로는 다음으로 시작해야 합니다.`$.`
- 각 값은 문자열, 숫자 또는 부울로 평가되어야 합니다.

JSONPath 표기법을 사용하여 컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 단원을 참조 하십시오.[IDT 컨텍스트 사용 \(p. 399\)](#).

## 실행 오류

실행 오류는 상태 머신이 상태를 실행할 때 발생하는 상태 머신 정의의 오류입니다. IDT는 각 오류에 대한 정보를 기록합니다.`test_manager.log`로그 메시지를 콘솔로 전송하고 스트리밍합니다.

다음 메서드를 사용하여 실행 오류를 처리할 수 있습니다.

- 추가 [Catch블록 \(p. 389\)](#) 상태 정의에서.
- 의 값을 확인하십시오. [hasExecutionErrors값 \(p. 389\)](#) 상태 시스템 컨텍스트에서.

### 캐치

를 사용하려면 `Catch`를 사용하여 상태 정의에 다음을 추가합니다.

```
"Catch": [
  {
    "ErrorEquals": [
      "<error-type>"
    ]
    "Next": "<state-name>"
  }
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

#### `Catch.ErrorEquals`

포착할 오류 유형의 배열입니다. 실행 오류가 지정된 값 중 하나와 일치하는 경우 상태 머신이에 지정된 상태로 전환됩니다. `Catch.Next`. 발생하는 오류 유형에 대한 자세한 내용은 각 상태 정의를 참조하십시오.

#### `Catch.Next`

현재 상태에 지정된 값 중 하나와 일치하는 실행 오류가 발생하는 경우 다음으로 전환할 다음 상태 `Catch.ErrorEquals`.

`Catch` 블록은 하나가 일치할 때까지 순차적으로 처리됩니다. 오류 없음이 `Catch` 블록에 나열된 것과 일치하지 않으면 상태 머신이 계속 실행됩니다. 실행 오류는 잘못된 상태 정의의 결과이므로 상태에 실행 오류가 발생하면 실패 상태로 전환하는 것이 좋습니다.

#### `hasExecutionError`

일부 상태에서 실행 오류가 발생하면 오류를 발생시키는 것 외에도 `hasExecutionError값:true` 상태 시스템 컨텍스트에서. 이 값을 사용하여 오류 발생 시점을 감지한 다음 `Choice` 상태 머신을 `Fail` 상태입니다.

이 메서드의 특성은 다음과 같습니다.

- 상태 머신이 지정된 값으로 시작하지 않습니다. `hasExecutionError`를 설정하면 특정 상태가 설정될 때 까지 이 값을 사용할 수 없습니다. 즉, 명시적으로 설정해야 합니다. `FallthroughOnError`에 `false`의 경우 `Choice`은 실행 오류가 발생하지 않을 경우 상태 시스템이 종지되지 않도록 이 값에 액세스합니다.
- 일단 설정되면 `true`, `hasExecutionError`이 (가) `false`로 설정되거나 컨텍스트에서 제거되지 않습니다. 즉, 이 값은 처음으로 설정된 경우에만 유용함을 의미합니다. `true` 모든 후속 주에서는 의미있는 가치를 제공하지 않습니다.
- 이 `hasExecutionError`값은 의 모든 브랜치 상태 머신과 공유됩니다. `Parallelstate`로, 액세스 순서에 따라 예기치 않은 결과가 발생할 수 있습니다.

이러한 특성 때문에 `Catch` 블록을 대신 사용할 수 있는 경우 이 메서드를 사용하지 않는 것이 좋습니다.

## 상태 머신 예제

이 단원에서는 상태 머신 구성 몇 가지 예를 들어보겠습니다

예제

- 상태 머신 예제: 단일 테스트 그룹 실행 (p. 390)
- 상태 머신 예제: 사용자가 선택한 테스트 그룹 실행 (p. 390)
- 상태 머신 예제: 제품 기능을 사용하여 단일 테스트 그룹 실행 (p. 392)
- 상태 머신 예제: 두 개의 테스트 그룹을 병렬로 실행 (p. 393)

### 상태 머신 예제: 단일 테스트 그룹 실행

상태 머신:

- id를 사용하여 테스트 그룹을 실행합니다. GroupA, 의 제품군에 있어야 합니다. group.json 파일.
- 실행 오류 및 전환 여부를 확인합니다. Fail 발견된 경우.
- 보고서 생성 및 전환 Succeed 오류가 없는 경우 Fail 그렇지 않습니다.

```
{  
    "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

상태 머신:

- 테스트 러너가 특정 테스트 그룹을 선택했는지 확인합니다. 테스트 러너는 테스트 그룹을 선택하지 않고 테스트 케이스를 선택할 수 없기 때문에 상태 머신은 특정 테스트 케이스를 확인하지 않습니다.
- 테스트 그룹을 선택한 경우:
  - 선택한 테스트 그룹 내에서 테스트 케이스를 실행합니다. 이렇게 하기 위해 상태 머신에서 테스트 그룹 또는 테스트 케이스를 명시적으로 지정하지 않습니다. RunTask 상태입니다.
  - 모든 테스트를 실행하고 종료한 후 보고서를 생성합니다.
- 테스트 그룹이 선택되지 않은 경우
  - 테스트 그룹에서 테스트 실행 GroupA.
  - 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": "Succeeded",  
            "Catch": [  
                {  
                    "ErrorEquals": [  
                        "ReportError"  
                    ],  
                    "Next": "Fail"  
                }  
            ]  
        }  
    }  
}
```

```
        ],
    },
    "Succeed": {
        "Type": "Succeed"
    },
    "Fail": {
        "Type": "Fail"
    }
}
```

### 상태 머신 예제: 제품 기능을 사용하여 단일 테스트 그룹 실행

상태 머신:

- 테스트 그룹을 실행합니다.GroupA.
- 실행 오류 및 전환 여부를 확인합니다.Fail 발견된 경우.
- 추가FeatureThatDependsOnGroupA에 대한 기능awsiotdevicetester\_report.xml 파일:
  - 다음의 경우, GroupA가 공 패스에서 피쳐가로 설정됩니다.supported.
  - 이 기능은 보고서에서 선택 사항으로 표시되지 않습니다.
- 보고서 생성 및 전환Succeed오류가 없는 경우Fail그렇지 않으면

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

```

        "Next": "Fail"
    }
]
},
"Succeed": {
    "Type": "Succeed"
},
"Fail": {
    "Type": "Fail"
}
}
}
}

```

### 상태 머신 예제: 두 개의 테스트 그룹을 병렬로 실행

상태 머신:

- 를 실행합니다.GroupA과GroupB테스트 그룹을 병렬로 실행합니다. 이ResultVar에 의해 컨텍스트에 저장된 변수RunTask브랜치 상태 머신의 상태를 다음 위치에서 사용할 수 있습니다.AddProductFeatures상태입니다.
- 실행 오류 및 전환 여부를 확인합니다.Fail발견된 경우. 이 상태 머신은 a를 사용하지 않습니다.Catch이 메서드가 분기 상태 시스템에서 실행 오류를 감지하지 못하기 때문에 block.
- 에 피처를 추가합니다.awsiotdevicetester\_report.xml전달된 그룹을 기반으로 하는 파일
  - 다음의 경우,GroupA가공 패스에서 피쳐가 로 설정됩니다.supported.
  - 이 기능은 보고서에서 선택 사항으로 표시되지 않습니다.
- 보고서 생성 및 전환Succeed오류가 없는 경우Fail그렇지 않으면

장치 폴에 두 개의 장치가 구성된 경우 둘 다GroupA과GroupB동시에 실행할 수 있습니다. 하지만, GroupA또는GroupB여러 테스트가 있으면 두 장치 모두 해당 테스트에 할당 될 수 있습니다. 하나의 장치만 구성된 경우 테스트 그룹이 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": "Succeeded",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
                "Next": "Failed"
            }
        ],
        "Succeeded": {
            "Type": "Succeeded"
        },
        "Failed": {
            "Type": "Failed"
        }
    }
}
```

## IDT 테스트 케이스 실행 파일 만들기

다음과 같은 방법으로 테스트 케이스 실행 파일을 만들고 테스트 스위트 폴더에 배치할 수 있습니다.

- 의 인수 또는 환경 변수를 사용하는 테스트 스위트의 경우 `test.json` 파일을 사용하여 실행할 테스트를 결정합니다. 전체 테스트 세트에 대해 단일 테스트 사례 실행 파일을 만들거나 테스트 세트의 각 테스트 그룹에 대한 테스트 실행 파일을 만들 수 있습니다.
- 지정된 명령을 기반으로 특정 테스트를 실행하려는 테스트 스위트의 경우 테스트 스위트의 각 테스트 케이스에 대해 하나의 테스트 케이스 실행 파일을 만듭니다.

테스트 작성기는 사용 사례에 적합한 접근 방식을 결정하고 그에 따라 테스트 케이스 실행 파일을 구조화할 수 있습니다. 각각에 올바른 테스트 케이스 실행 가능 경로를 제공하는지 확인하십시오. `test.jsonfile` 및 지정된 실행 파일이 올바르게 실행됩니다.

모든 장치가 테스트 케이스를 실행할 준비가 되면 IDT는 다음 파일을 읽습니다.

- `test.json` 선택한 테스트 케이스에 대해 시작할 프로세스와 설정할 환경 변수가 결정됩니다.
- `suite.json`에 대한 테스트 스위트는 설정할 환경 변수를 결정합니다.

IDT는에 지정된 명령과 인수에 따라 필요한 테스트 실행 프로세스를 시작합니다. `test.jsonfile`을 선택하고 필요한 환경 변수를 프로세스에 전달합니다.

## IDT 클라이언트 SDK 사용

IDT 클라이언트 SDK를 사용하면 IDT 및 테스트 종인 장치와 상호 작용할 수 있는 API 명령을 사용하여 테스트 실행 파일에 테스트 로직을 작성하는 방법을 간소화할 수 있습니다. IDT는 현재 다음 SDK를 제공합니다.

- IDT 클라이언트 SDK for Python
- IDT 클라이언트 SDK for Go

이러한 SDK는 `<device-tester-extract-location>/sdksfolder`. 새 테스트 사례 실행 파일을 만들 때 사용할 SDK를 테스트 케이스 실행 파일이 포함된 폴더에 복사하고 코드에서 SDK를 참조해야 합니다. 이 단원에서는 테스트 사례 실행 파일에서 사용할 수 있는 API 명령에 대해 간단히 설명합니다.

### 이 섹션의 내용

- 장치 상호 작용 (p. 396)
- IDT 상호 작용 (p. 396)

- [호스트 상호 작용 \(p. 396\)](#)

## 장치 상호 작용

다음 명령을 사용하면 추가 장치 상호 작용 및 연결 관리 기능을 구현하지 않고도 테스트 중인 장치와 통신할 수 있습니다.

### **ExecuteOnDevice**

테스트 스위트가 SSH 또는 Docker 셸 연결을 지원하는 장치에서 셸 명령을 실행할 수 있도록 허용합니다.

### **CopyToDevice**

테스트 스위트에서 IDT를 실행하는 호스트 컴퓨터에서 SSH 또는 Docker 셸 연결을 지원하는 디바이스의 지정된 위치로 로컬 파일을 복사할 수 있습니다.

### **ReadFromDevice**

테스트 제품군의 UART 연결을 지원하는 디바이스의 직렬 포트에서 읽을 수 있습니다.

## Note

IDT는 컨텍스트의 장치 액세스 정보를 사용하여 만들어진 장치에 대한 직접 연결을 관리하지 않으므로 테스트 케이스 실행 파일에서 이러한 장치 상호 작용 API 명령을 사용하는 것이 좋습니다. 그러나 이러한 명령이 테스트 사례 요구 사항을 충족하지 않으면 IDT 컨텍스트에서 장치 액세스 정보를 검색하고 이를 사용하여 테스트 스위트에서 장치에 직접 연결할 수 있습니다.

직접 연결하려면 [에서 정보를 검색하십시오](#)

`os.device.connectivity` 및 `resource.devices.connectivity` 테스트 중인 디바이스와 리소스 디바이스에 대한 필드입니다. IDT 컨텍스트 사용에 대한 자세한 내용은 단원을 참조하십시오 [IDT 컨텍스트 사용 \(p. 399\)](#).

## IDT 상호 작용

다음 명령을 사용하면 테스트 스위트가 IDT와 통신할 수 있습니다.

### **PollForNotifications**

테스트 스위트가 IDT의 알림을 확인할 수 있도록 허용합니다.

### **GetContextValue 및 GetContextString**

테스트 스위트가 IDT 컨텍스트에서 값을 검색할 수 있도록 허용합니다. 자세한 정보는 [IDT 컨텍스트 사용 \(p. 399\)](#)을 참조하십시오.

### **SendResult**

테스트 세트에서 테스트 사례 결과를 IDT에 보고할 수 있습니다. 이 명령은 테스트 스위트의 각 테스트 케이스가 끝날 때 호출해야 합니다.

## 호스트 상호 작용

다음 명령을 사용하면 테스트 스위트가 호스트 컴퓨터와 통신할 수 있습니다.

### **PollForNotifications**

테스트 스위트가 IDT의 알림을 확인할 수 있도록 허용합니다.

### **GetContextValue 및 GetContextString**

테스트 스위트가 IDT 컨텍스트에서 값을 검색할 수 있도록 허용합니다. 자세한 정보는 [IDT 컨텍스트 사용 \(p. 399\)](#)을 참조하십시오.

#### **ExecuteOnHost**

테스트 스위트가 로컬 시스템에서 명령을 실행할 수 있도록 허용하고 IDT가 테스트 케이스 실행 가능 수명 주기를 관리할 수 있도록 합니다.

### **IDT CLI 명령 활성화**

`o/run-suite` 명령 IDT CLI는 테스트 러너가 테스트 실행을 사용자 지정할 수 있는 몇 가지 옵션을 제공합니다. 테스트 러너가 이러한 옵션을 사용하여 사용자 지정 테스트 스위트를 실행할 수 있도록 하려면 IDT CLI에 대한 지원을 구현합니다. 지원을 구현하지 않으면 테스트 러너는 여전히 테스트를 실행할 수 있지만 일부 CLI 옵션이 제대로 작동하지 않습니다. 이상적인 고객 경험을 제공하기 위해 다음 인수에 대한 지원을 구현하는 것이 좋습니다.`run-suite` IDT CLI의 명령:

#### **timeout-multiplier**

테스트를 실행하는 동안 모든 시간 초과에 적용할 1.0보다 큰 값을 지정합니다.

테스트 러너는 이 인수를 사용하여 실행하려는 테스트 케이스의 시간 제한을 늘릴 수 있습니다. 테스트 러너가 이 인수를 지정할 때 `run-suite` 명령을 실행하면 IDT가 이 변수를 사용하여 `IDT_TEST_TIMEOUT` 환경 변수의 값을 계산하고 `config.timeoutMultiplier` IDT 컨텍스트의 필드입니다. 이 인수를 지원하려면 다음을 수행해야 합니다.

- 에서 제한 시간 값을 직접 사용하는 대신 `test.json` 파일을 사용하여 `IDT_TEST_TIMEOUT` 환경 변수를 읽어 올바르게 계산된 시간 초과 값을 가져옵니다.
- Retrieve the `config.timeoutMultiplier` IDT 컨텍스트에서 값을 입력하고 장기 실행 시간 초과에 적용합니다.

시간 초과 이벤트로 인해 조기 종료에 대한 자세한 내용은 단원을 참조하십시오. [종료 동작 지정 \(p. 398\)](#).

#### **stop-on-first-failure**

IDT가 실패할 경우 모든 테스트 실행을 중지하도록 지정합니다.

테스트 러너가 이 인수를 지정할 때 `run-suite` 명령을 실행하면 IDT는 오류가 발생하자마자 테스트 실행을 중지합니다. 그러나 테스트 케이스가 병렬로 실행되는 경우 예기치 않은 결과가 발생할 수 있습니다. 지원을 구현하려면 IDT에서 이 이벤트가 발생하면 테스트 논리가 실행 중인 모든 테스트 케이스를 중지하고 임시 리소스를 정리하고 테스트 결과를 IDT에 보고하도록 지시해야 합니다. 장애 발생 초기에 종료하는 방법에 대한 자세한 내용은 단원을 참조하십시오. [종료 동작 지정 \(p. 398\)](#).

#### **group-id 및 test-id**

IDT가 선택한 테스트 그룹 또는 테스트 케이스만 실행하도록 지정합니다.

테스트 러너는 이러한 인수와 함께 사용할 수 있습니다. `run-suite` 명령을 사용하여 다음 테스트 실행 동작을 지정합니다.

- 지정된 테스트 그룹 내에서 모든 테스트를 실행합니다.
- 지정된 테스트 그룹 내에서 선택한 테스트를 실행합니다.

이러한 인수를 지원하려면 테스트 스위트의 상태 머신에 특정 집합이 포함되어야 합니다. `RunTask`과 `Choice` 상태 시스템의 상태. 사용자 정의 상태 머신을 사용하지 않는 경우 기본 IDT 상태 머신에 필요한 상태가 포함되므로 추가 작업을 수행할 필요가 없습니다. 그러나 사용자 정의 상태 머신을 사용하는 경우 [상태 머신 예제: 사용자가 선택한 테스트 그룹 실행 \(p. 390\)](#) 상태 머신에 필요한 상태를 추가하는 샘플로 사용합니다.

IDT CLI 명령에 대한 자세한 내용은 단원을 참조하십시오. [사용자 지정 테스트 스위트 디버깅 및 실행 \(p. 409\)](#).

## 이벤트 로그 쓰기

테스트가 실행되는 동안 데이터를 `stdout`과 `stderr` 콘솔에 이벤트 로그 및 오류 메시지를 작성합니다. 콘솔 메시지의 형식에 대한 자세한 내용은 단원을 참조하십시오. [콘솔 메시지 형식 \(p. 411\)](#).

IDT가 테스트 세트 실행을 마치면 이 정보는 `test_manager.log`에 있는 파일을 엽니다. `<devicetester-extract-location>/results/<execution-id>/logsfolder`.

테스트 대상 장치의 로그를 포함하여 테스트 실행의 로그를 기록하도록 각 테스트 케이스를 구성할 수 있습니다. `<group-id>_<test-id>`에 있는 파일을 엽니다. `<device-tester-extract-location>/results/<execution-id>/logsfolder`. 이렇게 하려면 다음을 사용하여 IDT 컨텍스트에서 로그 파일의 경로를 검색합니다. `testData.logFilePath` 쿼리하고 해당 경로에 파일을 만들고 원하는 내용을 작성합니다. IDT는 실행 중인 테스트 케이스를 기반으로 경로를 자동으로 업데이트합니다. 테스트 사례에 대한 로그 파일을 생성하지 않도록 선택하면 해당 테스트 사례에 대한 파일이 생성되지 않습니다.

텍스트 실행 파일을 설정하여 필요에 따라 추가 로그 파일을 만들 수도 있습니다. `<device-tester-extract-location>/logsfolder`. 파일을 덮어쓰지 않도록 로그 파일 이름에 고유한 접두사를 지정하는 것이 좋습니다.

## 결과를 IDT에 보고

IDT는 테스트 결과를 `awsiotdevicetester_report.xml` 및 `suite-name_report.xml` 파일을 생성합니다. 이 보고서 파일은 `<device-tester-extract-location>/results/<execution-id>/`. 두 보고서 모두 테스트 세트 실행의 결과를 캡처합니다. IDT가 이러한 보고서에 사용하는 스키마에 대한 자세한 내용은 단원을 참조하십시오. [IDT 테스트 결과 및 로그 검토 \(p. 411\)](#)

의 컨텐츠를 채우려면 `suite-name_report.xml` 파일을 사용해야 합니다. `SendResult` 명령을 사용하여 테스트 실행이 완료되기 전에 테스트 결과를 IDT에 보고합니다. IDT가 테스트 결과를 찾을 수 없으면 테스트 사례에 오류가 발생합니다. 다음 파일은 테스트 결과를 IDT에 보내는 명령을 보여줍니다.

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

API를 통해 결과를 보고하지 않으면 IDT는 테스트 아티팩트 폴더에서 테스트 결과를 찾습니다. 이 폴더의 경로가에 저장됩니다. `testData.testArtifactsPath` IDT 컨텍스트에 제출되었습니다. 이 폴더에서 IDT는 첫 번째 알파벳 순으로 정렬된 XML 파일을 테스트 결과로 사용합니다.

테스트 논리가 JUnit XML 결과를 생성하는 경우 결과를 파싱한 다음 API를 사용하여 IDT에 제출하는 대신 결과를 IDT에 직접 제공하기 위해 아티팩트 폴더의 XML 파일에 테스트 결과를 기록할 수 있습니다.

이 방법을 사용하는 경우 테스트 논리가 테스트 결과를 정확하게 요약하고 결과 파일의 형식을 `suite-name_report.xml` 파일. IDT는 사용자가 제공한 데이터의 검증을 수행하지 않습니다. 단, 다음과 같은 예외가 있습니다.

- IDT는 의 모든 속성을 무시합니다. `testsuites` 태그. 대신 보고된 다른 테스트 그룹 결과에서 태그 속성을 계산합니다.
- 1개 이상 `testsuite` 태그가 안에 있어야 합니다. `testsuites`.

IDT는 모든 테스트 사례에 대해 동일한 객체 폴더를 사용하고 테스트 실행 사이에 결과 파일을 삭제하지 않기 때문에 IDT가 잘못된 파일을 읽는 경우 이 방법으로 잘못된 보고가 발생할 수도 있습니다. 모든 테스트 사례에서 생성된 XML 결과 파일에 대해 동일한 이름을 사용하여 각 테스트 사례의 결과를 덮어쓰고 IDT에서 사용할 수 있는 올바른 결과를 사용할 수 있는지 확인하는 것이 좋습니다. 테스트 스위트에서 보고에 대한 흔한 접근 방식을 사용할 수 있습니다. 즉, 일부 테스트 케이스에는 XML 결과 파일을 사용하고 다른 테스트 사례에 대해서는 API를 통해 결과를 제출할 수 있지만 이 방법을 사용하지 않는 것이 좋습니다.

## 종료 동작 지정

테스트 케이스에서 오류 또는 오류 결과를 보고하더라도 항상 종료 코드가 0으로 종료되도록 텍스트 실행 파일을 구성합니다. 테스트 케이스가 실행되지 않았거나 테스트 케이스 실행 파일이 결과를 IDT와 통신할 수

없는 경우에만 0이 아닌 종료 코드를 사용하십시오. IDT가 0이 아닌 종료 코드를 수신하면 테스트 케이스가 실행되지 않는 오류가 발생했음을 표시합니다.

IDT는 다음 이벤트가 완료되기 전에 테스트 케이스의 실행을 중지하도록 요청하거나 기대할 수 있습니다. 이 정보를 사용하여 테스트 케이스에서 이러한 각 이벤트를 감지하도록 테스트 사례 실행 파일을 구성합니다.

#### 제한 시간

테스트 케이스가 예 지정된 시간 초과 값보다 오래 실행될 때 발생합니다. `test.json` 파일 테스트 러너가 `timeout-multiplier` 인수를 사용하여 시간 초과 승수를 지정한 다음 IDT는 승수를 사용하여 시간 초과 값을 계산합니다.

이 이벤트를 감지하려면 `IDT_TEST_TIMEOUT` 환경 변수를 사용하십시오. 테스트 러너가 테스트를 시작하면 `IDT_TEST_TIMEOUT` 환경 변수의 값을 계산된 시간 초과 값(초)으로 설정하고 변수를 테스트 케이스 실행 파일에 전달합니다. 변수 값을 읽고 적절한 타이머를 설정할 수 있습니다.

#### 종단

테스트 러너가 IDT를 인터럽트할 때 발생합니다. 예를 들어, 키를 눌러 `Ctrl+C`.

터미널은 모든 하위 프로세스에 신호를 전파하므로 테스트 케이스에서 신호 처리기를 구성하여 인터럽트 신호를 감지할 수 있습니다.

또는 주기적으로 API를 폴링하여 의 값을 확인할 수 있습니다. `CancellationRequested`의 부울 `PollForNotifications` API 응답. IDT가 인터럽트 신호를 수신하면 `CancellationRequested` 부울로 변환 `true`.

#### 첫 번째 실패 시 중지

현재 테스트 케이스와 병렬로 실행되는 테스트 케이스가 실패하고 테스트 러너가 `stop-on-first-failure` 실패가 발생할 때 IDT가 중지되도록 지정하는 인수.

이 이벤트를 감지하기 위해 주기적으로 API를 폴링하여 `CancellationRequested`의 부울 `PollForNotifications` API 응답. IDT에 장애가 발생하여 첫 번째 실패 시 중지되도록 구성된 경우 `CancellationRequested` 부울로 변환 `true`.

이러한 이벤트가 발생하면 IDT는 현재 실행 중인 테스트 케이스 실행이 완료될 때까지 5분 동안 기다립니다. 실행 중인 모든 테스트 케이스가 5분 이내에 종료되지 않으면 IDT는 각 프로세스를 강제로 중지합니다. 프로세스가 종료되기 전에 IDT가 테스트 결과를 받지 못하면 테스트 케이스가 시간 초과된 것으로 표시됩니다. 가장 좋은 방법은 테스트 케이스가 이벤트 중 하나가 발생할 때 다음 작업을 수행하는지 확인해야 합니다.

1. 일반 테스트 로직 실행을 중지합니다.
2. 테스트 대상 디바이스의 테스트 아티팩트와 같은 임시 리소스를 정리합니다.
3. 테스트 실패 또는 오류와 같은 테스트 결과를 IDT에 보고합니다.
4. 종료.

## IDT 컨텍스트 사용

IDT가 테스트 스위트를 실행하면 테스트 스위트는 각 테스트 실행 방법을 결정하는 데 사용할 수 있는 데이터 집합에 액세스할 수 있습니다. 이 데이터를 IDT 컨텍스트라고 합니다. 예를 들어, 테스트 러너에서 제공하는 사용자 데이터 `userdata.json` IDT 컨텍스트에서 테스트 스위트에 파일을 사용할 수 있습니다.

IDT 컨텍스트는 읽기 전용 JSON 문서로 간주될 수 있습니다. 테스트 스위트는 객체, 배열, 숫자 등과 같은 표준 JSON 데이터 유형을 사용하여 데이터를 검색하고 컨텍스트에 데이터를 쓸 수 있습니다.

### 컨텍스트 스키마

IDT 컨텍스트는 다음 형식을 사용합니다.

```
{  
    "config": {  
        <config-json-content>  
        "timeoutMultiplier": <timeout-multiplier>,  
        "idtRootPath": <path/to/IDT/root>  
    },  
    "device": {  
        <device-json-device-element>  
    },  
    "devicePool": {  
        <device-json-pool-element>  
    },  
    "resource": {  
        "devices": [  
            {  
                <resource-json-device-element>  
                "name": "<resource-name>"  
            }  
        ]  
    },  
    "testData": {  
        "awsCredentials": {  
            "awsAccessKeyId": "<access-key-id>",  
            "awsSecretAccessKey": "<secret-access-key>",  
            "awsSessionToken": "<session-token>"  
        },  
        "logFilePath": "/path/to/log/file"  
    },  
    "userData": {  
        <userdata-json-content>  
    }  
}
```

#### **config**

의 정보[config.json파일 \(p. 408\)](#). config 필드에는 다음과 같은 추가 필드도 포함됩니다.

##### **config.timeoutMultiplier**

테스트 스위트에서 사용하는 모든 제한 시간 값에 대한 승수입니다. 이 값은 IDT CLI의 테스트 러너에 의해 지정됩니다. 기본값은 1입니다.

##### **config.idRootPath**

이 값은 다음을 구성하는 동안 IDT의 절대 경로 값에 대한 자리 표시자입니다.[userdata.json파일 \(p. 402\)](#) 선택한 디바이스에 대해

#### **device**

테스트 실행을 위해 선택한 장치에 대한 정보입니다. 이 정보는 devices 배열 요소[device.json파일 \(p. 402\)](#) 선택한 디바이스에 대해

#### **devicePool**

테스트 실행을 위해 선택한 장치 풀에 대한 정보입니다. 이 정보는 정의된 최상위 디바이스 풀 배열 요소와 동일합니다.[device.json](#) 선택한 장치 풀에 대한 파일입니다.

#### **resource**

의 리소스 디바이스에 대한 정보[resource.json파일](#).

##### **resource.devices**

이 정보는 devices에 정의된 배열[resource.json파일](#). Each devices 요소에는 다음 추가 필드가 포함됩니다.

#### **resource.device.name**

리소스 디바이스의 이름입니다. 이 값은 로 설정됩니다.`requiredResource.name`의 가치`test.json`파일.

#### **testData.awsCredentials**

이 AWS 테스트에서 연결하기 위해 사용하는 자격 증명 AWS 클라우드. 이 정보는 `config.json` 파일.

#### **testData.logFilePath**

테스트 사례에서 로그 메시지를 기록하는 로그 파일의 경로입니다. 테스트 제품군의 파일이 없는 경우 이 파일을 생성합니다.

#### **userData**

테스트 러너가 제공하는 정보 [userdata.json](#) 파일 (p. 405).

## 컨텍스트에서 데이터 액세스

구성 파일과 텍스트 실행 파일에서 JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. `GetContextValue` 과 `GetContextString` API. IDT 컨텍스트에 액세스하기 위한 JSONPath 문자열의 구문은 다음과 같이 다릅니다.

- `Insuite.json` 과 `test.json` 를 사용합니다. `{}{{query}}`. 즉, 루트 요소를 사용하지 마십시오. `$`. 표현을 시작합니다.
- `Instatemachine.json` 를 사용합니다. `{}{{$query}}`.
- API 명령에서는 다음을 사용합니다. `query` 또는 `{}{{$query}}` 명령에 따라 다릅니다. 자세한 내용은 SDK 의 인라인 설명서를 참조하십시오.

다음 표는 일반적인 푸바 JSONPath 표현식의 연산자에 대해 설명합니다.

연산자	설명
<code>\$</code>	루트 요소입니다. IDT의 최상위 컨텍스트 값은 객체 이므로 일반적으로 <code>\$</code> . 쿼리를 시작합니다.
<code>.childName</code>	이름이 있는 자식 요소에 액세스합니다. <code>.childName</code> 객체에서 배열에 적용하면 각 요소에 이 연산자가 적용된 새 배열을 생성합니다. 요소 이름은 대/소문자를 구분합니다. 예를 들어, 액세스하기 위한 쿼리 <code>awsRegion</code> 의 가치 <code>config</code> 객체가 있습니다. <code>\$.config.awsRegion</code> .
<code>[start:end]</code>	배열에서 요소를 필터링하여 <code>start</code> 색인 및 <code>end</code> 인덱스, 둘 다 포함.
<code>[index1, index2, ..., indexN]</code>	배열의 요소를 필터링하여 지정된 인덱스에서만 항목을 검색합니다.
<code>[?(expr)]</code>	다음을 사용하여 배열의 요소를 필터링합니다. <code>expr</code> 표현식. 이 표현식은 부울 값으로 평가되어야 합니다.

필터 표현식을 만들려면 다음 구문을 사용합니다.

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

이 구문에서:

- `jsonpath`는 표준 JSON 구문을 사용하는 JSONPath입니다.
- `value`은 표준 JSON 구문을 사용하는 모든 사용자 정의 값입니다.
- `operator`은 다음 연산자 중 하나입니다.
  - <(작음)
  - <=작거나 같음)
  - ==(같음)

표현식의 JSONPath 또는 값이 배열, 부울 또는 객체 값인 경우 사용할 수 있는 유일한 지원되는 이진 연산자입니다.

- >=(크거나 같음)
- >(큽)
- =~(정규 표현식 일치). 필터 표현식에서 이 연산자를 사용하려면 표현식 왼쪽의 JSONPath 또는 값이 문자열로 평가되고 오른쪽은 다음 패턴 값이어야 합니다.[RE2 구문](#).

{{형식으로 JSONPath 쿼리를 사용할 수 있습니다.##}}의 자리 표시자 문자열 `args`과 `environmentVariables`의 필드 `test.json` 파일 및 내부 `environmentVariables`의 필드 `suite.json` 파일을 생성합니다. IDT는 컨텍스트 조회를 수행하고 쿼리의 평가된 값으로 필드를 채웁니다. 예: `suite.jsonfile`, 자리 표시자 문자열을 사용하여 각 테스트 케이스에 따라 변경되는 환경 변수 값을 지정할 수 있으며 IDT는 환경 변수를 각 테스트 사례에 대한 을 바른 값으로 채웁니다. 그러나 에서 자리 표시자 문자열을 사용하는 경우 `test.json`과 `suite.jsonfiles`의 경우 쿼리에 대해 다음 고려 사항이 적용됩니다.

- 각 발생은 다음과 같이 해야 합니다. `devicePool` 쿼리의 키는 모두 소문자로 표시됩니다. 즉, 사용 `devicepool` 그 대신
- 배열의 경우 문자열 배열만 사용할 수 있습니다. 또한 어레이의 비표준이 아닌 `item1, item2, ..., itemN` 형식 배열에 요소가 하나만 포함되어 있으면 다음과 같이 직렬화됩니다. `item` 문자열 필드와 구별 할 수 없게 만듭니다.
- 자리 표시자를 사용하여 컨텍스트에서 객체를 검색할 수 없습니다.

이러한 고려 사항으로 인해 가능하면 API를 사용하여 의 자리 표시자 문자열 대신 테스트 논리의 컨텍스트에 액세스하는 것이 좋습니다. `test.json`과 `suite.json` 파일을 생성합니다. 그러나 경우에 따라 JSONPath 자리 표시자를 사용하여 단일 문자열을 검색하여 환경 변수로 설정하는 것이 더 편리할 수 있습니다.

## 테스트 러너의 설정 구성

사용자 지정 테스트 스위트를 실행하려면 테스트 러너가 실행하려는 테스트 스위트를 기반으로 해당 설정을 구성해야 합니다. 설정은 에 있는 구성 파일 템플릿에 따라 지정됩니다. `<device-tester-extract-location>/configs/folder`. 필요한 경우 테스트 러너도 설정해야 합니다. AWS IDT가 에 연결하는 데 사용할 자격 증명 AWS 클라우드.

테스트 작성기는 이러한 파일을 다음과 같이 구성해야 합니다. [테스트 제품군 디버깅 \(p. 409\)](#). 테스트 스위트를 실행하기 위해 필요에 따라 다음 설정을 구성할 수 있도록 테스트 러너에 대한 지침을 제공해야 합니다.

### device.json 구성

이 `device.json` 파일에는 테스트가 실행되는 디바이스에 대한 정보가 들어 있습니다 (예: IP 주소, 로그인 정보, 운영 체제, CPU 아키텍처).

테스트 러너는 다음 템플릿을 사용하여 이 정보를 제공할 수 있습니다. `device.json`에 있는 파일을 엽니다. `<device-tester-extract-location>/configs/folder`.

[

```
{  
    "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

디바이스 풀이라고 하는 디바이스 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 디바이스의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 디바이스가 사용됩니다.

#### sku

테스트 대상 디바이스를 고유하게 식별하는 영숫자 값입니다. SKU는 검증된 디바이스를 추적하는 데 사용됩니다.

#### Note

보드에 보드를 리스트하려는 경우 AWS 파트너 디바이스 카탈로그에서 지정하는 SKU가 목록 등록 프로세스에 사용하는 SKU와 일치해야 합니다.

### **features**

선택 사항. 디바이스의 지원되는 기능이 포함된 배열입니다. 장치 기능은 테스트 스위트에서 구성하는 사용자 정의 값입니다. 테스트 러너에 포함할 기능 이름 및 값에 대한 정보를 제공해야 합니다.`device.json` 파일. 예를 들어, 다른 장치에 대해 MQTT 서버로 작동하는 장치를 테스트하려는 경우 테스트 로직을 구성하여 이름이 지정된 기능에 대해 지원되는 특정 수준의 유효성을 검사할 수 있습니다.`MQTT_QOS`. 테스트 러너는 이 기능 이름을 제공하고 기능 값을 장치에서 지원하는 QOS 수준으로 설정합니다. 에서 제공된 정보를 검색할 수 있습니다.[IDT 컨텍스트 \(p. 399\)](#)와 함께`devicePool.features` 쿼리 또는 [상태 머신 컨텍스트 \(p. 387\)](#)와 함께`pool.features` 쿼리.

#### **features.name**

기능 이름입니다.

#### **features.value**

지원되는 기능 값입니다.

#### **features.configs**

필요한 경우 기능에 대한 구성 설정입니다.

#### **features.config.name**

구성 설정의 이름입니다.

#### **features.config.value**

지원되는 설정 값입니다.

### **devices**

테스트 할 풀에 있는 디바이스 배열입니다. 디바이스는 최소 1개 이상 필요합니다.

#### **devices.id**

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

#### **connectivity.protocol**

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 디바이스는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 유일한 값은 `ssh`과 `uart` 물리적 디바이스용 `docker` Docker 컨테이너용.

#### **connectivity.ip**

테스트 대상 디바이스의 IP입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

#### **connectivity.port**

선택 사항. SSH 연결에 사용할 포트 번호입니다.

기본값은 22입니다.

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 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki

- password

#### **connectivity.auth.credentials**

인증에 사용되는 자격 증명입니다.

#### **connectivity.auth.credentials.password**

테스트 대상 디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 connectivity.auth.method가 password로 설정된 경우에만 적용됩니다.

#### **connectivity.auth.credentials.privKeyPath**

테스트 대상 디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 connectivity.auth.method가 pki로 설정된 경우에만 적용됩니다.

#### **connectivity.auth.credentials.user**

테스트 대상 디바이스에 로그인하기 위한 사용자 이름입니다.

#### **connectivity.serialPort**

선택 사항. 장치가 연결된 직렬 포트입니다.

이 속성은 connectivity.protocol이 uart로 설정된 경우에만 적용됩니다.

#### **connectivity.containerId**

테스트 대상 도커 컨테이너의 컨테이너 ID 또는 이름입니다.

이 속성은 connectivity.protocol이 docker로 설정된 경우에만 적용됩니다.

#### **connectivity.containerUser**

선택 사항. 컨테이너 내에서 사용자에 대한 사용자의 이름. 기본값은 Dockerfile에서 제공하는 사용자입니다.

기본값은 22입니다.

이 속성은 connectivity.protocol이 docker로 설정된 경우에만 적용됩니다.

#### **Note**

테스트 러너가 테스트에 대해 잘못된 장치 연결을 구성하는지 확인하려면 검색하면 됩니다.  
pool.Devices[0].Connectivity.Protocol상태 머신 컨텍스트에서 예상값과 비교하십시오.Choice상태입니다. 잘못된 프로토콜이 사용되는 경우LogMessage상태 및 국가로의 전환Fail상태입니다.

또는 오류 처리 코드를 사용하여 잘못된 장치 유형에 대한 테스트 실패를 보고할 수 있습니다.

## (선택 사항) 사용자 데이터.json 구성

userdata.jsonfile에는 테스트 세트에 필요하지만 에서 지정되지 않은 추가 정보가 포함되어 있습니다.device.json파일. 이 파일의 형식은[userdata\\_scheme.json파일 \(p. 374\)](#)테스트 제품군의 집합에 정의되어 있습니다. 테스트 작성자인 경우 작성한 테스트 스위트를 실행할 사용자에게 이 정보를 제공해야 합니다.

## (선택 사항) 리소스.json 구성

resource.jsonfile에는 리소스 장치로 사용될 모든 장치에 대한 정보가 들어 있습니다. 리소스 디바이스는 테스트 중인 장치의 특정 기능을 테스트하는 데 필요한 장치입니다. 예를 들어 장치의 Bluetooth 기능을 테스트하려면 리소스 장치를 사용하여 기기가 성공적으로 연결할 수 있는지 테스트할 수 있습니다. 리소스 디바이스는 선택 사항이며 필요한 만큼의 리소스 디바이스를 요구할 수 있습니다. 테스트 작성자는[테스트.json 파일 \(p. 372\)](#)테스트에 필요한 리소스 디바이스 기능을 정의합니다. 테스트 러너

는resource.json파일을 사용하여 필요한 기능을 가진 리소스 디바이스 폴을 제공합니다. 작성한 테스트 스위트를 실행할 사용자에게 이 정보를 제공해야 합니다.

테스트 러너는 다음 템플릿을 사용하여 이 정보를 제공할 수 있습니다.resource.json에 있는 파일을 엽니다.<**device-tester-extract-location**>/configs/folder.

```
[  
  {  
    "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**

디바이스 폴이라고 하는 디바이스 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 폴에 속한 디바이스의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 폴에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 디바이스가 사용됩니다.

#### **features**

선택 사항. 디바이스의 지원되는 기능이 포함된 배열입니다. 이 필드에 필요한 정보는[테스트.json 파일 \(p. 372\)](#)테스트 스위트에서 실행할 테스트와 해당 테스트를 실행하는 방법을 결정합니다. 테스트 스위트에 기능이 필요하지 않은 경우 이 필드는 필요하지 않습니다.

#### **features.name**

기능 이름입니다.

**features.version**

기능 버전입니다.

**features.jobSlots**

장치를 동시에 사용할 수 있는 테스트 수를 나타내는 설정입니다. 기본값은 1입니다.

**devices**

테스트 할 풀에 있는 디바이스 배열입니다. 디바이스는 최소 1개 이상 필요합니다.

**devices.id**

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

**connectivity.protocol**

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 디바이스는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 유일한 값은 ssh과 uart 물리적 디바이스용 docker Docker 컨테이너용.

**connectivity.ip**

테스트 대상 디바이스의 IP 입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

**connectivity.port**

선택 사항. SSH 연결에 사용할 포트 번호입니다.

기본값은 22입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

**connectivity.auth**

연결에 대한 인증 정보입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

**connectivity.auth.method**

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki
- password

**connectivity.auth.credentials**

인증에 사용되는 자격 증명입니다.

**connectivity.auth.credentials.password**

테스트 대상 디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 connectivity.auth.method가 password로 설정된 경우에만 적용됩니다.

**connectivity.auth.credentials.privKeyPath**

테스트 대상 디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 connectivity.auth.method가 pki로 설정된 경우에만 적용됩니다.

**connectivity.auth.credentials.user**

테스트 대상 디바이스에 로그인하기 위한 사용자 이름입니다.

#### **connectivity.serialPort**

선택 사항. 장치가 연결된 직렬 포트입니다.

이 속성은 connectivity.protocol이 uart로 설정된 경우에만 적용됩니다.

#### **connectivity.containerId**

테스트 대상 도커 컨테이너의 컨테이너 ID 또는 이름입니다.

이 속성은 connectivity.protocol이 docker로 설정된 경우에만 적용됩니다.

#### **connectivity.containerUser**

선택 사항. 컨테이너 내에서 사용자에 대한 사용자의 이름. 기본값은 Dockerfile에서 제공하는 사용자입니다.

기본값은 22입니다.

이 속성은 connectivity.protocol이 docker로 설정된 경우에만 적용됩니다.

### (선택 사항) 구성.json 구성

이 config.json파일에는 IDT에 대한 구성 정보가 들어 있습니다. 일반적으로 테스트 러너는 이 파일을 수 정할 필요가 없습니다.AWSIDT에 대한 사용자 자격 증명 및 선택적으로AWS지역. 다음의 경우,AWS필요한 권한이 있는 자격 증명이 제공됩니다.AWS IoT Device Tester사용 지표를 수집 및 제출합니다.AWS. 이 기능은 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. 자세한 정보는 [IDT 사용량 지표 \(p. 415\)](#)을 참조 하십시오.

테스트 러너는 구성 할 수 있습니다.AWS다음 중 한 가지 방법으로 자격 증명을 사용합니다.

- 자격 증명 파일

IDT는 AWS CLI와 동일한 자격 증명 파일을 사용합니다. 자세한 내용은 [구성 및 자격 증명 파일](#)을 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 세션을 닫은 후에는 사용할 수 없습니다. IDT는 다음을 사용할 수 있습니다.`AWS_ACCESS_KEY_ID`과`AWS_SECRET_ACCESS_KEY`저장할 환경 변수AWS신입장

Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 `export`를 사용합니다.

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

Windows에서 이러한 변수를 설정하려면 `set`을 사용합니다.

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

를 구성하려면AWSIDT에 대한 자격 증명, 테스트 러너는auth의 단원config.json에 있는 파일을 업니다.<[device-tester-extract-location](#)>/configs/folder.

```
{  
    "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>`](#).

#### **log.location**

에 있는 log 폴더의 경로입니다. [`<device-tester-extract-##>`](#).

#### **configFiles.root**

구성 파일이 들어 있는 폴더의 경로입니다.

#### **configFiles.device**

의 경로입니다. `device.json` 파일.

#### **testPath**

테스트 제품군이 들어 있는 폴더의 경로입니다.

#### **reportPath**

IDT가 테스트 세트를 실행한 후 테스트 결과를 포함할 폴더의 경로입니다.

#### **awsRegion**

선택 사항. 이 AWS 테스트 스위트에서 사용할 리전입니다. 설정되지 않은 경우 테스트 스위트는 각 테스트 스위트에 지정된 기본 리전을 사용합니다.

#### **auth.method**

IDT가 검색하는 데 사용하는 방법 AWS 자격 증명. 지원되는 값은 다음과 같습니다. `file` 를 눌러 자격 증명 파일에서 자격 증명을 검색합니다. `environment` 환경 변수를 사용하여 자격 증명을 검색합니다.

#### **auth.credentials.profile**

자격 증명 파일에서 사용할 자격 증명 프로필입니다. 이 속성은 `auth.method`이 `file`로 설정된 경우에만 적용됩니다.

## 사용자 지정 테스트 스위트 디버깅 및 실행

이후 [필수 구성](#) (p. 402)가 설정되면 IDT가 테스트 스위트를 실행할 수 있습니다. 전체 테스트 제품군의 실행 시간은 하드웨어 및 테스트 제품군의 구성에 따라 다릅니다. 참고로, 라즈베리 파이 3B에서 전체 FreeRTOS 자격 테스트 제품군의 완료까지 약 30분이 걸립니다.

테스트 스위트를 작성할 때 IDT를 사용하여 디버그 모드에서 테스트 스위트를 실행하여 코드를 실행하기 전에 코드를 확인하거나 테스트 러너에게 제공 할 수 있습니다.

## 디버그 모드에서 IDT 실행

테스트 스위트는 IDT에 따라 장치와 상호 작용하고 컨텍스트를 제공하고 결과를 수신하므로 IDT 상호 작용 없이는 IDE에서 테스트 스위트를 디버그할 수 없습니다. 이를 위해 IDT CLI는 debug-test-suite 디버그 모드에서 IDT를 실행할 수 있는 명령입니다. 다음 명령을 실행하여 사용할 수 있는 옵션을 봅니다.debug-test-suite:

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

디버그 모드에서 IDT를 실행하면 IDT는 실제로 테스트 스위트를 시작하거나 테스트 오케스트레이터를 실행하지 않습니다. 대신 IDE와 상호 작용하여 IDE에서 실행 중인 테스트 스위트의 요청에 응답하고 로그를 콘솔에 인쇄합니다. IDT는 시간이 초과되지 않고 수동으로 중단될 때까지 종료될 때까지 기다립니다. 디버그 모드에서 IDT는 테스트 오케스트레이터를 실행하지 않으며 보고서 파일을 생성하지 않습니다. 테스트 스위트를 디버그하려면 IDT가 구성 파일에서 일반적으로 얻는 몇 가지 정보를 제공하기 위해 IDE를 사용해야 합니다. 다음 정보를 입력했는지 확인하십시오.

- 각 테스트에 대한 환경 변수 및 인수. IDT는 이 정보를 읽지 않습니다.test.json 또는 suite.json.
- 리소스 디바이스를 선택하기 위한 인수입니다. IDT는 이 정보를 읽지 않습니다.test.json.

테스트 제품군의 디버깅을 수행하려면 다음 단계를 완료합니다.

- 테스트 스위트를 실행하는 데 필요한 설정 구성 파일을 만듭니다. 예를 들어, 테스트 스위트에 device.json, resource.json, 및 user\_data.json 필요한 경우 모두 구성해야 합니다.
- 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 스위트의 요청을 기다린 다음 응답합니다. IDT는 또한 IDT 클라이언트 SDK의 사례 프로세스에 필요한 환경 변수를 생성합니다.

- IDE에서 다음을 사용하십시오.run 또는 debug 다음을 수행하도록 구성하세요.
  - IDT 생성 환경 변수의 값을 설정합니다.
  - 에서 지정한 환경 변수 또는 인수의 값을 설정합니다.test.json과 suite.json 파일.
  - 필요에 따라 중단점을 설정합니다.
- IDE에서 테스트 스위트를 실행합니다.

필요에 따라 여러 번 테스트 스위트를 디버깅하고 다시 실행할 수 있습니다. 디버그 모드에서는 IDT가 시간 초과되지 않습니다.

- 디버깅을 완료한 후 IDT를 중단하여 디버그 모드를 종료합니다.

## 테스트 실행을 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 지원되는 제품 (이 경우 FreeRTOS 버전) 과 현재 IDT 버전에 사용할 수 있는 FreeRTOS 인증 테스트 제품군 버전을 나열합니다.

#### **list-test-cases**

주어진 테스트 그룹의 테스트 케이스를 나열합니다. 다음 옵션이 지원됩니다.

- group-id. 검색할 테스트 그룹입니다. 이 옵션은 필수이며 단일 그룹을 지정해야 합니다.

#### **run-suite**

디바이스의 폴에 대해 테스트 제품군을 실행합니다. 일반적으로 사용되는 몇 가지 옵션은 다음과 같습니다.

- 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-id는 단일 그룹을 지정해야 합니다.
- pool-id. 테스트할 디바이스 폴. 테스트 실행자는 예 여려 디바이스 폴이 정의되어 있는 경우 폴을 지정해야 합니다.device.json파일.
- timeout-multiplier. 에 지정된 테스트 실행 시간 초과를 수정하도록 IDT를 구성합니다.test.json사용자 정의 승수를 사용하여 테스트용 file.
- stop-on-first-failure. 첫 번째 실패 시 실행을 중지하도록 IDT를 구성합니다. 이 옵션은 지정된 테스트 그룹을 디버깅하는 데 group-id와 함께 사용해야 합니다.
- userdata. 테스트 스위트를 실행하는 데 필요한 사용자 데이터 정보가 포함된 파일을 설정합니다. 이 작업은 경우에만 필요합니다 userdataRequired에서 true로 설정되어 있습니다(suite.json테스트 제품군의 파일입니다).

run-suite 옵션에 대한 자세한 내용은 다음 help 옵션을 사용하십시오.

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

#### **debug-test-suite**

디버그 모드로 테스트 제품군을 실행합니다. 자세한 정보는 [디버그 모드에서 IDT 실행 \(p. 410\)](#)을 참조하십시오.

## IDT 테스트 결과 및 로그 검토

이 섹션에서는 IDT가 콘솔 로그 및 테스트 보고서를 생성하는 형식에 대해 설명합니다.

### 콘솔 메시지 형식

AWS IoT Device Tester에서는 테스트 스위트를 시작할 때 콘솔에 메시지를 인쇄하기 위한 표준 형식을 사용합니다. 다음 발췌문은 IDT에 의해 생성된 콘솔 메시지의 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 구성됩니다.

**time**

기록된 이벤트에 대한 전체 ISO 8601 타임스탬프입니다.

**level**

기록된 이벤트에 대한 메시지 수준입니다. 일반적으로 로깅된 메시지 수준은 다음 중 하나입니다.  
.info,.warn 또는.error. IDT 발행.fatal 또는.panic 메시지가 예상되는 이벤트가 발생하여 일찍 종료되는 경우 메시지를 표시합니다.

**msg**

기록된 메시지입니다.

**executionId**

현재 IDT 프로세스에 대한 고유 ID 문자열입니다. 이 ID는 개별 IDT 실행을 구별하는 데 사용됩니다.

테스트 스위트에서 생성된 콘솔 메시지는 테스트 중인 장치와 테스트 세트, 테스트 그룹 및 IDT가 실행되는 테스트 케이스에 대한 추가 정보를 제공합니다. 다음 발췌문은 테스트 스위트에서 생성된 콘솔 메시지의 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 세트 특정 부분에는 다음 필드가 포함되어 있습니다.

**suiteId**

현재 실행 중인 테스트 제품군의 이름입니다.

**groupId**

현재 실행 중인 테스트 그룹의 ID입니다.

**testCaseId**

현재 실행 중인 테스트 사례의 ID입니다.

**deviceId**

현재 테스트 케이스에서 사용 중인 테스트 대상 장치의 ID입니다.

테스트 요약에는 테스트 세트, 실행된 각 그룹에 대한 테스트 결과, 생성된 로그 및 보고서 파일의 위치에 대한 정보가 포함되어 있습니다. 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 Tester보고서 스키마

`awsiotdevicetester_report.xml`는 다음 정보가 포함된 서명된 보고서입니다.

- IDT 버전
- 테스트 제품군 버전입니다.
- 보고서에 서명하는 데 사용되는 보고서 서명 및 키입니다.
- 예 지정된 디바이스 SKU 및 디바이스 폴 이름 `device.json` 파일.
- 테스트된 제품 버전 및 장치 기능
- 테스트 결과의 집계 요약 이 정보는 다음 정보에 포함된 정보와 동일합니다. `suite-name_report.xml` 파일.

```
<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. 표시된 기능 required 테스트 스위트에서 디바이스의 유효성을 검사하는 데 필요합니다. 다음 코드 조각은 `awsiotdevicetester_report.xml` 파일에 이 정보가 나타나는 방식을 보여 줍니다.

```
<feature name="ssh" value="supported" type="required"></feature>
```

로 표시된 기능 optional 은 (는) 검증에 필요하지 않습니다. 다음 코드 조각은 선택적 기능을 보여 줍니다.

```
<feature name="hs1" 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. 보고서에는 테스트 결과의 집계 요약 요약이 포함되어 있습니다.

```
<testsuites name="suite-name" results="<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>
```

두 가지 모두에 있는 보고서 섹션 `awsiotdevicetester_report.xml` 또는 `suite-name_report.xml`에는 실행된 테스트와 결과가 나열됩니다.

첫 번째 XML 태그 `<testsuites>`에는 테스트 실행의 요약이 포함됩니다. 예:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0" disabled="0">
```

### <testsuites> 태그에 사용되는 속성

#### **name**

테스트 제품군의 이름입니다.

#### **time**

테스트 세트를 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. 예:

```
<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. 예:

```
<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. 예:

```
<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

제공하는 경우 AWS 필요한 권한이 있는 자격 증명 AWS IoT Device Tester 사용 지표를 수집 및 제출합니다. AWS. 이 기능은 옵트인 기능이며 IDT 기능을 개선하는데 사용됩니다. IDT는 다음과 같은 정보를 수집합니다.

- 이AWSIDT를 실행하는 데 사용된 계정 ID
- 테스트 실행에 사용되는 IDT CLI 명령
- 실행 중인 테스트 스위트
- 예 있는 테스트 스위트<*device-tester-extract-##*>풀더
- 디바이스 폴에 구성된 디바이스 수입니다.
- 테스트 케이스 이름 및 실행 시간
- 테스트 통과 여부, 실패 여부, 오류 발생 여부 또는 건너뛰었는지 등의 테스트 결과 정보
- 테스트된 제품 특징
- 예기치 않은 종료 또는 초기 종료와 같은 IDT 종료 동작

IDT가 보내는 모든 정보도metrics.log에 있는 파일<*device-tester-extract-location*>/results/<*execution-id*>/folder. 로그 파일을 보고 테스트 실행 중에 수집된 정보를 볼 수 있습니다. 이 파일은 사용 지표를 수집하도록 선택한 경우에만 생성됩니다.

지표 수집을 비활성화하려면 추가 조치를 취할 필요가 없습니다. 간단히 저장하지 마십시오.AWS자격 증명 및 저장한 경우AWS자격 증명, 구성 안 함config.json 파일에 액세스합니다.

## AWS 자격 증명 구성

아직 을 가지고 있지 않은 경우AWS계정, 반드시[하나 만들기 \(p. 416\)](#). 이미 를 가지고 있는 경우AWS계정, 당신은 단순히[필요한 권한 구성 \(p. 416\)](#)IDT가 사용 지표를 다음으로 보낼 수 있도록 허용하는 계정의 경우AWS사용자를 대신하여.

### 1단계: AWS 계정 생성

이 단계에서는 을 생성하고 구성합니다.AWS계정. 이미 를 가지고 있는 경우AWS계정, 단원을 참조하십시오.[the section called “2단계: IDT에 대한 권한 구성” \(p. 416\)](#).

1. 열기[AWS홈 페이지](#)를 선택하고 생성AWSAccount.

#### Note

최근에 AWS에 로그인했다면 [Sign In to the Console]이 대신 표시될 수 있습니다.

2. 온라인 지시 사항을 따릅니다. 가입 절차에는 신용 카드 등록, 문자 메시지 또는 전화 수신, PIN 입력 등이 포함됩니다.

자세한 내용은 [새 Amazon Web Services 계정을 만들고 활성화하는 방법은 무엇입니까?](#)를 참조하십시오.

### 2단계: IDT에 대한 권한 구성

이 단계에서는 IDT가 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 사용하는 권한을 구성합니다. 이 AWS Management Console또는AWS Command Line Interface(AWS CLI) 용 IAM 정책 및 사용자를 생성한 다음 정책을 사용자에게 연결합니다.

- [IDT에 대한 권한 구성\(콘솔\) \(p. 416\)](#)
- [IDT에 대한 권한 구성\(AWS CLI\) \(p. 417\)](#)

#### IDT에 대한 권한을 구성하려면(콘솔)

콘솔을 사용하여 FreeRT용 IDT에 대한 권한을 구성하려면 다음 단계를 수행하십시오.

1. [IAM 콘솔](#)에 로그인합니다.

2. 특정 권한으로 역할을 생성하는 권한을 부여하는 고객 관리형 정책을 만듭니다.
  - a. 탐색 창에서 정책(Policies)을 선택한 후 정책 생성(Create policy)을 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 검토(Review policy)를 선택합니다.
- d. 이름(Name)에 **IDTUsageMetricsIAMPermissions**을 입력합니다. Summary(요약) 아래에서 정책에 의해 부여된 권한을 검토합니다.
- e. 정책 생성(Create policy)을 선택합니다.
3. IAM 사용자를 생성하고 사용자에게 권한을 연결합니다.
  - a. IAM 사용자를 생성합니다. 에서 1~5단계를 수행합니다.[IAM 사용자 생성\(콘솔\)](#)의 IAM 사용 설명서. 이미 IAM 사용자를 생성한 경우에는 다음 단계로 건너뜁니다.
  - b. IAM 사용자에게 권한을 연결합니다.
    - i. Set permissions(권한 설정) 페이지에서 Attach existing policies to user directly(사용자에게 직접 기존 정책 연결)를 선택합니다.
    - ii. 다음을 찾습니다.IDTUsageMetricsIAM 권한이전 단계에서 생성한 정책입니다. 확인란을 선택합니다.
  - c. [다음: 권한(Next: Tags)]를 선택합니다.
  - d. [다음: 권한(Next: 검토)를 선택하여 선택 사항의 요약을 봅니다.
  - e. 사용자 생성을 선택합니다.
  - f. 사용자의 액세스 키(액세스 키 ID와 보안 액세스 키)를 보려면 암호와 액세스 키 옆에 있는 Show(표시)를 선택합니다. 액세스 키를 저장하려면 Download .csv(csv 다운로드)를 선택한 후 안전한 위치에 파일을 저장합니다. 나중에 이 정보를 사용하여 AWS자격 증명 파일.

#### IDT에 대한 권한을 구성하려면(AWS CLI)

다음 단계에 따라AWS CLI를 놀러 FreeRTOS용 IDT에 대한 권한을 구성합니다.

1. 컴퓨터에 AWS CLI를 설치하고 구성합니다(아직 설치되어 있지 않은 경우). 다음 단계를 따르십시오.[다음 설치\(AWS CLI\)](#)의 AWS Command Line Interface 사용 설명서.

#### Note

이AWS CLI는 상호 작용을 할 때 사용할 수 있는 오픈 소스 도구입니다.AWS 명령줄 셸의 서비스

2. IDT 및 FreeRTOS 역할을 관리할 수 있는 권한을 부여하는 고객 관리형 정책을 만듭니다.

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 FreeRTOS에 필요한 권한을 연결합니다.

- a. IAM 사용자를 생성합니다.

```
aws iam create-user --user-name user-name
```

- b. Attach the IDTUsageMetricsIAMPermissions IAM 사용자에게 생성한 정책입니다. Replace## # ## IAM 사용자 이름과 함께 <account-id>의 ID가 포함된 명령에서 AWS 계정.

```
aws iam attach-user-policy --user-name user-name --policy-arn
arn:aws:iam::<account-id>:policy/IDTFreeRTOSIAMPermissions
```

4. 사용자에 대한 보안 액세스 키를 만듭니다.

```
aws iam create-access-key --user-name user-name
```

출력을 안전한 위치에 저장합니다. 나중에 이 정보를 사용하여 AWS 자격 증명 파일

## 를 제공하십시오 AWS IDT에 대한 자격 증명

IDT가 에 액세스하도록 허용하려면 AWS 자격 증명 및 지표 제출 AWS에서 다음을 수행합니다.

1. 저장 AWS IAM 사용자를 환경 변수로 사용하거나 자격 증명 파일에 대한 자격 증명:

- a. 환경 변수를 사용하려면 다음 명령을 실행합니다.

```
AWS_ACCESS_KEY_ID=access-key
AWS_SECRET_ACCESS_KEY=secret-access-key
```

- b. 자격 증명 파일을 사용하려면 다음 정보를 .aws/credentials file:

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

- 구성auth의 단원config.json파일. 자세한 내용은 [\(선택 사항\) 구성.json 구성 \(p. 408\)](#) 단원을 참조하세요.

## 문제 해결

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. FreeRTOS 콘솔 출력용 IDT를 사용해 실행 ID, 테스트 케이스 ID, 실패한 테스트 케이스의 테스트 그룹 ID를 찾은 뒤 라는 테스트 케이스용 로그 파일을 엽니다.results/`execution-id`/logs/`test_group_id`\_`test_case_id`.log. 이 파일에는 있는 정보는 다음과 같습니다.

- 전체 빌드 및 플래시 명령 출력.
- 테스트 실행 결과
- FreeRTOS 콘솔 출력을 위한 자세한 IDT

다음 문제 해결 워크플로우를 사용하는 것이 좋습니다.

- "###/##이 리소스에 액세스할 수 있는 권한이 없습니다."라는 오류가 표시되면 [AWS 계정 생성 및 구성 \(p. 326\)](#)에 지정된 대로 사용 권한을 구성해야 합니다.
- 실행 UUID 및 현재 실행 중인 작업과 같은 정보를 찾으려면 콘솔 출력을 읽습니다.
- FRO\_Report.xml 파일에서 각 테스트의 오류 문을 찾습니다. 이 디렉터리에는 각 테스트 그룹의 실행 로그가 포함되어 있습니다.
- 아래에 있는 로그 파일을 살펴봅니다./results/`execution-id`/logs.
- 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

FreeRTOS를 사용할 때는 바이너리를 실행하기 전에 올바른 구성 파일을 확보해야 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 FRO\_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.json의 필드가 누락되었거나 잘못된 형식입니다.	필수 필드가 누락되지 않았으며 나열된 파일에서 필요한 형식인지 확인합니다. 자세한 정보는 <a href="#">처음으로 마이크로 컨트롤러 보드의 테스트 준비 (p. 328)</a> 을 참조하십시오.
202	ValidationError	device.json, config.json 또는 userdata.json의 필드에 잘못된 값이 포함되어 있습니다.	보고서에서 오류 코드의 오른쪽에 표시되는 오류 메시지를 확인합니다. <ul style="list-style-type: none"> <li>잘못됨AWS리전 - 유효한 를 지정합니다.AWS해당 리전config.json파일에 대한 자세한 내용 AWS지역,리전 및 엔드포인트.</li> <li>잘못됨AWS자격 증명 - 유효한 설정AWS테스트 시스템의 자격 증명(환경 변수 또는 자격 증명 파일을 통해). 인증 필드가 올바르게 구성되었는지 확인합니다. 자세한 정보는 <a href="#">AWS 계정 생성 및 구성 (p. 326)</a>을 참조하십시오.</li> </ul>
203	CopySourceCodeError	FreeRTOS 소스 코드를 지정된 디렉터리에 복사할 수 없습니다.	다음 항목을 확인합니다. <ul style="list-style-type: none"> <li>userdata.json 파일에서 유효한 sourcePath가 지정되었는지 확인합니다.</li> <li>삭제하기buildFreeRTOS 소스 코드 디렉터리에서 폴더(해당 폴더가 있는 경우). 자세한 정보는 <a href="#">빌드, 플레이 및 테스트 설정 구성 (p. 332)</a>을 참조하십시오.</li> </ul>

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			<ul style="list-style-type: none"> <li>Windows에서는 파일 경로 이름에 대한 문자 제한이 있습니다. 파일 경로 이름이 길면 오류가 발생합니다.</li> </ul>
204	BuildSourceError	FreeRTOS 소스 코드를 컴파일할 수 없습니다.	<p>다음 항목을 확인합니다.</p> <ul style="list-style-type: none"> <li><code>userdata.json</code> 파일에서 <code>buildTool</code> 아래 있는 정보가 올바른지 확인합니다. 자세한 정보는 <a href="#">빌드, 플래시 및 테스트 설정 구성 (p. 332)</a>을 참조하십시오.</li> <li><code>cmake</code>를 빌드 도구로 사용하는 경우 <code>buildTool</code> 명령에서 <code>{}{enableTests}</code>가 지정되었는지 확인합니다. 자세한 정보는 <a href="#">빌드, 플래시 및 테스트 설정 구성 (p. 332)</a>을 참조하십시오.</li> <li>FreeRTOS용 IDT를 공백이 포함된 시스템의 파일 경로로 추출한 경우 <code>C:\Users\My Name\Desktop\</code>를 사용하면 경로가 제대로 구문 분석되도록 빌드 명령 안에 추가 파일 표가 필요할 수 있습니다. 플래시 명령에도 동일한 작업이 필요할 수 있습니다.</li> </ul>
205	FlashOrRunTestError	IDT FreeTO는 DUT에서 FreeTO를 플래시하거나 실행할 수 없습니다.	<code>userdata.json</code> 파일에서 <code>flashTool</code> 아래 있는 정보가 올바른지 확인합니다. 자세한 정보는 <a href="#">빌드, 플래시 및 테스트 설정 구성 (p. 332)</a> 을 참조하십시오.
206	StartEchoServerError	IDT FreeTO는 WiFi 또는 SecureSocket 테스트를 위해 에코 서버를 시작할 수 없습니다.	<code>userdata.json</code> 파일의 <code>echoServerConfiguration</code> 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 찾는데 문제가 있는 경우 IDE, Atom이나 Sublime과 같은 텍스트 편집기에서 또는 JSONLint와 같은 온라인 도구를 통해 수동으로 확인을 수행할 수 있습니다.

## 디버깅 무결성 검사 실패

FreeRTOSIntegrity 테스트 그룹을 실행하고 오류가 발생하면 먼저 다음 중 하나를 수정하지 않았는지 확인하십시오.`freertos` 디렉토리 파일, 문제가 발생하지 않았고 문제가 계속 발생하는 경우 을바른 분기를 사용하고 있는지 확인하십시오. IDT를 실행하는 경우 `list-supported-products` 명령을 사용하면 태그가 지정된 분기를 찾을 수 있습니다. `freertos` 사용해야 하는 repo입니다.

올바른 태그가 지정된 분기를 복제한 경우 `freertos` 저장소에도 여전히 문제가 있습니다. `submodule update` 명령에 대한 클론 워크플로 `freertosrepo`는 다음과 같습니다.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout -init -recursive
```

무결성 검사기에서 찾는 파일 목록은 `checksums.json`에 있는 파일 `freertos` 디렉토리입니다. 파일 및 폴더 구조를 수정하지 않고 FreeRTOS 포트를 한정하려면 '에 나열된 파일이 없는지 확인하십시오. `exhaustive`' 및 '`minimal`'의 단원 `checksums.json` 파일이 수정되었습니다. 구성된 SDK로 실행하려면 '아래에 있는 파일이 없는지 확인합니다. `minimal`' 섹션이 수정되었습니다.

SDK로 IDT를 실행하고 일부 파일을 수정한 경우 `freertos` 디렉토리에서 SDK를 올바르게 구성했는지 확인하십시오. `userdata` 파일. 그렇지 않으면 무결성 검사기가 `freertos` 디렉토리입니다.

## FullWiFi 테스트 그룹 오류 디버깅

FullWiFi 테스트 그룹에 오류가 표시되는 경우 `AFQP_WiFiConnectMultipleAP` 테스트가 실패합니다. 두 액세스 포인트가 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있지 않기 때문일 수 있습니다. 두 액세스 포인트가 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있는지 확인합니다.

## “필수 파라미터 누락” 오류 디버깅

새로운 기능이 FreeRTOS용 IDT에 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` 디렉토리 아래의 `test_group_id_test_case_id.log` 파일에 모든 누락 파라미터가 명시적으로 나열됩니다. IDT for FreeRTOS는 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 폴 이름이 실제로 존재하는지 확인합니다. 이 이름은 `device.json` 파일에서 직접 참조됩니다.
- 폴에 있는 디바이스에 올바른 구성 파라미터가 있는지 확인합니다.

## “테스트 결과의 시작을 찾을 수 없습니다” 오류 디버깅

IDT가 테스트 중인 장치에서 결과 출력을 구문 분석하려고 할 때 오류가 표시될 수 있습니다. 이 오류에는 여러 원인이 있을 수 있으므로 다음 영역이 정확한지 확인하십시오.

- 테스트 중인 디바이스가 호스트 컴퓨터에 안정적으로 연결되어 있는지 확인합니다. 로그 파일에서 이러한 오류를 보여주는 테스트를 확인하여 IDT가 수신하는 내용을 확인할 수 있습니다.
- 테스트 중인 장치가 느린 네트워크 또는 다른 인터페이스를 통해 연결되어 있거나 다른 FreeRTOS 테스트 그룹 출력과 함께 FreeRTOS 테스트 그룹 로그에 “—STARTING TESTS—” 플래그가 표시되지 않는 경우 `testStartDelayms` 사용자 데이터 데이터 구성에서 자세한 정보는 [빌드, 플래시 및 테스트 설정 구성 \(p. 332\)](#)을 참조하십시오.

## “리소스 액세스 권한 없음” 오류 디버깅

오류가 표시될 수 있습니다.”###/##터미널 출력 또는 에서 이 리소스에 액세스할 수 있는 권한이 없습니다.`test_manager.log`에 있는 파일/`results/execution-id/logs`. 이 문제를 해결하려면 AWSIoTDeviceTesterForFreeRTOSFullAccess 관리형 정책을 테스트 사용자에게 연결합니다. 자세한 정보는 [AWS 계정 생성 및 구성 \(p. 326\)](#)을 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.json`의 다른 포트를 테스트에 사용하도록 선택할 수 있습니다. 자세한 정보는 [빌드, 플래시 및 테스트 설정 구성 \(p. 332\)](#)을 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 다음 오류 메시지가 나타납니다. “테스트를 완료하는 데 60초 (url expired time) 이 소요됩니다. 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.

## 지원되는 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 유ти리티는 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/ttyUSB1`인 경우 `device.json` 파일에서 `/dev/ttyUSB1`을 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: None
- 정지 비트: 1
- 흐름 제어: None

이러한 설정은 IDT에서 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

최신버전의 Ubuntu (17.10 및 18.04)에는 glibc 코드 컴포저 스튜디오 7과 호환되지 않는 패키지입니다.x버전. 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 연결될 수 있습니다.
- 빌드 또는 플래시 프로세스 중에 로그 출력이 콘솔에 표시되지 않습니다.
- 헤드리스로 호출하더라도 빌드 명령이 GUI 모드에서 시작하려고 시도합니다.

## 로깅

FreeRTOS용 IDT는 단일 위치에 배치됩니다. 루트 IDT 디렉터리에서 다음과 같은 파일을 `results/execution-id/`에서 사용할 수 있습니다.

- `FRO_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/test_group_id__test_case_id.log`

`FRO_Report.xml` 및 `logs/test_group_id__test_case_id.log`은 검사해야 하는 가장 중요한 로그입니다. `FRO_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

일시 AWS IoT Device Tester가 실행될 때 실패가 간략한 메시지와 함께 콘솔로 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 IDT의 출력을 사용해 실행 ID, 테스트 케이스 ID, 실패한 테스트 케이스의 테스트 그룹 ID를 찾습니다. 그런 다음 이 정보를 사용하여 이름이 지정된 테스트 케이스의 로그 파일을 찾아서 열니다. `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 상세한 정보가 포함됩니다. AWS IoT Device Tester 콘솔 출력.

## S3 버킷 문제

를 누른 경우CTRL+CIDT를 실행하는 동안 IDT는 정리 프로세스를 시작합니다. 이러한 정리 중 일부는 IDT 테스트의 일부로 생성된 Amazon S3 리소스를 제거하는 것입니다. 정리를 완료할 수 없는 경우 생성된 Amazon S3 버킷이 너무 많은 문제가 발생할 수 있습니다. 즉, 다음에 IDT를 실행할 때 테스트가 실패하기 시작합니다.

를 누른 경우CTRL+CIDT를 종지하려면 이 문제를 방지하려면 정리 프로세스를 완료해야 합니다. 계정에서 수동으로 생성된 Amazon S3 버킷을 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 FRO_1.0.1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

IDT v1.7.0 and earlier

```
./devicetester_linux run-suite --suite-id FRO_1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

## 셀룰러 기능 및 AWS 요금

이 경우Cellular기능이 로 설정되어 있습니다. Yes 당신의device.JSON파일, FullSecureSockets는 테스트를 실행하기 위해 micro EC2 인스턴스를 사용하므로 추가 비용이 발생할 수 있습니다. AWS계정. 자세한 내용은 [Amazon EC2 요금](#)을 참조하세요.

## 자격 보고서 생성 정책

검증 보고서는 다음에서만 생성됩니다. AWS IoT Device Tester 지난 2년 이내에 릴리스된 FreeRTOS 버전을 지원하는 IDT 버전입니다. 지원 정책에 대해 궁금한 점이 있는 경우에 문의하십시오. [AWS Support](#).

## AWS IoT Device TesterFreeRTOS에 대한 Support 정책

AWS IoT Device TesterFreeRTOS는 검증하고 검증하는 테스트 자동화 도구입니다. 자격을 부여합니다. FreeRTOS 디바이스에 포함 시킬 수 있습니다. [AWS 파트너 디바이스 카탈로그](#). 최신 버전의 FreeRTOS 및 AWS IoT 디바이스를 테스트하고 검증하는 디바이스 테스터입니다. 우리는 지원합니다. AWS IoT 최신 버전의 FreeRTOS 및 이전 6개월 이내에 출시된 FreeRTOS 용 디바이스 테스터가 제공됩니다. 버전 AWS IoT Device Tester LTS 기반 버전을 지원하는 FreeRTOS는 2년 동안 지원됩니다. 현재, 의 버전만 AWS IoT Device TesterFreeRTOS 202012.00을 지원하는 것은 2년 동안 지원됩니다. FreeRTOS 최신 버전은에서 사용할 수 있습니다. [GitHub](#). AWS IoT 디바이스 테스터의 지원되는 버전은 지원되는 버전 AWS IoT Device Tester 용 FreeRTOS (p. 315) 단원을 참조하십시오.

IDT 프레임워크의 각 버전에서는 디바이스 검증을 위해 세 가지 버전의 테스트 제품군이 지원됩니다.

또한 의 지원되는 버전 중 하나를 사용할 수 있습니다.AWS IoT Device Tester해당 버전의 FreeRTOS 를 사용하여 디바이스를 테스트하고 검증할 수 있습니다. [FreeRTOS용 IDT의 지원되지 않는 버전 \(p. 317\)](#)은 계속 사용할 수 있지만 버그 수정 또는 업데이트가 제공되지 않습니다.

지원 정책에 대해 궁금한 점이 있는 경우 [AWS 고객 지원 센터](#)에 문의하십시오.

# AWS의 보안

AWS에서 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 매우 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 귀하의 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호합니다. AWS는 또 한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 [AWS 규정 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. AWS 서비스에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#)를 참조하십시오.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 데이터의 민감도, 조직의 요구 사항 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 AWS 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS를 구성하는 방법을 보여줍니다. 사용 방법도 알아봅니다. AWS를 모니터링하고 보호하는 데 도움이 되는 서비스 AWS 있습니다.

AWS IoT 보안에 대한 자세한 내용은 [AWS IoT의 보안 및 자격 증명](#)을 참조하십시오.

## Topics

- [AWS 리소스에 대한 자격 증명 및 액세스 관리](#) (p. 428)
- [규정 준수 검증](#) (p. 438)
- [AWS의 복원성](#) (p. 439)
- [FreeRTOS의 인프라 보안](#) (p. 439)

## AWS 리소스에 대한 자격 증명 및 액세스 관리

AWS Identity and Access Management(IAM)은 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 어떤 사용자가 AWS 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

### 주제

- [대상](#) (p. 428)
- [자격 증명을 통한 인증](#) (p. 429)
- [정책을 사용하여 액세스 관리](#) (p. 430)
- [자세히 알아보기](#) (p. 432)
- [방법 AWS IAM으로 작업하는 서비스](#) (p. 432)
- [자격 증명 기반 정책 예제](#) (p. 434)
- [자격 증명 및 액세스 문제 해결](#) (p. 436)

## 대상

AWS Identity and Access Management(IAM) 사용 방법은 AWS에서 하는 작업에 따라 달라집니다.

서비스 사용자— 사용하는 경우 AWS 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스

권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. AWS의 기능에 액세스할 수 없는 경우 [자격 증명 및 액세스 문제 해결 \(p. 436\)](#) 단원을 참조하십시오.

서비스 관리자— 담당 중인 경우 AWS 회사에서 리소스를 사용한다면 사용하는 서비스에 대한 전체 액세스 권한을 갖고 있을 것입니다. 서비스 관리자는 직원이 액세스해야 하는 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 AWS에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 [방법 AWS IAM으로 작업하는 서비스 \(p. 432\)](#) 단원을 참조하세요.

IAM 관리자 - IAM 관리자라면 AWS에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다.에 대한 자세한 내용 AWS IAM에서 사용할 수 있는 자격 증명 기반 정책은 단원을 참조하십시오.[정책 및 권한](#)의 AWS Identity and Access Management 사용 설명서.

## 자격 증명을 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. 를 사용하는 로그인에 대한 자세한 내용은 AWS Management Console를 참조하십시오.[IAM 콘솔 및 로그인 페이지](#)의 IAM 사용 설명서.

AWS 계정 계정 루트 사용자로, IAM 사용자로, 또는 IAM 역할을 수임하여 인증(AWS에 로그인)되어야 합니다. 회사의 싱글 사인온(SSO) 인증을 사용하거나 Google 또는 Facebook을 사용하여 로그인할 수도 있습니다. 이러한 경우 관리자는 이전에 IAM 역할을 사용하여 자격 증명 연동을 설정한 것입니다. 다른 회사의 자격 증명을 사용하여 AWS에 액세스하면 간접적으로 역할을 가정하는 것입니다.

[AWS Management Console](#)에 직접 로그인하려면 루트 사용자 이메일 또는 IAM 사용자 이름과 암호를 사용하세요. 루트 사용자 또는 IAM 사용자 액세스 키를 사용하여 AWS에 프로그래밍 방식으로 액세스할 수 있습니다. AWS는 자격 증명을 사용하여 암호화 방식으로 요청에 서명할 수 있는 SDK 및 명령줄 도구를 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 이렇게 하려면 인바운드 API 요청을 인증하기 위한 프로토콜인 서명 버전 4를 사용합니다. 요청 인증에 대한 자세한 내용은 AWS 일반 참조의 [Signature Version 4 서명 프로세스](#)를 참조하세요.

사용하는 인증 방법에 상관 없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS는 멀티 팩터 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 다음을 참조하세요.[에서 멀티 팩터 인증\(MFA\) 사용하기](#) AWS의 IAM 사용 설명서.

## AWS 계정 루트 사용자

AWS 계정을 처음 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 통합 인증 ID로 시작합니다. 이 자격 증명은 AWS 계정 루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업, 심지어 관리 작업의 경우에도 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 대신, [IAM 사용자를 처음 생성할 때만 루트 사용자를 사용하는 모범 사례](#)를 준수하세요. 그런 다음 루트 사용자 자격 증명을 안전하게 보관하고 몇 가지 계정 및 서비스 관리 태스크를 수행할 때만 사용합니다.

## IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. IAM 사용자에게 사용자 이름과 암호 또는 액세스 키 세트와 같은 장기 자격 증명이 있을 수 있습니다. 액세스 키를 생성하는 방법을 알아보려면 단원을 참조하십시오.[IAM 사용자의 액세스 키 관리](#)의 IAM 사용 설명서. IAM 사용자의 액세스 키를 생성할 때는 키 페어를 보고 안전하게 저장해야 합니다. 향후에 보안 액세스 키를 복구할 수 없습니다. 그 대신 새 액세스 키 페어를 생성해야 합니다.

[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 귀하는 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, 역할

은 임시 자격 증명만 제공합니다. 자세한 내용은 다음을 참조하세요.[IAM 사용자를 만들어야 하는 경우 \(역할이 아님\)](#)의 IAM 사용 설명서.

## IAM 역할

[IAM 역할](#)은 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. [역할을 전환](#)하여 AWS Management Console에서 IAM 역할을 임시로 수임할 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 단원을 참조하십시오.[IAM 역할 사용](#)의 IAM 사용 설명서.

임시 자격 증명이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 임시 IAM 사용자 권한 - IAM 사용자는 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 페더레이션 사용자 액세스 - IAM 사용자를 생성하는 대신 AWS Directory Service의 기존 자격 증명, 엔터프라이즈 사용자 딕렉터리 또는 웹 자격 증명 공급자를 사용할 수 있습니다. 이 사용자를 페더레이션 사용자라고 합니다. AWS에서는 [자격 증명 공급자](#)를 통해 액세스가 요청되면 페더레이션 사용자에게 역할을 할당합니다. 연합된 사용자에 대한 자세한 내용은 IAM 사용 설명서의 [연합된 사용자 및 역할](#)을 참조하세요.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 교차 계정 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스를 사용하면 역할을 (프록시로 사용하는 대신) 리소스에 정책을 직접 연결할 수 있습니다. 교차 계정 액세스를 위한 역할과 리소스 기반 정책 간의 차이점을 알아보려면 단원을 참조하십시오.[IAM 역할과 리소스 기반 정책의 차이](#)의 IAM 사용 설명서.
- AWS 서비스 액세스 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서에서 [AWS 서비스에 권한을 위임할 역할 생성](#) 섹션을 참조하세요.
- Amazon EC2에서 실행 중인 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 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 자격 증명을 얻을 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여](#)를 참조하세요.

[IAM 역할을 사용할지 여부를 알아보려면 단원을 참조하세요.](#)[IAM 역할을 만들어야 하는 경우 \(사용자가 아님\)](#)의 IAM 사용 설명서.

## 정책을 사용하여 액세스 관리

정책을 생성하고 IAM 자격 증명 또는 AWS 리소스에 연결하여 AWS에서 액세스를 제어합니다. 정책은 자격증명이나 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 엔터티(루트 사용자, IAM 사용자 또는 IAM 역할)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는지 여부를 결정합니다. 대부분의 정책은 AWS에 JSON 문서로서 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 내용은 단원을 참조하십시오.[JSON 정책 개요](#)의 IAM 사용 설명서.

IAM 관리자는 AWS 리소스에 액세스할 수 있는 사용자와 해당 리소스에 대해 수행할 수 있는 작업을 지정할 수 있습니다. 모든 IAM 개체(사용자 또는 역할)는 처음에는 권한이 없습니다. 다시 말해, 기본적으로 사용자는 아무 작업도 수행할 수 없으며, 자신의 암호를 변경할 수도 없습니다. 사용자에게 태스크를 수행할 권한을 부여하기 위해 관리자는 사용자에게 권한 정책을 연결해야 합니다. 또한 관리자는 의도한 권한을 가지고 있는 그룹에 사용자를 추가할 수 있습니다. 관리자가 그룹에 권한을 부여하면 그룹의 모든 사용자가 해당 권한을 받습니다.

IAM 정책은 태스크를 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 태스크를 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

## 자격 증명 기반 정책

자격 증명 기반 정책은 IAM 사용자, 역할 또는 그룹과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 단원을 참조하십시오.[IAM 정책 생성](#)의 IAM 사용 설명서.

자격 증명 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹 및 역할에게 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책을 사용할지 아니면 인라인 정책을 사용할지를 선택하는 방법은 단원을 참조하십시오.[관리형 정책과 인라인 정책의 선택](#)의 IAM 사용 설명서.

## 리소스 기반 정책

리소스 기반 정책은 Amazon S3 버킷과 같은 리소스에 연결하는 JSON 정책 문서입니다. 서비스 관리자는 이러한 정책을 사용하여 지정된 보안 주체(계정 멤버, 사용자 또는 역할)가 해당 리소스에 대해 수행할 수 있는 작업과 어떤 조건에서 수행할 수 있는지를 정의할 수 있습니다. 리소스 기반 정책은 인라인 정책입니다. 관리형 리소스 기반 정책은 없습니다.

## 액세스 제어 목록(ACL)

액세스 제어 정책(ACL)은 리소스에 액세스할 수 있는 권한을 가진 주체(계정 멤버, 사용자 또는 역할)를 제어합니다. ACL는 리소스 기반 정책과 비슷합니다. 다만 JSON 정책 문서 형식을 사용하지 않은 유일한 정책 유형입니다. Amazon S3, AWS WAF 및 Amazon VPC는 ACL을 지원하는 대표적인 서비스입니다. ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 [ACL\(액세스 제어 목록\) 개요](#)를 참조하세요.

## 기타 정책 유형

AWS는 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 유형은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 엔터티에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 자격 증명 기반 정책과 그 권한 경계의 교집합입니다. Principal 필드에서 사용자나 역할을 보안 주체로 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 내용은 단원을 참조하십시오.[IAM 개체에 대한 권한 범위](#)의 IAM 사용 설명서.
- 서비스 제어 정책(SCP) – SCP는 AWS Organizations에서 조직 또는 조직 단위(OU)에 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations는 기업이 소유하는 여러 개의 AWS 계정을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자를 비롯하여 멤버 계정의 엔터티에 대한 권한을 제한합니다. Organizations 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 [SCP 작동 방식](#)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 연합된 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 자격 증명 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 내용은 단원을 참조하십시오.[세션 정책](#)의 IAM 사용 설명서.

## 여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우 결과 권한은 이해하기가 더 복잡합니다. 자세한 방법은 다음과 같습니다. AWS 여러 정책 유형이 관련될 때 가 요청을 허용할지 여부를 결정합니다. [정책 평가 로직](#)의 IAM 사용 설명서.

## 자세히 알아보기

AWS 리소스의 자격 증명 및 액세스 관리에 대해 자세히 알아보려면 다음 페이지로 진행하십시오.

- [방법AWSIAM으로 작업하는 서비스 \(p. 432\)](#)
- [자격 증명 기반 정책 예제 \(p. 434\)](#)
- [자격 증명 및 액세스 문제 해결 \(p. 436\)](#)

## 방법AWSIAM으로 작업하는 서비스

IAM을 사용하여 액세스를 관리하기 전에 AWS 서비스를 사용하려면 어떤 IAM 기능을 사용할 수 있는지를 이해해야 합니다. 방법에 대한 상위 수준에서 살펴보려면 [AWSIAM으로 작업하는 서비스](#)의 IAM 사용 설명서입니다.

### 주제

- [자격 증명 기반 정책 \(p. 432\)](#)
- [AWS 리소스 기반 정책 \(p. 433\)](#)
- [태그 기반 권한 부여 \(p. 433\)](#)
- [IAM 역할 \(p. 434\)](#)

## 자격 증명 기반 정책

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 태스크와 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. JSON 정책에서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

### 작업

IAM 자격 증명 기반 정책의 Action 요소는 정책에 따라 허용되거나 거부되는 특정 작업에 대해 설명합니다. 일반적으로 정책 작업의 이름은 연결된 AWS API 작업의 이름과 동일합니다. 이 작업은 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에서 사용됩니다.

정책 작업은 작업 앞의 접두사를 사용합니다. 정책 문에는 Action 또는 NotAction 요소가 반드시 추가되어야 합니다. 각 서비스는 서비스에서 수행할 수 있는 작업을 설명하는 고유의 작업 세트를 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [  
    "service-prefix:action1",  
    "service-prefix:action2"]
```

와일드카드(\*)를 사용하여 여러 태스크를 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "service-prefix:Describe*"
```

목록을 보려면 [AWS동작, 참조에 대한 작업, 리소스 및 조건 키AWS서비스](#)의 IAM 사용 설명서.

### 리소스

Resource 요소는 작업이 적용되는 객체를 지정합니다. 문에는 Resource 또는 NotResource 요소가 반드시 추가되어야 합니다. ARN을 사용하거나 문이 모든 리소스에 적용됨을 표시하는 와일드카드(\*)를 사용하여 리소스를 지정합니다.

ARN 형식에 대한 자세한 내용은 [Amazon 리소스 이름\(ARN\)](#) 및 [AWS 서비스 네임스페이스](#)를 참조하세요.

특정 계정에 속하는 모든 인스턴스를 지정하려면 와일드카드(\*)를 사용합니다.

```
"Resource": "arn:aws:service-prefix:us-east-1:123456789012:resource-type/*"
```

리소스를 생성하기 위한 작업과 같은 일부 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우 와일드 카드(\*)를 사용해야 합니다.

```
"Resource": "*"
```

일부 API 작업에는 여러 리소스가 관련되므로 IAM 사용자는 모든 리소스를 사용할 수 있는 권한을 가지고 있어야 합니다. 단일 문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다.

```
"Resource": [  
    "resource1",  
    "resource2"]
```

각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 단원을 참조하십시오.[에 대한 작업, 리소스 및 조건 키AWS서비스](#).

## 조건 키

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 선택 사항입니다. 같음, 미만 등 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 빌드할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우 AWS는 논리적 AND 태스크를 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우 AWS는 논리적 OR 태스크를 사용하여 조건을 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 단원을 참조하십시오.[IAM 정책 요소: 변수와 태그](#)의 IAM 사용 설명서.

모두 보려면 AWS 전역 조건 키, 단원을 참조하십시오.[AWS 전역 조건 컨텍스트 키](#)의 IAM 사용 설명서.”

## AWS 리소스 기반 정책

리소스 기반 정책은 지정된 보안 주체가 리소스에 대해 수행할 수 있는 작업 및 관련 조건을 지정하는 JSON 정책 문서입니다. 리소스 기반 정책을 사용하여 리소스별로 다른 계정에 사용 권한을 부여할 수 있습니다.

교차 계정 액세스를 활성화하려는 경우 전체 계정이나 다른 계정의 IAM 개체를 [리소스 기반 정책의 보안 주체](#)로 지정할 수 있습니다. 리소스 기반 정책에 교차 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 다른 AWS 계정에 있는 경우 보안 주체 엔터티가 리소스에 액세스할 권한도 부여해야 합니다. 엔터티에 자격 증명 기반 정책을 연결하여 권한을 부여 합니다. 하지만 리소스 기반 정책이 동일 계정의 보안 주체에 액세스를 부여하는 경우 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 역할과 리소스 기반 정책의 차이](#)를 참조하십시오.

자세한 리소스 기반 정책 페이지의 예를 보려면 <https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html> 단원을 참조하십시오.

## 태그 기반 권한 부여

태그를 리소스에 연결하거나 요청에서 태그를 전달할 수 있습니다. 태그를 기반으로 액세스를 제어하려면 **prefix:ResourceTag/**key-name****, **aws:RequestTag/**key-name**** 또는 **aws:TagKeys** 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예제는 [태그를 기반으로 리소스 보기 \(p. 436\)](#)에서 확인할 수 있습니다.

## IAM 역할

[IAM 역할](#)은 특정 권한을 가지고 있는 AWS 계정 내 엔터티입니다.

### 임시 자격 증명 사용

임시 자격 증명을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. [AssumeRole](#) 또는 [GetFederationToken](#) 같은 AWS Security Token Service(AWS STS) API 작업을 호출하여 임시 보안 자격 증명을 가져옵니다.

### 서비스 연결 역할

[서비스 연결 역할](#)을 사용하면 AWS 제품이 다른 서비스의 리소스에 액세스하여 사용자 대신 태스크를 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

### 서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 [서비스 역할](#)을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 태스크를 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

## 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 AWS 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS CLI 또는 AWS API를 사용해 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 태스크를 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [JSON 탭에서 정책 생성](#)을 참조하세요.

### 주제

- [정책 모범 사례 \(p. 434\)](#)
- [AWS 콘솔 사용 \(p. 435\)](#)
- [사용자가 자신이 권한을 볼 수 있도록 허용 \(p. 435\)](#)
- [태그를 기반으로 리소스 보기 \(p. 436\)](#)

## 정책 모범 사례

자격 증명 기반 정책은 매우 강력합니다. 이 정책은 계정에서 사용자가 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부를 결정합니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. 자격 증명 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르십시오.

- 사용 시작하기 AWS 관리형 정책 — 사용을 시작하려면 AWS 신속한 서비스, 사용 AWS 지원에게 필요한 권한을 부여하는 관리형 정책을 제공합니다. 이 정책은 이미 계정에서 사용할 수 있으며 AWS에 의해 유지 관리 및 업데이트됩니다. 자세한 내용은 단원을 참조하십시오.[로 권한 사용 시작 AWS 관리형 정책](#)의 IAM 사용 설명서.
- 최소 권한 부여 — 사용자 지정 정책을 생성할 때는 작업을 수행하는 데 필요한 권한만 부여합니다. 최소한의 권한 조합으로 시작하여 필요에 따라 추가 권한을 부여합니다. 처음부터 권한을 많이 부여한 후 나중에

줄이는 방법보다 이 방법이 안전합니다. 자세한 내용은 IAM 사용 설명서에서 [최소 권한 부여](#)를 참조하세요.

- 민감한 작업에 대해 MFA 활성화 – 보안을 강화하기 위해 IAM 사용자가 중요한 리소스 또는 API 작업에 액세스하려면 멀티 팩터 인증(MFA)을 사용해야 합니다. 자세한 내용은 IAM 사용 설명서의 [AWS에서 멀티 팩터 인증\(MFA\) 사용](#)을 참조하세요.
- 보안 강화를 위해 정책 조건 사용 – 실제로 가능한 경우, 자격 증명 기반 정책이 리소스에 대한 액세스를 허용하는 조건을 정의합니다. 예를 들어 요청을 할 수 있는 IP 주소의 범위를 지정하도록 조건을 작성할 수 있습니다. 지정된 날짜 또는 시간 범위 내에서만 요청을 허용하거나, SSL 또는 MFA를 사용해야 하는 조건을 작성할 수도 있습니다. 자세한 내용은 단원을 참조하십시오.[IAM JSON 정책 요소: Condition](#)의 IAM 사용 설명서.

## AWS 콘솔 사용

AWS 서비스 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은 AWS 계정에서 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다. 최소 필수 권한보다 더 제한적인 자격 증명 기반 정책을 만들면 콘솔이 해당 정책에 연결된 개체(IAM 사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

해당 개체가 콘솔을 여전히 사용할 수 있도록 하려면 AWS 관리형 정책도 개체에 연결합니다. 자세한 내용은 단원을 참조하십시오.[사용자에게 권한 추가](#)의 IAM 사용 설명서.

AWS CLI 또는 AWS API만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

## 사용자가 자신이 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 자격 증명에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI나 AWS API를 사용하여 프로그래밍 방식으로 이 태스크를 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": "*"  
        }  
    ]  
}
```

```
        }
    ]  
}
```

## 태그를 기반으로 리소스 보기

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 리소스에 대한 액세스를 제어할 수 있습니다. 이 예제에서는 리소스 보기 허용하는 정책을 생성할 수 있는 방법을 보여 줍니다. 하지만 리소스 태그 Owner가 해당 사용자의 사용자 이름의 값을 가지고 있는 경우에만 권한이 부여됩니다. 이 정책은 콘솔에서 이 태스크를 완료하는 데 필요한 권한도 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListInputsInConsole",
            "Effect": "Allow",
            "Action": "prefix>ListInputs",
            "Resource": "*"
        },
        {
            "Sid": "ViewResourceIfOwner",
            "Effect": "Allow",
            "Action": "prefix>ListInputs",
            "Resource": "arn:aws:prefix:*:*:resource-name/*",
            "Condition": {
                "StringEquals": {"prefix:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

이 정책을 계정의 IAM 사용자에게 연결할 수 있습니다. richard-roe라는 사용자가 **resource-name**을 보려고 하는 경우 **resource-name**에 Owner=richard-roe 또는 owner=richard-roe 태그를 지정해야 합니다. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 태그 키 Owner는 Owner 및 owner 모두와 일치합니다. 자세한 내용은 단원을 참조하십시오.[IAM JSON 정책 요소: Condition](#)의 IAM 사용 설명서.

## 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

### 주제

- 작업을 수행할 권한이 없음 (p. 436)
- iam:PassRole를 수행하도록 인증되지 않음 (p. 437)
- 액세스 키를 보아야 합니다. (p. 437)
- 관리자이며 다른 사용자가 AWS 리소스에 액세스하도록 허용하려고 함 (p. 437)
- 내 AWS 계정 외부의 사람이 내 리소스에 액세스하도록 허용하려고 함 (p. 438)

## 작업을 수행할 권한이 없음

AWS Management Console에서 태스크를 수행할 권한이 없다는 메시지가 나타나는 경우 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 다음과 같은 경우 발생합니다. mateo@jackson:~\$ IAM 사용자가 콘솔을 사용하여 AWS에 대한 세부 정보를 보려고 합니다. # ## ### 그러나 가지고 있지 않습니다. prefix:Action 권한.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
perform: prefix:Action on resource: my-example-resource
```

이 경우 Mateo는 **prefix:Action**를 사용하여 **my-example-resource**에 액세스하는 것을 허용하도록 정책을 업데이트하라고 관리자에게 요청합니다.

## iam:PassRole를 수행하도록 인증되지 않음

iam:PassRole 태스크를 수행할 권한이 없다는 오류가 수신되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다. 역할을 서비스에 전달하는 것을 허용하도록 정책을 업데이트하라고 관리자에게 요청합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신, 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는라는 IAM 사용자가 marymajor는 콘솔을 사용하여 서비스에서 작업을 수행하려고 합니다. 하지만 태스크를 수행하려면 서비스에 서비스 역할이 부여한 권한이 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 Mary는 iam:PassRole 태스크를 수행하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

## 액세스 키를 보아야 합니다.

IAM 사용자 액세스 키를 생성한 후에는 언제든지 액세스 키 ID를 볼 수 있습니다. 하지만 보안 액세스 키는 다시 볼 수 없습니다. 보안 액세스 키를 잃어버린 경우 새로운 액세스 키 페어를 생성해야 합니다.

액세스 키는 액세스 키 ID(예: AKIAIOSFODNN7EXAMPLE)와 보안 액세스 키(예: wJalrXUtnFEMI/K7MDE.../bPxRfiCYEXAMPLEKEY)의 두 가지 부분으로 구성됩니다. 사용자 이름 및 암호와 같이 액세스 키 ID와 보안 액세스 키를 함께 사용하여 요청을 인증해야 합니다. 사용자 이름과 암호를 관리하는 것처럼 안전하게 액세스 키를 관리합니다.

### Important

[정식 사용자 ID를 찾는 데](#) 도움이 되더라도 액세스 키를 타사에 제공하지 마시기 바랍니다. 이로 인해 다른 사람에게 계정에 대한 영구 액세스를 제공하게 될 수 있습니다.

액세스 키 페어를 생성할 때는 액세스 키 ID와 보안 액세스 키를 안전한 위치에 저장하라는 메시지가 나타납니다. 보안 액세스 키는 생성할 때만 사용할 수 있습니다. 하지만 보안 액세스 키를 잃어버린 경우 새로운 액세스 키를 IAM 사용자에게 추가해야 합니다. 최대 두 개의 액세스 키를 가질 수 있습니다. 이미 두 개가 있는 경우 새로 생성하려면 먼저 키 페어 하나를 삭제해야 합니다. 지침을 보려면 단원을 참조하십시오. [액세스 키 관리](#)의 IAM 사용 설명서.

## 관리자이며 다른 사용자가 AWS 리소스에 액세스하도록 허용하려고 함

다른 사용자가 서비스에 액세스하도록 허용하려면 액세스 권한이 필요한 사용자 또는 애플리케이션에 대한 IAM 개체 (사용자 또는 역할)를 생성해야 합니다. 다른 사용자들은 해당 엔터티에 대한 자격 증명을 사용해 AWS에 액세스합니다. 그런 다음 AWS에 대한 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다.

바로 시작하려면 단원을 참조하세요. [첫 번째 IAM 위임 사용자 및 그룹 생성](#)의 IAM 사용 설명서.

## 내 AWS 계정 외부의 사람이 내 리소스에 액세스하도록 허용하려고 함

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스하는 데 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 ACL(액세스 제어 목록)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- 서비스가 이러한 기능을 지원하는지 여부를 알아보려면 [방법AWSIAM으로 작업하는 서비스 \(p. 432\)](#) 단원을 참조하십시오.
- 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 AWS 소유한 계정은 다음을 참조하십시오. [다른 IAM 사용자에 대한 액세스 권한 제공AWS소유한 계정의 IAM 사용 설명서](#).
- 서드 파티 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 AWS 계정, 참조 [액세스 권한 제공AWS타사가 소유한 계정의 IAM 사용 설명서](#).
- 자격 증명 연동을 통해 액세스 권한을 제공하는 방법을 알아보려면 단원을 참조하십시오. [외부에서 인증된 사용자에게 액세스 권한 제공\(자격 증명 연동\)](#)의 IAM 사용 설명서.
- 교차 계정 액세스를 위한 역할 및 리소스 기반 정책 사용의 차이점을 알아보려면 단원을 참조하십시오. [IAM 역할과 리소스 기반 정책의 차이](#)의 IAM 사용 설명서.

## 규정 준수 검증

서드 파티 감사자는 SOC, PCI, FedRAMP, HIPAA 등 여러 AWS 규정 준수 프로그램의 일환으로 AWS 서비스의 보안 및 규정 준수를 평가합니다.

FreeRTOS 또는 기타 여부를 알아보려면 AWS 서비스 특정 규정 준수 프로그램의 범위에 있는 경우 단원을 참조하십시오. [AWS 규정 준수 프로그램 제공 범위 내 서비스](#). 일반적인 내용은 [AWS 규정 준수 프로그램](#)을 참조하세요.

AWS Artifact를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 [AWS Artifact에서 보고서 다운로드](#)를 참조하세요.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- [보안 및 규정 준수 빠른 시작 안내서](#) – 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 AWS에서 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- [HIPAA 보안 및 규정 준수 기술 백서 설계](#) - 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 만드는 방법을 설명합니다.

### Note

모든 AWS 서비스에 HIPAA 자격이 있는 것은 아닙니다. 자세한 내용은 [HIPAA 적격 서비스 참조](#)를 참조하세요.

- [AWS 규정 준수 리소스](#) - 고객 조직이 속한 산업 및 위치에 적용될 수 있는 워크북 및 가이드 모음입니다.
- AWS Config 개발자 안내서의 [규칙을 사용하여 리소스 평가](#) – AWS Config 서비스는 내부 사례, 산업 지침 및 규제에 대한 리소스 구성의 준수 상태를 평가합니다.
- [AWS Security Hub](#) - 이 AWS 서비스는 보안 산업 표준 및 도법 사례 규정 준수 여부를 확인하는 데 도움이 되도록 AWS 내 보안 상태를 종합적으로 보여줍니다.
- [AWS Audit Manager](#) - 이 AWS 서비스는 AWS 사용을 지속해서 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 간소화할 수 있도록 지원합니다.

## AWS의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 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 기준의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하세요.

## FreeRTOS의 인프라 보안

AWS 관리형 서비스는 AWS에 설명된 글로벌 네트워크 보안 절차 [Amazon Web Services: 보안 프로세스 개요 백서](#).

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 AWS 서비스에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.0 이상을 지원해야 합니다. TLS 1.2 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 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 [AWS Security Token Service](#)(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

# FreeRTOS 사용 설명서

이러한 이전 버전의 FreeRTOS 사용 설명서는 FreeRTOS LTS (장기 지원) 릴리스에서 사용할 수 있습니다.

- [FreeRTOS 사용 설명서](#)FreeRTOS 버전 202012.00용

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.