
Amazon FreeRTOS

사용 설명서



Amazon FreeRTOS: 사용 설명서

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Amazon FreeRTOS란 무엇입니까?	1
Amazon FreeRTOS 아키텍처	1
FreeRTOS 커널	2
FreeRTOS 커널 기초	3
FreeRTOS 커널 스케줄러	3
메모리 관리	3
작업 간 조정	4
소프트웨어 타이머	7
저전력 지원	7
FreeRTOSConfig.h	7
Amazon FreeRTOS 라이브러리	8
OTA(Over-the-Air) 업데이트	8
OTA 리소스에 태그 지정	8
OTA 업데이트 사전 조건	9
OTA 자습서	21
OTA 업데이트 관리자 서비스	43
애플리케이션에 OTA 에이전트 통합	44
OTA 보안	47
OTA 문제 해결	48
Amazon FreeRTOS 소스 코드 다운로드	55
Amazon FreeRTOS 콘솔	55
Amazon FreeRTOS 콘솔	55
미리 정의된 Amazon FreeRTOS 구성	55
사용자 지정 Amazon FreeRTOS 구성	56
빠른 연결 워크플로우	57
태그 지정 구성	57
Amazon FreeRTOS 적격 하드웨어 플랫폼	58
개발 워크플로우	59
Embedded C용 AWS IoT 디바이스 SDK	59
추가 리소스	60
Amazon FreeRTOS 시작하기	61
Amazon FreeRTOS 데모 애플리케이션	61
첫 번째 단계	61
보드별 시작 안내서	61
문제 해결	61
Amazon FreeRTOS 애플리케이션 개발	61
첫 번째 단계	62
AWS 계정 및 권한 설정	62
AWS IoT에 MCU 보드 등록	63
Amazon FreeRTOS 다운로드	65
Amazon FreeRTOS 데모 구성	66
개발자 모드 키 프로비저닝	67
소개	67
옵션 #1: AWS IoT에서 프라이빗 키 가져오기	67
옵션 #2: 온보드 프라이빗 키 생성	67
문제 해결	69
일반 시작하기 문제 해결 도움말	69
터미널 에뮬레이터 설치	69
CMake와 Amazon FreeRTOS 사용	70
사전 조건	71
타사 코드 편집기 및 디버깅 도구를 사용하여 Amazon FreeRTOS 애플리케이션 개발	71
CMake를 사용하여 Amazon FreeRTOS 빌드	72
보드별 시작 안내서	76
Cypress CYW943907AEVAL1F 개발 키트	77

Cypress CYW954907AEVAL1F 개발 키트	80
Windows 시뮬레이터와 Microchip ECC608a 보안 요소	83
Espressif ESP32-DevKitC 및 ESP-WROVER-KIT	86
Infineon XMC4800 IoT 연결 키트	101
Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트	105
Marvell MW320 AWS IoT 시작 키트	110
Marvell MW322 AWS IoT 시작 키트	120
MediaTek MT7697Hx 개발 키트	131
Microchip Curiosity PIC32MZ EF	135
Nordic nRF52840-DK	138
Nuvoton NuMaker-IoT-M487	141
NXP LPC54018 IoT 모듈	147
Renesas Starter Kit+ for RX65N-2MB	150
STMicroelectronics STM32L4 Discovery Kit IoT 노드	153
Texas Instruments CC3220SF-LAUNCHXL	155
Windows Device Simulator	159
Xilinx Avnet MicroZed Industrial IoT Kit	162
Amazon FreeRTOS 라이브러리	168
Amazon FreeRTOS 이식 라이브러리	168
Amazon FreeRTOS 애플리케이션 라이브러리	180
Amazon FreeRTOS 공통 라이브러리	188
Amazon FreeRTOS 라이브러리 구성	192
공통 라이브러리	192
원자성 작업	193
선형 컨테이너	193
로깅	193
정적 메모리	194
작업 풀	194
Bluetooth Low Energy	197
개요	197
아키텍처	197
종속성 및 요구 사항	198
라이브러리 구성 파일	199
최적화	200
사용 제한	200
초기화	200
API 참조	202
사용 예	202
이식	204
Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK	206
AWS IoT Device Defender	207
개요	207
종속성 및 요구 사항	208
문제 해결	208
개발자 지원	209
사용 제한	209
초기화	210
Amazon FreeRTOS Device Defender API	210
사용 예	210
AWS IoT 디바이스 샘플	210
개요	210
종속성 및 요구 사항	210
API 참조	211
사용 예	211
AWS IoT Greengrass	212
개요	212
종속성 및 요구 사항	212

API 참조	213
사용 예	213
MQTT(v2.0.0)	214
개요	214
종속성 및 요구 사항	214
기능	214
API 참조	215
사용 예	215
MQTT(v1.0.0)	216
개요	216
종속성 및 요구 사항	216
기능	216
주요 구성	217
최적화	217
개발자 지원	218
Initialization(초기화)	218
API 참조	218
이식	219
HTTPS	219
개요	219
종속성 및 요구 사항	219
기능	219
API 참조	220
OTA 에이전트	220
개요	220
기능	220
API 참조	221
사용 예	221
이식	221
퍼블릭 키 암호화 표준(PKCS) #11	221
개요	221
기능	221
비대칭 암호화 지원	222
이식	223
보안 소켓	224
개요	224
종속성 및 요구 사항	224
기능	225
문제 해결	225
개발자 지원	225
사용 제한	225
Initialization(초기화)	225
API 참조	226
사용 예	226
이식	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	232
공통 I/O	232
Amazon FreeRTOS 데모	233

Amazon 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	242
일반 속성 서버	244
Microchip Curiosity PIC32MZEF용 부트로더	245
부트로더 상태	245
플래시 디바이스	246
애플리케이션 이미지 구조	247
이미지 헤더	247
이미지 설명자	248
이미지 트레일러	249
부트로더 구성	249
부트로더 빌드	250
AWS IoT Device Defender	250
AWS IoT Greengrass	251
OTA(Over-the-Air) 업데이트	253
Texas Instruments CC3220SF-LAUNCHXL	256
Microchip Curiosity PIC32MZEF	258
Espressif ESP32	261
HTTPS	262
개요	262
사용량 지침	263
AWS IoT 디바이스 샘플	263
보안 소켓	264
Amazon FreeRTOS용 AWS IoT 디바이스 테스터 사용	266
Amazon FreeRTOS 201912.00용 IDT v1.6.0 다운로드	267
이전 Amazon FreeRTOS용 IDT 버전	267
사전 조건	269
Amazon FreeRTOS 다운로드	269
Amazon FreeRTOS용 IDT 다운로드	269
AWS 계정 생성 및 구성	269
(선택 사항) AWS 명령줄 인터페이스(CLI) 설치	272
처음으로 마이크로 컨트롤러 보드의 테스트 준비	272
라이브러리 이식 계층 추가	272
AWS 자격 증명 구성	272
Amazon FreeRTOS용 IDT에서 디바이스 폴 만들기	272
빌드, 플래시 및 테스트 설정 구성	275
Bluetooth Low Energy 테스트 실행	280
사전 조건	280
Raspberry Pi 설정	280
Amazon FreeRTOS 디바이스 설정	282
BLE 테스트 실행	282
BLE 테스트 문제 해결	282
Amazon FreeRTOS Qualification Suite 실행	282
Amazon FreeRTOS용 IDT 명령	283
재검증을 위한 테스트	283
결과 및 로그 이해	283
결과 보기	284
문제 해결	285
디바이스 구성 문제 해결	286
제한 시간 오류 해결	291
AWS의 보안	292

ID 및 액세스 관리	292
대상	292
자격 증명을 통한 인증	293
정책을 이용한 액세스 관리	294
자세히 알아보기	296
AWS 서비스에서 IAM로 작업하는 방식	296
자격 증명 기반 정책 예제	298
문제 해결	300
규정 준수 확인	302
복원성	303
인프라 보안	303

Amazon FreeRTOS란 무엇입니까?

Amazon FreeRTOS는 연결, 보안, OTA(Over-the-Air) 업데이트용 라이브러리를 사용하여 FreeRTOS 커널을 확장하는 실시간 운영 체제입니다. 또한 Amazon FreeRTOS는 [적격 보드](#)에서 Amazon FreeRTOS 기능을 보여 주는 데모 애플리케이션을 제공합니다.

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

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

Amazon FreeRTOS 릴리스의 세 가지 유형은 메이저, 마이너 및 장기 지원(LTS)입니다. 메이저라는 명칭은 여러 라이브러리에 새로운 기능 또는 중요한 업데이트가 추가됨을 나타냅니다. 모든 릴리스는 YYYYMM.NN 형식의 날짜 기반 버전 관리를 사용합니다. 여기서

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

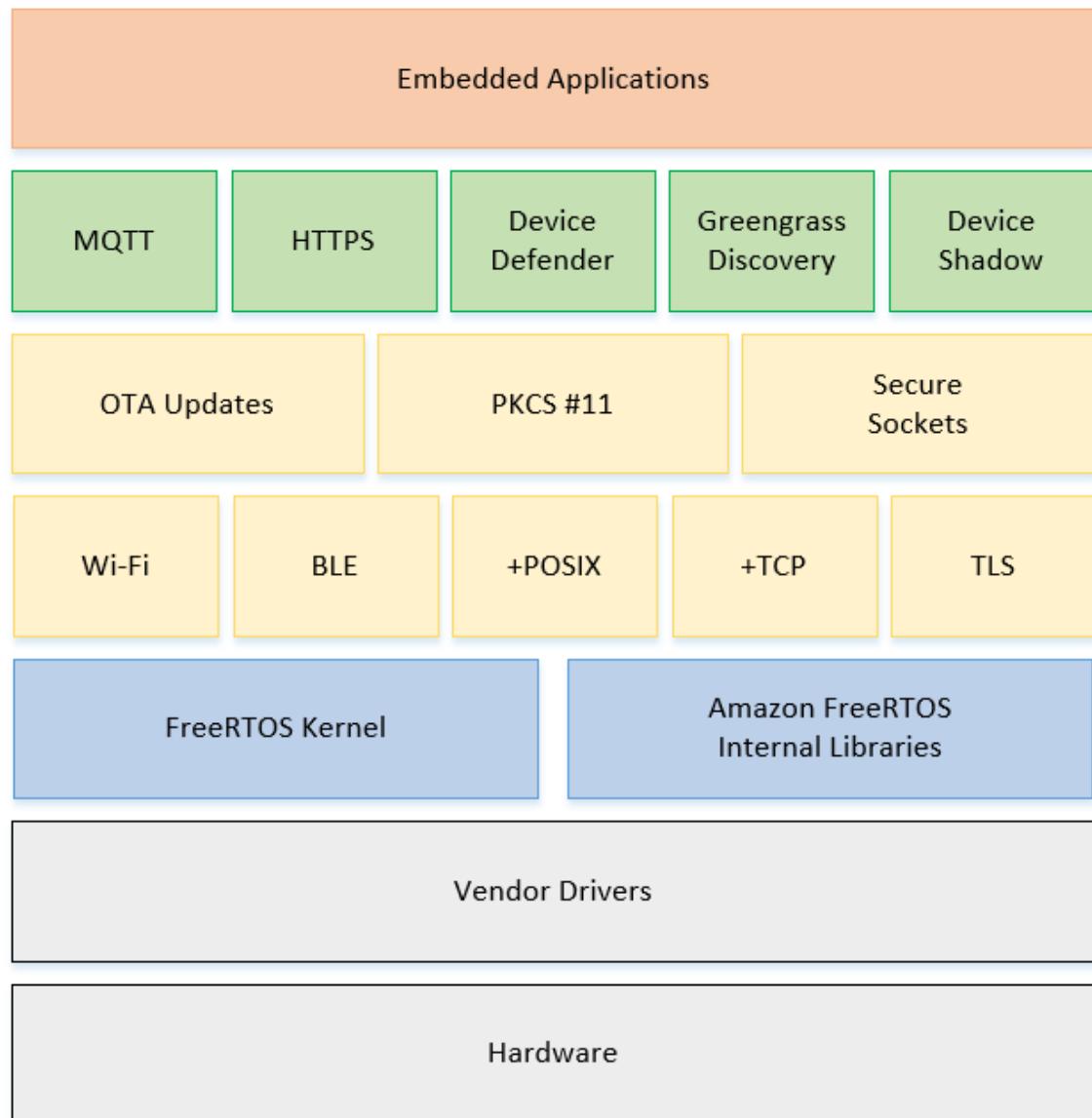
예를 들어, 2019년 6월의 두 번째 릴리스는 201906.01이 됩니다.

이전에 Amazon FreeRTOS는 의미 체계 버전 관리를 메이저 릴리스에 사용했습니다. Amazon FreeRTOS는 날짜 기반 버전 관리로 이동했지만(Amazon FreeRTOS 1.4.8에서 Amazon FreeRTOS 201906.00으로 업데이트됨), FreeRTOS 커널 및 각 개별 Amazon FreeRTOS 라이브러리는 여전히 의미 체계 버전 관리를 그대로 유지하고 있습니다. 의미 체계 버전 관리에서는 버전 번호(X.Y.Z) 자체가 릴리스가 메이저 릴리스인지, 마이너 릴리스인지, 또는 포인트 릴리스인지를 나타냅니다. 이로 인해 의미 체계 버전 관리가 개별 애플리케이션에 영향을 미치지 않는 변경 사항을 기반으로 메이저 릴리스를 표시하는 상황이 발생할 수 있습니다. 라이브러리의 의미 체계 버전을 사용하여 새 릴리스의 범위와 애플리케이션에 미치는 새 릴리스의 영향을 평가할 수 있습니다.

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

Amazon FreeRTOS 아키텍처

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



FreeRTOS 커널

FreeRTOS 커널은 다양한 아키텍처를 지원하는 실시간 운영 체제 커널이며, 임베디드 마이크로 컨트롤러 애플리케이션을 빌드하기에 적합합니다. 이 커널은 다음을 제공합니다.

- 멀티태스킹 스케줄러
- 여러 가지 메모리 할당 옵션(정적으로 할당되는 시스템 생성 가능)
- 작업 간 조정 프리미티브(작업 알림, 메시지 대기열, 각종 세마포어, 스트림 및 메시지 버퍼 등)

FreeRTOS 커널에 대한 최신 문서는 FreeRTOS.org를 참조하십시오. FreeRTOS.org에서는 [Quick Start Guide](#) 및 상세한 [Mastering the FreeRTOS Real Time Kernel](#)을 비롯하여, FreeRTOS 커널 사용에 대한 자세

한 자습서와 안내서를 다양하게 제공합니다. 이 안내서의 FreeRTOS 커널에 대한 자세한 내용은 [FreeRTOS 커널 기초 \(p. 3\)](#) 단원을 참조하십시오.

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)

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 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.

대기열

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

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

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

세마포어 및 뮤텍스

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

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

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

DTT 알림

작업 알림을 사용하면 세마포어와 같은 개별 통신 객체 없이 작업 간에 상호 작용하고, 인터럽트 서비스 루틴(ISR)과 동기화할 수 있습니다. 각 RTOS 작업에는 알림 내용(있는 경우)을 저장하는데 사용되는 32비트 알림 값이 있습니다. RTOS 작업 알림은 수신 작업을 차단 해제하고 수신 작업의 알림 값을 선택적으로 업데이트 할 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()`은 데이터를 작업 내 스트림 버퍼로 전송하는 데 사용되고, `xStreamBufferSendFromISR()`은 데이터를 인터럽트 서비스 루틴(ISR) 내 스트림 버퍼로 전송하는 데 사용됩니다.

`xStreamBufferSend()`은 블록 시간 지정을 허용합니다. `xStreamBufferSend()`이 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 \(p. 7\)](#)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()`을 구현하여 다른 CPU 코어에서 인터럽트를 생

성한 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 \(p. 7\)](#)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 스트

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

[FreeRTOSConfig.h \(p. 7\)](#)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()`을 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 \(p. 7\)](#)에서 자체 구현을 제공하여 `sbRECEIVE_COMPLETED()`의 기본 동작을 변경할 수 있습니다.

소프트웨어 타이머

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

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

저전력 지원

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

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

커널 구성

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

Amazon FreeRTOS 데모 및 테스트 애플리케이션용 FreeRTOSConfig.h 파일은 <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/FreeRTOSConfig.h 및 <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_tests/config_files/FreeRTOSConfig.h에 위치합니다.

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

Amazon FreeRTOS 라이브러리

Amazon FreeRTOS에는 다음을 위한 라이브러리가 포함되어 있습니다.

- MQTT 및 디바이스 새도우를 사용하여 AWS IoT 클라우드에 디바이스를 안전하게 연결합니다.
- AWS IoT Greengrass 코어를 검색하고 연결합니다.
- Wi-Fi 연결을 관리합니다.
- [Amazon FreeRTOS 무선\(OTA\) 업데이트 \(p. 8\)](#)를 수신하고 처리합니다.

자세한 내용은 [Amazon FreeRTOS 라이브러리](#) 항목을 참조하십시오.

Amazon FreeRTOS 무선(OTA) 업데이트

무선(OTA) 업데이트를 사용하면 플랫폼에 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 \(p. 43\)](#)에서는 디바이스에 업데이트가 사용 가능함을 알려주는 [AWS IoT 작업](#)을 생성합니다. OTA 데모 애플리케이션은 디바이스에서 실행되고 AWS IoT 작업에 대한 알림 항목을 구독하고 업데이트 메시지를 수신하는 Amazon FreeRTOS 작업을 생성합니다. 업데이트를 사용할 수 있는 경우 OTA 에이전트는 선택한 설정에 따라 HTTP 또는 MQTT 프로토콜을 사용하여 AWS IoT에 요청을 게시하고 업데이트를 수신합니다. OTA 에이전트는 다운로드한 파일의 디지털 서명을 확인하고 유효할 경우 펌웨어 업데이트를 설치합니다. Amazon FreeRTOS OTA 업데이트 데모 애플리케이션을 사용하지 않을 경우 [Amazon FreeRTOS OTA 에이전트 라이브러리 \(p. 220\)](#)를 자체 애플리케이션에 통합하여 펌웨어 업데이트 기능을 가져와야 합니다.

Amazon FreeRTOS OTA 업데이트를 사용하여 다음을 수행할 수 있습니다.

- 배포 전에 펌웨어에 디지털 방식으로 서명합니다.
- 새 펌웨어 이미지를 단일 디바이스, 디바이스 그룹 또는 전체 플랫폼에 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 등).

는 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

무선(OTA) 업데이트를 사용하려면 다음을 수행합니다.

- 업데이트를 저장할 Amazon S3 버킷 생성 ([p. 9](#))를 선택하십시오.
- OTA 업데이트 서비스 역할 생성 ([p. 9](#))를 선택하십시오.
- OTA 사용자 정책 생성 ([p. 11](#))를 선택하십시오.
- 코드 서명 인증서 생성 ([p. 12](#))를 선택하십시오.
- Code Signing for AWS IoT를 사용할 경우 [Code Signing for AWS IoT에 대한 액세스 권한 부여](#) ([p. 18](#))를 선택하십시오.
- OTA 라이브러리와 함께 Amazon FreeRTOS 다운로드 ([p. 19](#))를 선택하십시오.

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. <https://console.aws.amazon.com/s3/>에서 Amazon S3에 로그인합니다.
2. [Create bucket]을 선택합니다.
3. 버킷 이름을 입력한 후 다음을 선택합니다.
4. 모든 버전을 동일한 버킷에 유지하려면 버전 관리를 선택하고 다음을 선택합니다.
5. 다음을 선택하여 기본 권한을 수락합니다.
6. [Create bucket]을 선택합니다.

For more information about Amazon S3, see [Amazon Simple Storage Service 콘솔 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. <https://console.aws.amazon.com/iam/>에 로그인합니다.
2. 탐색 창에서 역할을 선택합니다.
3. [Create role]을 선택합니다.
4. Select type of trusted entity(신뢰할 수 있는 유형의 엔터티 선택) 아래에서 AWS 서비스를 선택합니다.
5. AWS 서비스 목록에서 IoT를 선택합니다.
6. Select your use case(사용 사례 선택) 아래에서 IoT를 선택합니다.
7. 다음: 태그를 선택합니다.
8. 다음: 검토를 선택합니다.
9. 역할 이름과 설명을 입력한 후 역할 생성을 선택합니다.

IAM 역할에 대한 자세한 내용은 [IAM 역할](#) 단원을 참조하십시오.

OTA 서비스 역할에 OTA 업데이트 권한을 추가하려면

1. IAM 콘솔 페이지의 검색 상자에 역할 이름을 입력한 후 목록에서 해당 역할을 선택합니다.
2. 정책 연결을 선택합니다.
3. 검색 상자에 "AmazonFreeRTOSOTAUpdate"를 입력하고 필터링된 정책 목록에서 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. Add inline policy(인라인 정책 추가)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. [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. Add inline policy(인라인 정책 추가)를 선택합니다.
3. [JSON] 탭을 선택합니다.
4. 다음 정책 문서를 복사하여 상자에 붙여 넣습니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObjectVersion",  
                "s3:GetObject",  
                "s3:PutObject"  
            ]  
        }  
    ]  
}
```

```
        ],
        "Resource": [
            "arn:aws:s3:::<example-bucket>/*"
        ]
    }
}
```

이 정책은 Amazon S3 객체를 읽을 수 있는 OTA 서비스 역할 권한을 부여합니다. <**example-bucket**>를 버킷 이름으로 바꿉니다.

5. [Review policy]를 선택합니다.
6. 정책 이름을 입력한 후 정책 생성을 선택합니다.

OTA 사용자 정책 생성

IAM 사용자에게 OTA 업데이트 수행 권한을 부여해야 합니다. IAM 사용자에게 다음에 대한 권한이 있어야 합니다.

- 펌웨어 업데이트를 저장할 S3 액세스
- AWS Certificate Manager에 저장된 인증서 액세스
- AWS IoT 스트리밍 서비스 액세스
- Amazon FreeRTOS OTA 업데이트 액세스
- AWS IoT 작업 액세스
- IAM에 액세스합니다.
- Code Signing for AWS IoT 액세스 [Code Signing for AWS IoT에 대한 액세스 권한 부여 \(p. 18\)](#)을(를) 참조하십시오.
- Amazon FreeRTOS 하드웨어 플랫폼 나열

IAM 사용자에게 필요한 권한을 부여하려면 OTA 사용자 정책을 생성한 다음 IAM 사용자에게 연결합니다. 자세한 내용은 [IAM 정책](#)을 참조하십시오.

OTA 사용자 정책을 생성하려면

1. <https://console.aws.amazon.com/iam/> 콘솔을 엽니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 목록에서 IAM 사용자를 선택합니다.
4. 권한 추가를 선택합니다.
5. [Attach existing policies directly]를 선택합니다.
6. [Create policy]를 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",
                "acm>ImportCertificate",
                "acm>ListCertificates",
                "acm>GetCertificate"
            ]
        }
    ]
}
```

```
        "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>"
}
]
```

<example-bucket>을 OTA 업데이트 펌웨어 이미지가 저장되는 Amazon S3 버킷의 이름으로 바꿉니다. <your-account-id>를 AWS 계정 ID로 바꿉니다. 콘솔의 오른쪽 상단에서 AWS 계정 ID를 찾을 수 있습니다. 계정 ID를 입력할 때 대시(-)를 제거합니다. <role-name>을 방금 생성한 IAM 서비스 역할의 이름으로 바꿉니다.

8. [Review policy]를 선택합니다.
9. 새 OTA 사용자 정책의 이름을 입력하고 정책 생성을 선택합니다.

IAM 사용자에 OTA 사용자 정책을 연결하려면

1. IAM 콘솔의 탐색 창에서 사용자를 선택한 다음 사용자를 선택합니다.
2. 권한 추가를 선택합니다.
3. [Attach existing policies directly]를 선택합니다.
4. 방금 생성한 OTA 사용자 정책을 검색하고 옆에 있는 확인란을 선택합니다.
5. [Next: Review]를 선택합니다.
6. 권한 추가를 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 인증 기관(CA)을 통해 인증서를 구매합니다.

플랫폼에 따라 필요한 코드 서명 인증서 유형이 다릅니다. 다음 단원에서는 서로 다른 Amazon FreeRTOS에 적격한 플랫폼에 대한 코드 서명 인증서를 생성하는 방법을 설명합니다.

주제

- Texas Instruments CC3220SF-LAUNCHXL을 위한 코드 서명 인증서 생성 (p. 13)
- Microchip Curiosity PIC32MZEF에 대한 코드 서명 인증서 생성 (p. 14)
- Espressif ESP32에 대한 코드 서명 인증서 생성 (p. 15)
- Nordic nrf52840-dk에 대한 코드 서명 인증서 생성 (p. 16)
- Amazon FreeRTOS Windows 시뮬레이터에 대한 코드 서명 인증서 생성 (p. 17)

- 사용자 지정 하드웨어에 대한 코드 서명 인증서 생성 (p. 18)

Texas Instruments CC3220SF-LAUNCHXL을 위한 코드 서명 인증서 생성

SimpleLink Wi-Fi CC3220SF Wireless Microcontroller Launchpad Development Kit는 펌웨어 코드 서명을 위한 두 가지 인증서 체인을 지원합니다.

- 프로덕션(인증서 카탈로그)

프로덕션 인증서 체인을 사용하려면 상용 코드 서명 인증서를 구매하고 [TI Uniflash 도구](#)를 사용하여 보드를 프로덕션 모드로 설정해야 합니다.

- 테스트 및 개발(인증서 실습)

실습 인증서 체인을 사용하면 자체 서명된 코드 서명 인증서를 사용하여 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 AWS Command Line Interface 사용 설명서의 [AWS CLI 설치](#)를 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)
```

SimpleLink CC3220 SDK의 인증서는 DER 형식입니다. 자체 서명된 코드 서명 인증서를 생성하려면 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 file://tisigner.crt.pem --private-key file://tisigner.key --certificate-chain file://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.

Microchip Curiosity PIC32MZEF에 대한 코드 서명 인증서 생성

Microchip Curiosity PIC32MZEF에서는 ECDSA 코드 서명 인증서로 자체 서명된 SHA256을 지원합니다.

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 참조하십시오.

- 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 genkey -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 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.

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 [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 참조하십시오.

- 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. AWS IoT용 코드 서명을 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 [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 참조하십시오.

1. 작업 디렉터리에서 다음 텍스트를 사용하여 cert_config.txt 파일을 생성합니다.
test_signer@amazon.com을 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 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.

Amazon FreeRTOS Windows 시뮬레이터에 대한 코드 서명 인증서 생성

Amazon FreeRTOS Windows 시뮬레이터에서 OTA 업데이트를 수행하려면 ECDSA P-256 키 및 SHA-256 해시와 함께 코드 서명 인증서가 필요합니다. 코드 서명 인증서가 없는 경우 다음 단계를 따라 인증서를 생성합니다.

Note

코드 서명 인증서를 생성하려면 컴퓨터에 [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 참조하십시오.

- 작업 디렉터리에서 다음 텍스트를 사용하여 cert_config.txt 파일을 생성합니다.
test_signer@amazon.com을 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
```

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.

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](#) 단원을 참조하십시오.

코드 서명 인증서를 생성한 후 AWS CLI를 사용하여 ACM으로 가져올 수 있습니다.

```
aws acm import-certificate --certificate file://code-sign.crt --private-key file://code-sign.key
```

이 명령의 출력에는 인증서에 대한 ARN이 표시됩니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

ACM에서 특정 알고리즘 및 키 크기를 사용하려면 인증서가 필요합니다. 자세한 내용은 [인증서를 가져오기 위한 사전 조건](#)을 참조하십시오. ACM에 대한 자세한 내용은 [AWS Certificate Manager로 인증서 가져오기](#)를 참조하십시오.

코드 서명 인증서 및 프라이빗 키의 내용을 복사하여 나중에 다운로드할 Amazon FreeRTOS 코드의 일부인 `aws_ota_codesigner_certificate.h` 파일에 붙여 넣고 서식을 지정해야 합니다.

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. Code Signing for Amazon FreeRTOS를 사용하려면 IAM 사용자 계정에 Code Signing for Amazon FreeRTOS에 대한 액세스 권한을 부여해야 합니다.

IAM 사용자 계정에 Code Signing for AWS IoT에 대한 권한을 부여하려면

1. <https://console.aws.amazon.com/iam/>에 로그인합니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 생성을 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. [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. [Next: Review]를 선택합니다.
12. 권한 추가를 선택합니다.

OTA 라이브러리와 함께 Amazon FreeRTOS 다운로드

Amazon FreeRTOS 콘솔 ([p. 55](#))에서 Amazon FreeRTOS를 다운로드하거나 [GitHub](#)에서 Amazon FreeRTOS를 다운로드할 수 있습니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.

콘솔에서 다운로드한 Amazon FreeRTOS 구성에 OTA 라이브러리를 포함하려면 사전 정의된 구성을 사용자 지정하거나 OTA 기능을 지원하는 플랫폼에 대해 구성을 생성하면 됩니다. Configure Amazon FreeRTOS Software(Amazon FreeRTOS 소프트웨어 구성) 구성 속성 페이지의 라이브러리에서 OTA Updates(OTA 업데이트)를 선택합니다. 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 \(\[p. 253\]\(#\)\)](#) 단원을 참조하십시오.

HTTP를 사용한 OTA 업데이트 사전 조건

이 단원에서는 HTTP를 사용하여 무선(OTA) 업데이트를 수행하기 위한 일반적인 요구 사항에 대해 설명합니다. Amazon FreeRTOS OTA는 HTTP 또는 MQTT 프로토콜을 사용하여 펌웨어 업데이트 이미지를 Amazon S3에서 디바이스로 전송할 수 있습니다. Amazon FreeRTOS에서 OTA 업데이트를 생성할 때 두 프로토콜을 모두 지정할 수 있습니다. 이 경우 각 디바이스에서 이미지를 전송하는 데 사용되는 프로토콜을 결정합니다.

Note

- 디바이스 펌웨어는 필요한 Amazon FreeRTOS 라이브러리(예: MQTT, HTTP, OTA 에이전트)를 포함해야 합니다. Amazon FreeRTOS 버전 201912.00 이상이 필요합니다.
- 펌웨어 이미지를 전송하는 데 HTTP 프로토콜을 사용할 수도 있지만 작업 실행 알림, 작업 문서, 실행 상태 업데이트 송수신 등 AWS IoT Core와의 다른 상호 작용에서 MQTT 라이브러리를 사용하기 때문에 MQTT 라이브러리가 여전히 필요합니다.
- OTA 업데이트 작업에 대해 MQTT 및 HTTP 프로토콜을 모두 지정하는 경우 각 개별 디바이스의 OTA 에이전트 소프트웨어 설정에 따라 펌웨어 이미지를 전송하는 데 사용되는 프로토콜이 결정됩니다. OTA 에이전트를 기본 MQTT 프로토콜 방법에서 HTTP 프로토콜로 변경하려면 디바이스의 Amazon FreeRTOS 소스 코드를 컴파일하는 데 사용되는 헤더 파일을 수정할 수 있습니다.

Configurations

기본적으로 \vendors\boards\board\aws_demos\config_files\aws_ota_agent_config.h 파일에서 다음 OTA 프로토콜 구성을 참조하십시오.

```
/**  
 * @brief The protocol selected for OTA control operations.  
 * This configuration parameter sets the default protocol for all the OTA control  
 * operations like requesting an OTA job, updating the job status, and so on.  
 *  
 * Note - Only MQTT is supported at this time for control operations.  
 */  
#define configENABLED_CONTROL_PROTOCOL      ( OTA_CONTROL_OVER_MQTT )  
/**  
 * @brief The protocol selected for OTA data operations.  
 * This configuration parameter sets the protocols selected for the data operations  
 * like requesting file blocks from the service.  
 *  
 * Note - Both MQTT and HTTP are supported for data transfer. This configuration parameter  
 * can be set to the following -  
 * Enable data over MQTT - ( OTA_DATA_OVER_MQTT )  
 * Enable data over HTTP - ( OTA_DATA_OVER_HTTP )  
 * Enable data over both MQTT & HTTP ( OTA_DATA_OVER_MQTT | OTA_DATA_OVER_HTTP )  
 */  
#define configENABLED_DATA_PROTOCOLS        ( OTA_DATA_OVER_MQTT )  
/**  
 * @brief The preferred protocol selected for OTA data operations.  
 *  
 * Primary data protocol will be the protocol used for downloading files if more than  
 * one protocol is selected while creating OTA job. Default primary data protocol is MQTT  
 * and the following update here switches to HTTP as primary.  
 *  
 * Note - use OTA_DATA_OVER_HTTP for HTTP as primary data protocol.  
 */  
#define configOTA_PRIMARY_DATA_PROTOCOL     ( OTA_DATA_OVER_MQTT )
```

HTTP를 통한 OTA 데이터 전송을 활성화하려면

1. configENABLED_DATA_PROTOCOLS를 OTA_DATA_OVER_HTTP로 변경합니다.
2. 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 )
```

Memory Usage

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 Amazon FreeRTOS xPortGetFreeHeapSize API를 사용하여 계산된 모든 지원되는 플랫폼의 힙 메모리 사용량 데이터입니다. OTA 라이브러리를 사용하기에 충분한 RAM이 있는지 확인해야 합니다.

Texas Instruments CC3220SF-LAUNCHXL

제어 작업(MQTT): 12KB

데이터 작업(HTTP): 10KB

Note

TI는 하드웨어에서 SSL을 수행하기 때문에 RAM을 훨씬 적게 사용하므로 mbedtls 라이브러리를 사용하지 않습니다.

Microchip Curiosity PIC32MZEF

제어 작업(MQTT): 65KB

데이터 작업(HTTP): 43KB

Espressif ESP32

제어 작업(MQTT): 65KB

데이터 작업(HTTP): 45KB

Note

ESP32의 BLE는 약 87KB의 RAM을 사용합니다. 위의 디바이스별 구성에서 언급한 모든 내용을 활성화할 수 있는 RAM이 충분하지 않습니다.

Windows 시뮬레이터

제어 작업(MQTT): 82KB

데이터 작업(HTTP): 63KB

Nordic nrf52840-dk

HTTP는 지원되지 않습니다.

OTA 자습서

이 단원에는 OTA 업데이트를 사용하여 Amazon FreeRTOS를 실행하는 디바이스에서 펌웨어를 업데이트하는 방법을 설명하는 자습서가 포함되어 있습니다. 펌웨어 이미지 외에도 OTA 업데이트를 사용하여 모든 유형의 파일을 AWS IoT에 연결된 디바이스로 보낼 수 있습니다.

AWS IoT 콘솔 또는 AWS CLI를 사용하여 OTA 업데이트를 생성할 수 있습니다. 콘솔을 사용하면 많은 작업이 자동으로 수행되므로 OTA를 가장 쉽게 시작할 수 있습니다. AWS CLI는 OTA 업데이트 작업을 자동화하

거나, 많은 수의 디바이스로 작업하거나, Amazon FreeRTOS에 적격하지 않은 디바이스를 사용 중인 경우에 유용합니다. Amazon FreeRTOS용 디바이스 검증에 대한 자세한 내용은 [Amazon FreeRTOS 파트너 웹 사이트](#)를 참조하십시오.

OTA 업데이트를 생성하려면

1. 초기 펌웨어 버전을 하나 이상의 디바이스에 배포합니다.
2. 펌웨어가 올바르게 실행되는지 확인합니다.
3. 펌웨어를 업데이트해야 하는 경우 코드를 변경하고 새 이미지를 빌드합니다.
4. 펌웨어에 수동으로 서명할 경우 서명 후 서명된 펌웨어 이미지를 Amazon S3 버킷에 업로드합니다. Code Signing for AWS IoT를 사용 중인 경우 서명되지 않은 펌웨어 이미지를 Amazon S3 버킷에 업로드합니다.
5. OTA 업데이트를 생성합니다.

OTA 업데이트를 생성할 때 이미지 전송 프로토콜(MQTT 또는 HTTP)을 지정하거나 둘 다 지정하여 디바이스가 선택할 수 있도록 합니다. 디바이스의 Amazon 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 Amazon FreeRTOS를 실행하고 있지 않은 경우 디바이스에서 실행되는 OTA 에이전트를 구현해야 합니다.

초기 펌웨어 설치

펌웨어를 업데이트하려면 OTA 에이전트 라이브러리를 사용하여 OTA 업데이트 작업을 수행하는 초기 펌웨어 버전을 설치해야 합니다. Amazon FreeRTOS를 실행하지 않을 경우 이 단계를 건너뜁니다. 대신 OTA 에이전트 구현을 디바이스에 복사해야 합니다.

주제

- [Texas Instruments CC3220SF-LAUNCHXL에 초기 펌웨어 버전 설치 \(p. 22\)](#)
- [Microchip Curiosity PIC32MZEF에 초기 펌웨어 버전 설치 \(p. 25\)](#)
- [Espressif ESP32에 초기 펌웨어 버전 설치 \(p. 27\)](#)
- [Nordic nRF52840 DK에 초기 펌웨어 버전 설치 \(p. 29\)](#)
- [Windows 시뮬레이터의 초기 펌웨어 \(p. 30\)](#)
- [사용자 지정 보드에 초기 펌웨어 버전 설치 \(p. 30\)](#)

Texas Instruments CC3220SF-LAUNCHXL에 초기 펌웨어 버전 설치

이 단계는 [Texas Instruments CC3220SF-LAUNCHXL에 대해 Amazon FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행 \(p. 256\)](#)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 재설정합니다.
2. [TI Uniflash 도구](#)를 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 <amazon-freertos>/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug 디렉터리에 위치합니다. 열기를 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 집합(위치 = 0)에 두고 보드를 재설정한 후 터미널 에뮬레이터를 다시 연결하여 Code Composer Studio에서 출력을 디버깅할 때와 출력이 동일한지 확인합니다. 터미널 출력의 애플리케이션 버전 번호를 메모해 두십시오. 나중에 이 버전 번호를 사용하여 OTA 업데이트에 의해 펌웨어가 업데이트되었는지 확인합니다.

터미널에 다음과 같은 출력이 표시됩니다.

```
0 0 [Tmr Svc] Simple Link task created
```

```
Device came up in Station mode

1 369 [Tmr Svc] Starting key provisioning...
2 369 [Tmr Svc] Write root certificate...
3 467 [Tmr Svc] Write device private key...
4 568 [Tmr Svc] Write device certificate...
SL Disconnect...

5 664 [Tmr Svc] Key provisioning done...
Device came up in Station mode

Device disconnected from the AP on an ERROR...!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 11:22:a1:b2:c3:d4

[NETAPP EVENT] IP acquired by the device

Device has connected to Guest

Device IP Address is 111.222.3.44

6 1716 [OTA] OTA demo version 0.9.0
7 1717 [OTA] Creating MQTT Client...
8 1717 [OTA] Connecting to broker...
9 1717 [OTA] Sending command to MQTT task.
10 1717 [MQTT] Received message 10000 from queue.
11 2193 [MQTT] MQTT Connect was accepted. Connection established.
12 2193 [MQTT] Notifying task.
13 2194 [OTA] Command sent to MQTT task passed.
14 2194 [OTA] Connected to broker.
15 2196 [OTA Task] Sending command to MQTT task.
16 2196 [MQTT] Received message 20000 from queue.
17 2697 [MQTT] MQTT Subscribe was accepted. Subscribed.
18 2697 [MQTT] Notifying task.
19 2698 [OTA Task] Command sent to MQTT task passed.
20 2698 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/get/
accepted

21 2699 [OTA Task] Sending command to MQTT task.
22 2699 [MQTT] Received message 30000 from queue.
23 2800 [MQTT] MQTT Subscribe was accepted. Subscribed.
24 2800 [MQTT] Notifying task.
25 2801 [OTA Task] Command sent to MQTT task passed.
26 2801 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-next

27 2814 [OTA Task] [OTA] Check For Update #0
28 2814 [OTA Task] Sending command to MQTT task.
29 2814 [MQTT] Received message 40000 from queue.
30 2916 [MQTT] MQTT Publish was successful.
31 2916 [MQTT] Notifying task.
32 2917 [OTA Task] Command sent to MQTT task passed.
33 2917 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
34 2917 [OTA Task] [OTA] Missing job parameter: execution
35 2917 [OTA Task] [OTA] Missing job parameter: jobId
36 2918 [OTA Task] [OTA] Missing job parameter: jobDocument
37 2918 [OTA Task] [OTA] Missing job parameter: ts_ota
38 2918 [OTA Task] [OTA] Missing job parameter: files
39 2918 [OTA Task] [OTA] Missing job parameter: streamname
40 2918 [OTA Task] [OTA] Missing job parameter: certfile
41 2918 [OTA Task] [OTA] Missing job parameter: filepath
42 2918 [OTA Task] [OTA] Missing job parameter: filesize
43 2919 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
44 2919 [OTA Task] [OTA] Missing job parameter: fileid
45 2919 [OTA Task] [OTA] Missing job parameter: attr
```

```
47 3919 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
48 4919 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
49 5919 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
```

Microchip Curiosity PIC32MZEF에 초기 펌웨어 버전 설치

이 단계는 Microchip Curiosity PIC32MZEF에 대해 Amazon FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행 (p. 258)에 설명된 대로 aws_demos 프로젝트를 이미 빌드했다는 가정하에 작성되었습니다.

데모 애플리케이션을 보드에 번인하려면

- aws_demos 프로젝트를 다시 빌드하고 오류 없이 컴파일되는지 확인합니다.
-  도구 모음에서 아이콘을 선택합니다.
- 프로그래밍 프로세스가 완료된 후 ICD 4 디버거를 연결 해제하고 보드를 재설정합니다. 터미널 에뮬레이터를 다시 연결하여 MPLAB X IDE에서 데모를 디버깅할 때와 출력이 동일한지 확인합니다.

터미널에 다음과 비슷한 출력이 표시됩니다.

```
Bootloader version 00.09.00
[prvBOOT_Init] Watchdog timer initialized.
[prvBOOT_Init] Crypto initialized.

[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] No application image or magic code present at: 0xbd000000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000

[prvValidateImage] Validating image at Bank : 1
[prvValidateImage] No application image or magic code present at: 0xbd100000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd100000

[prvBOOT_ValidateImages] Booting default image.

>0 36246 [IP-task] 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
```

다음 절차에서는 참조 부트로더와 암호화 서명이 있는 애플리케이션으로 구성된 통합 16진수 파일 또는 출고 시 이미지를 생성합니다. 부트로더는 부팅 시 애플리케이션의 암호화 서명을 확인하고 OTA 업데이트를 지원합니다.

출고 시 이미지를 빌드 후 플래시하려면

1. [Source Forge\(소스 Forge\)](#)에서 설치된 SRecord 도구가 있는지 확인합니다. `srec_cat` 및 `srec_info` 프로그램이 포함된 디렉터리가 시스템 경로에 있는지 확인합니다.
2. 출고 시 이미지에 대한 OTA 시퀀스 번호와 애플리케이션 버전을 업데이트합니다.
3. `aws_demos` 프로젝트를 빌드합니다.
4. `factory_image_generator.py` 스크립트를 실행하여 출고 시 이미지를 생성합니다.

```
factory_image_generator.py -b mplab.production.bin -p MCHP-Curiosity-PIC32MZEF -k
private_key.pem -x aws_bootloader.X.production.hex
```

이 명령은 다음 파라미터를 사용합니다.

- `mplab.production.bin`: 애플리케이션 이진 파일
- `MCHP-Curiosity-PIC32MZEF`: 플랫폼 이름
- `private_key.pem`: 코드 서명 프라이빗 키
- `aws_bootloader.X.production.hex`: 부트로더 16진수 파일

`aws_demos` 프로젝트를 빌드할 때 애플리케이션 이진 이미지와 부트로더 16진수 파일이 프로세스의 일부로 빌드됩니다. `vendors/microchip/boards/curiosity_pic32mzef/aws_demos/` 디렉터리 아래의 각 프로젝트에는 이러한 파일이 포함된 `dist/pic32mz_ef_curiosity/production/` 디렉터리가 있습니다. 생성된 통합 16진수 파일의 이름은 `mplab.production.unified.factory.hex`입니다.

5. MPLab IPE 도구를 사용하여 생성된 16진수 파일을 디바이스에 프로그래밍합니다.
6. 이미지가 업로드될 때 보드의 UART 출력을 보고 출고 시 이미지가 작동하는지 확인할 수 있습니다. 모든 것이 올바르게 설정되었으면 이미지 부트가 성공적으로 표시됩니다.

```
[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] Valid magic code at: 0xbd000000
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000
[prvValidateImage] Addresses are valid.
[prvValidateImage] Crypto signature is valid.
```

```
[...]  
[prvBOOT_ValidateImages] Booting image with sequence number 1 at 0xbd000000
```

7. 인증서가 잘못 구성되었거나 OTA 이미지가 제대로 서명되지 않은 경우 다음과 같은 메시지가 표시된 후 칩의 부트로더가 잘못된 업데이트를 지웁니다. 코드 서명 인증서가 일치하는지 확인하고 이전 단계를 주의해서 검토합니다.

```
[prvValidateImage] Validating image at Bank : 0  
[prvValidateImage] Valid magic code at: 0xbd000000  
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000  
[prvValidateImage] Addresses are valid.  
[prvValidateImage] Crypto signature is not valid.  
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000  
[BOOT_FLASH_EraseBank] Bank erased at : 0xbd000000
```

Espressif ESP32에 초기 펌웨어 버전 설치

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 [Amazon FreeRTOS 시작하기](#)에서 설명한 MQTT 데모 프로젝트를 실행하여 보드와 도구 체인이 올바르게 설정되어 있는지 확인할 수 있습니다.

초기 출고 시 이미지를 보드에 플래시하려면

1. <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/ aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
2. [OTA 업데이트 사전 조건 \(p. 9\)](#)에서 생성한 SHA-256/ECDSA PEM 형식 코드 서명 인증서를 demos/include/aws_ota_codesigner_certificate.h에 복사합니다. 다음 방식으로 형식을 정해야 합니다.

```
static const char signingcredentialsIGNING_CERTIFICATE_PEM[ ] = "-----BEGIN  
CERTIFICATE-----\n"  
    ...base64 data...\n"-----END CERTIFICATE-----";
```

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 clean을 실행해야 할 수 있습니다. make flash monitor를 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->0x3ffb4a8
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+I를 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 \(p. 138\)](#) 및 [무선\(OTA\) 업데이트 사전 조건](#)의 단계를 이미 수행했다는 가정하에 작성되었습니다. OTA 업데이트를 시도하기 전에 [Amazon FreeRTOS 시작하기](#)에서 설명한 MQTT 데모 프로젝트를 실행하여 보드와 도구 체인이 올바르게 설정되어 있는지 확인할 수 있습니다.

초기 출고 시 이미지를 보드에 플래시하려면

1. `<amazon-freertos>/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/ aws_demo_config.h`를 업니다.
2. `#define CONFIG_MQTT_DEMO_ENABLED`을 `#define democonfigOTA_UPDATE_DEMO_ENABLED`로 바꿉니다.
3. OTA 업데이트 데모를 선택한 상태에서 [Nordic nRF52840-DK 시작하기 \(p. 138\)](#)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. Amazon FreeRTOS 소스 코드의 구조에 대한 자세한 내용은 [Amazon FreeRTOS 데모 \(p. 233\)](#) 단원을 참조하십시오.

Amazon 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

Amazon FreeRTOS에 포함된 OTA 에이전트에서 업데이트 버전을 확인한 후 기존 펌웨어 버전보다 최신 버전이 있는 경우에만 해당 버전을 설치합니다. 다음 단계에서는 OTA 데모 애플리케이션의 펌웨어 버전을 높이는 방법을 보여 줍니다.

1. IDE에서 `aws_demos` 프로젝트를 엽니다.
2. `demos/include/aws_application_version.h`를 열고 `APP_VERSION_BUILD` 토큰 값을 늘립니다.
3. Microchip Curiosity PIC32MZEF를 사용 중인 경우 `vendors/microchip/boards/curiosity_pic32mzef/bootloader/bootloader/utility/user-config/ota-descriptor.config`에서 OTA 시퀀스 번호를 늘립니다. 새 OTA 이미지가 생성될 때마다 OTA 시퀀스 번호가 증가해야 합니다.
4. 프로젝트를 다시 빌드합니다.

[업데이트를 저장할 Amazon S3 버킷 생성 \(p. 9\)](#)에 설명된 대로 펌웨어 업데이트를 생성된 Amazon S3 버킷에 복사해야 합니다. Amazon S3에 복사할 파일 이름은 사용 중인 하드웨어 플랫폼에 따라 다릅니다.

- Texas Instruments CC3220SF-LAUNCHXL: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Microchip Curiosity PIC32MZEF: `vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/dist/pic32mz_ef_curiosity/production/mplab.production.ota.bin`
- Espressif ESP32: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

OTA 업데이트 생성(AWS IoT 콘솔)

1. AWS IoT 콘솔의 탐색 창에서 관리를 선택한 다음 작업을 선택합니다.
2. Create를 선택합니다.
3. Create an Amazon FreeRTOS Over-the-Air (OTA) update job(Amazon FreeRTOS 무선(OTA) 업데이트 작업 생성)에서 OTA 업데이트 작업 생성을 선택합니다.
4. OTA 업데이트를 단일 디바이스 또는 디바이스 그룹에 배포할 수 있습니다. 업데이트 할 디바이스 선택에서 선택을 선택합니다. 단일 디바이스를 업데이트 하려면 사물 탭을 선택합니다. 디바이스 그룹을 업데이트 하려면 사물 그룹 탭을 선택합니다.
5. 단일 디바이스를 업데이트 할 경우 디바이스와 연결된 IoT 사물 옆에 있는 확인란을 선택합니다. 디바이스 그룹을 업데이트 할 경우 디바이스와 연결된 사물 그룹 옆에 있는 확인란을 선택합니다. [Next]를 선택 합니다.
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

Amazon 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 내용은 [펌웨어 업데이트에 디지털 방식으로 서명 \(p. 33\)](#)를 참조하십시오.

- b. 코드 서명 인증서에서 선택을 선택하여 이전에 가져온 인증서를 선택하거나 가져오기를 선택하여 새 인증서를 가져옵니다.
- c. 디바이스에 있는 코드 서명 인증서의 경로 이름 아래에 디바이스에 있는 코드 서명 인증서의 정규화된 경로 이름을 입력합니다. 인증서 위치는 플랫폼마다 다릅니다. [초기 펌웨어 설치 \(p. 22\)](#)의 지침을 수행할 때 코드 서명 인증서를 배치한 위치입니다.

Important

Texas Instruments CC3220SF-LAUNCHXL에서 코드 서명 인증서가 파일 시스템의 루트에 있는 경우 파일 이름 앞에 슬래시(/)를 포함하지 마십시오. 그렇지 않으면 인증 중에 file not found 오류가 발생하고 OTA 업데이트가 실패합니다.

10. S3에서 펌웨어 이미지 선택 또는 업로드에서 선택을 선택합니다. Amazon S3 버킷 목록이 표시됩니다. 펌웨어 업데이트가 포함된 버킷을 선택한 후 버킷에서 펌웨어 업데이트를 선택합니다.

Note

Microchip Curiosity PIC32MZEF 데모 프로젝트에서는 기본 이름 `mplab.production.bin` 및 `mplab.production.ota.bin`으로 두 이진 이미지를 생성합니다. OTA 업데이트를 위해 이미지를 업로드할 때 두 번째 파일을 사용합니다.

11. 디바이스에 있는 펌웨어 이미지의 경로 이름에 펌웨어 이미지를 디바이스에 복사할 위치의 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. OTA 업데이트 작업의 IAM 역할에서 S3 버킷에 액세스할 수 있고 다음 정책을 가진 역할을 선택합니다.

- AWSIoTThingsRegistration
 - AmazonFreeRTOSOTAUpdate
13. [Next]를 선택합니다.
 14. 작업 유형에서 작업이 선택한 디바이스/그룹에 배포된 후 완료됩니다(스냅샷)를 선택합니다.
 15. OTA 업데이트 작업의 ID와 설명을 입력한 후 생성을 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 펌웨어 이미지를 디바이스에 복사할 위치의 정규화된 경로 이름을 입력합니다. 이 위치는 플랫폼마다 다를 수 있습니다.
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 이름 아래에 디바이스에 있는 코드 서명 인증서의 정규화된 경로 이름을 입력합니다. 이 경로 이름은 플랫폼마다 다를 수 있습니다.
3. 디바이스에 있는 펌웨어 이미지의 경로 이름에 펌웨어 이미지를 디바이스에 복사할 위치의 정규화된 경로 이름을 입력합니다. 이 경로 이름은 플랫폼마다 다를 수 있습니다.
4. 서명 아래에 PEM 형식 서명을 붙여 넣습니다.
5. 원래 해시 알고리즘에서 파일 서명을 생성할 때 사용한 해시 알고리즘을 선택합니다.
6. 원래 암호화 알고리즘에서 파일 서명을 생성할 때 사용한 알고리즘을 선택합니다.
7. Amazon S3에서 펌웨어 이미지 선택에서 Amazon S3 버킷과 Amazon S3 버킷에 있는 서명된 펌웨어 이미지를 선택합니다.

코드 서명 정보를 지정한 후 OTA 업데이트 작업 유형, 서비스 역할 및 업데이트 ID를 지정합니다.

Note

OTA 업데이트에 대한 작업 ID에 개인 식별 정보(OTA)를 사용하지 마십시오. 개인 식별 정보의 예는 다음과 같습니다.

- 이름
- IP 주소
- 이메일 주소
- 위치
- 은행 세부 정보
- 의료 정보

1. 작업 유형에서 작업이 선택한 디바이스/그룹에 배포된 후 완료됩니다(스냅샷)를 선택합니다.
2. OTA 업데이트 작업의 IAM 역할에서 OTA 서비스 역할을 선택합니다.
3. 작업에 대한 영수증 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 방법에 대한 자습서는 [OTA\(Over-the-Air\) 업데이트 데모 애플리케이션 \(p. 253\)](#) 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 참조하십시오. AWS IoT용 코드 서명에서 지원되지 않는 암호화 알고리즘을 사용하려는 경우 펌웨어 이진 파일을 Amazon S3에 업로드하기 전에 이 파일에 서명해야 합니다.

AWS IoT용 코드 서명을 사용하여 펌웨어 이미지에 서명

AWS IoT용 코드 서명을 사용하여 펌웨어 이미지에 서명하려면 [AWS SDK 또는 명령줄 도구](#) 중 하나를 사용하면 됩니다. AWS IoT용 코드 서명에 대한 자세한 내용은 [AWS IoT용 코드 서명](#)을 참조하십시오.

코드 서명 도구를 설치하여 구성한 이후에 서명되지 않은 펌웨어 이미지를 Amazon S3 버킷에 복사하고 다음 CLI 명령을 사용하여 코드 서명 작업을 시작합니다. put-signing-profile 명령은 재사용 가능한 코드 서명 프로필을 생성합니다. start-signing-job 명령은 서명 작업을 시작합니다.

```
aws signer put-signing-profile \
--profile-name <your_profile_name> \
--signing-material certificateArn=arn:aws:acm::<your-region>:<your-aws-account-id>:certificate/<your-certificate-id> \
--platform <your-hardware-platform> \
--signing-parameters certname=<your_certificate_path_on_device>
```

```
aws signer start-signing-job \
--source \
's3={bucketName=<your_s3_bucket>,key=<your_s3_object_key>,version=<your_s3_object_version_id>}' \
--destination 's3={bucketName=<your_destination_bucket>}' \
--profile-name <your_profile_name>
```

Note

<your-source-bucket-name>과 <your-destination-bucket-name>은 동일한 Amazon S3 버킷일 수 있습니다.

다음은 put-signing-profile 및 start-signing-job 명령의 파라미터입니다.

source

S3 버킷에서 서명되지 않은 펌웨어 위치를 지정합니다.

- **bucketName**: S3 버킷의 이름
- **key**: S3 버킷의 펌웨어 키(파일 이름)
- **version**: S3 버킷에 있는 펌웨어의 S3 버전 펌웨어 버전과 다릅니다. Amazon S3 콘솔로 이동한 후 버킷을 선택하고 페이지 상단의 버전 옆에 있는 표시를 선택하여 찾을 수 있습니다.

destination

S3 버킷에 있는 서명된 펌웨어의 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 **platformId**입니다.

사용 가능한 플랫폼 및 **platformId** 값 목록을 반환하려면 `aws signer list-signing-platforms` 명령을 사용합니다.

서명 작업이 시작되고 서명된 펌웨어 이미지를 대상 Amazon S3 버킷에 쓩니다. 서명된 펌웨어 이미지의 파일 이름은 GUID입니다. 스트림을 생성할 때 이 파일 이름이 필요합니다. Amazon S3 콘솔로 이동한 후 버킷을 선택하여 파일 이름을 찾을 수 있습니다. GUID 파일 이름을 가진 파일이 표시되지 않는 경우 브라우저를 새로 고칩니다.

이 명령은 작업 ARN 및 작업 ID를 표시합니다. 이러한 값은 나중에 필요합니다. 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 S3의 여러 위치에 저장된 여러 데이터를 사용하여 OTA 업데이트를 수행 할 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. 버킷 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.

create-stream 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>
```

다음은 create-stream CLI 명령의 인수입니다.

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`

Amazon S3 버킷에 대한 액세스 권한을 부여하는 IAM 역할입니다.

서명된 펌웨어 이미지의 Amazon S3 객체 키를 찾으려면 `aws signer describe-signing-job --job-id <my-job-id>` 명령을 사용합니다. 여기서 `my-job-id`는 `create-signing-job` CLI 명령에 의해 표시되는 작업 ID입니다. `describe-signing-job` 명령의 출력에는 서명된 펌웨어 이미지의 키가 포함됩니다.

```
... text deleted for brevity ...  
"signedObject": {  
  "s3": {  
    "bucketName": "ota-bucket",  
    "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"  
  }  
}  
... text deleted for brevity ...
```

OTA 업데이트 생성

`create-ota-update` CLI 명령을 사용하여 OTA 업데이트 작업을 생성합니다.

```
aws iot create-ota-update \  
  --ota-update-id "<my_ota_update>" \  
  --target-selection SNAPSHOT \  
  --protocols "[MQTT, HTTP]" \  
  --description "<a cli ota update>" \  
  --files file://<ota.json> \  
  --targets arn:aws:iot:<your-aws-region>:<your-aws-account>:thing/<your-thing-name> \  
  --role-arn arn:aws:iam::<your-aws-account>:role/<your-ota-service-role>
```

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

`ota-update-id`

임의 OTA 업데이트 ID입니다.

target-selection

유효한 값은 다음과 같습니다.

- **SNAPSHOT**: 이 작업은 선택한 AWS IoT 사물 또는 그룹에 업데이트를 배포한 후 종료됩니다.
- **CONTINUOUS**: 이 작업은 선택한 그룹에 추가된 모든 디바이스에 업데이트를 계속 배포합니다.

protocols

펌웨어 이미지 전송을 지원하는 프로토콜 목록입니다. 다음 값을 지정할 수 있습니다.

- **MQTT**: 펌웨어 이미지가 MQTT 프로토콜을 사용하여 디바이스로 전송됩니다.
- **HTTP**: 펌웨어 이미지가 HTTP 프로토콜을 사용하여 디바이스로 전송됩니다. 펌웨어 이미지는 Amazon S3에서 호스팅되어야 합니다.
- **MQTT 및 HTTP**: 펌웨어 이미지는 MQTT 또는 HTTP 프로토콜을 사용하여 디바이스로 전송할 수 있습니다. 디바이스는 사용할 프로토콜을 결정할 수 있습니다.

description

OTA 업데이트에 대한 텍스트 설명입니다.

files

OTA 업데이트에 대한 데이터를 포함하는 JSON 파일에 대한 하나 이상의 참조입니다. JSON 파일은 다음 속성을 포함할 수 있습니다.

- **fileName**: 정규화된 펌웨어 이미지 파일 이름입니다. Texas Instruments CC3220SF-LAUNCHXL의 경우 이 이름은 /sys/mcuflashimg.bin입니다. Microchip의 경우 이 이름은 mplab.production.bin입니다.
- **fileLocation**: 펌웨어 업데이트에 대한 정보를 포함합니다.
 - **stream**: 펌웨어 업데이트를 포함하는 스트림입니다.
 - **streamId**: create-stream CLI 명령에 지정된 스트림 ID입니다.
 - **fileId**: create-stream에 전달된 JSON 파일에 지정된 파일 ID입니다.
 - **s3Location**: 펌웨어 업데이트의 Amazon S3 내 위치입니다.
 - **bucket**: 펌웨어 업데이트를 포함하는 Amazon S3 버킷입니다.
 - **key**: 펌웨어 업데이트 키입니다.
 - **version**: 펌웨어 업데이트 버전입니다.
- **codeSigning**: 코드 서명 작업에 대한 정보를 포함합니다.
 - **awsSignerJobId**: start-signing-job 명령에 따라 생성되는 서명자 작업 ID입니다.
 - **startSigningJobParamater**: 코드 서명 작업을 시작하는 데 필요한 정보입니다.
 - **signingProfileParameter**: 서명 작업 프로필을 생성하는 데 필요한 정보입니다.
 - **certificateArn**: 코드 서명 작업을 생성하는 데 사용되는 인증서의 ACM ARN입니다.
 - **platformId**: 사용 중인 하드웨어 플랫폼의 ID입니다.
 - **certificatePathOnDevice**: 디바이스에 있는 인증서의 경로입니다.
 - **signingProfileName**: 코드 서명 프로필 이름입니다. 이 이름을 가진 프로필이 없는 경우 signingProfileParameter에 대한 값을 제공해야 합니다. 지정된 이름을 가진 프로필이 있고 signingProfileParameter에 대한 값을 제공한 경우 해당 값이 서명 프로필에 사용된 값과 정확히 일치해야 합니다.
 - **destination**: 서명된 아티팩트가 배치되는 위치입니다.
 - **s3Destination**: 서명된 아티팩트가 배치되는 Amazon S3 버킷입니다.
 - **bucket**: Amazon S3 버킷입니다.
 - **prefix**: 코드 서명 아티팩트의 접두사입니다. 기본 이름은 signedImage/입니다. 사용자 폴더 아래에 signedImage 폴더가 생성됩니다.
 - **customCodeSigning**: 사용자 지정 서명에 대한 정보를 포함합니다.
 - **signature**: 사용자 지정 서명을 포함합니다.

- `inlineDocument`: 사용자 지정 서명입니다.
- `certificateChain`: 사용자 지정 서명에 대한 인증서 체인을 포함합니다.
 - `certificateName`: 디바이스에 있는 코드 서명 인증서의 경로 이름입니다.
 - `inlineDocument`: 인증서 체인입니다.
- `hashAlgorithm`: 서명을 생성하는 데 사용된 해시 알고리즘입니다.
- `signatureAlgorithm`: 코드 서명에 사용되는 서명 알고리즘입니다.
- `attributes`: 임의 키/값 페어입니다.

`targets`

OTA 업데이트가 업데이트 할 디바이스를 지정하는 하나 이상의 AWS IoT 사물 ARN입니다.

`role-arn`

서비스 역할의 ARN입니다.

다음은 AWS IoT용 코드 서명을 사용하는 `create-ota-update` 명령에 전달되는 JSON 파일의 예입니다.

```
[  
 {  
   "fileName": "firmware.bin",  
   "fileLocation": {  
     "stream": {  
       "streamId": "004",  
       "fileId":123  
     }  
   },  
   "codeSigning": {  
     "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"  
   }  
 }  
 ]
```

다음은 인라인 파일을 사용하여 사용자 지정 코드 서명 자료를 제공하는 `create-ota-update` CLI 명령에 전달되는 JSON 파일의 예입니다.

```
[  
 {  
   "fileName": "firmware.bin",  
   "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 Amazon FreeRTOS OTA에서 코드 서명 작업을 시작하고 코드 서명 프로필 및 스트림을 생성하는 데 사용되는 create-ota-update CLI 명령에 전달되는 JSON 파일의 예입니다.

```
[  
  {  
    "fileName": "<your_firmware_path_on_device>",  
    "fileVersion": "1",  
    "fileLocation": {  
      "s3Location": {  
        "bucket": "<your_bucket_name>",  
        "key": "<your_object_key>",  
        "version": "<your_S3_object_version>"  
      }  
    },  
    "codeSigning":{  
      "startSigningJobParameter":{  
        "signingProfileName": "myTestProfile",  
        "signingProfileParameter": {  
          "certificateArn": "<your_certificate_arn>",  
          "platform": "<your_platform_id>",  
          "certificatePathOnDevice": "<certificate_path>"  
        },  
        "destination": {  
          "s3Destination": {  
            "bucket": "<your_destination_bucket>"  
          }  
        }  
      }  
    }  
  }  
]
```

다음은 기존 프로필로 코드 서명 작업을 시작하고 지정된 스트림을 사용하는 OTA 업데이트를 생성하는 create-ota-update CLI 명령에 전달되는 JSON 파일의 예입니다.

```
[  
  {  
    "fileName": "<your_firmware_path_on_device>",  
    "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 Amazon FreeRTOS OTA에서 기존 코드 서명 작업 ID로 스트림을 생성하는 데 사용되는 create-ota-update CLI 명령에 전달되는 JSON 파일의 예입니다.

```
[
```

```
[  
  {  
    "fileName": "<your_firmware_path_on_device>",  
    "fileVersion": "1",  
    "codeSigning":{  
      "awsSignerJobId": "<your_signer_job_id>"  
    }  
  }  
]
```

다음은 OTA 업데이트를 생성하는 create-ota-update CLI 명령에 전달되는 JSON 파일의 예입니다. 업데이트는 지정된 S3 객체에서 스트림을 생성하고 사용자 지정 코드 서명을 사용합니다.

```
[  
  {  
    "fileName": "<your_firmware_path_on_device>",  
    "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 CLI 명령을 사용하여 모든 OTA 업데이트의 목록을 가져올 수 있습니다.

```
aws iot list-ota-updates
```

list-ota-updates 명령의 출력은 다음과 같습니다.

```
{  
  "otaUpdates": [  
    {  
      "otaUpdateId": "my_ota_update2",  
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update2",  
      "creationDate": 1522778769.042  
    },  
    {  
      "otaUpdateId": "my_ota_update1",  
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update1",  
      "creationDate": 1522775938.956  
    },  
    {  
      "otaUpdateId": "my_ota_update",  
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update",  
      "creationDate": 1522775938.956  
    }  
  ]  
}
```

```
        "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update",
        "creationDate": 1522775151.031
    }
}
```

OTA 업데이트에 대한 정보 가져오기

get-ota-update CLI 명령을 사용하여 OTA 업데이트의 생성 또는 삭제 상태를 확인할 수 있습니다.

```
aws iot get-ota-update --ota-update-id <your-ota-update-id>
```

get-ota-update 명령에 대한 출력은 다음과 같습니다.

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota-update-001",
        "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/ota-update-001",
        "creationDate": 1575414146.286,
        "lastModifiedDate": 1575414149.091,
        "targets": [
            "arn:aws:iot:us-west-2:123456789012:thing/myDevice"
        ],
        "protocols": [ "HTTP" ],
        "awsJobExecutionsRolloutConfig": {
            "maximumPerMinute": 0
        },
        "awsJobPresignedUrlConfig": {
            "expiresInSec": 1800
        },
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
                "fileName": "my_firmware.bin",
                "fileLocation": {
                    "s3Location": {
                        "bucket": "my-bucket",
                        "key": "my_firmware.bin",
                        "version": "AvP3bfJC9gyqnwoxPHuTqM5GWENT4iii"
                    }
                },
                "codeSigning": {
                    "awsSignerJobId": "b7a55a54-fae5-4d3a-b589-97ed103737c2",
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "my-profile-name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "some-ota-bucket",
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_COMPLETE",
        "awsIotJobId": "AFR_OTA-ota-update-001",
        "awsIotJobArn": "arn:aws:iot:us-west-2:123456789012:job/AFR_OTA-ota-update-001"
    }
}
```

`otaUpdateStatus`에 대해 반환되는 값은 다음과 같습니다.

`CREATE_PENDING`

OTA 업데이트 생성이 보류 중입니다.

`CREATE_IN_PROGRESS`

OTA 업데이트가 생성되고 있습니다.

`CREATE_COMPLETE`

OTA 업데이트가 생성되었습니다.

`CREATE_FAILED`

OTA 업데이트를 생성하지 못했습니다.

`DELETE_IN_PROGRESS`

OTA 업데이트를 삭제하고 있습니다.

`DELETE_FAILED`

OTA 업데이트를 삭제하지 못했습니다.

Note

OTA 업데이트가 생성된 후 실행 상태를 가져오려면 `describe-job-execution` 명령을 사용해야 합니다. 자세한 내용은 [작업 실행 설명](#) 단원을 참조하십시오.

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. 다음 예제와 같이 `delete-stream` 명령을 사용하여 이 스트림을 삭제할 수 있습니다.

```
aws iot delete-stream --stream-id <your_stream_id>
```

OTA 업데이트 삭제

OTA 업데이트를 생성할 때 다음과 같은 항목이 생성됩니다.

- OTA 업데이트 작업 데이터베이스 내 항목
- 업데이트를 수행할 AWS IoT 작업
- 업데이트 종인 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

OTA 업데이트를 생성하면 Amazon FreeRTOS에서 AWS IoT 작업을 생성합니다. 또한 작업을 처리하는 각 디바이스에 대해 작업 실행이 생성됩니다. delete-job CLI 명령을 사용하여 작업 및 관련 작업 실행을 삭제할 수 있습니다.

```
aws iot delete-job --job-id <your-job-id> --no-force
```

no-force 파라미터는 종료 상태(완료됨 또는 취소됨)인 작업만 삭제할 수 있도록 지정합니다. force 파라미터를 전달하여 종료 상태가 아닌 작업을 삭제할 수 있습니다. 자세한 내용은 [DeleteJob API](#) 단원을 참조하십시오.

Note

IN_PROGRESS 상태인 작업을 삭제하면 디바이스에서 IN_PROGRESS 상태인 작업 실행이 중단되고 디바이스가 비결정적 상태로 유지될 수 있습니다. 삭제된 작업을 실행하는 각 디바이스를 알려진 상태로 복구할 수 있는지 확인합니다.

작업에 대해 생성된 작업 실행의 수와 기타 요소에 따라 작업을 삭제하는 데 몇 분 정도 걸릴 수 있습니다. 작업이 삭제되는 동안 상태는 DELETION_IN_PROGRESS입니다. 상태가 이미 DELETION_IN_PROGRESS인 작업을 삭제하거나 취소하려고 하면 오류가 발생합니다.

delete-job-execution을 사용하여 작업 실행을 삭제할 수 있습니다. 일부 디바이스에서 작업을 처리할 수 없는 경우에 작업 실행을 삭제하려고 할 수 있습니다. 이렇게 하면 다음 예제와 같이 단일 디바이스에 대한 작업 실행이 삭제됩니다.

```
aws iot delete-job-execution --job-id <your-job-id> --thing-name <your-thing-name> --execution-number <your-job-execution-number> --no-force
```

delete-job CLI 명령과 마찬가지로 --force 파라미터를 delete-job-execution에 전달하여 작업 실행을 강제로 삭제할 수 있습니다. 자세한 내용은 [DeleteJobExecution API](#) 단원을 참조하십시오.

Note

IN_PROGRESS 상태인 작업 실행을 삭제하면 디바이스에서 IN_PROGRESS 상태인 작업 실행이 중단되고 디바이스가 비결정적 상태로 유지될 수 있습니다. 삭제된 작업을 실행하는 각 디바이스를 알려진 상태로 복구할 수 있는지 확인합니다.

OTA 업데이트 데모 애플리케이션을 사용하는 방법에 대한 자세한 내용은 [OTA\(Over-the-Air\) 업데이트 데모 애플리케이션 \(p. 253\)](#) 단원을 참조하십시오.

OTA 업데이트 관리자 서비스

무선(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.
- AWS 계정과 연결된 모든 OTA 업데이트를 나열합니다.
- OTA 업데이트를 삭제합니다.

OTA 업데이트는 OTA 업데이트 관리자 서비스에 의해 유지보수되는 데이터 구조입니다. OTA 업데이트는 다음을 포함합니다.

- OTA 업데이트 ID
- OTA 업데이트 설명(선택 사항)
- 업데이트할 디바이스 목록(대상).
- OTA 업데이트 유형: 연속 또는 스냅샷 필요한 업데이트 유형에 대한 자세한 내용은 AWS IoT 개발자 안내서의 [작업 단원](#)을 참조하십시오.
- OTA 업데이트를 수행하는 데 사용되는 프로토콜은 [MQTT], [HTTP] 또는 [MQTT, HTTP]입니다. MQTT 및 HTTP를 지정하면 디바이스 설정에 따라 사용되는 프로토콜이 결정됩니다.
- 대상 디바이스로 전송할 파일 목록
- AWS IoT 작업 서비스에 액세스할 수 있는 IAM 역할
- 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 OTA 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. [Amazon FreeRTOS용 코드 서명](#)을 사용하여 펌웨어 업데이트에 서명하거나, 자체 코드 서명 도구를 사용할 수 있습니다.

프로세스를 더 자세히 제어하려면 [CreateStream](#) API를 사용하여 MQTT를 통해 업데이트를 전송할 때 스트림을 생성할 수 있습니다. 경우에 따라 Amazon FreeRTOS 에이전트 [코드](#)를 수정하여 보내고 받는 블록의 크기를 조정할 수 있습니다.

OTA 업데이트를 생성하면 OTA 관리자 서비스에서 디바이스에 업데이트가 사용 가능함을 알려주는 [AWS IoT 작업](#)을 생성합니다. Amazon 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. Amazon FreeRTOS를 사용하지 않을 경우 자체 OTA 에이전트를 구현하여 업데이트를 수신 및 다운로드하고 설치 작업을 수행해야 합니다.

애플리케이션에 OTA 에이전트 통합

무선(OTA) 에이전트는 OTA 업데이트 기능을 제품에 추가하기 위해 작성해야 하는 코드 양을 간소화하기 위해 설계되었습니다. 통합 작업에서는 기본적으로 OTA 에이전트를 초기화하고, 선택적으로 OTA 완료 이벤트 메시지에 응답하기 위한 사용자 지정 클백 함수를 생성합니다.

Note

OTA 업데이트 기능을 애플리케이션에 통합하는 작업은 다소 간단하지만, OTA 업데이트 시스템에서 디바이스 코드 통합보다 더 많은 것을 이해해야 합니다. AWS IoT 사물, 자격 증명, 코드 서명 인증서, 디바이스 프로비저닝 및 OTA 업데이트 작업을 사용하여 AWS 계정을 구성하는 방법을 숙지하려면 [Amazon FreeRTOS 사전 조건](#)을 참조하십시오.

연결 관리

OTA 에이전트는 AWS IoT 서비스와 관련된 모든 제어 통신 작업에 MQTT 프로토콜을 사용하지만 MQTT 연결을 관리하지는 않습니다. OTA 에이전트가 애플리케이션의 연결 관리 정책을 간섭하지 않도록 하려면 MQTT 연결(연결 해제 및 다시 연결 기능 포함)을 기본 사용자 애플리케이션에서 처리해야 합니다. 이 파일은 MQTT 또는 HTTP 프로토콜을 통해 다운로드할 수 있습니다. OTA 작업을 생성할 때 프로토콜을 선택할 수 있습니다. MQTT를 선택하면 OTA 에이전트는 제어 작업 및 파일 다운로드에 동일한 연결을 사용합니다. HTTP를 선택하면 OTA 에이전트가 HTTP 연결을 처리합니다.

MQTT를 사용한 간단한 OTA 데모

다음은 에이전트가 MQTT 브로커에 연결하고 OTA 에이전트를 초기화하는 방법을 보여주는 간단한 OTA 데모에서 발췌한 내용입니다. 이 예에서는 기본 OTA 완료 콜백을 사용하고 일부 통계를 초당 하나씩 출력하도록 데모를 구성합니다. 간결하게 나타내기 위해 이 데모에서는 일부 세부 정보를 생략합니다.

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.

```

void vRunOTAUpdateDemo( const IoTNetworkInterface_t * pNetworkInterface,
                        void * pNetworkCredentialInfo )
{
    IoTMqttConnectInfo_t xConnectInfo = IOT_MQTT_CONNECT_INFO_INITIALIZER;
    OTA_State_t eState;
    OTA_ConnectionContext_t xOTAConnectionCtx = { 0 };

    configPRINTF( ( "OTA demo version %u.%u.%u\r\n",
                    xAppFirmwareVersion.u.x.ucMajor,
                    xAppFirmwareVersion.u.x.ucMinor,
                    xAppFirmwareVersion.u.x.usBuild ) );
    configPRINTF( ( "Creating MQTT Client...\r\n" ) );

/* Create the MQTT Client. */

    for( ; ; )
    {
        xNetworkConnected = prxCreatenetworkConnection();

        if( xNetworkConnected )
        {
            configPRINTF( ( "Connecting to broker...\r\n" ) );
            memset( &xConnectInfo, 0, sizeof( xConnectInfo ) );

            if( xConnection.ulNetworkType == AWSIOT_NETWORK_TYPE_BLE )
            {
                xConnectInfo.awsIotMqttMode = false;
                xConnectInfo.keepAliveSeconds = 0;
            }
            else
            {
                xConnectInfo.awsIotMqttMode = true;
                xConnectInfo.keepAliveSeconds = otaDemoKEEPALIVE_SECONDS;
            }

            xConnectInfo.cleanSession = true;
            xConnectInfo.clientIdentifierLength = ( uint16_t )
strlen( clientcredentialIOT_THING_NAME );
            xConnectInfo.pClientIdentifier = clientcredentialIOT_THING_NAME;

/* Connect to the broker. */
            if( IoTMqtt_Connect( & xConnection.xNetworkInfo ),

```

```

    & xConnectInfo,
    otaDemoCONN_TIMEOUT_MS, &( xConnection.xMqttConnection ) )

== IOT_MQTT_SUCCESS )
{
    configPRINTF( ( "Connected to broker.\r\n" ) );
    xOTAConnectionCtx.pvControlClient = xConnection.xMqttConnection;
    xOTAConnectionCtx.pxNetworkInterface = ( void * ) pNetworkInterface;
    xOTAConnectionCtx.pvNetworkCredentials = pNetworkCredentialInfo;

    OTA_AgentInit( ( void * ) ( &xOTAConnectionCtx ), ( const uint8_t * )
( clientcredentialIOT_THING_NAME ), App_OTACompleteCallback, ( TickType_t ) ~0 );

    while( ( eState = OTA_GetAgentState() ) != eOTA_AgentState_Stopped )
    {
        /* Wait forever for OTA traffic but allow other tasks to run and output
statistics only once per second. */
        vTaskDelay( myappONE_SECOND_DELAY_IN_TICKS );
        configPRINTF( ( "State: %s Received: %u     Queued: %u     Processed: %u
Dropped: %u\r\n", pcStateStr[ eState ],
                           OTA_GetPacketsReceived(), OTA_GetPacketsQueued(),
OTA_GetPacketsProcessed(), OTA_GetPacketsDropped() ) );
    }

    IotMqtt_Disconnect( xConnection.xMqttConnection, false );
}
else
{
    configPRINTF( ( "ERROR: MQTT_AGENT_Connect() Failed.\r\n" ) );
}

vMqttDemoDeleteNetworkConnection( &xConnection );

/* After failure to connect or a disconnect, wait an arbitrary one second
before retry. */
vTaskDelay( myappONE_SECOND_DELAY_IN_TICKS );
}
else
{
    configPRINTF( ( "Failed to create MQTT client.\r\n" ) );
}
}
}

```

다음은 이 데모 애플리케이션의 상위 수준 흐름입니다.

- MQTT 에이전트 컨텍스트를 생성합니다.
 - AWS IoT 엔드포인트에 연결합니다.
 - OTA 에이전트를 초기화합니다.
 - OTA 업데이트 작업 루프를 허용하고 1초에 1회 통계를 출력합니다.
 - 에이전트가 중지하면 1초 기다렸다가 연결을 다시 시도합니다.

OTA 완료 이벤트에 대해 사용자 지정 콜백 사용

이전 예에서는 OTA_AgentInit API에 대한 세 번째 파라미터를 NULL로 지정하여 OTA 완료 이벤트에 대해 기본 제공 콜백 핸들러를 사용했습니다. 완료 이벤트에 대해 사용자 지정 처리를 구현하려면 콜백 핸들러의 함수 주소를 OTA_AgentInit API에 전달해야 합니다. OTA 프로세스 중에 에이전트는 다음 이벤트 열거형 중 하나를 콜백 핸들러에 전송할 수 있습니다. 이벤트를 처리하는 방법과 시기는 애플리케이션 개발자가 결정해야 합니다.

/* *

```
* @brief OTA Job callback events.  
*  
* After an OTA update image is received and authenticated, the agent calls the user  
* callback (set with the OTA_AgentInit API) with the value eOTA_JobEvent_Activate to  
* signal that the device must be rebooted to activate the new image. When the device  
* boots, if the OTA job status is in self test mode, the agent calls the user callback  
* with the value eOTA_JobEvent_StartTest, signaling that any additional self tests  
* should be performed.  
*  
* If the OTA receive fails for any reason, the agent calls the user callback with  
* the value eOTA_JobEvent_Fail instead to allow the user to log the failure and take  
* any action deemed appropriate by the user code.  
*/  
typedef enum {  
    eOTA_JobEvent_Activate, /*! OTA receive is authenticated and ready to activate. */  
    eOTA_JobEvent_Fail,    /*! OTA receive failed. Unable to use this update. */  
    eOTA_JobEvent_StartTest /*! OTA job is now in self test, perform user tests. */  
} OTA_JobEvent_t;
```

OTA 에이전트는 기본 애플리케이션에서 활성 처리 중에 백그라운드에서 업데이트를 수신할 수 있습니다. 이러한 이벤트를 전달하는 목적은 즉시 조치할 수 있는지, 아니면 일부 다른 애플리케이션별 처리가 완료 될 때까지 지연할지 여부를 애플리케이션에서 결정할 수 있도록 하기 위한 것입니다. 그러면 활성 처리(예: vacuum 수행) 중에 디바이스의 예기치 않은 중단으로 펌웨어 업데이트 후에 재설정하는 것을 방지할 수 있습니다. 다음은 콜백 핸들러에 의해 수신되는 작업 이벤트입니다.

eOTA_JobEvent_Activate event

이 이벤트가 콜백 핸들러에 수신된 경우 즉시 디바이스를 재설정하거나 나중에 디바이스 재설정을 위한 호출을 예약할 수 있습니다. 따라서 필요한 경우 디바이스 재설정 및 자체 테스트 단계를 연기할 수 있습니다.

eOTA_JobEvent_Fail event

이 이벤트가 콜백 핸들러에 수신된 경우 업데이트가 실패한 것입니다. 이 경우 아무것도 수행할 필요가 없습니다. 로그 메시지를 출력하거나 애플리케이션 관련 작업을 수행할 수 있습니다.

eOTA_JobEvent_StartTest event

자체 테스트 단계에서는 새로 업데이트된 펌웨어를 실행하고 올바르게 작동하는지 결정하기 이전에 자체 테스트를 실시한 후 최신 영구 애플리케이션 이미지로 커밋할 수 있습니다. 새 업데이트를 수신하여 인증하고 디바이스를 재설정한 후 테스트 준비가 되면 OTA 에이전트가 eOTA_JobEvent_StartTest 이벤트를 콜백 함수에 전송합니다. 개발자는 업데이트 후 디바이스 펌웨어가 올바르게 작동하는지 확인하는 데 필요한 테스트를 추가할 수 있습니다. 자체 테스트에서 디바이스 펌웨어가 안정적인 것으로 간주될 경우 코드는 OTA_SetImageState(eOTA_ImageState_Accepted) 함수를 호출하여 펌웨어를 새 영구 이미지로 커밋해야 합니다.

디바이스에 테스트를 수행하는 데 필요한 특수 하드웨어 또는 메커니즘이 없는 경우 기본 콜백 핸들러를 사용할 수 있습니다. eOTA_JobEvent_Activate 이벤트가 수신되면 기본 핸들러가 즉시 디바이스를 재설정합니다.

OTA 보안

다음은 무선(OTA) 보안의 세 가지 측면입니다.

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.

Amazon FreeRTOS OTA 업데이트 관리자 서비스는 Code Signing for AWS IoT를 사용하여 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 따라 스트리밍 중에 Amazon S3에 저장된 펌웨어 업데이트에 액세스할 수 있도록 AWS IoT 서비스 보안 주체에게 읽기 권한을 부여해야 합니다.

Code Signing for AWS IoT

AWS IoT 콘솔은 [AWS IoT용 코드 서명](#)을 사용하여 AWS IoT에서 지원되는 디바이스에 대한 펌웨어 이미지에 자동으로 서명합니다.

AWS IoT용 코드 서명은 ACM으로 가져오는 인증서와 프라이빗 키를 사용합니다. 테스트를 위해 자체 서명된 인증서를 사용할 수 있지만, 잘 알려진 상용 인증 기관(CA)에서 인증서를 얻는 것이 좋습니다.

코드 서명 인증서는 X.509 버전 3 Key Usage 및 Extended Key Usage 확장을 사용합니다. Key Usage 확장은 Digital Signature로 설정되어 있고, Extended Key Usage 확장은 Code Signing으로 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

[Amazon Web Services용 도구](#)에서 AWS IoT SDK용 코드 서명을 다운로드할 수 있습니다.

OTA 문제 해결

다음 단원에서는 OTA 업데이트 관련 문제를 해결하는 데 유용한 정보를 제공합니다.

주제

- [OTA 업데이트를 위한 Cloudwatch Logs 설정 \(p. 48\)](#)
- [AWS CloudTrail를 사용하여 AWS IoT OTA API 호출 로깅 \(p. 52\)](#)
- [AWS CLI를 사용하여 OTA 오류 코드 가져오기 \(p. 54\)](#)
- [Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결 \(p. 55\)](#)

OTA 업데이트를 위한 Cloudwatch Logs 설정

OTA 업데이트 서비스에서는 Amazon CloudWatch를 통한 로깅을 지원합니다. AWS IoT 콘솔을 사용하여 OTA 업데이트를 위한 Amazon CloudWatch 로깅을 활성화하고 구성할 수 있습니다. CloudWatch Logs에 대한 자세한 내용은 [Cloudwatch Logs](#)를 참조하십시오.

로깅을 활성화하려면 IAM 역할을 생성하고 OTA 업데이트 로깅을 구성해야 합니다.

Note

OTA 업데이트 로깅을 활성화하기 전에 CloudWatch Logs 액세스 권한을 이해해야 합니다. CloudWatch Logs 액세스 권한이 있는 사용자는 디버깅 정보를 볼 수 있습니다. 자세한 내용은 [Amazon CloudWatch Logs에 대한 인증 및 액세스 제어](#)를 참조하십시오.

로깅 역할 생성 및 로깅 활성화

AWS IoT 콘솔을 사용하여 로깅 역할을 생성하고 로깅을 활성화합니다.

1. 탐색 창에서 설정을 선택합니다.
2. 로그에서 편집을 선택합니다.
3. 세부 사항 수준에서 디버깅을 선택합니다.
4. 역할 설정에서 새로 생성을 선택하여 로깅을 위한 IAM 역할을 생성합니다.
5. 이름 아래에 역할의 고유한 이름을 입력합니다. 모든 필요한 권한을 가진 역할이 생성됩니다.
6. [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

CloudWatch 콘솔에서 로그를 볼 수 있습니다.

CloudWatch Logs에서 OTA 업데이트를 보려면

1. 탐색 창에서 로그를 선택합니다.
2. 로그 그룹에서 AWSIoTLogsV2를 선택합니다.

OTA 업데이트 로그는 다음 속성을 포함할 수 있습니다.

accountId

로그를 생성한 AWS 계정 ID입니다.

actionType

로그를 생성한 작업입니다. 이 속성은 다음 값 중 하나로 설정될 수 있습니다.

- CreateOTAUpdate: OTA 업데이트가 생성되었습니다.
- DeleteOTAUpdate: OTA 업데이트가 삭제되었습니다.
- StartCodeSigning: 코드 서명 작업이 시작되었습니다.
- CreateAWSJob: AWS IoT 작업이 생성되었습니다.
- CreateStream: 스트림이 생성되었습니다.
- GetStream: 스트림 요청이 AWS IoT 스트리밍 서비스로 전송되었습니다.
- DescribeStream: 스트림에 대한 정보 요청이 AWS IoT 스트리밍 서비스로 전송되었습니다.

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": "875157236366",  
    "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 스트리밍 서비스에서 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

Amazon FreeRTOS가 CloudTrail과 통합되었습니다. 이 서비스는 모든 AWS IoT OTA API 호출을 캡처하고 사용자가 지정하는 Amazon S3 버킷에 로그 파일을 전송합니다. CloudTrail은 사용자 코드에서 AWS IoT OTA API로 전송되는 API 호출을 캡처합니다. CloudTrail에서 수집하는 정보를 사용하여 AWS IoT OTA에 어떤 요청이 이루어졌는지, 어떤 소스 IP 주소에서 요청했는지, 누가 언제 요청했는지 등을 확인할 수 있습니다.

그 구성 및 활성화 방법을 포함하여 CloudTrail에 대한 자세한 내용은 [AWS CloudTrail User Guide](#)를 참조하십시오.

CloudTrail의 Amazon FreeRTOS 정보

AWS 계정에서 CloudTrail 로깅이 활성화된 경우, AWS IoT OTA 작업에 대한 API 호출이 CloudTrail 로그 파일에서 추적되어 다른 AWS 제품 레코드와 함께 로그 파일에 기록됩니다. CloudTrail은 기간 및 파일 크기를 기준으로 새 파일을 만들고 기록하는 시점을 결정합니다.

다음 AWS IoT OTA 컨트롤 플레이션 작업은 CloudTrail에서 로깅합니다.

- [CreateStream](#)
- [DescribeStream](#)
- [ListStreams](#)
- [UpdateStream](#)
- [DeleteStream](#)
- [CreateOTAUpdate](#)
- [GetOTAUpdate](#)
- [ListOTAUpdates](#)
- [DeleteOTAUpdate](#)

Note

AWS IoT OTA 데이터 영역 작업(디바이스 측)은 CloudTrail에서 로깅하지 않습니다. 이러한 작업을 모니터링하려면 CloudWatch를 사용하십시오.

모든 로그 항목에는 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 [CloudTrail userIdentity 요소](#) 단원을 참조하십시오. AWS IoT OTA 작업은 [AWS IoT OTA API 참조](#)에 나와 있습니다.

원하는 기간만큼 Amazon S3 버킷에 로그 파일을 저장할 수 있지만 Amazon S3 수명 주기 규칙을 정의하여 자동으로 로그 파일을 보관하거나 삭제할 수도 있습니다. 기본적으로 로그 파일은 Amazon S3 서버 측 암호화(SSE)를 통해 암호화됩니다.

로그 파일이 전송될 때 알림을 받으려면 Amazon SNS 알림을 게시하도록 CloudTrail을 구성할 수 있습니다. 자세한 내용은 [CloudTrail에 대한 Amazon SNS 알림 구성](#) 단원을 참조하십시오.

여러 AWS 리전 및 여러 AWS 계정의 AWS IoT OTA 로그 파일을 하나의 Amazon S3 버킷에 집계할 수도 있습니다.

자세한 내용은 [여러 리전에서 CloudTrail 로그 파일 받기](#) 및 [여러 계정에서 CloudTrail 로그 파일 받기](#) 단원을 참조하십시오.

Amazon 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.

다음은 CreateOTAUpdate 작업 호출의 로그를 보여 주는 CloudTrail 로그 항목을 나타낸 예제입니다.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::<your_aws_account>:user/<your_user_id>",
        "accountId": "<your_aws_account>",
        "accessKeyId": "<your_access_key_id>",
        "userName": "<your_username>",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-23T17:27:08Z"
            }
        },
        "invokedBy": "apigateway.amazonaws.com"
    },
    "eventTime": "2018-08-23T17:27:19Z",
    "eventSource": "iot.amazonaws.com",
    "eventName": "CreateOTAUpdate",
    "awsRegion": "<your_aws_region>",
    "sourceIPAddress": "apigateway.amazonaws.com",
    "userAgent": "apigateway.amazonaws.com",
    "requestParameters": {
        "targets": [
            "arn:aws:iot:<your_aws_region>:<your_aws_account>:thing/Thing_CMH"
        ],
        "roleArn": "arn:aws:iam::<your_aws_account>:role/Role_FreeRTOSJob",
        "files": [
            {
                "fileName": "/sys/mcuflashimg.bin",
                "fileSource": {
                    "fileId": 0,
                    "streamId": "<your_stream_id>"
                },
                "codeSigning": {
                    "awsSignerJobId": "<your_signer_job_id>"
                }
            }
        ]
    }
}
```

```
        },
    ],
    "targetSelection": "SNAPSHOT",
    "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
},
"responseElements": {
    "otaUpdateArn": "arn:aws:iot:<your_aws_region>:<your_aws_account>:otaupdate/
FreeRTOSJob_CMH-23-1535045232806-92",
    "otaUpdateStatus": "CREATE_PENDING",
    "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
},
"requestID": "c9649630-a6f9-11e8-8f9c-e1cf2d0c9d8e",
"eventID": "ce9bf4d9-5770-4cee-acf4-0e5649b845c0",
"eventType": "AwsApiCall",
"recipientAccountId": "<recipient_aws_account>"
}
```

AWS CLI를 사용하여 OTA 오류 코드 가져오기

1. AWS CLI를 설치하고 구성합니다.
2. "aws configure"를 실행하고 다음 정보를 입력합니다.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

3. 실행합니다.

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

여기서 **JobID**는 상태를 가져오려는 작업에 대한 전체 작업 ID 문자열이고 **ThingName**은 디바이스가 AWS IoT에 등록된 AWS IoT 사물 이름입니다.

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.html에 나열되어 있습니다. 접두사가 "kOTA_err_"인 오류 코드를 참조하십시오. 하위 코드는 플랫폼별 코드이거나 일반 오류에 대한 자세한 내용을 제공할 수 있습니다.

Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결

CC3220SF Launchpad 플랫폼은 소프트웨어 번조 탑지 메커니즘을 제공합니다. 이 플랫폼은 무결성 위반 시마다 증가하는 보안 알림 카운터를 사용합니다. 보안 알림 카운터가 미리 결정된 임계값(기본값 15)에 도달하면 디바이스가 잠기고 호스트는 SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT 비동기 이벤트를 수신합니다. 잠긴 디바이스는 액세스가 제한됩니다. 디바이스를 복구하려면 디바이스를 다시 프로그래밍하거나 출하 시 설정으로 복원 프로세스를 사용하여 출하 시 이미지로 되돌릴 수 있습니다. network_if.c에서 비동기 이벤트 핸들러를 업데이트하여 원하는 동작을 프로그래밍해야 합니다.

Amazon FreeRTOS 소스 코드 다운로드

Amazon FreeRTOS 콘솔에서 Amazon FreeRTOS 검증 플랫폼에 대해 구성된 Amazon FreeRTOS 버전을 다운로드할 수 있습니다. 검증 플랫폼 목록은 [Amazon FreeRTOS 적격 하드웨어 플랫폼 \(p. 58\)](#) 또는 [Amazon FreeRTOS 파트너](#) 웹 사이트를 참조하십시오.

[GitHub](#)에서 Amazon FreeRTOS를 복제하거나 다운로드할 수도 있습니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.

Amazon FreeRTOS 콘솔

Amazon FreeRTOS 콘솔에서 마이크로 컨트롤러 기반 디바이스용 애플리케이션을 작성하는 데 필요한 모든 것을 담은 패키지를 구성하고 다운로드할 수 있습니다.

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

자세한 내용은 [Amazon FreeRTOS 콘솔 \(p. 55\)](#) 단원을 참조하십시오.

Amazon FreeRTOS 콘솔

Amazon FreeRTOS 콘솔에서 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.

미리 정의된 Amazon FreeRTOS 구성

미리 정의된 구성을 사용하면 필요한 라이브러리에 대해 고민할 필요 없이 지원되는 사용 사례를 활용하여 빠르게 시작할 수 있습니다. 미리 정의된 구성을 사용하려면 [Amazon FreeRTOS 콘솔](#)로 이동하여 사용할 구성을 찾은 다음 다운로드를 선택합니다.

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

미리 정의된 구성에서 사용자 지정 구성을 생성하려면

1. [Amazon FreeRTOS 콘솔](#)로 이동합니다.

2. 탐색 창에서 소프트웨어를 선택합니다.
3. Amazon FreeRTOS 디바이스 소프트웨어에서 구성 다운로드를 선택합니다.
4. 사용자 지정하려는 미리 정의된 구성 옆에 있는 줄임표(...)를 선택한 다음 사용자 지정을 선택합니다.
5. Configure Amazon FreeRTOS Software(Amazon FreeRTOS 소프트웨어 구성) 페이지에서 Amazon FreeRTOS 버전, 하드웨어 플랫폼 및 라이브러리를 선택하고 새 구성에 이름을 지정하고 설명을 입력합니다.
6. 페이지 맨 아래에서 생성 후 다운로드를 선택합니다.

사용자 지정 Amazon FreeRTOS 구성

사용자 지정 구성은 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 구성은 생성하려면

1. [Amazon FreeRTOS 콘솔](#)로 이동하여 새로 생성을 선택합니다.
2. 사용할 Amazon 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 Amazon FreeRTOS 라이브러리에 대해 라이브러리 추가를 선택합니다. 다른 라이브러리를 필요로 하는 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 구성은 편집하려면

1. [Amazon FreeRTOS 콘솔](#)로 이동합니다.
2. 탐색 창에서 소프트웨어를 선택합니다.
3. Amazon FreeRTOS 디바이스 소프트웨어에서 구성 다운로드를 선택합니다.
4. 편집하려는 구성 옆에 있는 줄임표(...)를 선택한 다음 편집을 선택합니다.
5. Configure Amazon FreeRTOS Software(Amazon FreeRTOS 소프트웨어 구성) 페이지에서 구성의 Amazon FreeRTOS 버전, 하드웨어 플랫폼, 라이브러리 및 설명을 변경할 수 있습니다.
6. 페이지 맨 아래에서 저장 후 다운로드를 선택합니다.

사용자 지정 구성은 삭제하려면

1. [Amazon FreeRTOS 콘솔](#)로 이동합니다.
2. 탐색 창에서 소프트웨어를 선택합니다.
3. Amazon FreeRTOS 디바이스 소프트웨어에서 구성 다운로드를 선택합니다.
4. 삭제하려는 구성 옆에 있는 줄임표(...)를 선택한 다음 삭제를 선택합니다.

빠른 연결 워크플로우

Amazon FreeRTOS 콘솔은 또한 미리 정의된 구성이 있는 모든 보드를 위한 빠른 연결 워크플로 옵션을 포함합니다. 빠른 연결 워크플로우는 AWS IoT 및 AWS IoT Greengrass를 위한 Amazon FreeRTOS 데모 애플리케이션을 구성하고 실행하도록 도와줍니다. 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 Amazon FreeRTOS 구성에 태그를 적용할 수 있습니다. 구성에 태그를 적용하려면 콘솔로 이동합니다. 태그 아래에서 태그의 이름과 값을 입력합니다.

예를 들어, 태그를 사용하여 IAM 정책 구성에 대한 액세스 권한을 관리할 수 있습니다. 자세한 내용은 [IAM 정책에 태그 사용 \(p. 57\)](#) 단원을 참조하십시오.

태그를 사용하여 AWS IoT 리소스를 관리하는 방법에 대한 자세한 내용은 AWS IoT 개발자 안내서의 [IAM 정책에 태그 사용](#) 단원을 참조하십시오.

IAM 정책에 태그 사용

Amazon FreeRTOS 콘솔을 사용하여 작업에 사용하는 IAM 정책에서 태그 기반의 리소스 수준 권한을 적용할 수 있습니다. 이를 통해 사용자가 생성, 수정 또는 사용할 수 있는 구성을 더욱 정확하게 제어할 수 있습니다. AWS IoT의 태그 지정 및 IAM 정책에 대한 자세한 내용은 AWS IoT 개발자 안내서의 [IAM 정책에 태그 사용](#) 단원을 참조하십시오.

IAM 정책 정의에서 리소스 태그를 기반으로 사용자 액세스(권한)를 제어하기 위해 다음 조건 컨텍스트 키 및 값과 함께 Condition 요소(Condition 블록이라고도 함)를 사용합니다.

- 특정 태그가 지정된 Amazon FreeRTOS 구성에 대해 사용자 작업을 허용하거나 거부하려면 `aws:ResourceTag/tag-key: tag-value`를 사용합니다.
- Amazon FreeRTOS 콘솔에서 구성을 생성하거나 수정할 때 특정 태그를 사용해야 하거나 사용하지 않아야 할 경우 `aws:RequestTag/tag-key: tag-value`를 사용합니다.
- Amazon FreeRTOS 콘솔에서 구성을 생성하거나 수정할 때 특정 태그 키 집합을 사용해야 하거나 사용하지 않아야 할 경우 `aws:TagKeys: [tag-key, ...]`를 사용합니다.

자세한 내용은 AWS Identity and Access Management 사용 설명서의 [태그를 사용한 액세스 제어](#) 단원을 참조하십시오. IAM 내 JSON 정책의 요소, 변수 및 평가 로직에 대한 자세한 구문, 설명 및 예제는 [IAM JSON 정책 참조](#) 단원을 참조하십시오.

다음은 태그 기반 제한 2개를 적용하는 정책 예제입니다. 이 정책으로 제한되는 IAM 사용자는 다음과 같습니다.

- 리소스에 태그 `env=prod`을 지정할 수 없습니다. 이 예제에서 `"aws:RequestTag/env" : "prod"` 행을 참조하십시오.
- 기존 태그 `env=prod`가 지정된 리소스를 수정 또는 액세스할 수 없습니다. 이 예제에서 `"aws:ResourceTag/env" : "prod"` 행을 참조하십시오.

```
{  
    "Version" : "2012-10-17",  
    "Statement" : [  
        {  
            "Effect" : "Deny",  
            "Action" : "freertos:*",  
            "Resource" : "*",  
            "Condition" : {  
                "StringEquals" : {  
                    "aws:RequestTag/env" : "prod",  
                    "aws:ResourceTag/env" : "prod"  
                }  
            }  
        }  
    ]  
}
```

```
        "aws:RequestTag/env" : "prod"
    }
}
{
    "Effect" : "Deny",
    "Action" : "freertos:*",
    "Resource" : "*",
    "Condition" : {
        "StringEquals" : {
            "aws:ResourceTag/env" : "prod"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iot:/*"
    ],
    "Resource": "*"
}
]
```

또한 다음과 같이 목록에서 태그를 둘러싸 지정된 태그 키에 대해 여러 태그 값을 지정할 수도 있습니다.

```
"StringEquals" : {
    "aws:ResourceTag/env" : ["dev", "test"]
}
```

Note

태그를 기준으로 리소스에 대한 사용자 액세스를 허용 또는 거부하는 경우 동일한 리소스에서 태그를 추가 또는 제거할 수 있도록 사용자를 명시적으로 거부할 것을 고려해야 합니다. 그렇지 않으면 사용자가 제한을 피해 태그를 수정하여 리소스에 대한 액세스 권한을 얻을 수 있습니다.

Amazon FreeRTOS 적격 하드웨어 플랫폼

다음 하드웨어 플랫폼은 Amazon FreeRTOS에 대해 인증되었습니다.

- ATECC608a Zero Touch Provisioning Kit for AWS IoT
- Cypress CYW943907AEVAL1F 개발 키트
- Cypress CYW954907AEVAL1F 개발 키트
- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT 연결 키트
- Marvell MW320 AWS IoT 시작 키트
- Marvell MW322 AWS IoT 시작 키트
- MediaTek MT7697Hx 개발 키트
- Microchip Curiosity PIC32MZEF 번들
- Nordic nRF52840-DK
- NXP LPC54018 IoT Module
- OPTIGA Trust X Security Solution
- Renesas RX65N RSK IoT Module

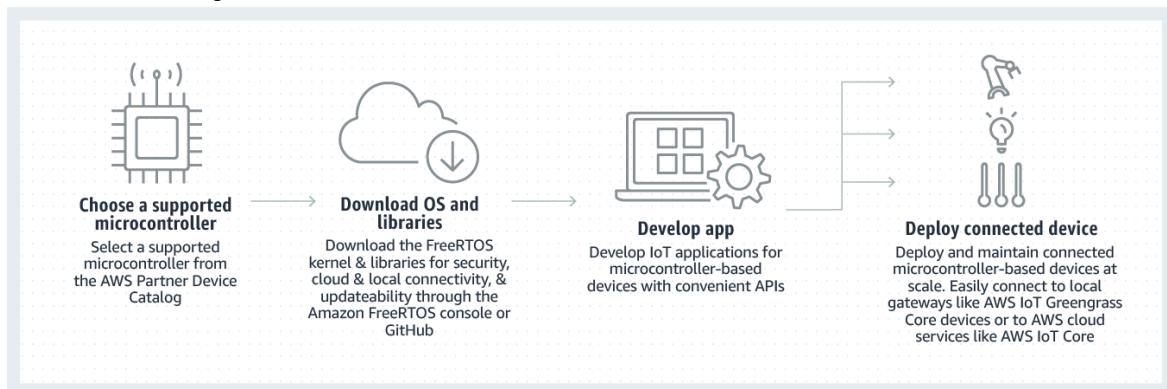
- STMicroelectronics STM32L4 Discovery Kit IoT 노드
- Texas Instruments CC3220SF-LAUNCHXL
- 최소 듀얼 코드 및 유선 이더넷 연결이 있는 Microsoft Windows 7 이상
- Xilinx Avnet MicroZed Industrial IoT Kit

적격 디바이스 목록은 [AWS Partner Device Catalog](#)에도 나와 있습니다.

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

개발 워크플로우

Amazon 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.



Embedded C용 AWS IoT 디바이스 SDK

Embedded C용 AWS IoT 디바이스 SDK는 안전하게 AWS IoT 플랫폼에 연결하기 위해 포함된 애플리케이션에서 사용할 수 있는 C 소스 파일의 모음입니다. SDK에는 전송 클라이언트, TLS 구현, 사용 예제가 포함됩니다. 또한 AWS 클라우드의 AWS IoT 서비스와 상호 작용하는 라이브러리도 포함됩니다.

Embedded C용 AWS IoT 디바이스 SDK에는 다음 라이브러리가 포함됩니다. 이러한 라이브러리에는 해당 Amazon FreeRTOS 라이브러리와 동일한 API가 있습니다.

- AWS IoT Device Defender
- MQTT
- AWS IoT 디바이스 샘플
- 공통 라이브러리
 - 선형 컨테이너
 - 로깅
 - 정적 메모리
 - 작업 풀

소스 코드로 배포되는 이 SDK는 애플리케이션 코드, 기타 라이브러리 및 RTOS와 함께 고객 펌웨어에 내장됩니다. 자세한 내용은 [Embedded C용 AWS IoT 디바이스 SDK GitHub](#)를 참조하십시오.

SDK 소스 코드를 환경에 이식하는 방법에 대한 자세한 내용은 [Embedded C용 AWS IoT 디바이스 SDK 이식 안내서](#)를 참조하십시오.

API 참조의 경우 [AWS IoT 디바이스 SDK C API 참조](#)를 참조하십시오.

추가 리소스

다음은 유용한 리소스입니다.

- Amazon FreeRTOS 엔지니어링 팀에 Amazon FreeRTOS에 대한 질문이 있는 경우 [Amazon FreeRTOS GitHub 페이지](#)에서 문제를 개설할 수 있습니다.
- AWS 커뮤니티에서 AWS 및 Amazon FreeRTOS에 대한 기술적 질문을 하려면 [AWS 토론 포럼](#)을 방문하십시오.
- AWS에 대한 기술 지원을 받으려면 [AWS 지원 센터](#)를 방문하십시오.
- AWS 결제, 계정 서비스, 이벤트, 침해 또는 기타 AWS 관련 문제를 문의하려면 [문의처](#) 페이지를 방문하십시오.

Amazon FreeRTOS 시작하기

이 Amazon FreeRTOS 시작하기 자습서에서는 호스트 컴퓨터에 Amazon FreeRTOS를 다운로드하고 구성한 후 [적격 마이크로컨트롤러 보드](#)에서 단일 데모 애플리케이션을 컴파일하여 실행하는 방법을 보여 줍니다.

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.

Amazon FreeRTOS 데모 애플리케이션

이 자습서의 데모 애플리케이션은 `/demos/mqtt/iot_demo_mqtt.c` 파일에 정의된 Hello World MQTT 데모입니다. 이 애플리케이션은 [Amazon FreeRTOS MQTT 라이브러리 \(p. 216\)](#)를 사용하여 AWS 클라우드에 연결하고 주기적으로 [AWS IoT MQTT 브로커](#)에 의해 호스팅되는 MQTT 주제에 메시지를 게시합니다.

Amazon FreeRTOS 데모 애플리케이션은 한 번에 하나만 실행할 수 있습니다. Amazon FreeRTOS 데모 프로젝트를 직접 빌드하는 경우 `<amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_demo_config.h` 헤더 파일에서 활성화된 첫 번째 데모가 실행되는 애플리케이션입니다. 이 자습서에서는 어떤 데모도 활성화 또는 비활성화할 필요가 없습니다. Hello World MQTT 데모는 기본적으로 활성화되어 있습니다.

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

첫 번째 단계

시작하려면 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오.

보드별 시작 안내서

[첫 번째 단계 \(p. 62\)](#)를 완료했으면 플랫폼의 하드웨어 및 소프트웨어 개발 환경을 설정한 후 보드에서 데모를 컴파일하고 실행할 수 있습니다. 보드별 지침은 [보드별 시작 안내서 \(p. 76\)](#) 단원을 참조하십시오.

문제 해결

시작하기 과정에서 발생하는 문제 해결에 대한 도움말은 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오. 보드별 문제 해결 도움말은 [보드별 시작 안내서 \(p. 76\)](#)에서 해당 보드의 시작 안내서를 참조하십시오.

Amazon FreeRTOS 애플리케이션 개발

IDE를 사용하여 Amazon FreeRTOS 인증 디바이스에서 코드를 편집, 디버깅, 컴파일, 플래시 및 실행할 수 있습니다. 각 보드별 시작 안내서에는 특정 플랫폼용 IDE 설정 지침이 포함되어 있습니다.

타사 코드 편집기 및 디버거를 사용하여 애플리케이션을 개발하고, CMake를 사용하여 소스 코드를 빌드하고 실행할 수도 있습니다. CMake를 Amazon FreeRTOS 개발용 빌드 도구로 사용하는 방법에 대한 자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

첫 번째 단계

Amazon FreeRTOS를 시작 하려면 AWS 계정, AWS IoT 및 Amazon FreeRTOS 클라우드 서비스에 액세스 할 수 있는 권한을 가진 IAM 사용자가 필요하며 지원되는 하드웨어 플랫폼 중 하나가 필요합니다. 또한 Amazon FreeRTOS를 다운로드하고, 해당 보드의 Amazon FreeRTOS 데모 프로젝트를 AWS IoT와 연동하도록 구성해야 합니다. 다음 단원에서는 이러한 요구 사항을 소개합니다.

1. AWS 계정 및 권한 설정 (p. 62)

AWS 계정 및 권한 설정 (p. 62)의 지침을 완료했으면, [Amazon FreeRTOS 콘솔](#)에서 빠른 연결 워크플로우를 수행하여 보드를 AWS 클라우드에 빠르게 연결할 수 있습니다. 빠른 연결 워크플로를 수행할 경우 이 목록의 나머지 단계를 완료하지 않아도 됩니다. 현재 다음 보드에서는 Amazon FreeRTOS 콘솔에서 Amazon FreeRTOS의 구성을 사용할 수 없습니다.

- Cypress CYW943907AEVAL1F 개발 키트
- Cypress CYW954907AEVAL1F 개발 키트

2. AWS IoT에 MCU 보드 등록 (p. 63)

3. Amazon FreeRTOS 다운로드 (p. 65)
4. Amazon FreeRTOS 데모 구성 (p. 66)

Note

Espressif ESP32-DevKitC 또는 ESP-WROVER-KIT를 사용하는 경우 이 첫 단계를 건너뛰고 [Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 시작하기 \(p. 86\)](#)로 진행하십시오.
Nordic nRF52840-DK를 사용하는 경우 이 첫 단계를 건너뛰고 [Nordic nRF52840-DK 시작하기 \(p. 138\)](#)로 진행하십시오.

AWS 계정 및 권한 설정

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

IAM 사용자를 AWS 계정에 추가하려면 [IAM 사용 설명서](#)를 참조하십시오. AWS IoT 및 Amazon FreeRTOS에 대한 액세스 권한을 IAM 사용자 계정에 부여하려면 다음 IAM 정책을 IAM 사용자 계정에 연결합니다.

- `AmazonFreeRTOSFullAccess`
- `AWSIoTFullAccess`

`AmazonFreeRTOSFullAccess` 정책을 IAM 사용자에게 연결하려면

1. [IAM 콘솔](#)로 이동하고 탐색 창에서 사용자를 선택합니다.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 `AmazonFreeRTOSFullAccess`를 입력하고 목록에서 해당 항목을 선택한 다음 Next: Review(다음: 검토)를 선택합니다.
6. 권한 추가를 선택합니다.

AWSIoTFullAccess 정책을 IAM 사용자에게 연결하려면

1. [IAM 콘솔](#)로 이동하고 탐색 창에서 사용자를 선택합니다.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 **AWSIoTFullAccess**를 입력하고 목록에서 해당 항목을 선택한 다음 Next: Review(다음: 검토)를 선택합니다.
6. 권한 추가를 선택합니다.

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

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

AWS 계정과 권한을 설정한 후 계속해서 [Amazon FreeRTOS 콘솔](#)의 빠른 연결 워크플로우 또는 [AWS IoT에 MCU 보드 등록 \(p. 63\)](#)으로 진행할 수 있습니다.

AWS IoT에 MCU 보드 등록

AWS 클라우드와 통신하기 위해서는 보드가 AWS IoT에 등록되어 있어야 합니다. 보드를 AWS IoT에 등록하려면 다음 요소가 필요합니다.

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.

[Amazon FreeRTOS 콘솔](#)에서 빠른 연결 워크플로우를 사용하는 경우 정책, AWS IoT 사물, 키 및 인증서가 생성됩니다. 빠른 연결 워크플로우를 사용할 경우 다음 절차를 무시해도 됩니다.

보드를 수동으로 등록하려면 아래 절차를 수행합니다.

AWS IoT 정책을 생성하려면

1. IAM 정책을 생성하려면 AWS 리전과 AWS 계정 번호를 알아야 합니다.

AWS 계정 번호를 확인하려면 [AWS Management Console](#)을 열고 오른쪽 위 모서리에서 계정 이름 아래의 메뉴를 찾아 확장한 후 내 계정을 선택합니다. 계정 ID는 Account Settings(계정 설정) 아래에 표시됩니다.

AWS 계정에 대한 AWS 리전을 확인하려면 AWS Command Line Interface를 사용합니다. 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
```

출력은 다음과 같아야 합니다.

```
{
```

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

이 예제에서 리전은 us-west-2입니다.

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

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

이 정책은 다음 권한을 부여합니다.

iot:Connect

모든 클라이언트 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. 생성을 선택합니다.

디바이스에 대한 IoT 사물, 프라이빗 키 및 인증서를 생성하려면

1. AWS IoT 콘솔로 이동합니다.

2. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다.
3. 계정에 등록된 IoT 사물이 없는 경우 You don't have any things yet(아직 사물이 없습니다) 페이지가 표시됩니다. 이 페이지가 나타나면 Register a thing(사물 등록)을 선택합니다. 그렇지 않은 경우 [Create]를 선택합니다.
4. Creating AWS IoT things(IoT 사물 생성) 페이지에서 Create a single thing(단일 사물 생성)을 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 [Amazon FreeRTOS 다운로드 \(p. 65\)](#)를 수행할 수 있습니다.

Amazon FreeRTOS 다운로드

Amazon FreeRTOS 콘솔이나 [Amazon FreeRTOS GitHub 리포지토리](#)에서 Amazon FreeRTOS를 다운로드 할 수 있습니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.

Amazon FreeRTOS 콘솔에서 빠른 연결 워크플로우를 수행할 경우에는 콘솔에서 워크플로우의 다운로드 지침을 수행하고 다음 절차는 무시하십시오.

Amazon FreeRTOS 콘솔에서 Amazon FreeRTOS를 다운로드하려면

1. [Amazon FreeRTOS 콘솔](#)로 이동합니다.
2. 미리 정의된 구성에서 AWS IoT- ##### ##을 찾은 후 다운로드를 선택합니다.
3. 폴더에 다운로드한 파일의 압축을 풀고 폴더 경로를 기록해둡니다. 이 자습서에서는 이 폴더를 <[amazon-freertos](#)>라고 합니다.

Note

Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. Amazon FreeRTOS 다운로드의 가장 긴 경로는 122자입니다. Amazon FreeRTOS 프로젝트에서 파일을 수용하려면 <[amazon-freertos](#)> 디렉터리에 대한 경로가 98자 이하여야 합니다. 예를 들어, c:\Users\Username\Dev\AmazonFreeRTOS는 사용할 수 있지만 c:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS를 사용하면 빌드 실패가 발생합니다.

Note

Cypress CYW954907AEVAL1F 또는 CYW943907AEVAL1F 개발 키트를 시작하는 경우 GitHub에서 Amazon FreeRTOS를 다운로드해야 합니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오. 현재, 이러한 보드에 대한 Amazon FreeRTOS 구성은 Amazon FreeRTOS 콘솔에서 사용할 수 없습니다.

Note

Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. 이름이 긴 Amazon FreeRTOS 다운로드 디렉터리 경로는 빌드 오류를 발생시킬 수 있습니다.

시작하기 설명서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

Amazon FreeRTOS를 다운로드했으면 [Amazon FreeRTOS 데모 구성 \(p. 66\)](#)으로 계속 진행합니다.

Amazon FreeRTOS 데모 구성

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

[Amazon FreeRTOS 콘솔](#)에서 빠른 연결 워크플로우를 수행할 경우에는 콘솔에서 워크플로우의 구성 지침을 수행하고 다음 절차는 무시하십시오.

AWS IoT 엔드포인트를 구성하려면

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

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

AWS IoT 엔드포인트가 Endpoint(엔드포인트)에 표시됩니다. URL은 <1234567890123>-ats.iot.<us-east-1>.amazonaws.com과 같아야 합니다. 이 엔드포인트를 기록해둡니다.

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

디바이스에 AWS IoT 사물 이름이 있어야 합니다. 이 이름을 기록해둡니다.

4. /demos/include/aws_clientcredential.h를 업니다.
5. 다음 상수의 값을 지정합니다.

- #define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
- #define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"

Wi-Fi를 구성하려면

보드를 Wi-Fi 연결을 통해 인터넷에 연결하는 경우 Amazon FreeRTOS에 네트워크에 연결할 Wi-Fi 자격 증명을 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 ##### ## ##

유료한 보안 유형은 다음과 같습니다.

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

AWS IoT 자격 증명의 형식을 지정하려면

Amazon FreeRTOS는 등록된 사물과 연결된 AWS IoT 인증서 및 프라이빗 키와 디바이스 대신 AWS IoT와 성공적으로 통신할 수 있는 권한 정책이 필요합니다.

Note

AWS IoT 자격 증명을 구성하려면 디바이스를 등록할 때 AWS IoT 콘솔에서 다운로드한 프라이빗 키와 인증서가 필요합니다. 디바이스를 AWS IoT 사물로 등록한 후 AWS IoT 콘솔에서 디바이스 인증서를 검색할 수 있지만 프라이빗 키를 검색할 수 없습니다.

Amazon 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)에서 AWS IoT 콘솔에서 다운로드한 `<ID>-certificate.pem.crt`를 선택합니다.
3. Private Key PEM file(프라이빗 키 PEM 파일)에서 AWS IoT 콘솔에서 다운로드한 `<ID>-private.pem.key`를 선택합니다.
4. Generate and save aws_clientcredential_keys.h(aws_clientcredential_keys.h 생성 및 저장)를 선택한 다음 파일을 `demos/include`에 저장합니다. 이 파일은 디렉터리의 기존 파일을 덮어씁니다.

Note

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

Amazon FreeRTOS를 구성했으면 해당 보드에 대한 시작 안내서를 참조하여 Amazon FreeRTOS 데모를 컴파일하고 실행할 수 있습니다. 시작하기 자습서에서 사용하는 데모 애플리케이션은 Hello World MQTT 데모이며 `demos/mqtt/aws_hello_world.c`에 있습니다.

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 [Amazon FreeRTOS 데모 구성 \(p. 66\)](#)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 [Amazon FreeRTOS 데모 구성 \(p. 66\)](#)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

[보드별 시작 안내서 \(p. 76\)](#) 의 보드 가이드에 설명된 대로 Hello World MQTT 데모를 엽니다.
프로젝트에서 `aws_dev_mode_key_provisioning.c` 파일을 열고 기본적으로 0으로 설정된
`keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR`의 정의를 1로 변경합니다.

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

그런 다음 데모 프로젝트를 빌드 및 실행하고 다음 단계를 계속합니다.

퍼블릭 키 추출

디바이스가 아직 프라이빗 키 및 클라이언트 인증 서로 프로비저닝되지 않았으므로, 데모에서 AWS IoT에 대한 인증이 실패합니다. 그러나 Hello World MQTT 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 키 쌍을 생성하며 이전 단계를 반복해야 합니다.

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

퍼블릭 키 인프라 설정

[CA 인증서 등록](#)의 지침에 따라 디바이스 랩 인증서에 대한 인증서 계층 구조를 만듭니다. CA 인증서를 사용하여 디바이스 인증서 생성 섹션에 설명된 시퀀스를 실행하기 전에 종지합니다.

이 경우 ROM 크기를 줄이기 위해 CSR을 생성하고 서명하는데 필요한 X.509 인코딩 로직이 Amazon 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 인증서를 발급하도록 인증 기관을 구성하는 경우 동일한 메커니즘을 프로덕션에서 사용할 수 있습니다. 이 키는 지정된 관리자의 통제 하에 있어야 합니다.

인증서 가져오기

인증서를 발급한 후 다음 단계는 인증서를 디바이스로 가져오는 것입니다. 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

[자체 인증서 사용](#)에 설명된 대로 `deviceCert.pem`을 AWS IoT 레지스트리로 가져옵니다. 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 MQTT Hello World 데모를 다시 실행합니다. 이제 AWS IoT MQTT 브로커와 성공적으로 연결됩니다.

시작하기 문제 해결

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

주제

- [일반 시작하기 문제 해결 도움말 \(p. 69\)](#)
- [터미널 에뮬레이터 설치 \(p. 69\)](#)

보드별 문제 해결은 해당 보드의 [Amazon FreeRTOS 시작하기 \(p. 61\)](#) 설명서를 참조하십시오.

일반 시작하기 문제 해결 도움말

- Hello World 데모 프로젝트를 실행한 후 AWS IoT 콘솔에 메시지가 표시되지 않으면 다음과 같이 해보십시오.
 - 터미널 창을 열고 샘플을 로깅 출력을 봅니다. 이 단계는 무엇이 잘못되었는지 확인하는 데 도움이 됩니다.
 - 자격 증명이 유효한지 확인합니다.

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	없음
증지	1비트
흐름 제어	없음

보드의 직렬 포트 찾기

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

Windows

```
chgport
```

Linux

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

macOS

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

CMake와 Amazon FreeRTOS 사용

CMake를 사용하여 Amazon FreeRTOS 애플리케이션 소스 코드에서 프로젝트 빌드 파일을 생성할 수 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

Amazon FreeRTOS와 함께 CMake를 사용하는 방법에 대한 자세한 내용은 아래 주제를 참조하십시오.

주제

- [사전 조건 \(p. 71\)](#)
- [타사 코드 편집기 및 디버깅 도구를 사용하여 Amazon FreeRTOS 애플리케이션 개발 \(p. 71\)](#)
- [CMake를 사용하여 Amazon FreeRTOS 빌드 \(p. 72\)](#)

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 사용할 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 Amazon FreeRTOS 애플리케이션 개발

코드 편집기, 디버깅 확장 또는 타사 디버깅 도구를 사용하여 Amazon 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 프로젝

트를 빌드할 수 있습니다. CMake를 사용한 Amazon FreeRTOS 애플리케이션 빌드에 대한 자세한 내용은 [CMake를 사용하여 Amazon FreeRTOS 빌드 \(p. 72\)](#) 단원을 참조하십시오.

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"
  }
]
```

CMake를 사용하여 Amazon FreeRTOS 빌드

CMake는 기본적으로 호스트 운영 체제를 대상 시스템으로 사용합니다. CMake를 교차 컴파일에 사용하려면 도구 체인 파일이 필요합니다. 이 파일은 사용할 컴파일러를 지정합니다. Amazon FreeRTOS에서는 [<amazon-freeertos>/tools/cmake/toolchains](#)에 기본 도구 체인 파일을 제공합니다. 이 파일을 CMake에 제공하는 방법은 CMake 명령줄 인터페이스를 사용하는지 또는 GUI를 사용하는지에 따라 달립니다. 추가 세부 정보는 아래 [빌드 파일 생성\(CMake 명령줄 도구\) \(p. 72\)](#) 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.

Amazon FreeRTOS 빌드 파일 생성에 대한 자세한 내용은 [빌드 파일 생성\(CMake 명령줄 도구\) \(p. 72\)](#) 및 [빌드 파일 생성\(CMake GUI\) \(p. 73\)](#) 단원을 참조하십시오.

2. 네이티브 빌드 시스템을 호출하여 프로젝트를 실행 파일로 만듭니다.

Amazon FreeRTOS 빌드 파일 생성에 대한 자세한 내용은 [생성된 빌드 파일에서 Amazon FreeRTOS 빌드 \(p. 75\)](#) 단원을 참조하십시오.

빌드 파일 생성(CMake 명령줄 도구)

CMake 명령줄 도구(cmake)를 사용하여 Amazon FreeRTOS에 대한 빌드 파일을 생성할 수 있습니다. 빌드 파일을 생성하려면 대상 보드, 컴파일러, 소스 코드와 빌드 디렉터리의 위치를 지정해야 합니다. -DVENDOR 옵션을 사용하여 대상 보드를 지정합니다. -DCOMPILER 옵션을 사용하여 컴파일러를 지정합니다. -S 옵션을 사용하여 소스 코드의 위치를 지정합니다. -B 옵션을 사용하여 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 디렉터리로 소스 파일을 빌드할 수 있습니다.

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

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 -G Ninja
```

또는

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build -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 이를 일부만 사용해도 됩니다. 예를 들어, 다음 명령은 동일한 소스에서 동일한 빌드 파일을 생성합니다.

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

```
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -S . -B build
```

```
cmake -DVENDOR=t -DBOARD=cc -DCOMPILER=arm-ti -S . -B build
```

기본 디렉터리 cmake/toolchains에 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
```

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
```

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

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

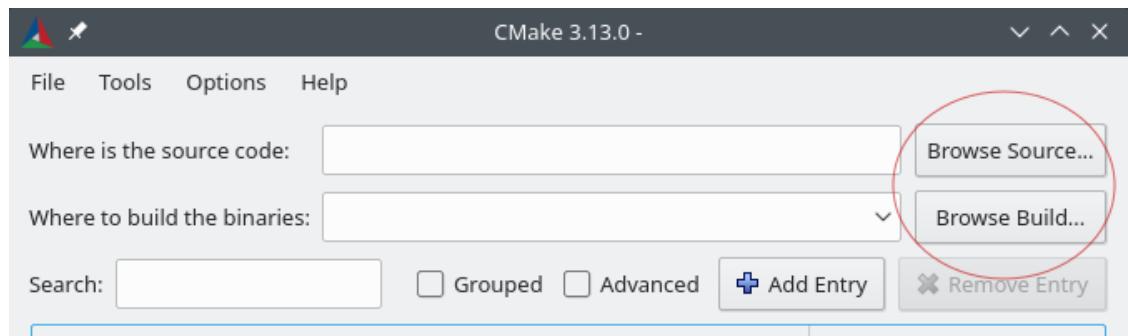
CMAKE_BUILD_TYPE을 release로 설정하여 컴파일 옵션에 최적화 플래그를 추가할 수도 있습니다.

빌드 파일 생성(CMake GUI)

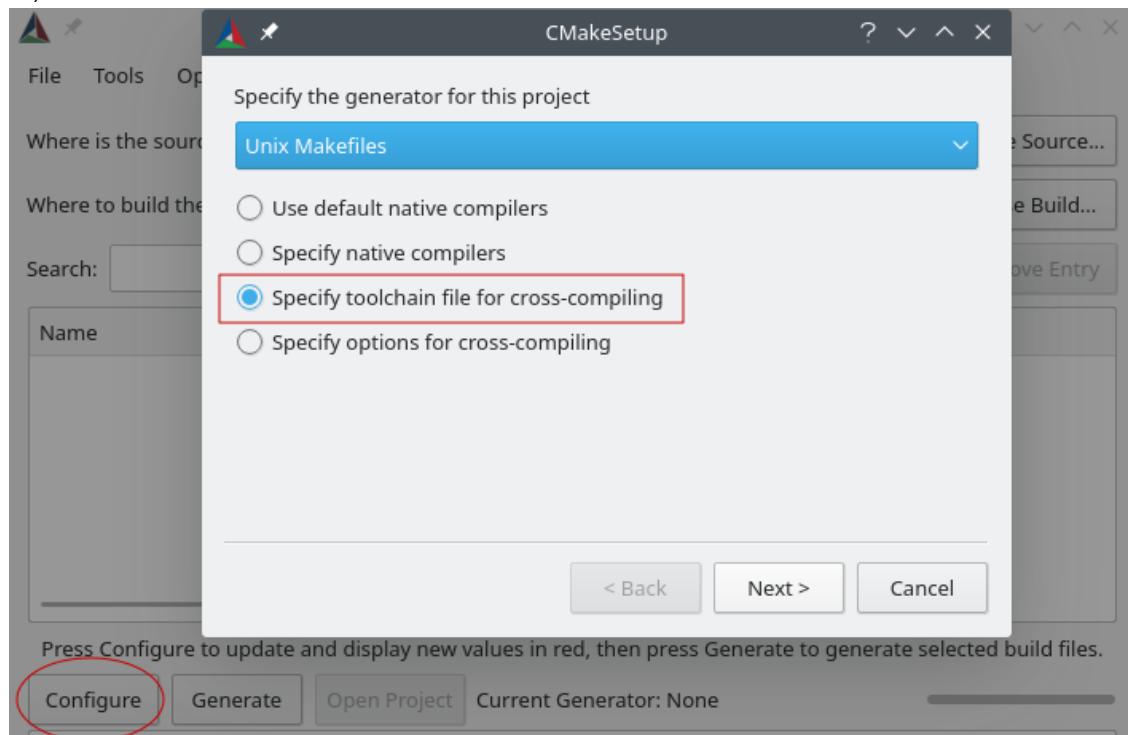
CMake GUI를 사용하여 Amazon FreeRTOS 빌드 파일을 생성할 수 있습니다.

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. 구성 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 File(파일) 메뉴에서 Delete Cache(캐시 삭제)를 선택하여 CMake 캐시를 삭제합니다.



4. Specify toolchain file for cross-compiling(교차 컴파일을 위한 도구 체인 파일 지정)을 선택하고 다음을 선택합니다.
5. 도구 체인 파일(예: <amazon-freertos>/tools/cmake/toolchains/arm-ti.cmake)을 선택하고 완료를 선택합니다.

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

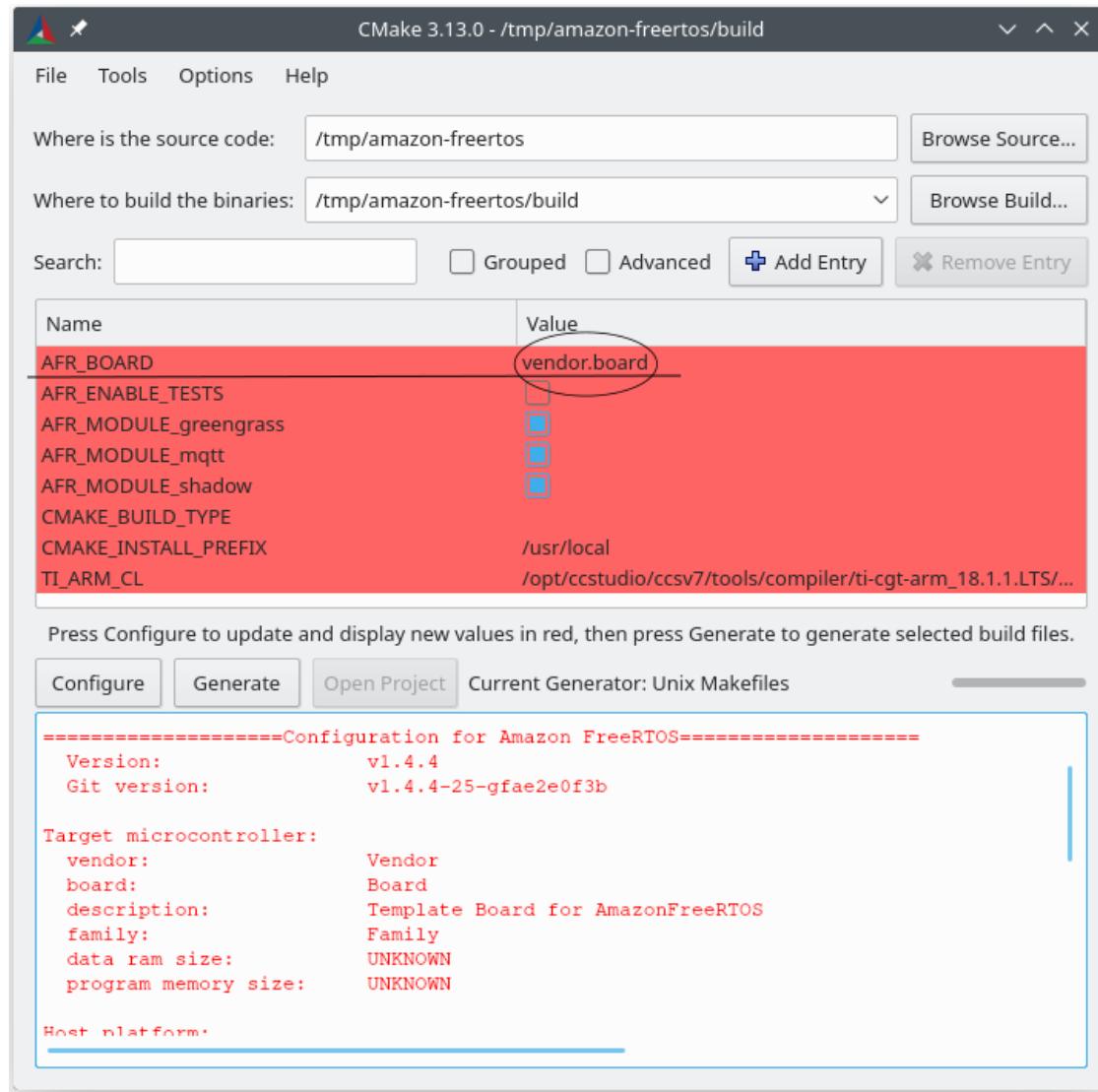
Note

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 Amazon FreeRTOS 빌드

네이티브 빌드 시스템을 사용하여 빌드

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

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

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

CMake를 사용하여 빌드

CMake 명령줄 도구를 사용하여 Amazon 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.

보드별 시작 안내서

[첫 번째 단계 \(p. 62\)](#)를 완료한 후 Amazon FreeRTOS 시작하기에 대한 보드별 지침은 해당 보드의 안내서를 참조하십시오.

- Cypress CYW943907AEVAL1F 개발 키트 시작하기 (p. 77)
- Cypress CYW954907AEVAL1F 개발 키트 시작하기 (p. 80)
- Infineon XMC4800 IoT Connectivity Kit 시작하기 (p. 101)
- Marvell MW320 AWS IoT 시작 키트 시작 (p. 110)
- Marvell MW322 AWS IoT 시작 키트 시작 (p. 120)
- MediaTek MT7697Hx 개발 키트에서 시작하기 (p. 131)
- Microchip Curiosity PIC32MZ EF 시작하기 (p. 135)
- Nuvoton NuMaker-IoT-M487 시작하기 (p. 141)
- NXP LPC54018 IoT 모듈 시작하기 (p. 147)
- Renesas Starter Kit+ for RX65N-2MB 시작하기 (p. 150)
- STMicroelectronics STM32L4 Discovery Kit IoT Node 시작하기 (p. 153)
- Texas Instruments CC3220SF-LAUNCHXL 시작하기 (p. 155)
- Windows Device Simulator 시작하기 (p. 159)
- Xilinx Avnet MicroZed Industrial IoT Kit 시작하기 (p. 162)

Note

다음 Amazon FreeRTOS 시작하기 자습 안내서에서 [첫 번째 단계 \(p. 62\)](#)는 수행할 필요가 없습니다.

- Windows 시뮬레이터와 Microchip ECC608a 보안 요소 시작하기 (p. 83)
- Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 시작하기 (p. 86)
- Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트 시작하기 (p. 105)
- Nordic nRF52840-DK 시작하기 (p. 138)

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 MQTT Hello World 데모를 설정하고 실행하는 단계를 안내합니다. 현재, 이 보드의 Amazon FreeRTOS 포트는 TCP 서버와 클라이언트 데모를 지원하지 않습니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <[amazon-freeertos](#)>라고 합니다.

Important

Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. Amazon FreeRTOS 프로젝트에서 파일을 수용하려면 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로가 43자 이하여야 합니다. [Amazon FreeRTOS 다운로드 \(p. 65\)](#)에서 언급했듯이 Cypress용 Amazon FreeRTOS 포트는 현재 [GitHub](#)에서만 사용할 수 있습니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

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

WICED Studio SDK 다운로드 및 설치

이 시작 안내서에서는 Cypress WICED Studio SDK를 사용하여 Amazon 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

- 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 <amazon-freertos>/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

직렬 연결 설정

호스트 시스템과 보드 간의 직렬 연결을 설정하려면

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

터미널 설치 및 직렬 연결 설정에 대한 자세한 내용은 [터미널 에뮬레이터 설치 \(p. 69\)](#)를 참조하십시오.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

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

WICED Studio에서 Amazon FreeRTOS 데모 프로젝트를 빌드하고 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. Select root directory(루트 디렉터리 선택)에서 Browse...(찾아보기...)를 선택하고 경로 <amazon-freertos>/projects/cypress/CYW943907AEVAL1F/wicedstudio로 이동한 후 OK(확인)를 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.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

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

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

이 오류 문제를 해결하려면 다음을 수행합니다.

1. `WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\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
```

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

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 MQTT Hello World 데모를 설정하고 실행하는 단계를 안내합니다. 현재, 이 보드의 Amazon FreeRTOS 포트는 TCP 서버와 클라이언트 데모를 지원하지 않습니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

Important

Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. Amazon FreeRTOS 프로젝트에서 파일을 수용하려면 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로가 43자 이하여야 합니다.

[Amazon FreeRTOS 다운로드 \(p. 65\)](#)에서 언급했듯이 Cypress용 Amazon FreeRTOS 포트는 현재 [GitHub](#)에서만 사용할 수 있습니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

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

WICED Studio SDK 다운로드 및 설치

이 시작 안내서에서는 Cypress WICED Studio SDK를 사용하여 Amazon 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 미너 버전 번호입니다. 이 문서에서는 `WICED-Studio-6.2`의 폴더 이름을 가정하지만, 설치한 버전의 정확한 이름을 사용해야 합니다. `WICED_STUDIO_SDK_PATH`

환경 변수를 정의할 때 WICED Studio IDE의 설치 경로가 아닌, WICED Studio SDK의 전체 설치 경로를 지정해야 합니다. Windows 및 macOS에서는 SDK의 **wiced-studio-m.n** 폴더가 기본적으로 Documents 폴더에 생성됩니다.

Windows에서 환경 변수를 만들려면

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

Linux 또는 macOS에서 환경 변수를 만들려면

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

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

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

```
cd <amazon-freertos>/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

직렬 연결 설정

호스트 시스템과 보드 간의 직렬 연결을 설정하려면

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

터미널 설치 및 직렬 연결 설정에 대한 자세한 내용은 [터미널 에뮬레이터 설치 \(p. 69\)](#)를 참조하십시오.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

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

WICED Studio에서 Amazon FreeRTOS 데모 프로젝트를 빌드하고 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. Select root directory(루트 디렉터리 선택)에서 Browse...(찾아보기...)를 선택하고 <amazon-freertos>/projects/cypress/CYW954907AEVAL1F/wicedstudio 경로로 이동한 후 OK(확인)를 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.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

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

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

이 오류 문제를 해결하려면 다음을 수행합니다.

1. `WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\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
```

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Windows 시뮬레이터와 Microchip ECC608a 보안 요소 시작하기

이 자습서에서는 Windows 시뮬레이터와 Microchip ECC608a 보안 요소를 시작하기 위한 지침을 제공합니다.

다음 하드웨어가 필요합니다.

- [Microchip ECC608a 보안 요소 클릭보드](#)
- [SAMD21 XPlained Pro](#)
- [mikroBUS Xplained Pro 어댑터](#)

시작하기 전에 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 이 자습서에서 Amazon FreeRTOS 다운로드 딜렉토리에 대한 경로는 [`<amazon-freeertos>`](#)라고 합니다.

개요

이 자습서에 포함된 단계는 다음과 같습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. Amazon FreeRTOS 데모 애플리케이션을 이진 이미지로 교차 컴파일합니다.
4. 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.

Microchip ECC608a 하드웨어 설정

Microchip ECC608a 디바이스와 상호 작용하려면 먼저 SAMD21을 프로그래밍해야 합니다.

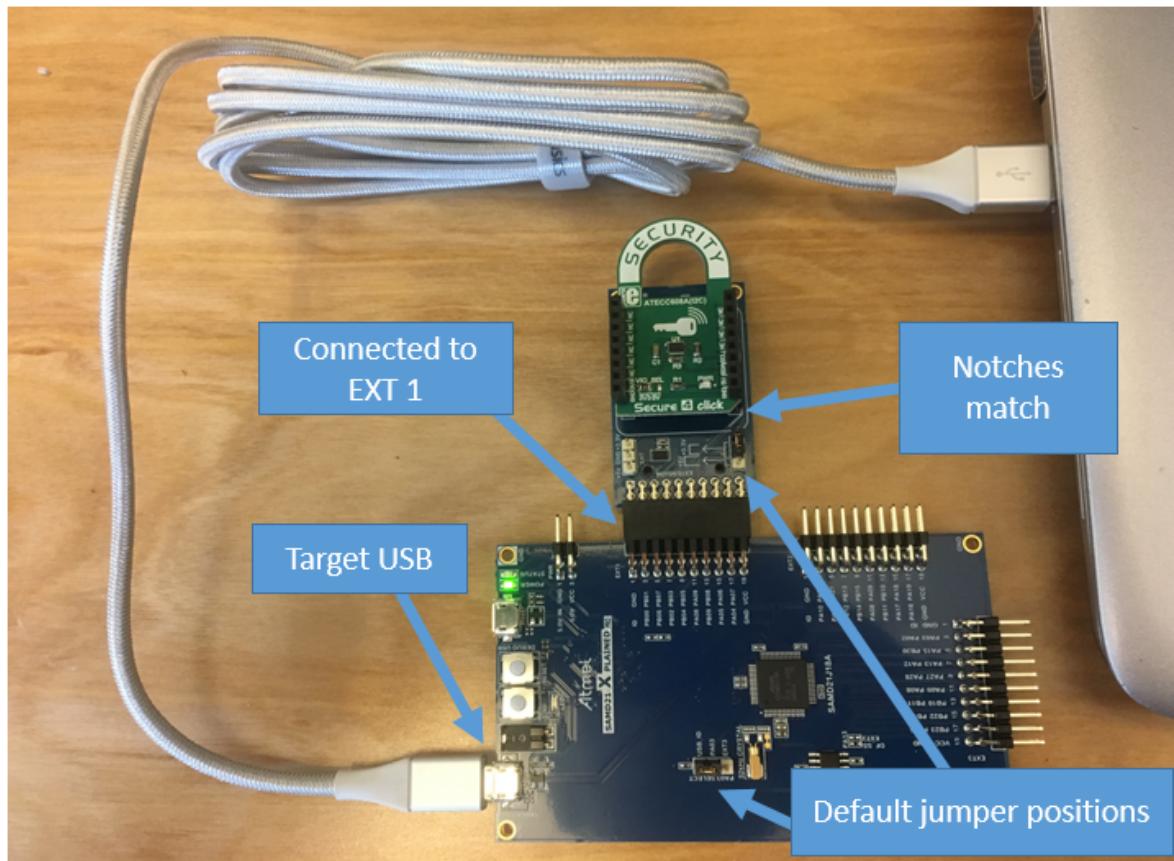
SAMD21 XPlained Pro 보드를 설정하려면

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 mikroBUSX XPlained Pro 어댑터에 연결합니다. 클릭 보드의 노치 모서리가 어댑터 보드의 노치 아이콘과 일치해야 합니다.
4. 마이크로 USB 케이블을 대상 USB에 연결합니다.

설정이 다음과 같아야 합니다.



개발 환경 설정

1. 아직 없는 경우 [AWS 계정을 생성](#)하십시오. IAM 사용자를 AWS 계정에 추가하려면 [IAM 사용 설명서](#)를 참조하십시오.

IAM 사용자 계정에 AWS IoT 및 Amazon FreeRTOS에 대한 액세스 권한을 부여하려면 다음 단계에서 IAM 사용자 계정에 다음 IAM 정책을 연결합니다.

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

2. [AmazonFreeRTOSFullAccess](#) 정책을 IAM 사용자에게 연결합니다.

- a. [IAM 콘솔](#)로 이동하고 탐색 창에서 사용자를 선택합니다.
- b. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
- c. 권한 추가를 선택합니다.
- d. [Attach existing policies directly]를 선택합니다.
- e. 검색 상자에 [AmazonFreeRTOSFullAccess](#)를 입력하고 목록에서 해당 항목을 선택한 다음 Next: Review(다음: 검토)를 선택합니다.
- f. 권한 추가를 선택합니다.

3. [AWSIoTFullAccess](#) 정책을 IAM 사용자에게 연결합니다.

- a. [IAM 콘솔](#)로 이동하고 탐색 창에서 사용자를 선택합니다.
- b. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
- c. 권한 추가를 선택합니다.

- d. [Attach existing policies directly]를 선택합니다.
- e. 검색 상자에 **AWSIoTFullAccess**를 입력하고 목록에서 해당 항목을 선택한 다음 Next: Review(다음: 검토)를 선택합니다.
- f. 권한 추가를 선택합니다.

IAM에 대한 자세한 내용은 [IAM 사용 설명서](#)의 [IAM 권한 및 정책](#)을 참조하십시오.

4. [Amazon FreeRTOS GitHub 리포지토리](#)에서 Amazon FreeRTOS 리포지토리를 다운로드합니다.

GitHub에서 Amazon FreeRTOS를 다운로드하려면:

1. [Amazon FreeRTOS GitHub 리포지토리](#)를 찾아봅니다.
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
```

4. <[amazon-freertos](#)> 디렉터리에서 사용할 분기를 체크아웃합니다.

Note

Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. Amazon FreeRTOS 다운로드 디렉터리 경로가 길면 빌드 오류가 발생할 수 있습니다.

시작하기 설명서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

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.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Important

Microchip ECC608a 디바이스에는 `c_InitToken` 호출 동안 처음 프로젝트가 실행될 때 디바이스에 잠기는 초기화가 한 회만 됩니다. 그러나 Amazon 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 Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행하려면

1. Visual Studio에서 프로젝트를 로드합니다.

파일 메뉴에서 열기를 선택합니다. 파일/솔루션을 선택하고 <[amazon-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.

Build(빌드) 메뉴에서 Build Solution(솔루션 빌드)을 선택하고 솔루션이 오류 없이 빌드되는지 확인합니다. Debug(디버깅), Start Debugging(디버깅 시작)을 선택하여 프로젝트를 실행합니다. 첫 번째 실행에서 디바이스 인터페이스를 구성하고 다시 컴파일해야 합니다. 자세한 내용은 [네트워크 인터페이스 구성 \(p. 161\)](#) 단원을 참조하십시오.

4. Microchip ECC608a를 프로비저닝합니다.

マイクロチップ은 ATECC608a 부품 설정에 도움이 되는 여러 스크립팅 도구를 제공합니다. <[amazon-freertos](#)>\vendors\microchip\secure_elements\app\example_trust_chain_tool로 이동하여 README.md 파일을 업니다.

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

1. AWS에 인증 기관을 생성하고 등록합니다.
2. Microchip ECC608a에 키를 생성하고 퍼블릭 키 및 디바이스 일련 번호를 내보냅니다.
3. 디바이스에 대한 인증서를 생성하고 해당 인증서를 AWS에 등록합니다.
4. CA 인증서 및 디바이스 인증서를 디바이스에 로드합니다.
5. Amazon FreeRTOS 샘플을 빌드하고 실행합니다.

데모 프로젝트를 다시 실행합니다. 이제 연결되어야 합니다!

문제 해결

일반적인 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 시작하기

이 자습서에서는 ESP32-WROOM-32, ESP32-SOLO-1 또는 ESP-WROVER 모듈과 ESP-WROVER-KIT-VB가 장착된 Espressif ESP32-DevKitC를 시작하기 위한 지침을 제공합니다. AWS Partner Device Catalog에 기재된 파트너로부터 구매하려면 [ESP32-WROOM-32 DevKitC](#), [ESP32-SOLO-1](#) 또는 [ESP32-WROVER-KIT](#) 링크를 사용하십시오. 이 개발 보드 버전이 Amazon FreeRTOS에서 지원됩니다. 이러한 보드에 대한 자세한 정보는 Espressif 웹 사이트에서 [ESP32-DevKitC](#) 또는 [ESP-WROVER-KIT](#)를 참조하십시오.

Note

현재 ESP32-WROVER-KIT 및 ESP DevKitC용 Amazon FreeRTOS 포트는 다음 기능을 지원하지 않습니다.

- 대칭 다중 처리(SMP).

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. Amazon 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 Amazon FreeRTOS를 시작하려면 먼저 AWS 계정과 권한을 설정해야 합니다.

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

IAM 사용자를 AWS 계정에 추가하려면 [IAM 사용 설명서](#)를 참조하십시오. AWS IoT 및 Amazon FreeRTOS에 대한 액세스 권한을 IAM 사용자 계정에 부여하려면 다음 IAM 정책을 IAM 사용자 계정에 연결합니다.

- `AmazonFreeRTOSFullAccess`
- `AWSIoTFullAccess`

`AmazonFreeRTOSFullAccess` 정책을 IAM 사용자에게 연결하려면

1. [IAM 콘솔](#)로 이동하고 탐색 창에서 사용자를 선택합니다.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 `AmazonFreeRTOSFullAccess`를 입력하고 목록에서 해당 항목을 선택한 다음 Next: Review(다음: 검토)를 선택합니다.
6. 권한 추가를 선택합니다.

`AWSIoTFullAccess` 정책을 IAM 사용자에게 연결하려면

1. [IAM 콘솔](#)로 이동하고 탐색 창에서 사용자를 선택합니다.
2. 검색 텍스트 상자에 사용자 이름을 입력한 다음 목록에서 해당 이름을 선택합니다.
3. 권한 추가를 선택합니다.
4. [Attach existing policies directly]를 선택합니다.
5. 검색 상자에 `AWSIoTFullAccess`를 입력하고 목록에서 해당 항목을 선택한 다음 Next: Review(다음: 검토)를 선택합니다.
6. 권한 추가를 선택합니다.

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

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

Espressif 하드웨어 설정

ESP32-DevKitC 개발 보드 하드웨어 설정에 대한 정보는 [ESP32-DevKitC 시작 안내서](#)를 참조하십시오.

ESP-WROVER-KIT 개발 보드 하드웨어 설정에 대한 정보는 [ESP-WROVER-KIT 시작 안내서](#)를 참조하십시오.

Note

Espressif 안내서의 시작하기 섹션으로 진행하지 마십시오. 대신 아래 단계를 따르십시오.

개발 환경 설정

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다.

도구 체인 설정

도구 체인을 설정하려면 호스트 시스템의 운영 체제별 지침을 따릅니다.

Note

다음 단계에서 "Get ESP-IDF" 지침에 이르면 중지하고 이 페이지의 지침으로 돌아오십시오. 이전에 "Get ESP-IDF" 지침을 수행하고 ESP-IDF를 설치했으면 계속하기 전에 시스템에서 `IDF_PATH` 환경 변수를 지워야 합니다.

- [Windows용 도구 체인의 표준 설정](#)
- [macOS용 도구 체인의 표준 설정](#)
- [Linux용 도구 체인의 표준 설정](#)

Note

ESP-IDF 버전 3.3(Amazon FreeRTOS에서 지원되는 버전)은 ESP32 컴파일러의 최신 버전을 지원하지 않습니다. ESP-IDF 버전 3.3과 호환되는 파일을 사용해야 합니다(위의 링크 참조). 컴파일러의 버전을 확인하려면 "`xtensa-esp32-elf-gcc --version`"을 실행합니다.

CMake 설치

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

[CMake.org](#)에서 CMake의 최신 버전을 다운로드할 수 있습니다. 소스 및 이진 배포를 둘 다 사용할 수 있습니다.

Amazon FreeRTOS와 함께 CMake를 사용하는 방법에 대한 자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

직렬 연결 설정

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

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

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.

Amazon FreeRTOS 다운로드 및 구성

환경이 설정된 후 [GitHub](#) 또는 [Amazon FreeRTOS 콘솔](#)에서 Amazon FreeRTOS를 다운로드할 수 있습니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.

Amazon FreeRTOS 데모 애플리케이션 구성

- macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행하는 경우 mingw32.exe를 엽니다.
- Python 2.7.10 이상이 설치되어 있는지 확인하려면 python --version을 실행합니다. 설치된 버전이 표시됩니다. 컴퓨터에 Python 2.7.10 이상이 설치되어 있지 않으면 [Python 웹 사이트](#)에서 설치할 수 있습니다.
- AWS IoT 명령을 실행하려면 AWS CLI가 필요합니다. Windows를 실행하는 경우 easy_install awscli를 사용하여 AWS CLI를 mingw32 환경에 설치합니다.

macOS 또는 Linux를 실행하는 경우 [AWS 명령줄 인터페이스 설치](#)를 참조하십시오.

- aws configure를 실행하고 AWS 액세스 키 ID, 보안 액세스 키 및 기본 리전 이름으로 AWS CLI를 구성합니다. 자세한 내용은 [AWS CLI 구성](#)을 참조하십시오.
- 다음 명령을 사용하여 AWS SDK for Python(boto3)을 설치합니다.
 - Windows에서는 mingw32 환경에서 easy_install boto3을 실행합니다.
 - macOS 또는 Linux에서 pip install tornado nose --user를 실행한 후 pip install boto3 --user를 실행합니다.

Amazon FreeRTOS에는 AWS IoT에 연결하기 위해 Espressif 보드를 더 쉽게 설정할 수 있게 해주는 SetupAWS.py 스크립트가 포함되어 있습니다. 스크립트를 구성하려면 [`<amazon-freertos>/tools/aws_config_quick_start/configure.json`](#)을 열고 다음 속성을 설정합니다.

`afr_source_dir`

컴퓨터의 [`<amazon-freertos>`](#) 디렉터리에 대한 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)

구성 스크립트를 실행하려면

- macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행하는 경우 mingw32.exe를 엽니다.
- [`<amazon-freertos>/tools/aws_config_quick_start`](#) 디렉터리로 이동한 후 python SetupAWS.py setup을 실행합니다.

이 스크립트는 다음 작업을 수행합니다.

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

`SetupAWS.py`에 대한 자세한 내용은 <[amazon-freertos](#)>/tools/aws_config_quick_start 디렉터리에 있는 README.md를 참조하십시오.

Amazon FreeRTOS 데모 프로젝트 빌드, 플래시 및 실행

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

Amazon FreeRTOS 빌드

(Windows를 사용하는 경우 다음 단원을 참조하십시오.)

CMake를 사용하여 빌드 파일을 생성한 다음 Make를 사용하여 애플리케이션을 빌드합니다.

CMake를 사용하여 데모 애플리케이션의 빌드 파일을 생성하려면

1. Amazon FreeRTOS 다운로드 디렉터리의 루트로 디렉터리를 변경합니다.
2. 빌드 파일을 생성할 때는 다음 명령을 사용합니다.

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

Note

디버깅을 위해 애플리케이션을 빌드하려면 이 명령에 `-DCMAKE_BUILD_TYPE=Debug` 플래그를 추가합니다.

테스트 애플리케이션 빌드 파일을 생성하려면 `-DAFR_ENABLE_TESTS=1` 플래그를 추가합니다.

Lightweight IP(LwIP) 지원을 추가하려면 `-DAFR_ESP_LWIP=1` 플래그를 추가합니다.

Make를 사용하여 애플리케이션을 빌드하려면

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

```
make all -j4
```

Note

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

Windows에서 Amazon FreeRTOS 빌드

Windows에서는 CMake에 대한 빌드 생성기를 지정해야 합니다. 그렇지 않으면 CMake에서 기본적으로 Visual Studio로 지정됩니다. Espressif는 Windows, Linux 및 MacOS에서 작동하는 Ninja 빌드 시스템을 공식

적으로 권장합니다. cmd 또는 powershell과 같은 기본 Windows 환경에서 CMake 명령을 실행해야 합니다. MSYS2 또는 WSL과 같은 가상 Linux 환경에서는 CMake 명령 실행이 지원되지 않습니다.

CMake를 사용하여 빌드 파일을 생성한 다음 Make를 사용하여 애플리케이션을 빌드합니다.

CMake를 사용하여 데모 애플리케이션의 빌드 파일을 생성하려면

1. Amazon FreeRTOS 다운로드 디렉터리의 루트로 디렉터리를 변경합니다.
2. 빌드 파일을 생성할 때는 다음 명령을 사용합니다.

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

Note

디버깅을 위해 애플리케이션을 빌드하려면 이 명령에 -DCMAKE_BUILD_TYPE=Debug 플래그를 추가합니다.

테스트 애플리케이션 빌드 파일을 생성하려면 -DAFR_ENABLE_TESTS=1 플래그를 추가합니다.

애플리케이션을 빌드하려면

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

```
ninja
```

또는 일반 CMake 인터페이스를 사용하여 애플리케이션을 빌드합니다.

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

Note

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

Amazon FreeRTOS 플래시 및 실행

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

보드의 플래시를 지우려면 [amazon-freertos](#) 디렉터리로 이동하여 다음 명령을 사용하십시오.

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

애플리케이션 바이너리를 보드에 플래시하려면 make를 사용하십시오.

```
make flash
```

IDF 스크립트를 사용하여 보드를 플래시할 수도 있습니다.

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

모니터링하려면 다음 명령을 사용합니다.

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

Note

이러한 명령을 결합할 수 있습니다. 예:

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

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

Bluetooth Low-Energy 데모 실행

Amazon FreeRTOS는 [Bluetooth Low Energy](#) 연결을 지원합니다.

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

Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 설정하려면

1. [Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK](#)에 있는 지침에 따라 호스트 컴퓨터에 모바일 플랫폼용 SDK를 다운로드하고 설치합니다.
2. [Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션](#)에 있는 지침에 따라 모바일 디바이스에 데모 모바일 애플리케이션을 설정합니다.

보드에서 MQTT over Bluetooth Low Energy 데모를 실행하는 방법에 대한 자세한 내용은 [MQTT over Bluetooth Low Energy 데모 애플리케이션](#)을 참조하십시오.

보드에서 Wi-Fi 프로비저닝 데모를 실행하는 방법에 대한 자세한 내용은 [Wi-Fi 프로비저닝 데모 애플리케이션](#)을 참조하십시오.

ESP32를 위한 자체 CMake 프로젝트에서 Amazon FreeRTOS 사용

자체 CMake 프로젝트에서 Amazon FreeRTOS를 사용하려면 하위 디렉터리로 설정하고 애플리케이션과 함께 빌드할 수 있습니다. 먼저 [GitHub](#), 또는 [Amazon FreeRTOS 콘솔](#)에서 Amazon FreeRTOS 복사본을 가져옵니다. 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 amazon-freertos
```

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

```
# Pull the latest changes from the remote tracking branch.  
git submodule update --remote -- amazon-freertos  
# Commit the submodule change because it is pointing to a different revision now.  
git add amazon-freertos  
git commit -m "Update Amazon FreeRTOS to a new release"
```

프로젝트에 다음과 같은 디렉터리 구조가 있다고 가정합니다.

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

다음은 Amazon FreeRTOS와 함께 애플리케이션을 빌드하는 데 사용할 수 있는 최상위 CMakeLists.txt 파일의 예입니다.

```
cmake_minimum_required(VERSION 3.13)  
project(freertos_examples)  
add_executable(my_app src/main.c)  
# Tell IDF build to link against this target.  
set(IDF_PROJECT_EXECUTABLE my_app)  
  
# Add amazon freertos as a subdirectory. AFR_BOARD tells which board to target.  
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")  
add_subdirectory(amazon-freertos)  
  
# Link against the mqtt library so that we can use it. Dependencies are transitively  
# linked.  
target_link_libraries(my_app PRIVATE AFR::mqtt)
```

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

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

애플리케이션을 보드에 플래시하려면 다음을 실행합니다.

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

Amazon FreeRTOS에서 구성 요소 사용

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

```
=====Configuration for Amazon FreeRTOS=====  
Version: 201910.00  
Git version: 201910.00-388-gcb3612cb7  
  
Target microcontroller:  
  vendor: Espressif  
  board: ESP32-DevKitC
```

```

description: Development board produced by Espressif that comes in two
variants either with ESP-WROOM-32 or ESP32-WROVER module

family: ESP32
data ram size: 520KB
program memory size: 4MB

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

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

```

"빌드 할 모듈" 목록에서 모든 구성 요소를 참조할 수 있습니다. 애플리케이션에 연결하려면 이름 앞에 AFR::네임스페이스를 추가합니다(예: AFR::mqtt, AFR::ota, 등).

ESP-IDF에 사용자 지정 구성 요소 추가

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

```

- amazon-freertos
- components
  - foo
    - include
      - foo.h
    - src
      - foo.c
      - CMakeLists.txt
- src
  - main.c
- CMakeLists.txt

```

다음은 구성 요소의 CMakeLists.txt 파일의 예입니다.

```

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

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

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

```

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

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

ESP 구성 요소의 경우 `COMPONENT_REQUIRES` 및 `COMPONENT_PRIV_REQUIRES` 변수를 설정하여 수행됩니다. ESP-IDF Programming Guide의 [When Writing a Component\(구성 요소 작성 시\)](#)를 참조하십시오.

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

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

```
# 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/foo" ABSOLUTE  
)  
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})
```

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

Amazon FreeRTOS에 대한 구성 재정의

현재 Amazon FreeRTOS 소스 트리 외부에서 config를 재정의하도록 잘 정의된 접근 방식은 없습니다. 기본적으로 CMake는 `<amazon-freertos>/vendors/espressif/boards/esp32/aws_demos/config_files/` 및 `<amazon-freertos>/demos/include/` 디렉터리를 찾습니다. 그러나 해결 방법을 사용하여 컴파일러가 다른 디렉터리를 먼저 검색하도록 지시할 수 있습니다. 예를 들어, Amazon FreeRTOS 구성에 다른 폴더를 추가할 수 있습니다.

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

`amazon-freertos-configs`의 파일은 `<amazon-freertos>/vendors/espressif/boards/esp32/aws_demos/config_files/` 및 `<amazon-freertos>/demos/include/i` 디렉터리에서 복사됩니다. 그런 다음 최상위 `CMakeLists.txt` 파일에서 컴파일러가 이 디렉터리를 먼저 검색하도록 `add_subdirectory(amazon-freertos)` 앞에 이 줄을 추가합니다.

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

ESP-IDF를 위한 자체 sdkconfig 제공

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

```
cmake -S . -B build -IDF_SDKCONFIG_DEFAULTS=<path_to_your_sdkconfig_defaults> -DCMAKE_TOOLCHAIN_FILE=amazon-freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

자체 sdkconfig.default 파일에 대한 위치를 지정하지 않으면 Amazon FreeRTOS에서 [<amazon-freertos>/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults](#)에 위치한 기본 파일을 사용합니다.

요약

foo라는 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)

add_executable(my_app src/main.c)

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

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

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

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

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

문제 해결

- macOS를 실행하는 경우 운영 체제에서 ESP-WROVER-KIT를 인식하지 못하면 D2XX 드라이버가 설치되어 있지 않은지 확인합니다. 이 드라이버를 제거하려면 [macOS X용 FTDI 드라이버 설치 가이드](#)의 지침을 따르십시오.
- ESP-IDF에서 제공하는 모니터 유ти리티(make monitor를 사용하여 호출)를 사용하면 주소를 디코딩할 수 있습니다. 이러한 이유로 이 유ти리티는 애플리케이션이 충돌하는 경우 의미 있는 역추적을 얻는 데 도움이 될 수 있습니다. 자세한 내용은 Espressif 웹사이트의 [주소 자동 디코딩](#)을 참조하십시오.
- 특수 JTAG 하드웨어를 사용할 필요 없이 gdb와 통신하기 위해 GDBStub을 활성화할 수도 있습니다. 자세한 내용은 Espressif 웹 사이트의 [GDBStub용 GDB 시작](#)을 참조하십시오.
- JTAG 하드웨어 기반 디버깅이 필요한 경우 OpenOCD 기반 환경 설정에 대한 자세한 내용은 [Espressif 웹사이트](#)에서 이용 가능한 ESP32용 JTAG 디버깅 문서를 참조하십시오.
- macOS에서 pip를 사용하여 pyserial을 설치할 수 없는 경우 [pyserial 웹 사이트](#)에서 다운로드합니다.

- 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 Amazon FreeRTOS에 포함되어 있으므로 외부에서 다운로드할 필요가 없습니다. IDF_PATH 환경 변수가 설정된 경우 Amazon FreeRTOS를 빌드하기 전에 제거하는 것이 좋습니다.
- Window에서 프로젝트를 빌드하려면 3-4분 정도 걸릴 수 있습니다. make 명령에서 -j4 스위치를 사용하여 빌드 시간을 줄일 수 있습니다.

```
make flash monitor -j4
```

- 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.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅

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

ESP-DevKitC JTAG 설정

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

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

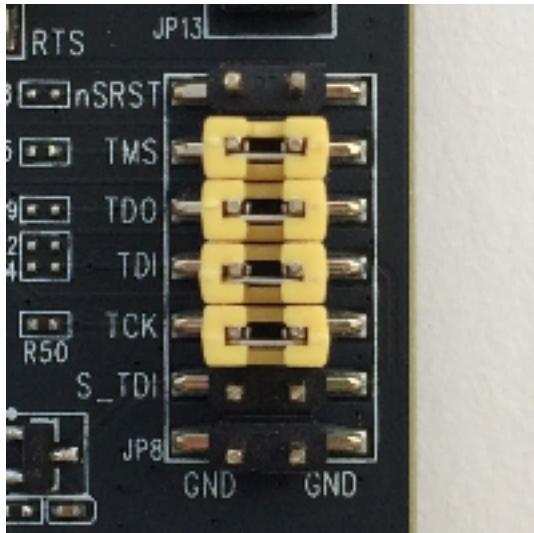
ESP-WROVER-KIT JTAG 설정

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

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

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

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



Windows에서 디버깅

Windows에서 디버깅을 설정하려면

1. [Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅 \(p. 97\)](#)의 설명과 같이 FTDI C232HM-DDHSL-0의 USB 쪽을 컴퓨터 및 다른 쪽에 연결합니다. FTDI C232HM-DDHSL-0 디바이스가 Universal Serial Bus Controllers(법용 직렬 버스 컨트롤러)의 Device Manager(장치 관리자)에 나타나야 합니다.
2. 법용 직렬 버스 디바이스 목록에서 C232HM-DDHSL-0 디바이스를 마우스 오른쪽 버튼으로 클릭하고 속성을 선택합니다.

Note

디바이스가 USB 직렬 포트로 나열될 수 있습니다.

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

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

FTDIBUS\COMPORT&VID_0403&PID_6014

이 예에서 공급업체 ID는 0403이고 제품 ID는 6014입니다.

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

`ftdi_vid_pid 0x0403 0x6014`

4. [Windows용 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. 실행 `zadig.exe Options`(옵션) 메뉴에서 List All Devices(모든 디바이스 나열)를 선택합니다.
 - c. 드롭다운 메뉴에서 C232HM-DDHSL-0를 선택합니다.
 - d. 녹색 화살표 오른쪽의 대상 드라이버 필드에서 WinUSB를 선택합니다.
 - e. 대상 드라이버 필드 아래의 드롭다운 상자에서 화살표를 선택한 후 Install Driver(드라이버 설치)를 선택합니다. Replace Driver(드라이버 바꾸기)를 선택합니다.
7. 명령 프롬프트를 열고 `projects/espressif/esp32/make/aws_demos`로 이동한 후 다음을 실행합니다.

ESP32-WROOM-32 및 ESP32-WROVER:

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

ESP32-SOLO-1:

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

이 명령 프롬프트를 열어 둡니다.

8. 새 명령 프롬프트를 열고 msys32 디렉터리로 이동한 다음 `mingw32.exe`를 실행합니다. mingw32 터미널에서 `projects/espressif/esp32/make/aws_demos`로 이동하고 `make flash monitor`를 실행합니다.
9. 다른 mingw32 터미널을 열고 `projects/espressif/esp32/make/aws_demos`로 이동한 후 해당 보드에서 데모의 실행이 시작될 때까지 기다립니다. 시작되면 `xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf`를 실행합니다. 프로그램이 `main` 기능에서 중지해야 합니다.

Note

ESP32는 최대 2개의 중단점을 지원합니다.

macOS에서 디버깅

1. macOS용 [FTDI 드라이버](#)를 다운로드합니다.
2. [OpenOCD](#)를 다운로드합니다.
3. 다운로드한 .tar 파일의 압축을 풀고 `.bash_profile`에서 경로를 `<OCD_INSTALL_DIR>/openocd-esp32/bin`으로 설정합니다.
4. 다음 명령을 사용하여 libusb를 macOS에 설치합니다.

```
brew install libusb
```

5. 다음 명령을 사용하여 직렬 포트 드라이버를 언로드합니다.

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

6. 10.9 이후의 macOS 버전을 실행하는 경우 다음 명령을 사용하여 Apple의 FTDI 드라이버를 언로드합니다.

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

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

```
system_profiler SPUSBDataType
```

system_profiler의 출력은 다음과 같아야 합니다.

DEVICE:

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

8. projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg를 엽니다. 디바이스의 공급업체 ID 및 제품 ID는 ftdi_vid_pid로 시작하는 줄에서 지정합니다. 이전 단계에서 출력한 system_profiler의 ID와 일치하도록 ID를 변경합니다.
9. 터미널 창을 열고 projects/espressif/esp32/make/aws_demos로 이동한 후 다음 명령을 사용하여 OpenOCD를 실행합니다.

ESP32-WROOM-32 및 ESP32-WROVER:

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

ESP32-SOLO-1:

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

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

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

11. projects/espressif/esp32/make/aws_demos로 이동하여 다음 명령을 실행합니다.

```
make flash monitor
```

12. 다른 새 터미널을 열고 projects/espressif/esp32/make/aws_demos로 이동한 후 다음 명령을 실행합니다.

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

프로그램이 main()에서 종지해야 합니다.

Linux에서 디버깅

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/ttYS*`를 입력하여 컴퓨터에 연결된 모든 USB 디바이스를 나열합니다. 이 단계는 보드의 USB 포트가 운영 체제에서 인식되는지 확인하는 데 도움이 됩니다. 다음과 같이 출력되어야 합니다.

```
$ls -l /dev/ttYS*
crw-rw---- 1 root dialout 188, 0 Jul 10 19:04 /dev/ttYS0
```

```
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 dialout을 plugdev로 변경했는지 확인합니다.

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

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

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

ESP32-WROOM-32 및 ESP32-WROVER:

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

ESP32-SOLO-1:

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

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

```
make flash monitor
```

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

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

프로그램이 `main()`에서 중지되어야 합니다.

Infineon XMC4800 IoT Connectivity Kit 시작하기

이 자습서에서는 Infineon XMC4800 IoT 연결 키트를 시작하기 위한 지침을 제공합니다. Infineon XMC4800 IoT 연결 키트가 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

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 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 [`<amazon-freertos>`](#)라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

- 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
- Amazon 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

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

DAVE 설치

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

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.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Amazon FreeRTOS 데모를 DAVE로 가져옵니다.

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.

Amazon FreeRTOS 다운로드의 압축을 폰 디렉터리에서 데모 프로젝트는 projects/infineon/xmc4800_iotkit/dave4/aws_demos에 있습니다.

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.

Amazon 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_demos를 다시 빌드하고 구성 변경 사항이 적용되었는지 확인합니다.
3. 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.
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 종단점에서 중지되면 Run(실행) 메뉴에서 Resume(다시 시작)을 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----
```

CMake로 Amazon FreeRTOS 데모 빌드

Amazon FreeRTOS용 IDE 개발을 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

Note

이 단원에서는 MingW를 사용하여 Windows에서 CMake를 네이티브 빌드 시스템으로 사용하는 방법을 다룹니다. 다른 운영 체제 및 옵션과 함께 CMake를 사용하는 방법에 대한 자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

CMake로 Amazon FreeRTOS 데모를 빌드하려면

- 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.

- CMake 및 MingW를 설치합니다.

지침은 [CMake 사전 조건 \(p. 71\)](#)을 참조하십시오.

- 생성된 빌드 파일을 포함할 폴더(<BUILD_FOLDER>)를 생성합니다.
- 디렉터리를 Amazon FreeRTOS 다운로드 디렉터리(<amazon-freertos>)로 변경하고 다음 명령을 사용하여 빌드 파일을 생성합니다.

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

- 디렉터리를 빌드 디렉터리(<BUILD_FOLDER>)로 변경하고 다음 명령을 사용하여 이진 파일을 빌드합니다.

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

이 명령은 출력 바이너리 aws_demos.hex를 빌드 디렉터리로 빌드합니다.

6. [JLINK \(p. 102\)](#)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 \(p. 102\)](#)을 통해 볼 수 있어야 합니다.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

아직 설치하지 않은 경우 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

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 Connectivity Kit 시작하기 \(p. 101\)](#) 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 방문하여 [파트너](#)에서 구입하시기 바랍니다.
2. 보안 확장 팩:
 - 보안 요소 - Infineon OPTIGA Trust X.
AWS 파트너 디바이스 카탈로그를 방문하여 [파트너](#)에서 구입하시기 바랍니다.
 - 개인화 보드 - Infineon OPTIGA 개인화 보드.
 - 어댑터 보드 - Infineon MyIoT 어댑터.

여기 나와 있는 단계를 따르려면 보드와의 직렬 연결을 열어서 로깅 및 디버깅 정보를 확인해야 합니다. (한 단계에서는 보드의 직렬 디버깅 출력에서 퍼블릭 키를 복사하여 파일에 붙여 넣어야 합니다.) 이를 위해서는 XMC4800 IoT 연결 키트에 추가로 3.3V USB/직렬 변환기가 필요합니다. [JBtek EL - PN-47310126](#) USB/직렬 변환기는 이 데모에서 작동하는 것으로 알려져 있습니다. 또한 Infineon MyIoT 어댑터 보드에 직렬 케이블을 연결하려면 3개의 수-수 [점퍼 와이어](#)(수신(RX), 전송(TX) 및 접지(GND))가 필요합니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 지침은 [옵션 #2: 온보드 프라이빗 키 생성 \(p. 67\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 딜렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

개요

이 자습서에 포함된 단계는 다음과 같습니다.

1. 마이크로컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 머신에 설치합니다.
2. Amazon FreeRTOS 데모 애플리케이션을 이진 이미지로 교차 컴파일합니다.
3. 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.
4. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

개발 환경 설정

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

DAVE 설치

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

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 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

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링 할 수 있습니다. 디바이스가 데모 프로젝트를 실행하기 전에 설정할 수 있습니다.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Amazon FreeRTOS 데모를 DAVE로 가져옵니다.

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 디렉터리 선택)를 선택하고 Browse(가져오기)를 선택한 다음 XMC4800 데모 프로젝트를 선택합니다.

Amazon FreeRTOS 다운로드의 압축을 폰 디렉터리에서 데모 프로젝트는 `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`에 있습니다.

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

4. [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.

Amazon FreeRTOS 데모 프로젝트 실행

1. 프로젝트 메뉴에서 Rebuild Active Project(활성 프로젝트 다시 빌드)를 선택하여 aws_demos를 다시 빌드하고 구성 변경 사항이 적용되었는지 확인합니다.
2. Project Explorer(프로젝트 탐색기)에서 aws_demos를 마우스 오른쪽 버튼으로 클릭하고 Debug As(다른 형식으로 디버그)를 선택한 다음 DAVE C/C++ Application(DAVE C/C++ 애플리케이션)을 선택합니다.
3. GDB SEGGER J-Link Debugging(GDB SEGGER J-Link 디버깅)을 두 번 클릭하여 디버그 확인을 생성합니다. Debug(디버그)를 선택합니다.
4. 디버거가 main()의 종단점에서 중지되면 Run(실행) 메뉴에서 Resume(다시 시작)을 선택합니다.

여기서 옵션 #2: 온보드 프라이빗 키 생성 (p. 67)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----
```

CMake로 Amazon 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 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

Amazon FreeRTOS용 IDE 개발을 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake로 Amazon FreeRTOS 데모를 빌드하려면

- GNU Arm Embedded Toolchain을 설정합니다.

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

Note

objcopy 유틸리티에 [보고된 버그](#)가 있으므로 "8-2018-q4-major" 이외의 버전을 다운로드하는 것이 좋습니다.

- 다운로드한 도구 체인 설치 관리자를 열고 마법사의 지침에 따릅니다.
 - 설치 마법사의 마지막 페이지에서 Add path to environment variable(환경 변수에 경로 추가)를 선택하여 시스템 경로 환경 변수에 도구 체인 경로를 추가합니다.
- CMake 및 MingW를 설치합니다.
- 지침은 [CMake 사전 조건 \(p. 71\)](#)을 참조하십시오.
- 생성된 빌드 파일을 포함할 폴더(<BUILD_FOLDER>)를 생성합니다.
 - 디렉터리를 Amazon FreeRTOS 다운로드 디렉터리(<amazon-freertos>)로 변경하고 다음 명령을 사용하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S . -B <BUILD_FOLDER> -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

- 디렉터리를 빌드 디렉터리(<BUILD_FOLDER>)로 변경하고 다음 명령을 사용하여 바이너리를 빌드합니다.

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

이 명령은 출력 바이너리 aws_demos.hex를 빌드 디렉터리로 빌드합니다.

- JLINK (p. 102)를 사용하여 이미지를 플래시 및 실행합니다.
 - 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
```

- 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 \(p. 102\)](#)을 통해 볼 수 있어야 합니다. [옵션 #2: 온보드 프라이빗 키 생성 \(p. 67\)](#)의 퍼블릭 키 추출 단계로 계속 진행합니다. 모든 단계가 완료되

면 AWS IoT 콘솔로 이동합니다. 디바이스에서 보낸 MQTT 메시지가 이전에 설정한 MQTT 클라이언트에 표시되어야 합니다.

문제 해결

일반적인 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Marvell MW320 AWS IoT 시작 키트 시작

이 자습서에서는 Marvell MW320 AWS IoT 시작 키트를 시작하기 위한 지침을 제공합니다.

Marvell MW320 AWS IoT 시작 키트에는 단일 마이크로 컨트롤러 칩에 802.11b/g/n Wi-Fi를 통합하는 Cortex M4 마이크로 컨트롤러 개발 보드인 Marvell MW320이 포함되어 있습니다. 이 키트는 FCC 인증을 받았으며 현재 구입할 수 있습니다. 또한 MW320 모듈은 FCC 인증을 받았으며 사용자 지정 및 대량 판매가 가능합니다.

Marvell MW320 AWS IoT 시작 키트가 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입 하시기 바랍니다.

Note

이 자습서에서는 Marvell MW320용 애플리케이션 개발 및 디버깅을 위해 Ubuntu 16.04를 사용합니다. 다른 운영 체제에서도 작동하지만 공식적으로 지원되지는 않습니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

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

Amazon FreeRTOS에는 필요한 타사 라이브러리를 설치하고 애플리케이션을 빌드 및 보드에 플래시하기 위한 몇 가지 스크립트가 포함됩니다. 이러한 스크립트는 vendors/marvell/WMSDK/mw320/sdk 디렉터리에 있습니다.

또한 AWS IoT 시작 키트에는 사전 플래시된 무선 마이크로컨트롤러 데모 프로젝트 펌웨어가 포함됩니다.

M320 AWS IoT 시작 키트 및 Amazon FreeRTOS 포트와 함께 번들로 제공되는 소프트웨어뿐 아니라 다음과 같은 소프트웨어 구성 요소가 호스트 머신에 설치되어 있어야 합니다.

- GCC 도구 체인(애플리케이션을 크로스 컴파일하기 위해). 버전 4_9_2015q3 이상이 지원됩니다.
- OpenOCD(보드의 플래시 메모리를 프로그래밍하고 펌웨어 이미지를 다운로드하여 메모리에서 실행하는 다양한 JTAG 함수에 액세스하기 위해)
- CMake 빌드 시스템. 버전 3.13 이상이 지원됩니다.

- (선택 사항) 지원되는 IDE(애플리케이션 개발과 디버깅에 사용하기 위해)

installpkgs.sh와 함께 필요한 타사 라이브러리 설치

vendors/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh 스크립트가 머신 유형을 자동으로 감지하고, 다음과 같은 몇 가지 필수 라이브러리를 설치하려고 시도합니다.

- C 라이브러리
- USB 라이브러리
- FTDI 라이브러리
- ncurses
- Python
- LaTeX

Note

installpkgs.sh 스크립트는 32비트 및 64비트 Ubuntu 환경에 apt-get을 사용하고 32비트 및 64비트 Fedora 환경에 yum을 사용하여 패키지를 설치하기 위한 지침을 포함합니다. 배포에서 스크립트를 실행하는 데 문제가 있으면 스크립트 파일을 열고 필수 패키지 목록을 찾아 수동으로 설치하십시오.

루트 권한을 사용하여 Amazon FreeRTOS 다운로드의 루트 디렉터리에서 다음 명령을 실행하십시오.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Linux 호스트 머신에서 권한을 구성하여 sudo를 사용하지 않고 flashprog 및 ramload 작업을 허용할 수 있습니다. 이를 위해서는 다음 명령을 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

Note

Eclipse IDE를 사용하는 경우 이러한 권한을 구성해야 합니다.

도구 체인 설정

이 보드의 Amazon FreeRTOS 포트는 기본적으로 GNU 도구 체인을 사용하도록 구성됩니다. Makefile이 있는 모든 컴파일러 도구 체인을 호출하기 위해서는 GNU 컴파일러 도구 체인 바이너리가 사용자의 PATH 변수에 포함되어야 합니다. 또한 GNU 도구 체인에는 arm-none-eabi- 접두사가 붙습니다.

JTAG와 인터페이스하는 OpenOCD 소프트웨어로 디버깅하기 위해 GCC 도구 체인을 GDB(GNU Debugger)와 함께 사용할 수 있습니다.

Linux 시스템에 GCC 도구 체인을 설정하려면

1. launchpad에서 도구 체인 tarball을 다운로드합니다. 파일 이름은 gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2입니다.
2. 선택한 디렉터리에 파일을 복사합니다. 디렉터리 경로에 공백이 없어야 합니다.
3. 다음 명령으로 파일의 압축을 해제합니다.

```
tar -vxf file-name
```

4. 도구 체인 바이너리를 PATH에 추가합니다.

예를 들어 \$HOME 디렉터리의 .profile 파일을 열고 다음 줄을 파일 끝에 추가합니다.

```
PATH="$PATH:<path>/gcc-arm-none-eabit-4_9_2015_q3/bin"
```

여기서 <path>는 gcc-arm-none-eabit-4_9_2015_q3 폴더의 전체 디렉터리 경로입니다.

Note

Ubuntu의 일부 배포에는 Debian 버전의 GCC 크로스 컴파일러가 포함됩니다. 기본 크로스 컴파일러가 배포에 포함되어 있으면 이를 제거하고 단계에 따라 GCC 컴파일러 도구 체인을 설정하십시오.

OpenOCD 설정

OpenOCD 버전 0.9가 필요합니다. 호스트 시스템에 이전 버전이 설치되어 있으면 배포의 제거 프로세스를 사용하여 제거하십시오.

표준 Linux 패키지 관리자를 사용하여 OpenOCD를 설치할 수 있습니다. 예를 들면 다음과 같습니다.

```
apt-get install openocd
```

시스템에 설치된 OpenOCD 기본 버전은 Linux 커널 버전에 따라 다릅니다.

패키지 관리자를 사용하여 OpenOCD 버전 0.9를 설치할 수 없으면 이 단계를 따르십시오.

1. [OpenOCD.org](#)에서 OpenOCD 버전 0.9 소스 코드를 다운로드합니다.
2. openocd 다운로드를 추출한 후 디렉터리를 추출된 폴더로 변경합니다.
3. FTDI 및 JLink를 활성화합니다.

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

4. openocd를 만듭니다.

```
make install
```

CMake 설치

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

[CMake.org](#)에서 CMake의 최신 버전을 다운로드할 수 있습니다. 소스 및 이진 배포를 둘 다 사용할 수 있습니다.

Amazon FreeRTOS와 함께 CMake를 사용하는 방법에 대한 자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

직렬 연결 설정

호스트 시스템과 보드 간의 직렬 연결을 설정하려면

1. USB 케이블 한쪽 끝을 호스트 시스템에 연결하고 다른 쪽 끝을 보드에 연결합니다.

호스트 시스템에서 보드를 감지해야 합니다. 명령줄에서 dmesg 명령을 실행하거나 /var/log/messages 파일을 열어 보드가 감지되었는지 확인할 수 있습니다. 성공적으로 연결되면 다음과 유사한 메시지가 표시됩니다.

```
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
```

Note

Marvell 개발 보드에는 USB 인터페이스 두 개를 호스트에 노출하는 FTDI 칩이 있습니다. 첫 번째 인터페이스(ttyUSB0)는 MCU의 JTAG 기능에 연결됩니다. 두 번째 인터페이스(ttyUSB1)는 MCU의 물리적 UARTx 포트에 연결됩니다.

2. 다음 설정을 사용하여 ttyUSB1 인터페이스와의 직렬 연결을 엽니다.

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

예를 들어 minicom을 사용하는 경우 다음을 수행합니다.

- a. 설정 모드에서 minicom을 시작합니다.

```
minicom -s
```

- b. Serial Port Setup(직렬 포트 설정)으로 이동합니다.
- c. 다음 설정을 구성합니다:

```
| 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
```

Exit(종료)로 이동하여 직렬 콘솔에서 메시지 표시를 시작합니다.

터미널 에뮬레이터를 설치하여 직렬 연결을 설정하는 데 대한 자세한 내용은 [터미널 에뮬레이터 설치 \(p. 69\)](#) 단원을 참조하십시오.

Amazon FreeRTOS 데모 프로젝트 빌드, 플래시 및 실행

Amazon FreeRTOS의 M320 포트와 함께 포함된 유ти리티 스크립트와 CMake를 사용하여 명령줄에서 Amazon FreeRTOS 데모 프로젝트를 빌드, 플래시 및 실행할 수 있습니다. 또는 IDE를 사용하여 프로젝트를 빌드할 수 있습니다.

CMake로 데모 빌드 파일 생성

Amazon FreeRTOS 다운로드 루트에서 다음 명령을 실행하여 CMake로 데모 빌드 파일을 생성합니다.

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

또는

```
cmake -DVENDOR=marvell -DBOARD=mw322 -DCOMPILER=arm-gcc -S . -B build -DAFR_ENABLE_TESTS=0
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
=====Configuration for Amazon FreeRTOS=====
Version:          1.4.7
Git version:      v1.4.7-128-gc28d0a266

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:              Darwin-17.7.0
  Toolchain:        arm-gcc
  Toolchain path:  /usr/local
  CMake generator: Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build: bufferpool, crypto, freertos_plus_tcp, greengrass, kernel,
                     mqtt, pkcs11, secure_sockets, shadow, tls, wifi
  Enabled by user:  greengrass, mqtt, pkcs11, secure_sockets, shadow, wifi
  Enabled by dependency: bufferpool, common, crypto, demo_base, freertos,
                         freertos_plus_tcp, kernel, pkcs11_mbedtls, tls, utils
  3rdparty dependencies: jsmn, mbedtls, pkcs11
  Available demos:   demo_greengrass, demo_key_provisioning, demo_logging,
                     demo_mqtt_hello_world, demo_mqtt_pubsub, demo_shadow, demo_tcp
  Available tests:
```

make로 데모 빌드

다음 명령을 실행하여 데모를 빌드합니다.

```
cd build
```

```
make all -j4
```

다음과 유사한 출력 화면이 표시되어야 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/jsonn/jsonn.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$
```

유사한 명령 세트를 사용하여 테스트 프로젝트를 빌드할 수 있습니다.

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

또는

```
cmake -DVENDOR=marvell -DBOARD=mw322 -DCOMPILER=arm-gcc -S . -B build -DAFR_ENABLE_TESTS=1
```

```
cd build
```

```
make all -j4
```

Note

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

애플리케이션 플래시

보드의 플래시 메모리를 프로그래밍하기 위해 flashprog.py 스크립트가 사용됩니다. 스크립트는 Python 2.7로 작성됩니다.

데모 애플리케이션 이미지를 보드로 플래시하려면 먼저 레이아웃 파일과 Boot2 부트로더를 사용하여 보드의 플래시 메모리를 준비하십시오.

레이아웃 파일과 Boot2 부트로더를 로드하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. flashprog.py Python 스크립트를 -l 및 --boot2 옵션과 함께 실행합니다.

```
./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
```

vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt에 정의된 기본 레이아웃 구성에 따라 flashprog 스크립트가 플래시에 레이아웃을 씁니다. 플래시에 대한 파티셔닝된 정보가 레이아웃에 보관됩니다.

또한 스크립트는 플래시에 부트로더를 씁니다. 부트로더는 vendors/marvell/WMSDK/mw320/sdk/boot2/bin/boot2.bin에 있습니다. 보드에 플래시된 후 부트로더가 마이크로컨트롤러 펌웨어 이미지를 로드합니다.

다음과 유사한 출력 화면이 표시되어야 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 0.245458s (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
```

레이아웃 파일과 부트로더를 보드에 플래시한 후 특정 펌웨어를 보드에 플래시하십시오. Wi-Fi 칩셋을 사용하는 경우 해당 펌웨어가 플래시 메모리에 있어야 합니다.

Wi-Fi 펌웨어를 플래시하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. flashprog.py Python 스크립트를 --wififw 옵션과 함께 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./vendors/
marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

flashprog 스크립트가 펌웨어를 보드로 플래시합니다.

다음과 유사한 출력 화면이 표시되어야 합니다.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary)....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked

target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

레이아웃, 부트로더 및 Wi-Fi 펌웨어가 보드에 플래시되면 데모 애플리케이션을 보드에 플래시하고 실행할 수 있습니다.

데모를 플래시하고 실행하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. flashprog.py Python 스크립트를 --mcufw 및 -r 옵션과 함께 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw ./build/marvell/mw320/aws_demos.bin -r
```

flashprog 스크립트가 데모를 보드로 플래시합니다. 스크립트를 `-r` 옵션과 함께 실행하면 보드가 재설정됩니다.

보드를 재설정합니다.

로그에 데모 애플리케이션이 보여야 합니다. 다음과 유사하게 출력됩니다.

```
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_nrst_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...
```

Note

`aws_tests` 애플리케이션을 플래시하려면 동일한 명령을 사용하되 `aws_tests.bin` 파일을 `aws_demos.bin` 대신 지정하십시오.

애플리케이션만 변경하는 경우에는 레이아웃, 부트로더 및 Wi-Fi 펌웨어를 다시 로드할 필요가 없습니다. 레이아웃을 변경하는 경우 모든 구성 요소를 다시 로드해야 할 수 있습니다.

데모를 빌드, 플래시 및 실행할 때 다음과 유사한 출력이 표시됩니다.

```
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 70968 [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----
```

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

GNU Debugger에 연결

GDB(GNU Debugger)에 연결하려면

1. 디렉터리를 변경합니다.

```
cd <amazon-freertos>/vendors/marvell/WMSDK/mw320
```

2. arm-none-eabi-gdb 명령으로 GDB에 연결합니다.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../build/vendors/marvell/boards/mw300_rd/aws_demos.axf
```

Amazon FreeRTOS 테스트 애플리케이션을 디버깅하는 경우 `aws_tests.axf`를 대신 대상으로 지정 합니다.

SRAM에 애플리케이션 로드

`ramload.py` 스크립트로 디바이스의 SRAM(Static Random-Access Memory)에 데모를 로드한 후 디바이스에서 애플리케이션을 실행할 수 있습니다. `ramload.py`를 사용하여 애플리케이션을 로드 및 실행하면

`flashprog.py` 스크립트를 사용하여 플래시 메모리에 로드하는 것보다 속도가 빨라 반복적인 개발을 보다 효율적으로 수행할 수 있습니다.

Note

`ramload.py` 스크립트는 Python 2.7로 작성됩니다.

SRAM에 로드하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. `ramload.py` Python 스크립트를 `aws_demos.axf` 파일에서 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/ramload.py ./build/vendors/marvell/boards/mw300_rd/aws_demos.axf
```

로그에 데모 애플리케이션이 보여야 합니다. 다음과 유사하게 출력됩니다.

```
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: 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)
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
```

Note

SRAM에 로드된 이미지는 재부팅할 때 지워집니다.

기타 로그 활성화

이 보드를 시작할 때 발생하는 문제를 해결하기 위해 다른 로깅 메시지를 활성화해야 할 수도 있습니다.

보드 관련 로그를 활성화하려면

1. 작업 중인 프로젝트(예: `aws_tests` 또는 `aws_demos`)의 `main.c` 파일을 엽니다.
2. `prvMiscInitialization` 함수에서 `wmstdio_init(UART0_ID, 0)` 호출을 활성화합니다.

Wi-Fi 로그를 활성화하려면

1. Open `vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h`.
2. 매크로 `CONFIG_WLCMGR_DEBUG`를 활성화합니다.

IDE 개발 및 디버깅 사용

IDE 설정

애플리케이션을 개발 및 디버깅하고 프로젝트를 시작화하기 위해 IDE를 사용할 수 있습니다.

예를 들어 Eclipse IDE를 사용하는 경우 `perm_fix.sh` 스크립트를 사용하여 몇 가지 권한을 구성합니다.

```
./vendors/marvell/WMSDK/mw320/tools/bin/perm_fix.sh
```

Eclipse를 설정하려면

1. [Oracle](#)에서 JRE(Java 런타임 환경)를 설치합니다.
Eclipse를 실행하려면 JRE가 필요합니다. JRE 버전(32비트 또는 64비트)은 설치하는 Eclipse 버전(32비트 또는 64비트)과 일치해야 합니다.
2. [Eclipse.org](#)에서 C/C++ 개발자용 Eclipse IDE를 다운로드합니다. Eclipse 버전 4.9.0 이상이 지원됩니다.
3. 다운로드한 아카이브 폴더를 추출하고 플랫폼별 Eclipse 실행 파일을 실행하여 IDE를 시작합니다.

IDE로 데모 빌드

`make`를 사용하여 명령줄에서 직접 데모를 빌드하는 대신 IDE에서 데모 프로젝트의 빌드 파일을 열고 빌드 할 수 있습니다. IDE에서 파일을 열면 빌드하기 전에 프로젝트를 시작화할 수 있습니다.

Note

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

Eclipse로 프로젝트를 빌드하려면

1. Eclipse를 엽니다.
2. 프로젝트를 생성할 Workspace를 선택합니다.
3. Select a wizard 페이지에서 C/C++를 확장한 후 Makefile Project with Existing Code를 선택합니다.
4. Import existing code 페이지에서 `aws_demos` 소스 코드 위치를 찾아 `aws_demos`를 선택한 후 Finish를 선택합니다.
5. Project Explorer에서 `aws_demos`를 마우스 오른쪽 버튼으로 클릭한 후 프로젝트를 빌드합니다.

빌드가 성공하면 `aws_demos.bin` 실행 파일이 생성됩니다.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Marvell MW322 AWS IoT 시작 키트 시작

이 자습서에서는 Marvell MW322 AWS IoT 시작 키트를 시작하기 위한 지침을 제공합니다.

Marvell MW322 AWS IoT 시작 키트에는 단일 마이크로 컨트롤러 칩에 802.11b/g/n Wi-Fi를 통합하는 Cortex M4 마이크로 컨트롤러 개발 보드인 Marvell MW322가 포함되어 있습니다. 이 키트는 FCC 인증을 받았으며 현재 구입할 수 있습니다. 또한 MW322 모듈은 FCC 인증을 받았으며 사용자 지정 및 대량 판매가 가능합니다.

Marvell MW322 AWS IoT 시작 키트가 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입 하시기 바랍니다.

Note

이 자습서에서는 Marvell MW322용 애플리케이션 개발 및 디버깅을 위해 Ubuntu 16.04를 사용합니다. 다른 운영 체제에서도 작동하지만 공식적으로 지원되지는 않습니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 딜렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

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.
- Amazon 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

Amazon FreeRTOS에는 필요한 타사 라이브러리를 설치하고 애플리케이션을 빌드 및 보드에 플래시하기 위한 몇 가지 스크립트가 포함됩니다. 이러한 스크립트는 `vendors/marvell/WMSDK/mw320/sdk` 딜렉터리에 있습니다.

또한 AWS IoT 시작 키트에는 사전 설치된 무선 마이크로컨트롤러 데모 프로젝트 펌웨어가 포함됩니다.

M320 AWS IoT 시작 키트 및 Amazon FreeRTOS 포트와 함께 번들로 제공되는 소프트웨어뿐 아니라 다음과 같은 소프트웨어 구성 요소가 호스트 머신에 설치되어 있어야 합니다.

- GCC 도구 체인(애플리케이션을 크로스 컴파일하기 위해). 버전 4_9_2015q3 이상이 지원됩니다.
- OpenOCD(보드의 플래시 메모리를 프로그래밍하고 펌웨어 이미지를 다운로드하여 메모리에서 실행하는 다양한 JTAG 함수에 액세스하기 위해)
- CMake 빌드 시스템. 버전 3.13 이상이 지원됩니다.
- (선택 사항) 지원되는 IDE(애플리케이션 개발과 디버깅에 사용하기 위해)

installpkgs.sh와 함께 필요한 타사 라이브러리 설치

`vendors/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh` 스크립트가 머신 유형을 자동으로 감지하고, 다음과 같은 몇 가지 필수 라이브러리를 설치하려고 시도합니다.

- C 라이브러리
- USB 라이브러리
- FTDI 라이브러리
- ncurses
- Python
- LaTeX

Note

`installpkgs.sh` 스크립트는 32비트 및 64비트 Ubuntu 환경에 `apt-get`을 사용하고 32비트 및 64비트 Fedora 환경에 `yum`을 사용하여 패키지를 설치하기 위한 지침을 포함합니다. 배포에서 스크

립트를 실행하는 데 문제가 있으면 스크립트 파일을 열고 필수 패키지 목록을 찾아 수동으로 설치하십시오.

루트 권한을 사용하여 Amazon FreeRTOS 다운로드의 루트 디렉터리에서 다음 명령을 실행하십시오.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Linux 호스트 머신에서 권한을 구성하여 sudo를 사용하지 않고 flashprog 및 ramload 작업을 허용할 수 있습니다. 이를 위해서는 다음 명령을 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

Note

Eclipse IDE를 사용하는 경우 이러한 권한을 구성해야 합니다.

도구 체인 설정

이 보드의 Amazon FreeRTOS 포트는 기본적으로 GNU 도구 체인을 사용하도록 구성됩니다. Makefile이 다른 컴파일러 도구 체인을 호출하기 위해서는 GNU 컴파일러 도구 체인 바이너리가 사용자의 PATH 변수에 포함되어야 합니다. 또한 GNU 도구 체인에는 arm-none-eabi- 접두사가 붙습니다.

JTAG와 인터페이스하는 OpenOCD 소프트웨어로 디버깅하기 위해 GCC 도구 체인을 GDB(GNU Debugger)와 함께 사용할 수 있습니다.

Linux 시스템에 GCC 도구 체인을 설정하려면

1. [launchpad](#)에서 도구 체인 tarball을 다운로드합니다. 파일 이름은 gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2입니다.
2. 선택한 디렉터리에 파일을 복사합니다. 디렉터리 경로에 공백이 없어야 합니다.
3. 다음 명령으로 파일의 압축을 해제합니다.

```
tar -vxf file-name
```

4. 도구 체인 바이너리를 PATH에 추가합니다.

예를 들어 \$HOME 디렉터리의 .profile 파일을 열고 다음 줄을 파일 끝에 추가합니다.

```
PATH="$PATH:<path>/gcc-arm-none-eabit-4_9_2015_q3/bin"
```

여기서 <path>는 gcc-arm-none-eabit-4_9_2015_q3 폴더의 전체 디렉터리 경로입니다.

Note

Ubuntu의 일부 배포에는 Debian 버전의 GCC 크로스 컴파일러가 포함됩니다. 기본 크로스 컴파일러가 배포에 포함되어 있으면 이를 제거하고 단계에 따라 GCC 컴파일러 도구 체인을 설정하십시오.

OpenOCD 설정

OpenOCD 버전 0.9가 필요합니다. 호스트 시스템에 이전 버전이 설치되어 있으면 배포의 제거 프로세스를 사용하여 제거하십시오.

표준 Linux 패키지 관리자를 사용하여 OpenOCD를 설치할 수 있습니다. 예를 들면 다음과 같습니다.

```
apt-get install openocd
```

시스템에 설치된 OpenOCD 기본 버전은 Linux 커널 버전에 따라 다릅니다.

파키지 관리자를 사용하여 OpenOCD 버전 0.9를 설치할 수 없으면 이 단계를 따르십시오.

1. [OpenOCD.org](#)에서 OpenOCD 버전 0.9 소스 코드를 다운로드합니다.
2. openocd 다운로드를 추출한 후 디렉터리를 추출된 폴더로 변경합니다.
3. FTDI 및 JLink를 활성화합니다.

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

4. openocd를 만듭니다.

```
make install
```

CMake 설치

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

[CMake.org](#)에서 CMake의 최신 버전을 다운로드할 수 있습니다. 소스 및 이진 배포를 둘 다 사용할 수 있습니다.

Amazon FreeRTOS와 함께 CMake를 사용하는 방법에 대한 자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

직렬 연결 설정

호스트 시스템과 보드 간의 직렬 연결을 설정하려면

1. USB 케이블 한쪽 끝을 호스트 시스템에 연결하고 다른 쪽 끝을 보드에 연결합니다.

호스트 시스템에서 보드를 감지해야 합니다. 명령줄에서 dmesg 명령을 실행하거나 /var/log/messages 파일을 열어 보드가 감지되었는지 확인할 수 있습니다. 성공적으로 연결되면 다음과 유사한 메시지가 표시됩니다.

```
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
```

Note

Marvell 개발 보드에는 USB 인터페이스 두 개를 호스트에 노출하는 FTDI 칩이 있습니다. 첫 번째 인터페이스(ttyUSB0)는 MCU의 JTAG 기능에 연결됩니다. 두 번째 인터페이스(ttyUSB1)는 MCU의 물리적 UARTx 포트에 연결됩니다.

2. 다음 설정을 사용하여 `ttyUSB1` 인터페이스와의 직렬 연결을 엽니다.

터미널 설정	값
전송 속도	115200
테스트	8비트
파리티	없음
종지	1비트
흐름 제어	없음

예를 들어 minicom을 사용하는 경우 다음을 수행합니다.

- a. 설정 모드에서 minicom을 시작합니다.

```
minicom -s
```

- b. Serial Port Setup(직렬 포트 설정)으로 이동합니다.
c. 다음 설정을 구성합니다:

```
| 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
```

Exit(종료)로 이동하여 직렬 콘솔에서 메시지 표시를 시작합니다.

터미널 에뮬레이터를 설치하여 직렬 연결을 설정하는 데 대한 자세한 내용은 [터미널 에뮬레이터 설치 \(p. 69\)](#) 단원을 참조하십시오.

Amazon FreeRTOS 데모 프로젝트 빌드, 플래시 및 실행

Amazon FreeRTOS의 M320 포트와 함께 포함된 유ти리티 스크립트와 CMake를 사용하여 명령줄에서 Amazon FreeRTOS 데모 프로젝트를 빌드, 플래시 및 실행할 수 있습니다. 또는 IDE를 사용하여 프로젝트를 빌드할 수 있습니다.

CMake로 데모 빌드 파일 생성

Amazon FreeRTOS 다운로드 루트에서 다음 명령을 실행하여 CMake로 데모 빌드 파일을 생성합니다.

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

또는

```
cmake -DVENDOR=marvell -DBOARD=mw322 -DCOMPILER=arm-gcc -S . -B build -DAFR_ENABLE_TESTS=0
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
=====Configuration for Amazon FreeRTOS=====
Version:          1.4.7
Git version:      v1.4.7-128-gc28d0a266

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:              Darwin-17.7.0
  Toolchain:       arm-gcc
  Toolchain path: /usr/local
  CMake generator: Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build: bufferpool, crypto, freertos_plus_tcp, greengrass, kernel,
                     mqtt, pkcs11, secure_sockets, shadow, tls, wifi
  Enabled by user:  greengrass, mqtt, pkcs11, secure_sockets, shadow, wifi
  Enabled by dependency:  bufferpool, common, crypto, demo_base, freertos,
                         freertos_plus_tcp, kernel, pkcs11_mbedtls, tls, utils
  3rdparty dependencies: jsmn, mbedtls, pkcs11
  Available demos:   demo_greengrass, demo_key_provisioning, demo_logging,
                     demo_mqtt_hello_world, demo_mqtt_pubsub, demo_shadow, demo_tcp
  Available tests:  =====
```

make로 데모 빌드

다음 명령을 실행하여 데모를 빌드합니다.

```
cd build
```

```
make all -j4
```

다음과 유사한 출력 화면이 표시되어야 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_dyne
mic_buffers.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/demo_runner/aws_demo_.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$
```

유사한 명령 세트를 사용하여 테스트 프로젝트를 빌드할 수 있습니다.

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

또는

```
cmake -DVENDOR=marvell -DBOARD=mw322 -DCOMPILER=arm-gcc -S . -B build -DAFR_ENABLE_TESTS=1
```

```
cd build
```

```
make all -j4
```

Note

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

애플리케이션 플래시

보드의 플래시 메모리를 프로그래밍하기 위해 flashprog.py 스크립트가 사용됩니다. 스크립트는 Python 2.7로 작성됩니다.

데모 애플리케이션 이미지를 보드로 플래시하려면 먼저 레이아웃 파일과 Boot2 부트로더를 사용하여 보드의 플래시 메모리를 준비하십시오.

레이아웃 파일과 Boot2 부트로더를 로드하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. flashprog.py Python 스크립트를 -l 및 --boot2 옵션과 함께 실행합니다.

```
./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
```

vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt에 정의된 기본 레이아웃 구성에 따라 flashprog 스크립트가 플래시에 레이아웃을 쓹습니다. 플래시에 대한 파티셔닝된 정보가 레이아웃에 보관됩니다.

또한 스크립트는 플래시에 부트로더를 쓱니다. 부트로더는 vendors/marvell/WMSDK/mw320/sdk/boot2/bin/boot2.bin에 있습니다. 보드에 플래시된 후 부트로더가 마이크로컨트롤러 펌웨어 이미지를 로드합니다.

다음과 유사한 출력 화면이 표시되어야 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 0.245458s (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
```

레이아웃 파일과 부트로더를 보드에 플래시한 후 특정 펌웨어를 보드에 플래시하십시오. Wi-Fi 칩셋을 사용하는 경우 해당 펌웨어가 플래시 메모리에 있어야 합니다.

Wi-Fi 펌웨어를 플래시하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. `flashprog.py` Python 스크립트를 `--wififw` 옵션과 함께 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

`flashprog` 스크립트가 펌웨어를 보드로 플래시합니다.

다음과 유사한 출력 화면이 표시되어야 합니다.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked

target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

레이아웃, 부트로더 및 Wi-Fi 펌웨어가 보드에 플래시되면 데모 애플리케이션을 보드에 플래시하고 실행할 수 있습니다.

데모를 플래시하고 실행하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. `flashprog.py` Python 스크립트를 `--mcufw` 및 `-r` 옵션과 함께 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw ./build//marvell/mw320/aws_demos.bin -r
```

`flashprog` 스크립트가 데모를 보드로 플래시합니다. 스크립트를 `-r` 옵션과 함께 실행하면 보드가 재설정됩니다.

보드를 재설정합니다.

로그에 데모 애플리케이션이 보여야 합니다. 다음과 유사하게 출력됩니다.

```
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: 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...
```

Note

aws_tests 애플리케이션을 플래시하려면 동일한 명령을 사용하되 aws_tests.bin 파일을 aws_demos.bin 대신 지정하십시오.
애플리케이션만 변경하는 경우에는 레이아웃, 부트로더 및 Wi-Fi 펌웨어를 다시 로드할 필요가 없습니다. 레이아웃을 변경하는 경우 모든 구성 요소를 다시 로드해야 할 수 있습니다.

데모를 빌드, 플래시 및 실행할 때 다음과 유사한 출력이 표시됩니다.

```
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 70968 [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----
```

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

GNU Debugger에 연결

GDB(GNU Debugger)에 연결하려면

1. 디렉터리를 변경합니다.

```
cd <amazon-freertos>/vendors/marvell/WMSDK/mw320
```

2. arm-none-eabi-gdb 명령으로 GDB에 연결합니다.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../build/vendors/marvell/boards/mw300_rd/aws_demos.axf
```

Amazon FreeRTOS 테스트 애플리케이션을 디버깅하는 경우 `aws_tests.axf`를 대신 대상으로 지정 합니다.

SRAM에 애플리케이션 로드

`ramload.py` 스크립트로 디바이스의 SRAM(Static Random-Access Memory)에 데모를 로드한 후 디바이스에서 애플리케이션을 실행할 수 있습니다. `ramload.py`를 사용하여 애플리케이션을 로드 및 실행하면

`flashprog.py` 스크립트를 사용하여 플래시 메모리에 로드하는 것보다 속도가 빨라 반복적인 개발을 보다 효율적으로 수행할 수 있습니다.

Note

`ramload.py` 스크립트는 Python 2.7로 작성됩니다.

SRAM에 로드하려면

1. Amazon FreeRTOS 다운로드의 루트로 디렉터리를 변경합니다.
2. `ramload.py` Python 스크립트를 `aws_demos.axf` 파일에서 실행합니다.

```
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/ramload.py ./build/vendors/marvell/boards/mw300_rd/aws_demos.axf
```

로그에 데모 애플리케이션이 보여야 합니다. 다음과 유사하게 출력됩니다.

```
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: 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)
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
```

Note

SRAM에 로드된 이미지는 재부팅할 때 지워집니다.

기타 로그 활성화

이 보드를 시작할 때 발생하는 문제를 해결하기 위해 다른 로깅 메시지를 활성화해야 할 수도 있습니다.

보드 관련 로그를 활성화하려면

1. 작업 중인 프로젝트(예: `aws_tests` 또는 `aws_demos`)의 `main.c` 파일을 엽니다.
2. `prvMiscInitialization` 함수에서 `wmstdio_init(UART0_ID, 0)` 호출을 활성화합니다.

Wi-Fi 로그를 활성화하려면

1. Open `vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h`.
2. 매크로 `CONFIG_WLCMGR_DEBUG`를 활성화합니다.

IDE 개발 및 디버깅 사용

IDE 설정

애플리케이션을 개발 및 디버깅하고 프로젝트를 시각화하기 위해 IDE를 사용할 수 있습니다.

예를 들어 Eclipse IDE를 사용하는 경우 `perm_fix.sh` 스크립트를 사용하여 몇 가지 권한을 구성합니다.

```
./vendors/marvell/WMSDK/mw320/tools/bin/perm_fix.sh
```

Eclipse를 설정하려면

1. [Oracle](#)에서 JRE(Java 런타임 환경)를 설치합니다.

Eclipse를 실행하려면 JRE가 필요합니다. JRE 버전(32비트 또는 64비트)은 설치하는 Eclipse 버전(32비트 또는 64비트)과 일치해야 합니다.

2. [Eclipse.org](#)에서 C/C++ 개발자용 Eclipse IDE를 다운로드합니다. Eclipse 버전 4.9.0 이상이 지원됩니다.
3. 다운로드한 아카이브 폴더를 추출하고 플랫폼별 Eclipse 실행 파일을 실행하여 IDE를 시작합니다.

IDE로 데모 빌드

`make`를 사용하여 명령줄에서 직접 데모를 빌드하는 대신 IDE에서 데모 프로젝트의 빌드 파일을 열고 빌드 할 수 있습니다. IDE에서 파일을 열면 빌드하기 전에 프로젝트를 시각화할 수 있습니다.

Note

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

Eclipse로 프로젝트를 빌드하려면

1. Eclipse를 엽니다.
2. 프로젝트를 생성할 Workspace를 선택합니다.
3. Select a wizard 페이지에서 C/C++를 확장한 후 Makefile Project with Existing Code를 선택합니다.
4. Import existing code 페이지에서 `aws_demos` 소스 코드 위치를 찾아 `aws_demos`를 선택한 후 Finish를 선택합니다.
5. Project Explorer에서 `aws_demos`를 마우스 오른쪽 버튼으로 클릭한 후 프로젝트를 빌드합니다.

빌드가 성공하면 `aws_demos.bin` 실행 파일이 생성됩니다.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

MediaTek MT7697Hx 개발 키트에서 시작하기

이 자습서에서는 MediaTek MT7697Hx 개발 키트를 시작하기 위한 지침을 제공합니다. MediaTek MT7697Hx 개발 키트가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 딜렉터리에 대한 경로는 [`<amazon-freertos>`](#)라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
2. Amazon 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 MediaTek MT7697Hx 개발 키트의 USB 포트에 연결하십시오.

Keil MDK를 다운로드하고 설치하십시오.

GUI 기반의 Keil MDK(Microcontroller Development Kit)를 사용하여 보드에 Amazon FreeRTOS 프로젝트를 구성, 빌드 및 실행할 수 있습니다. Keil MDK는 μVision IDE 및 μVision Debugger를 포함합니다.

Note

Keil MDK는 Windows 7, Windows 8 및 Windows 10 64비트 컴퓨터에서만 지원됩니다.

Keil MDK를 다운로드하고 설치하려면

1. [Keil MDK 시작하기](#) 페이지로 이동하여 Download MDK-Core(MDK 코어 다운로드)를 선택합니다.
2. Keil로 등록하려면 사용자의 정보를 입력하고 제출합니다.
3. MDK 실행 파일을 마우스 오른쪽 버튼으로 클릭하고 Keil MDK 설치 관리자를 컴퓨터에 저장합니다.
4. Keil MDK 설치 관리자를 열고 단계에 따라 완료합니다. MediaTek 디바이스 팩(MT76x7 시리즈)을 설치해야 합니다.

직렬 연결 설정

MediaTek MT7697Hx 개발 키트와 직렬 연결을 설정하려면 Arm Mbed Windows 직렬 포트 드라이버를 설치해야 합니다. [Mbed](#)에서 드라이버를 다운로드할 수 있습니다. Windows serial driver(Windows 직렬 드라이버) 페이지의 단계에 따라 MediaTek MT7697Hx 개발 키트의 드라이버를 다운로드하고 설치합니다.

드라이버를 설치한 후에는 COM 포트가 Windows Device Manager에 나타납니다. 디버깅을 위해 HyperTerminal 또는 TeraTerm과 같은 터미널 유ти리티 도구를 사용하여 포트에 대한 세션을 열 수 있습니다.

Note

드라이버를 설치한 후 보드 연결에 문제가 있을 경우 시스템을 재부팅해야 할 수 있습니다.

Keil MDK로 Amazon FreeRTOS 데모 프로젝트 빌드 및 실행

Keil μVision에 Amazon FreeRTOS 데모 프로젝트를 빌드하려면

1. 시작 메뉴에서 Keil μVision 5를 엽니다.
2. `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx` 프로젝트 파일을 엽니다.

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

코드가 빌드되면 `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf`에서 데모 실행 파일을 볼 수 있습니다.

Amazon FreeRTOS 데모 프로젝트를 실행하려면

- MediaTek MT7697Hx 개발 키트를 PROGRAM 모드로 설정합니다.

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

- 터미널 유ти리티를 사용하여 직렬 콘솔 창을 엽니다. 직렬 포트를 115200bps, 패리티 없음, 8비트 및 1 정지 비트로 설정합니다.
- MediaTek MT7697Hx 개발 키트에서 재설정 버튼을 선택합니다.

클라우드에서 MQTT 메시지 모니터링

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

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

- [AWS IoT 콘솔](#)에 로그인합니다.
- 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
- 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

Keil µVision에서 Amazon FreeRTOS 프로젝트 디버깅

현재 Keil µVision이 포함된 MediaTek의 Amazon FreeRTOS 데모 프로젝트를 디버깅하기 전에 Keil µVision에 포함된 MediaTek 패키지를 편집해야 합니다.

Amazon FreeRTOS 프로젝트를 디버깅하기 위해 MediaTek 패키지를 편집하려면

- 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.

3. 디버깅 탭을 선택한 후 CMSIS-DAP Debugger(CMSIS-DAP 디버거)를 선택합니다.
4. `vendors MEDIATEK/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c`를 열고 매크로 `MTK_DEBUGGER`를 1로 설정합니다.
5. 데모 프로젝트를 μ Vision에 다시 빌드합니다.
6. MediaTek MT7697Hx 개발 키트를 PROGRAM 모드로 설정합니다.

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

7. 메뉴에서 Flash(플래시)를 선택한 후 다운로드를 선택합니다.

다운로드가 완료되면 μ Vision에서 알립니다.

8. MediaTek MT7697Hx 개발 키트에서 재설정 버튼을 누릅니다.
9. μ Vision 메뉴에서 디버깅을 선택한 후 Start/Stop Debug Session(시작/정지 디버깅 세션)을 선택합니다. 디버깅 세션을 시작하면 Call Stack + Locals(스택 + 로컬 호출) 창이 열립니다.
10. 메뉴에서 디버깅을 선택한 후 중지를 선택하여 코드 실행을 일시 중지합니다. 프로그램 카운터는 다음 줄에서 정지합니다.

```
{ volatile int wait_ice = 1 ; while ( wait_ice ) ; }
```

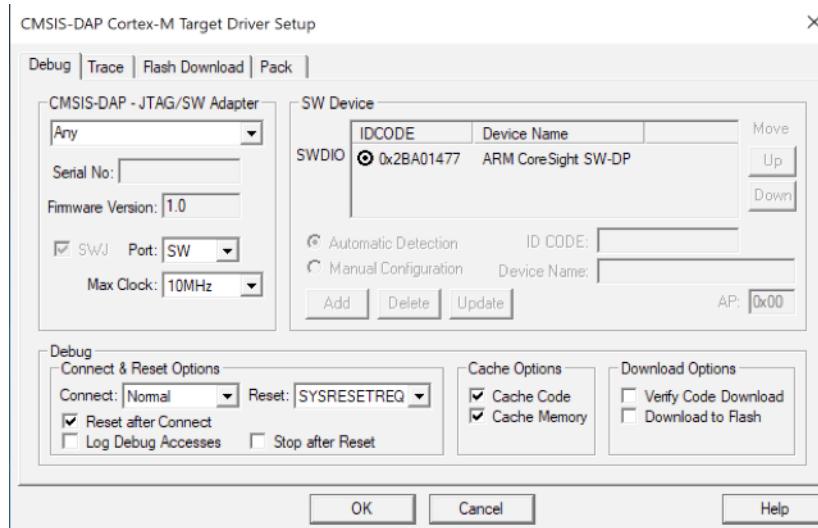
11. Call Stack + Locals(스택 + 로컬 호출) 창에서 `wait_ice` 값을 0로 변경합니다.
12. 프로젝트의 소스 코드에 중단점을 설정하고 코드를 실행합니다.

IDE 디버거 설정 문제 해결

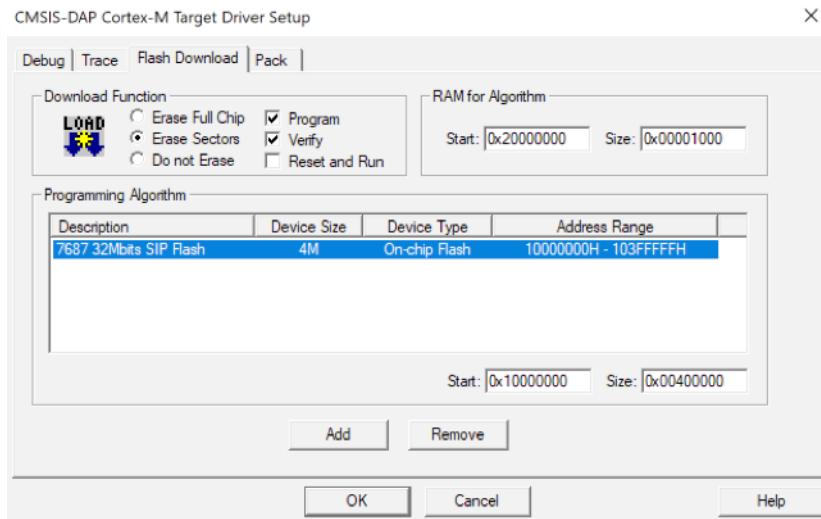
애플리케이션 디버깅에 문제가 있는 경우 디버거 설정이 잘못된 것일 수 있습니다.

디버거 설정이 올바른지 확인하려면

1. Keil μ Vision을 엽니다.
2. aws_demos 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 옵션을 선택한 다음, Utilities(유ти리티) 탭 아래에서 “-- Use Debug Driver --”(디버그 드라이버 사용) 옆에 있는 설정을 선택합니다.
3. 디버깅 탭 아래에서 설정이 다음과 같이 표시되는지 확인합니다.



4. Flash Download(플래시 다운로드) 탭 아래에서 설정이 다음과 같이 표시되는지 확인합니다.



Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Microchip Curiosity PIC32MZ EF 시작하기

이 자습서에서는 Microchip Curiosity PIC32MZ EF를 시작하기 위한 지침을 제공합니다. Microchip Curiosity PIC32MZ EF 번들이 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

번들에는 다음 항목이 포함됩니다.

- [Curiosity PIC32MZ EF Development Board](#)
- [MikroElectronika USB UART Click Board](#)
- [MikroElectronika WiFi 7 Click Board](#)
- [PIC32 LAN8720 PHY 도터 보드](#)

또한 디버깅을 위해 다음 항목도 필요합니다.

- [MPLAB Snap In-Circuit Debugger](#)
- [\(선택 사항\) PICkit 3 Programming Cable Kit](#)

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

Important

Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. Amazon FreeRTOS 프로젝트에서 파일을 수용하려면 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로가 43자 이하여야 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.

2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. Amazon 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. MikroElectronika USB UART Click Board를 Microchip Curiosity PIC32MZ EF의 microBUS 1 커넥터에 연결합니다.
 2. PIC32 LAN8720 PHY 도터 보드를 Microchip Curiosity PIC32MZ EF의 J18 헤더에 연결합니다.
 3. USB A - USB 미니 B 케이블을 사용하여 MikroElectronika USB UART Click Board를 컴퓨터에 연결합니다.
 4. MikroElectronika WiFi 7 Click Board를 Microchip Curiosity PIC32MZ EF의 microBUS 2 커넥터에 연결합니다.
 5. 아직 연결하지 않은 경우 앵글 커넥터를 Microchip Curiosity PIC32MZ EF의 ICSP 헤더에 납땜합니다.
 6. PICkit 3 Programming Cable Kit의 ICSP 케이블 한쪽 끝을 Microchip Curiosity PIC32MZ EF에 연결합니다.
- PICkit 3 Programming Cable Kit가 없는 경우 M-F Dupont 와이어 점퍼를 사용하여 연결합니다. 흰색 원은 Pin 1의 위치를 나타냅니다.
7. ICSP 케이블(또는 점퍼)의 다른쪽 끝을 MPLAB Snap Debugger에 연결합니다. 8핀 SIL Programming Connector의 Pin 1은 보드 오른쪽 하단에 검은색 사각형으로 표시됩니다.

Microchip Curiosity PIC32MZ EF의 Pin 1에 연결된 케이블(흰색 원으로 표시됨)이 MPLAB Snap Debugger의 Pin 1에 맞아야 합니다.

MPLAB Snap Debugger에 대한 자세한 내용은 [MPLAB Snap In-Circuit Debugger Information Sheet](#)를 참조하십시오.

8. 이더넷 케이블의 한 쪽 끝을 LAN8720 PHY 도터 보드에 연결합니다. 다른 쪽 끝을 라우터 또는 다른 인터넷 포트에 연결합니다.

개발 환경 설정

Note

이 디바이스의 Amazon 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.
 - [Windows용 MPLAB X 통합 개발 환경](#)
 - [macOS용 MPLAB X 통합 개발 환경](#)
 - [Linux용 MPLAB X 통합 개발 환경](#)
3. MPLAB XC32 Compiler를 설치합니다.
 - [Windows용 MPLAB XC32/32++ Compiler](#)
 - [macOS용 MPLAB XC32/32++ 컴파일러](#)
 - [Linux용 MPLAB XC32/32++ Compiler](#)

4. UART 터미널 에뮬레이터를 시작하고 다음 설정으로 연결을 엽니다.

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

Amazon FreeRTOS 데모 프로젝트 빌드 및 실행

MPLAB IDE에서 Amazon FreeRTOS 데모 열기

1. MPLAB IDE를 엽니다. 두 개 이상의 컴파일러 버전이 설치된 경우 IDE에서 사용할 컴파일러를 선택해야 합니다.
2. 파일 메뉴에서 Open Project(프로젝트 열기)를 선택합니다.
3. projects/microchip/curiosity_pic32mzef/mplab/aws_demos로 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.

Amazon 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.

CMake로 Amazon FreeRTOS 데모 빌드

Amazon FreeRTOS용 IDE 개발을 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake로 Amazon FreeRTOS 데모를 빌드하려면

1. 생성된 빌드 파일을 포함할 폴더(<BUILD_FOLDER>)를 생성합니다.
2. 소스 코드에서 빌드 파일을 생성하려면 다음 명령을 사용하십시오.

```
cmake -DVENDOR=microchip -DBOARD=curiosity_pic32mzef -DCOMPILER=xc32
      -DMCHP_HEXMATE_PATH=path/microchip/mplabx/v5.10/mplab_platform/bin -
      DAFR_TOOLCHAIN_PATH=path/microchip/xc32/v2.15/bin -S <amazon-freertos> -
      B <BUILD_FOLDER> -DAFR_ENABLE_TESTS=1
```

Note

Hexmate 및 도구 체인 바이너리에 대한 올바른 경로를 지정해야 합니다.

3. 디렉터리를 빌드 디렉터리(<BUILD_FOLDER>)로 변경하고 해당 디렉터리에서 make를 실행합니다.

자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 업니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Nordic nRF52840-DK 시작하기

이 자습서에서는 Nordic nRF52840-DK를 시작하기 위한 지침을 제공합니다. Nordic nRF52840-DK가 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

시작하려면 [Amazon FreeRTOS Bluetooth Low Energy용 AWS IoT 및 Amazon Cognito 설정 \(p. 234\)](#) 단원을 수행해야 합니다.

Amazon FreeRTOS Bluetooth Low Energy 데모를 실행하려면 Bluetooth 및 Wi-Fi 기능이 있는 iOS 또는 Android 모바일 디바이스도 필요합니다.

Note

iOS 디바이스를 사용하는 경우 데모 모바일 애플리케이션을 빌드하려면 Xcode가 필요합니다.
Android 디바이스를 사용하는 경우 Android Studio를 사용하여 데모 모바일 애플리케이션을 빌드할 수 있습니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

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

Nordic 하드웨어 설정

호스트 컴퓨터를 J2라는 레이블이 지정된 USB 포트에 연결합니다. 이 포트는 Nordic nRF52840 보드의 코인셀 배터리 훌더 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

Amazon FreeRTOS는 Segger Embedded Studio를 Nordic nRF52840-DK의 개발 환경으로 지원합니다.

환경을 설정하려면 호스트 컴퓨터에 Segger Embedded Studio를 다운로드하고 설치해야 합니다.

Segger Embedded Studio를 다운로드하고 설치하려면

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

Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션 설정

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

Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 설정하려면

1. [Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK \(p. 206\)](#)의 지침에 따라 모바일 플랫폼용 SDK를 다운로드하여 호스트 컴퓨터에 설치합니다.
2. [Amazon FreeRTOS Bluetooth Low Energy Mobile 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 \(p. 69\)](#)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 \(p. 69\)](#)의 지침에 따라 터미널을 직렬 연결로 보드에 연결하도록 구성합니다.

Amazon FreeRTOS 다운로드 및 구성

하드웨어와 환경을 설정한 후 Amazon FreeRTOS를 다운로드할 수 있습니다.

Amazon FreeRTOS 다운로드

Nordic nRF52840-DK용 Amazon FreeRTOS를 다운로드하려면 [Amazon FreeRTOS GitHub 페이지](#)로 이동하고 리포지토리를 복제합니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.

Note

Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. 이름이 긴 Amazon FreeRTOS 다운로드 디렉터리 경로는 빌드 오류를 발생시킬 수 있습니다.

이 자습서에서 `amazon-freertos` 디렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

프로젝트 구성

데모를 실행하려면 AWS IoT로 작업하도록 프로젝트를 구성해야 합니다. AWS IoT로 작업하도록 프로젝트를 구성하려면 디바이스를 AWS IoT 사물로 등록해야 합니다. [Amazon FreeRTOS Bluetooth Low Energy용 AWS IoT 및 Amazon Cognito 설정 \(p. 234\)](#)을 수행할 때 디바이스가 등록되어 있어야 합니다.

AWS IoT 엔드포인트를 구성하려면

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

AWS IoT 엔드포인트가 엔드포인트 텍스트 상자에 표시됩니다. URL은 <[1234567890123](#)-ats.iot.<[us-east-1](#)>.amazonaws.com와 같아야 합니다. 이 엔드포인트를 기록해둡니다.

3. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다. 디바이스의 AWS IoT 사물 이름을 기록해둡니다.
4. AWS IoT 엔드포인트와 AWS IoT 사물 이름을 기록했으면 IDE에서 <[amazon-freertos](#)>/demos/include/aws_clientcredential.h를 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. `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demos_config.h`를 열고 `CONFIG_MQTT_DEMO_ENABLED`를 정의합니다.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Amazon 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 Amazon FreeRTOS Bluetooth Low Energy 데모를 빌드하고 실행하려면

1. Segger Embedded Studio를 엽니다. 상단 메뉴에서 파일을 선택하고 Open Solution(솔루션 열기)을 선택한 후 `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject` 프로젝트 파일로 이동합니다.
2. Segger Embedded Studio 터미널 에뮬레이터를 사용하는 경우 상단 메뉴에서 도구를 선택한 후 Terminal Emulator(터미널 에뮬레이터), Terminal Emulator(터미널 에뮬레이터)를 선택하여 직렬 연결에 서 정보를 표시합니다.

다른 터미널 도구를 사용하는 경우 직렬 연결에서 출력할 도구를 모니터링할 수 있습니다.

3. Project Explorer에서 aws_demos 데모 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 빌드를 선택합니다.

Note

Segger Embedded Studio를 처음 사용하는 경우 "No license for commercial use"라는 경고가 표시될 수 있습니다. Segger Embedded Studio는 Nordic Semiconductor 디바이스에서 무료로 사용할 수 있습니다. Activate Your Free License를 선택하고 지침을 따릅니다.

4. 디버깅을 선택한 다음 이동을 선택합니다.

데모가 시작되면 Bluetooth Low Energy를 통해 모바일 장치와 페어링되기를 기다립니다.

5. [MQTT over Bluetooth Low Energy 데모 애플리케이션](#)에 대한 지침에 따라 Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 모바일 MQTT 프록시로 사용하여 데모를 완료합니다.

문제 해결

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Nuvoton NuMaker-IoT-M487 시작하기

이 자습서에서는 Nuvoton NuMaker-IoT-M487 개발 보드를 시작하기 위한 지침을 제공합니다. Nuvoton NuMaker-IoT-M487 개발 보드는 NuMicro M487 시리즈 마이크로컨트롤러와 함께 포함되어, 이 개발 보드에는 기본 제공 RJ45 이더넷 및 Wi-Fi 모듈이 포함되어 있습니다. Nuvoton NuMaker-IoT-M487이 없는 경우 [AWS 파트너 디바이스 카탈로그](#)를 방문하여 파트너에서 구입하시기 바랍니다.

시작하려면 먼저 개발 보드를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 소프트웨어를 구성해야 합니다. 지침은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 딜렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

1. 마이크로컨트롤러 보드용 임베디드 애플리케이션을 개발하고 디버깅하기 위한 소프트웨어를 호스트 머신에 설치합니다.
2. Amazon 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.

NuMaker-IoT-M487용 개발 도구를 설치하려면

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. Nu-Link_Keil_Driver_V3.00.6951(또는 최신 버전)를 설치합니다. 이 도구는 [Nuvoton 개발 도구](#) 페이지에 있습니다.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Amazon 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 <[amazon-freertos](#)>\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos 디렉터리에서 대상 프로젝트 aws_demos.uvproj를 선택합니다.
 - 이더넷 데모 프로젝트를 열려면 <[amazon-freertos](#)>\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth 디렉터리에서 대상 프로젝트 aws_demos_eth.uvproj를 선택합니다.
4. 보드를 플래시하기 위한 설정이 올바른지 확인하려면 aws_demo 프로젝트를 마우스 오른쪽 버튼으로 클릭한 다음 Options(옵션)를 선택합니다. (자세한 내용은 [문제 해결 \(p. 144\)](#) 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 클라우드에 전송하는 메시지를 모니터링할 수 있습니다.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. Subscription topic(구독 주제)에 freertos/demos/echo를 입력한 다음 Subscribe to topic(주제 구독)을 선택합니다.

Amazon 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 Amazon FreeRTOS 사용

CMake를 사용하여 Amazon FreeRTOS 데모 애플리케이션 또는 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake 빌드 시스템을 설치했는지 확인합니다. [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#)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.

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드에 전송하는 메시지를 모니터링할 수 있습니다.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. Subscription topic(구독 주제)에 freertos/demos/echo를 입력한 다음 Subscribe to topic(주제 구독)을 선택합니다.

소스 파일에서 빌드 파일을 생성하고 데모 프로젝트를 실행하려면

1. 호스트 머신에서 명령 프롬프트를 열고 <amazon-freertos> 폴더로 이동합니다.
2. 생성된 빌드 파일을 포함할 폴더를 만듭니다. 여기서는 이 폴더를 <BUILD_FOLDER>라고 합니다.
3. Wi-Fi 또는 이더넷 데모에 대한 빌드 파일을 생성합니다.

- Wi-Fi의 경우:

Amazon 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:

Amazon 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>
```

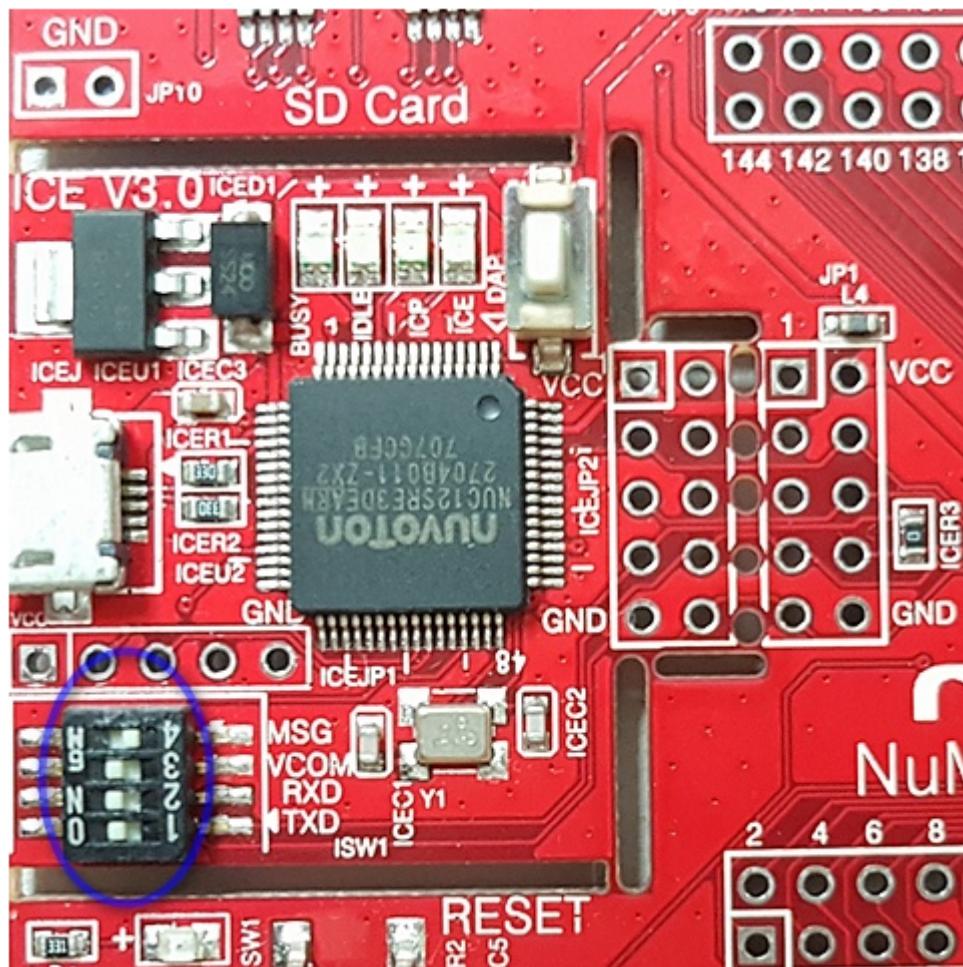
이 시점에서는 이진 파일 aws_demos.bin이 <BUILD_FOLDER>/vendors/Nuvoton/boards/numaker_iot_m487_wifi 폴더에 있어야 합니다.

5. 플래시 모드에 맞게 보드를 구성하려면 MSG 스위치(ICE에서 ISW1의 4번)가 ON으로 전환되어 있는지 확인합니다. 보드에 플러그인하면 창(드라이브)이 할당됩니다. ([문제 해결 \(p. 144\)](#)을 참조하십시오.)
6. 터미널 에뮬레이터를 열어 UART를 통해 메시지를 봅니다. [터미널 에뮬레이터 설치 \(p. 69\)](#)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

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



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

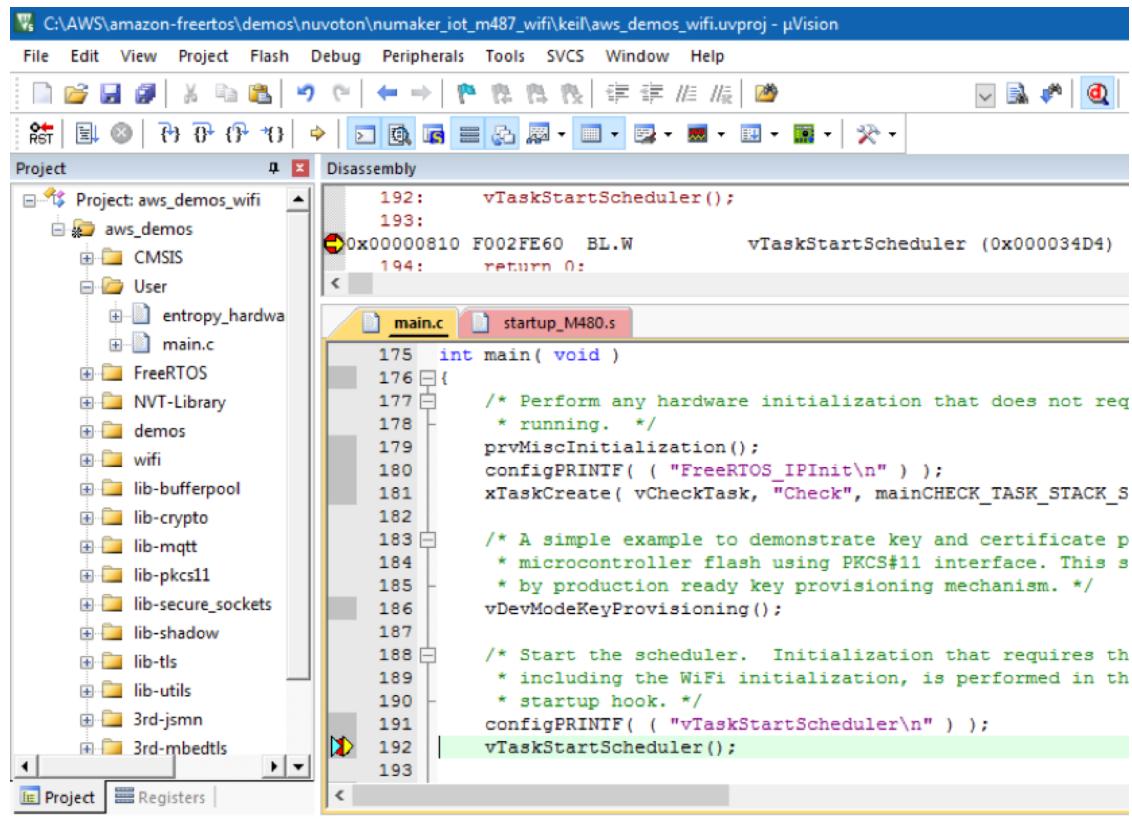
Keil µVision에서 Amazon FreeRTOS 프로젝트 디버깅

Keil µVision에서 디버깅 세션을 시작하려면

- Keil µVision을 엽니다.
- 단계에 따라 [Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.](#) (p. 142)에서 Amazon FreeRTOS 데모 프로젝트를 빌드합니다.
- 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 종단점을 설정한 다음 코드를 실행합니다. 프로젝트는 다음과 비슷해야 합니다.

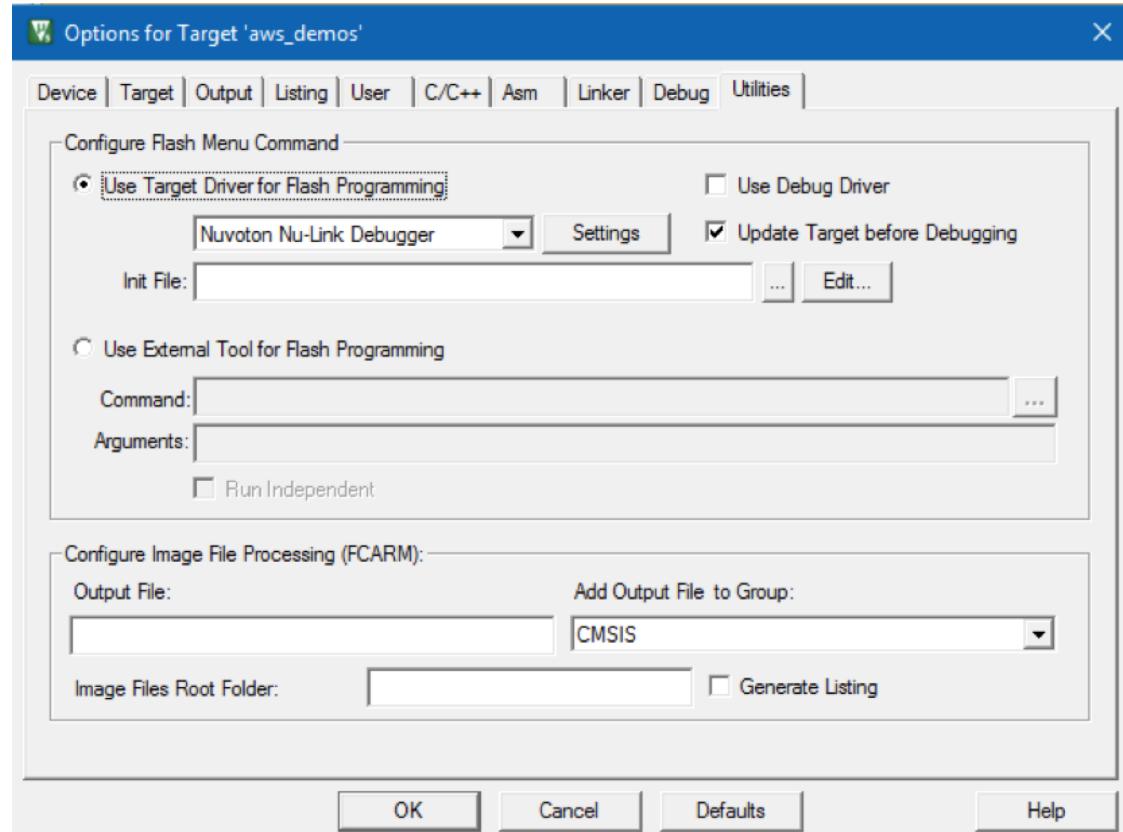


µVision 디버그 설정 문제 해결

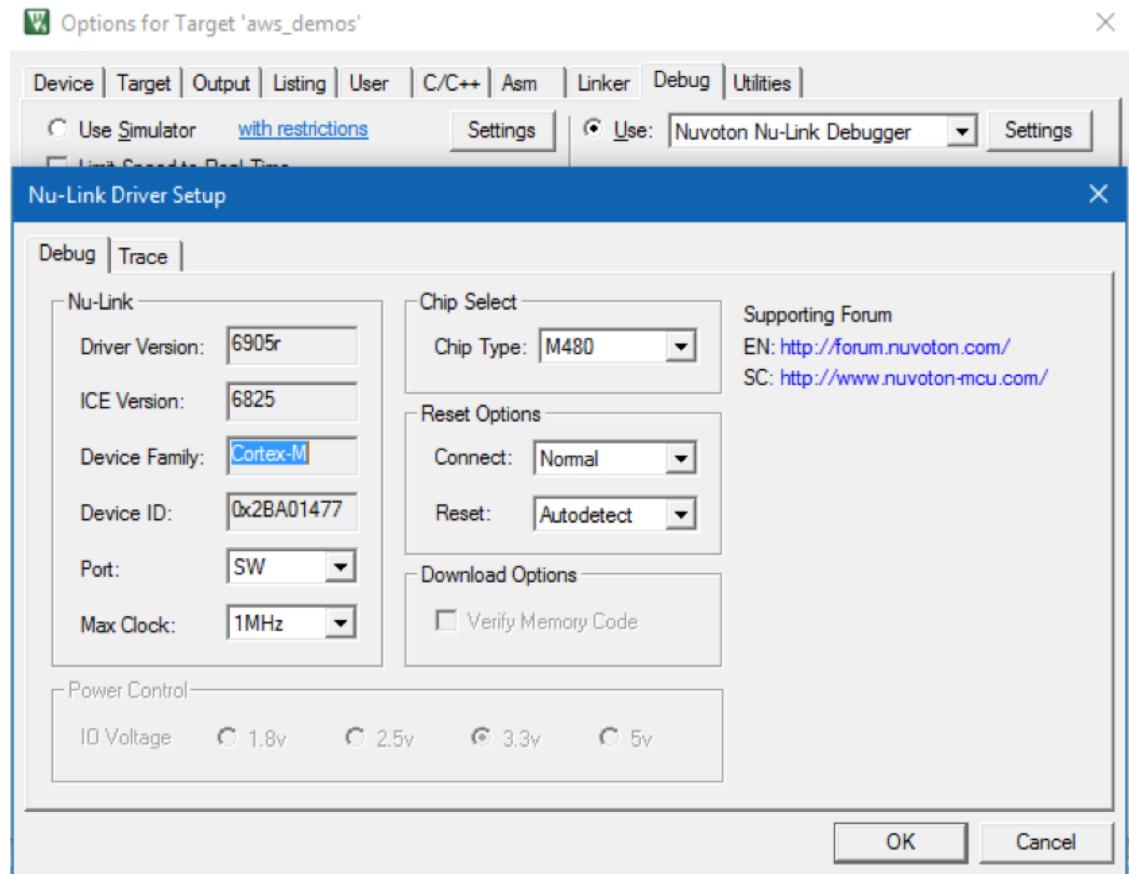
애플리케이션을 디버깅하는 동안 문제가 발생하는 경우 Keil µVision에서 디버그 설정이 올바르게 설정되었는지 확인합니다.

µVision 디버그 설정이 올바른지 확인하려면

1. Keil µVision을 엽니다.
2. IDE에서 aws_demo 프로젝트를 마우스 오른쪽 버튼으로 클릭한 다음 Options(옵션)를 선택합니다.
3. Utilities(유틸리티) 탭에서 Use Target Driver for Flash Programming(플래시 프로그래밍에 대상 드라이버 사용)이 선택되어 있는지 확인하고 Nuvoton Nu-Link Debugger(Nuvoton Nu-Link 디버거)가 대상 드라이버로 설정되어 있는지 확인합니다.



4. Debug(디버깅) 탭에서 Nuvoton Nu-Link Debugger(Nuvoton Nu-Link 디버거) 옆의 Settings(설정)를 선택합니다.



5. Chip Type(칩 유형)이 M480으로 설정되어 있는지 확인합니다.

NXP LPC54018 IoT 모듈 시작하기

이 자습서에서는 NXP LPC54018 IoT 모듈을 시작하기 위한 지침을 제공합니다. NXP LPC54018 IoT 모듈이 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다. USB 케이블을 사용하여 NXP LPC54018 IoT Module을 컴퓨터에 연결합니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. Amazon 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 코드 실행을 시작하고 디버깅하려면 JTAG 디버거가 필요합니다. Amazon FreeRTOS는 OM40006 IoT 모듈을 사용하여 테스트되었습니다. 지원되는 디버거에 대한 자세한 내용은 [OM40007 LPC54018 IoT 모듈](#) 제품 페이지에서 사용할 수 있는 NXP LPC54018 IoT 모듈용 사용 설명서를 참조하십시오.

- 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

Amazon FreeRTOS는 NXP LPC54018 IoT 모듈에 대해 IAR Embedded Workbench 및 MCUXpresso라는 두 개의 IDE를 지원합니다.

시작하기 전에 이러한 IDE 중 하나를 설치합니다.

IAR Embedded Workbench for ARM을 설치하려면

- [Software for NXP Kits\(NXP 키트용 소프트웨어\)](#)로 이동하여 Download Software(소프트웨어 다운로드)를 선택합니다.

Note

IAR Embedded Workbench for ARM에는 Microsoft Windows가 필요합니다.

- 설치 관리자의 앱축을 풀고 실행합니다. 다음에 나타나는 메시지를 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 SDK](#)로 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.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Amazon FreeRTOS 데모를 IDE로 가져옵니다.

Amazon FreeRTOS 샘플 코드를 IAR Embedded Workbench IDE로 가져오려면

1. IAR Embedded Workbench를 열고 File(파일) 메뉴에서 Open Workspace(작업 공간 열기)를 선택합니다.
2. search-directory(검색 디렉터리) 텍스트 상자에 projects/nxp/lpc54018iotmodule/iar/aws_demos를 입력하고 aws_demos.eww를 선택합니다.
3. Project(프로젝트) 메뉴에서 Rebuild All(모두 다시 빌드)을 선택합니다.

Amazon FreeRTOS 샘플 코드를 MCUXpresso IDE로 가져오려면

1. MCUXpresso를 열고 File(파일) 메뉴에서 Open Projects From File System(파일 시스템에서 프로젝트 열기)을 선택합니다.
2. 디렉터리 텍스트 상자에 projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos를 입력하고 마침을 선택합니다.
3. Project(프로젝트) 메뉴에서 Build All(모두 빌드)을 선택합니다.

Amazon FreeRTOS 데모 프로젝트 실행

IAR Embedded Workbench IDE로 Amazon FreeRTOS 데모 프로젝트를 실행하려면

1. IDE의 프로젝트 메뉴에서 Make를 선택합니다.
2. 프로젝트 메뉴에서 Download and Debug(다운로드 및 디버깅)를 선택합니다.
3. Debug(디버그) 메뉴에서 Start Debugging(디버깅 시작)을 선택합니다.
4. 디버거가 main의 중단점에서 중지되면 Debug(디버그) 메뉴에서 Go(이동)를 선택합니다.

Note

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

MCUexpresso IDE로 Amazon FreeRTOS 데모 프로젝트를 실행하려면

1. IDE의 프로젝트 메뉴에서 빌드를 선택합니다.
2. 처음 디버깅하는 경우 aws_demos 프로젝트를 선택하고 디버깅 도구 모음에서 파란색 디버그 버튼을 선택합니다.
3. 발견된 디버그 프로보가 표시됩니다. 사용할 프로보를 선택한 다음 OK(확인)를 선택하여 디버깅을 시작합니다.

Note



디버거가 `main()`의 중단점에서 중지되면 디버그 다시 시작 버튼 을 한 번 눌러 디버깅 세션을 재설정합니다. (이 단계는 NXP54018-IoT-Module용 MCUXpresso 디버거의 버그로 인해 필요합니다).

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

클라우드에서 MQTT 메시지 모니터링

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

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Renesas Starter Kit+ for RX65N-2MB 시작하기

이 자습서에서는 Renesas Starter Kit+ for RX65N-2MB를 시작하기 위한 지침을 제공합니다. Renesas RSK+ for RX65N-2MB가 없는 경우 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉토리에 대한 경로는 `<amazon-freeertos>`라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. Amazon FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

Renesas 하드웨어 설정

RSK+ for RX65N-2MB를 설정하려면

1. 양의 +5V 전원 어댑터를 RSK+ for RX65N-2MB의 PWR 커넥터에 연결합니다.
2. 컴퓨터를 RSK+ for RX65N-2MB의 USB2.0 FS 포트에 연결합니다.
3. 컴퓨터를 RSK+ for RX65N-2MB의 USB-to-serial 포트에 연결합니다.
4. 라우터 또는 인터넷에 연결된 이더넷 포트를 RSK+ for RX65N-2MB의 이더넷 포트에 연결합니다.

E2 Lite 디버거 모듈을 설정하려면

1. 14핀 리본 케이블을 사용하여 E2 Lite 디버거 모듈을 RSK+ for RX65N-2MB의 'E1/E2 Lite' 포트에 연결합니다.
2. USB 케이블을 사용하여 E2 Lite 디버거 모듈을 호스트 머신에 연결합니다. E2 Lite 디버거가 보드와 컴퓨터들 다에 연결된 경우 디버거에서 녹색 'ACT' LED가 깜박입니다.
3. 디버거가 호스트 컴퓨터와 RSK+ for RX65N-2MB에 연결되고 나면 E2 Lite 디버거 드라이버가 설치되기 시작합니다.

드라이버를 설치하려면 관리자 권한이 필요합니다.



개발 환경 설정

RSK+ for RX65N-2MB에 대한 Amazon FreeRTOS 구성 설정하려면 Renesas e²studio IDE 및 CC-RX 컴파일러를 사용합니다.

Note

Renesas e²studio IDE 및 CC-RX 컴파일러는 Windows 7, 8, 10 운영 체제에서만 지원됩니다.

e²studio 다운로드 및 설치

1. [Renesas e²studio 설치 관리자](#) 다운로드 페이지로 이동하여 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 e²studio 설치 관리자를 컴퓨터에 다운로드합니다.
4. 설치 관리자를 열고 단계에 따라 완료합니다.

자세한 내용은 Renesas 웹사이트에서 [e²studio](#)를 참조하십시오.

RX Family C/C++ 컴파일러 패키지 다운로드 및 설치

1. [RX Family C/C++ 컴파일러 패키지](#) 다운로드 페이지로 이동하여 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.

Amazon FreeRTOS 샘플 빌드 및 실행

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

e²studio에서 Amazon FreeRTOS 데모 빌드

e²studio에서 데모를 가져와서 빌드하기

1. 시작 메뉴에서 e²studio를 시작합니다.
2. Select a directory as a workspace(작업 영역으로 디렉터리 선택) 창에서 작업하려는 폴더를 찾아 시작을 선택합니다.
3. 처음 e²studio를 열면 Toolchain Registry(도구 체인 등록) 창이 열립니다. Renesas Toolchains(Renesas 도구 체인)을 선택하고 **CC-RX v3.00.00**이 선택되어 있는지 확인합니다. 등록을 선택한 다음 확인을 선택합니다.
4. 처음으로 e²studio를 열 경우 Code Generator Registration(코드 생성기 등록) 창이 나타납니다. 확인을 선택합니다.
5. Code Generator COM component register(코드 생성기 COM 구성요소 등록)창이 나타납니다. Please restart e²studio to use Code Generator에서 확인을 선택합니다.
6. e²studio 다시 시작 창이 나타납니다. 확인을 선택합니다.
7. e²studio가 다시 시작됩니다. Select a directory as a workspace(작업 영역으로 디렉터리 선택) 창에서 시작을 선택합니다.
8. e²studio 시작 화면에서 e²studio 워크벤치로 이동 화살표 아이콘을 선택합니다.
9. Project Explorer 창을 마우스 오른쪽 버튼으로 클릭하고 가져오기를 선택합니다.
10. 가져오기 마법사에서 General(일반), Existing Projects into Workspace(기존 프로젝트를 작업 공간으로)를 선택한 후 다음을 선택합니다.
11. 찾아보기를 선택하고 `projects/renesas/rx65n-rsk/e2studio/aws_demos` 디렉터리를 찾은 후 마침을 선택합니다.
12. 프로젝트 메뉴에서 프로젝트, Build All(모두 빌드)를 선택합니다.

빌드 콘솔이 라이선스 관리자가 설치되어 있지 않다는 경고 메시지를 표시합니다. CC-RX 컴파일러 용 라이선스 키가 없다면 이 메시지를 무시할 수 있습니다. 라이선스 관리자를 설치하려면 [License Manager\(라이선스 관리자\)](#) 다운로드 페이지를 참조하십시오.

Amazon FreeRTOS 프로젝트 실행

e²studio에서 프로젝트 실행하기

1. E2 Lite 디버거 모듈을 RSK+ for RX65N-2MB에 연결했는지 확인합니다.
2. 상단 메뉴에서 Run(실행), Debug Configuration(디버그 구성)을 선택합니다.
3. Renesas GDB Hardware Debugging(Renesas GDB 하드웨어 디버깅)을 확장하고 aws_demos HardwareDebug를 선택합니다.
4. Debugger(디버거) 탭을 선택한 다음 Connection Settings(연결 설정) 탭을 선택합니다. 연결 설정이 올바른지 확인합니다.
5. Debug(디버그)를 선택하여 코드를 보드에 다운로드하고 디버깅을 시작합니다.

e2-server-gdb.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. e²studio가 Renesas Debug Perspective(Renesas 디버그 관점)로 변경하도록 요청할 수 있습니다. Yes를 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.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 업니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

Renesas가 릴리스한 최신 프로젝트는 [GitHub](#)에서 amazon-freertos 리포지토리의 `renesas-rx` 포크를 확인하십시오.

문제 해결

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

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 AWS Partner Device Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

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 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 딜렉토리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

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

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Amazon 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/stm321475_discovery/ac6/aws_demos를 입력합니다.
4. aws_demos 프로젝트가 기본적으로 선택되어야 합니다.
5. Finish(완료)를 선택하여 프로젝트를 STM32 System Workbench로 가져옵니다.
6. 프로젝트 메뉴에서 Build All(모두 빌드)을 선택합니다. 오류 없이 프로젝트가 컴파일되는지 확인합니다.

Amazon FreeRTOS 데모 프로젝트 실행

1. USB 케이블을 사용하여 STMicroelectronics STM32L4 Discovery Kit IoT Node를 컴퓨터에 연결합니다.
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 종단점에서 중지되면 실행 메뉴에서 다시 시작을 선택합니다.

CMake와 Amazon FreeRTOS 사용

Amazon FreeRTOS용 IDE 개발을 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

먼저 생성된 빌드 파일을 포함할 폴더(<BUILD_FOLDER>)를 생성합니다.

빌드 파일을 생성할 때는 다음 명령을 사용합니다.

```
cmake -DVENDOR=st -DBOARD=stm321475_discovery -DCOMPILER=arm-gcc -S <amazon-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
```

Amazon FreeRTOS에서 CMake를 사용하는 방법에 대한 자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

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

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

최신 Wi-Fi 펌웨어를 다운로드하려면 [STM32L4 Discovery 키트 IoT 노드, 저전력 무선, Bluetooth Low Energy, NFC, SubGHz, Wi-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.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

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 Catalog를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 `<amazon-freertos>`라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

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

2. Amazon FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

개발 환경 설정

아래 단계에 따라 개발 환경을 설정하고 Amazon FreeRTOS를 시작합니다.

Amazon FreeRTOS는 TI CC3220SF-LAUNCHXL 개발 키트 Code Composer Studio와 IAR Embedded Workbench 버전 8.32라는 2개의 IDE를 지원합니다. 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. Product Families to Install(설치할 제품군)에서 SimpleLink Wi-Fi CC32xx Wireless MCUs(SimpleLink Wi-Fi CC32xx 무선 MCU)를 선택합니다.
5. 다음 페이지에서 프로브 디버깅에 대한 기본 설정을 수락하고 Finish(완료)를 선택합니다.

Code Composer Studio를 설치하는 동안 문제가 발생할 경우 [TI 개발 도구 지원](#), [Code Composer Studio FAQ](#) 및 [CCS 문제 해결](#)을 참조하십시오.

IAR Embedded Workbench 설치

1. ARM용 IAR Embedded Workbench [버전 8.32용 Windows 설치 프로그램](#)을 다운로드하여 실행합니다. Debug probe drivers(프로브 드라이버 디버깅)에서 TI XDS가 선택되어 있는지 확인합니다.
2. 설치를 완료하고 프로그램을 시작합니다. License Wizard(라이선스 마법사) 페이지에서 Register with IAR Systems to get an evaluation license(IAR 시스템에 등록하여 평가 라이선스 받기)를 선택하거나 고유의 IAR 라이선스를 사용합니다.

SimpleLink CC3220 SDK 설치

[SimpleLink CC3220 SDK](#)를 설치합니다. SimpleLink Wi-Fi CC3220 SDK에는 CC3220SF 프로그래밍 가능한 MCU용 드라이버, 40개 이상의 샘플 애플리케이션 및 샘플을 사용하는 데 필요한 설명서가 포함되어 있습니다.

Uniflash 설치

[Uniflash](#)를 설치합니다. CCS Uniflash는 TI MCU에서 온칩 플래시 메모리를 프로그래밍하는 데 사용되는 독립형 도구입니다. Uniflash에는 GUI, 명령줄 및 스크립팅 인터페이스가 있습니다.

최신 서비스 팩 설치

1. TI CC3220SF-LAUNCHXL에서 SOP 점퍼를 중간 핀 집합(위치 = 1)에 놓고 보드를 재설정합니다.
2. Uniflash를 시작합니다. CC3220SF LaunchPad 보드가 Detected Devices(검색된 디바이스)에 나타나면 Start(시작)를 선택합니다. 보드가 검색되지 않으면 New Configuration(새 구성)의 보드 목록에서 CC3220SF-LAUNCHXL을 선택한 후 Start Image Creator(이미지 생성시 시작)를 선택합니다.
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. Browse(찾아보기)를 선택한 다음 CC3220SF SimpleLink SDK를 설치한 위치로 이동합니다. 서비스 팩은 `ti/simplelink_cc32xx_sdk_<VERSION>/tools/cc32xx_tools/servicepack-cc3x20/sp_<VERSION>.bin`에 있습니다.
8.
 Burn() 버튼을 선택한 다음 프로그램 이미지(생성 및 프로그램)을 선택하여 서비스 팩을 설치합니다. SOP 점퍼를 다시 0 위치로 전환하고 보드를 재설정해야 합니다.

Wi-Fi 프로비저닝 구성

보드에 대한 Wi-Fi 설정을 구성하려면 다음 중 하나를 수행합니다.

- [Amazon FreeRTOS 데모 구성 \(p. 66\)](#)에 설명된 Amazon FreeRTOS 데모 애플리케이션을 구성하십시오.
- Texas Instruments의 [SmartConfig](#)를 사용합니다.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

TI Code Composer에서 Amazon FreeRTOS 데모 프로젝트 빌드 및 실행

Amazon FreeRTOS 데모를 TI Code Composer로 가져오려면

1. TI Code Composer를 열고 OK(확인)를 선택하여 기본 작업 영역 이름을 수락합니다.
2. Getting Started(시작하기) 페이지에서 Import Project(프로젝트 가져오기)를 선택합니다.
3. Select search-directory(검색 디렉터리 선택)에 `projects/ti/cc3220_launchpad/ccs/aws_demos`를 입력합니다. `aws_demos` 프로젝트가 기본적으로 선택되어야 합니다. 프로젝트를 TI Code Composer로 가져오려면 Finish(완료)를 선택합니다.
4. Project Explorer(프로젝트 탐색기)에서 `aws_demos`를 두 번 클릭하여 프로젝트를 활성화합니다.
5. Project(프로젝트)에서 Build Project(프로젝트 빌드)를 선택하여 오류나 경고 없이 프로젝트가 컴파일되는지 확인합니다.

TI Code Composer에서 Amazon FreeRTOS 데모를 실행하려면

1. Texas Instruments CC3220SF-LAUNCHXL의 Sense On Power(SOP) 점퍼가 0 위치에 있는지 확인합니다. 자세한 내용은 [CC3220 SimpleLink 사용 설명서](#)를 참조하십시오.
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 중단점에서 중지되면 실행 메뉴로 이동하여 다시 시작을 선택합니다.

IAR Embedded Workbench에서 Amazon FreeRTOS 데모 프로젝트 빌드 및 실행

Amazon FreeRTOS 데모를 IAR Embedded Workbench로 가져오려면

1. IAR Embedded Workbench를 열고 File(파일)을 선택한 다음 Open Workspace(작업 공간 열기)를 선택합니다.

2. projects/ti/cc3220_launchpad/iar/aws_demos로 이동하고 aws_demos.eww를 선택한 다음 확인을 선택합니다.
3. 프로젝트 이름(aws_demos)을 마우스 오른쪽 버튼으로 클릭한 다음 Make(만들기)를 선택합니다.

IAR Embedded Workbench에서 Amazon FreeRTOS 데모를 실행하려면

1. Texas Instruments CC3220SF-LAUNCHXL의 Sense On Power(SOP) 점퍼가 0 위치에 있는지 확인합니다. 자세한 내용은 [CC3220 SimpleLink 사용 설명서](#)를 참조하십시오.
2. USB 케이블을 사용하여 Texas Instruments CC3220SF-LAUNCHXL을 컴퓨터에 연결합니다.
3. 프로젝트를 다시 빌드합니다.

프로젝트를 다시 빌드하려면 Project(프로젝트) 메뉴에서 Make(만들기)를 선택합니다.

4. Project(프로젝트) 메뉴에서 Download and Debug(다운로드 및 디버그)를 선택합니다. "Warning: Failed to initialize EnergyTrace(경고: EnergyTrace를 초기화하지 못함)"가 표시되는 경우 이 메시지를 무시할 수 있습니다. EnergyTrace에 대한 자세한 내용은 [MSP EnergyTrace Technology](#)를 참조하십시오.
5. 디버거가 main()의 중단점에서 중지되면 디버깅 메뉴로 이동하여 이동을 선택합니다.

CMake와 Amazon FreeRTOS 사용

Amazon FreeRTOS용 IDE 개발을 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake로 Amazon 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 <amazon-freertos> -B <BUILD_FOLDER>
```

IAR 컴파일러를 사용하는 경우 다음 명령을 사용하십시오.

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-iar -S <amazon-freertos> -B <BUILD_FOLDER>
```

자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

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.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Windows Device Simulator 시작하기

이 자습서에서는 Amazon FreeRTOS Windows Device Simulator를 시작하기 위한 지침을 제공합니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 디렉터리에 대한 경로는 <amazon-freertos>라고 합니다.

Amazon FreeRTOS는 지정한 플랫폼용 Amazon FreeRTOS 라이브러리와 샘플 애플리케이션이 포함된 zip 파일로 릴리스됩니다. Windows 머신에서 샘플을 실행하려면 Windows에서 실행하도록 이식된 라이브러리와 샘플을 다운로드합니다. 이 파일 세트를 Windows용 Amazon FreeRTOS 시뮬레이터라고 합니다.

개발 환경 설정

1. 최신 버전의 [WinPCap](#)을 설치합니다.
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. Desktop development with C++ 구성 요소의 선택 사항 섹션에서 이 옵션을 선택할 수 있습니다.

3. 활성 유선 이더넷 연결이 있는지 확인합니다.
4. (선택 사항) Amazon FreeRTOS 프로젝트를 빌드하기 위해 CMake 기반 빌드 시스템을 사용하려면 최신 버전의 [CMake](#)를 설치하십시오. Amazon FreeRTOS를 사용하려면 CMake 버전 3.13 이상이 필요합니다.

Amazon FreeRTOS 데모 프로젝트를 빌드하고 실행합니다.

Visual Studio 또는 CMake를 사용하여 Amazon FreeRTOS 프로젝트를 빌드할 수 있습니다.

Visual Studio IDE를 사용하여 Amazon 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_demos를 마우스 오른쪽 버튼으로 클릭하고 Retarget Projects(프로젝트 대상 재지정)를 선택합니다. Review Solution Actions(솔루션 작업 검토) 창이 열립니다. 컴퓨터에 있는 Windows SDK 버전(드롭다운의 초기 값)을 선택하고 확인을 선택합니다.

3. 프로젝트를 빌드 및 실행합니다.

구축 메뉴에서 솔루션 구축을 선택하고 솔루션이 오류나 경고 없이 빌드되는지 확인합니다. 디버깅, Start Debugging(디버깅 시작)을 선택하여 프로젝트를 실행합니다. 처음 실행하는 경우 네트워크 인터페이스를 선택 (p. 161)해야 합니다.

CMake로 Amazon 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. Amazon FreeRTOS 소스 코드 디렉터리를 설정합니다.

GUI의 Where is the source code(소스 코드 위치)에서 Amazon FreeRTOS 소스 코드 디렉터리 (<amazon-freertos>)를 설정합니다.

Where to build the binaries(바이너리 빌드 위치)에 대해 <amazon-freertos>/build를 설정합니다.

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 생성기 옵션은 그대로 두고 마침을 선택합니다.
4. CMake 프로젝트를 생성하여 엽니다.

프로젝트를 구성한 후, CMake GUI는 생성된 프로젝트에 사용할 수 있는 모든 옵션을 표시합니다. 이 자습서에서는 옵션을 기본값으로 두어도 됩니다.

생성을 선택하여 Visual Studio 솔루션을 만든 다음 프로젝트 열기를 선택하여 Visual Studio에서 프로젝트를 엽니다.

Visual Studio에서 aws_demos 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 Set as StartUp Project(시작 프로젝트로 설정)를 선택합니다. 이를 통해 프로젝트를 빌드하고 실행할 수 있습니다. 처음 실행하는 경우 [네트워크 인터페이스를 선택 \(p. 161\)](#)해야 합니다.

Amazon FreeRTOS와 함께 CMake를 사용하는 방법에 대한 자세한 내용은 [CMake와 Amazon FreeRTOS 사용 \(p. 70\)](#) 단원을 참조하십시오.

네트워크 인터페이스 구성

데모 프로젝트의 첫 번째 실행 시 사용할 네트워크 인터페이스를 선택해야 합니다. 프로그램에서 네트워크 인터페이스가 열거됩니다. 유선 이더넷 인터페이스의 번호를 찾습니다. 출력은 다음과 같아야 합니다.

```
0 0 [None] FreeRTOS_IPInit
1 0 [None] vTaskStartScheduler
1. rpcap://\Device\NPF_{AD01B877-A0C1-4F33-8256-EE1F4480B70D}
(Network adapter 'Intel(R) Ethernet Connection (4) I219-LM' on local host)

2. rpcap://\Device\NPF_{337F7AF9-2520-4667-8EFF-2B575A98B580}
(Network adapter 'Microsoft' on local host)

The interface that will be opened is set by "configNETWORK_INTERFACE_TO_USE", which
should be defined in FreeRTOSConfig.h

ERROR: configNETWORK_INTERFACE_TO_USE is set to 0, which is an invalid value.
Please set configNETWORK_INTERFACE_TO_USE to one of the interface numbers listed above,
then re-compile and re-start the application. Only Ethernet (as opposed to WiFi)
interfaces are supported.
```

유선 이더넷 인터페이스의 번호를 식별한 후 애플리케이션 창을 닫습니다. 위의 예에서 사용할 번호는 1입니다.

FreeRTOSConfig.h를 열고 configNETWORK_INTERFACE_TO_USE를 유선 네트워크 인터페이스에 해당하는 번호로 설정합니다.

클라우드에서 MQTT 메시지 모니터링

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 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

문제 해결

Windows의 일반적인 문제 해결

Visual Studio로 데모 프로젝트를 빌드할 때 다음 오류가 발생할 수 있습니다.

Error "The Windows SDK version X.Y was not found" when building the provided Visual Studio solution.

프로젝트는 컴퓨터에 있는 Windows SDK 버전을 대상으로 해야 합니다.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Xilinx Avnet MicroZed Industrial IoT Kit 시작하기

이 자습서에서는 Xilinx Avnet MicroZed Industrial IoT 키트를 시작하기 위한 지침을 제공합니다. Xilinx Avnet MicroZed Industrial IoT 키트가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 [파트너](#)에서 구입하시기 바랍니다.

시작하려면 먼저 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 Amazon FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 [첫 번째 단계 \(p. 62\)](#) 단원을 참조하십시오. 이 자습서에서 Amazon FreeRTOS 다운로드 딕터리에 대한 경로는 <[amazon-freertos](#)>라고 합니다.

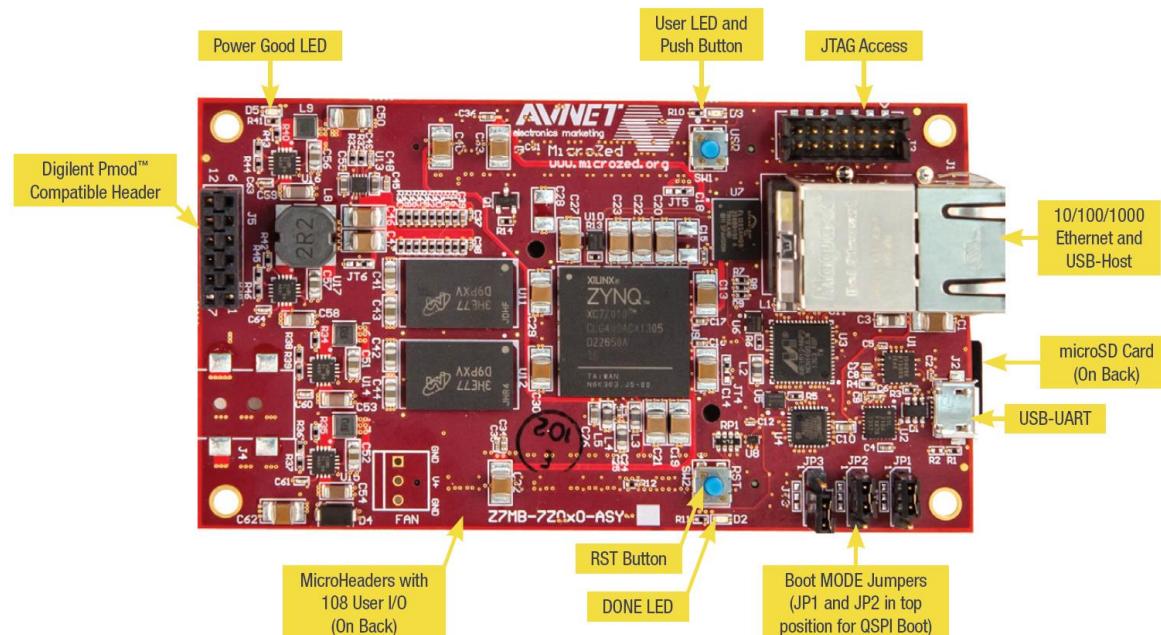
개요

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.
- Amazon FreeRTOS 데모 애플리케이션을 이진 이미지로 크로스 컴파일합니다.
- 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

MicroZed 하드웨어 설정

다음 다이어그램은 MicroZed 하드웨어를 설정할 때 유용할 수 있습니다.



MicroZed 보드를 설정하려면

1. 컴퓨터를 MicroZed 보드의 USB-UART 포트에 연결합니다.
2. 컴퓨터를 MicroZed 보드의 JTAG Access 포트에 연결합니다.
3. 라우터 또는 인터넷에 연결된 이더넷 포트를 MicroZed 보드의 이더넷 및 USB-Host 포트에 연결합니다.

개발 환경 설정

MicroZed 키트에 대한 Amazon FreeRTOS 구성 설정을 하려면 Xilinx Software Development Kit(XSDK)를 사용해야 합니다. XSDK는 Windows 및 Linux에서 지원됩니다.

XSDK를 다운로드하고 설치합니다.

Xilinx 소프트웨어를 설치하려면 무료 Xilinx 계정이 필요합니다.

XSDK를 다운로드하려면

1. [Software Development Kit Standalone WebInstall Client](#) 다운로드 페이지로 이동합니다.
2. 운영 체제에 적합한 옵션을 선택합니다.
3. Xilinx 로그인 페이지로 이동합니다.

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 MicroZed's USB-UART 연결을 감지하지 못하는 경우 CP210x USB-to-UART Bridge VCP 드라이버를 수동으로 설치합니다. 자침은 [Silicon Labs CP210x USB-to-UART Installation Guide](#)를 참조하십시오.

XSDK에 대한 자세한 내용은 Xilinx 웹 사이트에서 [Getting Started with Xilinx SDK](#)를 참조하십시오.

Amazon FreeRTOS 데모 프로젝트 빌드 및 실행

XSDK IDE에서 Amazon FreeRTOS 데모 열기

1. 작업 영역 디렉터리가 `projects/xilinx/microzed/xsdk`로 설정된 상태로 XSDK IDE를 시작합니다.
2. 시작 페이지를 닫습니다. 메뉴에서 Project를 선택한 후 Build Automatically를 지웁니다.
3. 메뉴에서 File을 선택한 후 Import를 선택합니다.
4. Select 페이지에서 General을 확장하고 Existing Projects into Workspace를 선택한 후 Next를 선택합니다.
5. Import Projects 페이지에서 Select root directory를 선택한 후 데모 프로젝트의 루트 디렉터리를 입력합니다. 디렉터리를 찾아보려면 Browse를 선택합니다.

루트 디렉터리를 지정하면 Import Projects 페이지에 해당 디렉터리의 프로젝트가 표시됩니다. 사용 가능한 모든 프로젝트가 기본적으로 선택되어 있습니다.

Note

Import Projects 페이지의 상단에 경고가 표시되는 경우("Some projects cannot be imported because they already exist in the workspace") 이를 무시할 수 있습니다.

6. 프로젝트가 모두 선택된 상태에서 Finish를 선택합니다. XSDK IDE가 aws_demos 프로젝트를 MicroZed 보드에서 빌드 및 실행하는 데 필요한 모든 프로젝트를 업니다.
7. 메뉴에서 Window를 선택한 후 Preferences를 선택합니다.
8. 탐색 창에서 Run/Debug를 확장하고 String Substitution을 선택한 후 New를 선택합니다.
9. New String Substitution Variable의 Name에 **AFR_ROOT**를 입력합니다. Value에 aws_demos의 루트 경로를 입력합니다. OK를 선택한 후 OK를 선택하여 변수를 저장하고 Preferences를 닫습니다.

Amazon 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.

클라우드에서 MQTT 메시지 모니터링

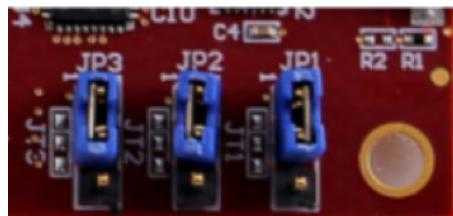
AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링 할 수 있습니다.

AWS IoT MQTT 클라이언트를 사용하여 MQTT 주제를 구독하려면

1. [AWS IoT 콘솔](#)에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 `iotdemo/#`을 입력한 다음 주제 구독을 선택합니다.

JTAG 디버깅

1. MicroZed 보드의 부팅 모드 점퍼를 JTAG 부팅 모드로 설정합니다.

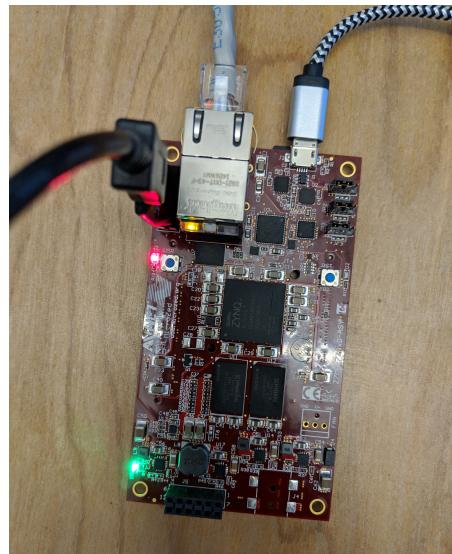


2. USB-UART 포트 바로 아래에 있는 MicroSD 카드 슬롯에 MicroSD 카드를 삽입합니다.

Note

디버깅하기 전에 MicroSD 카드에 있는 모든 콘텐츠를 백업하십시오.

보드는 다음과 같은 모습이어야 합니다.



3. XSDK IDE에서 aws_demos를 마우스 오른쪽 버튼으로 클릭하고 Debug As를 선택한 후 1 Launch on System Hardware (System Debugger)를 선택합니다.
4. 디버거가 main()의 중단점에서 중지되면 메뉴에서 Run을 선택한 후 Resume을 선택합니다.

Note

애플리케이션을 처음 실행할 때 새 인증서-키 페어가 비휘발성 메모리로 가져오기 됩니다. 후속 실행의 경우 이미지와 BOOT.bin 파일을 다시 빌드하기 전에 main.c 파일에서 vDevModeKeyProvisioning()을 주석 처리할 수 있습니다. 그러면 실행할 때마다 인증서와 키를 스토리지에 복사하지 않아도 됩니다.

MicroSD 카드 또는 QSPI 플래시에서 MicroZed 보드를 부팅하여 Amazon FreeRTOS 데모 프로젝트를 실행하도록 선택할 수 있습니다. 자침은 [Amazon FreeRTOS 데모 프로젝트의 부팅 이미지 생성 \(p. 165\)](#) 및 [Amazon FreeRTOS 데모 프로젝트 실행 \(p. 165\)](#) 단원을 참조하십시오.

Amazon 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. Output BIF file path(출력 BIF 파일 경로) 옆의 찾아보기를 선택한 후 <amazon-freertos>/vendors/xilinx/microzed/aws_demos/aws_demos.bif에 있는 aws_demos.bif를 선택합니다.
4. [추가]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.

Amazon FreeRTOS 데모 프로젝트 실행

Amazon FreeRTOS 데모 프로젝트를 실행하려면 MicroSD 카드 또는 QSPI 플래시에서 MicroZed 보드를 부팅하면 됩니다.

Amazon FreeRTOS 데모 프로젝트를 실행하기 위해 MicroZed 보드를 설정할 때 [MicroZed 하드웨어 설정](#) (p. 162)의 다이어그램을 참조하십시오. MicroZed 보드를 컴퓨터에 연결했는지 확인합니다.

MicroSD 카드에서 Amazon FreeRTOS 프로젝트 부팅

Xilinx MicroZed Industrial IoT Kit와 함께 제공되는 MicroSD 카드를 포맷합니다.

1. `BOOT.bin` 파일을 MicroSD 카드에 복사합니다.
2. USB-UART 포트 바로 아래의 MicroSD 카드 슬롯에 카드를 삽입합니다.
3. MicroZed 부팅 모드 점퍼를 SD 부팅 모드로 설정합니다.

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 Amazon FreeRTOS 데모 프로젝트 부팅

1. MicroZed 보드의 부팅 모드 점퍼를 JTAG 부팅 모드로 설정합니다.



2. 컴퓨터가 USB-UART 및 JTAG Access 포트에 연결되어 있는지 확인합니다. 녹색 Power Good LED 조명이 켜져야 합니다.
3. XSDK IDE의 메뉴에서 Xilinx를 선택한 후 Program Flash를 선택합니다.
4. Program Flash Memory에서 하드웨어 플랫폼이 자동으로 채워져야 합니다. Connection에서 보드를 호스트 컴퓨터와 연결할 MicroZed 하드웨어 서버를 선택합니다.

Note

Xilinx Smart Lync JTAG 케이블을 사용하는 경우 XSDK IDE에 하드웨어 서버를 만들어야 합니다. New를 선택한 후 서버를 정의합니다.

5. Image File에 `BOOT.bin` 이미지 파일의 디렉터리 경로를 입력합니다. 파일을 찾아보려면 Browse를 선택합니다.
6. Offset에 `0x0`을 입력합니다.
7. FSBL File에 `fsbl1.elf` 파일의 디렉터리 경로를 입력합니다. 파일을 찾아보려면 Browse를 선택합니다.
8. Program을 선택하여 보드를 프로그래밍합니다.
9. QSPI 프로그래밍을 완료한 후 USB-UART 케이블을 제거하여 보드의 전원을 끕니다.
10. MicroZed 보드의 부팅 모드 점퍼를 QSPI 부팅 모드로 설정합니다.
11. USB-UART 포트 바로 아래에 있는 MicroSD 카드 슬롯에 카드를 삽입합니다.

Note

MicroSD 카드에 있는 모든 콘텐츠를 백업하십시오.

12. RST 버튼을 눌러 디바이스를 재설정하고 애플리케이션 부팅을 시작합니다. USB-UART 포트에서 USB-UART 케이블을 뽑은 후 케이블을 다시 삽입할 수도 있습니다.

문제 해결

잘못된 경로와 관련된 빌드 오류가 발생한 경우 [Amazon FreeRTOS 데모 프로젝트 빌드 \(p. 164\)](#)에 설명된 대로 프로젝트를 정리하고 다시 빌드해 보십시오.

Windows를 사용하는 경우 Windows XSDK IDE에서 문자열 대체 변수를 설정할 때 슬래시를 사용해야 합니다.

Amazon FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 [시작하기 문제 해결 \(p. 69\)](#) 단원을 참조하십시오.

Amazon FreeRTOS 라이브러리

Amazon FreeRTOS 라이브러리는 FreeRTOS 커널 및 내부 라이브러리에 추가 기능을 제공합니다. 내장 형 애플리케이션의 네트워킹 및 보안을 위해 Amazon FreeRTOS 라이브러리를 사용할 수 있습니다. 또한 Amazon FreeRTOS 라이브러리는 애플리케이션이 AWS IoT 서비스와 상호 작용할 수 있도록 지원합니다.

libraries 디렉터리에는 Amazon FreeRTOS 라이브러리의 소스 코드가 들어 있습니다. 라이브러리 기능의 구현을 지원하는 헬퍼 함수가 있습니다. 이 헬퍼 함수를 변경하지 않는 것이 좋습니다.

Amazon FreeRTOS 이식 라이브러리

다음 이식 라이브러리는 Amazon FreeRTOS 콘솔에서 다운로드할 수 있는 Amazon FreeRTOS 구성에 포함되어 있습니다. 이러한 라이브러리는 플랫폼에 따라 다릅니다. 라이브러리의 콘텐츠는 하드웨어 플랫폼에 따라 변경됩니다. 디바이스에 이러한 라이브러리를 이식하는 방법에 대한 자세한 내용은 [Amazon FreeRTOS 이식 안내서](#)를 참조하십시오.

Amazon FreeRTOS 이식 라이브러리

라이브러리
Amazon FreeRTOS
Bluetooth
Low Energy
WiFi
TCP/IP
TLS
Amazon FreeRTOS
Bluetooth
Low Energy
라이브러리를 사용하여 게이트웨이

Amazon FreeRTOS 라이브러리를 통해 AWS IoT MQTT 브로커와 통신할 수 있습니다. 자세한 내용은 [Amazon FreeRTOS Bluetooth Low Energy 라이브러리 \(p. 197\)](#) 단원을 참조하십시오.

설명
문서
제작
주
제작
리

Amazon
FreeRTOS

AWS

IoT

OTA

OTA)

에

이

전

트

라

이

브

러

리

는

Amazon
FreeRTOS

디

바

이

스

를

AWS

IoT

OTA

에

이

전

트

에

연

결

합

니

다.

자세한내용은

Amazon
FreeRTOS

OTA

에

이

전

트

Amazon FreeRTOS 이식 라이브러리 (p. 220)
단원을 참조하십시오.

Amazon
FreeRTOS
라이브러리

FreeRTOS

+POSIX

API

Amazon

FreeRTOS

라이브러리를

사용하

여

POSIX

호환

애플리케이션

을

Amazon

FreeRTOS

에

코스

템

에

이식

할

수

있

습

니

다.

자세한내용은

FreeRTOS

+POSIX

를

참조하

십

설명
주
리
시
오.
[Secure
Sockets
API](#)
제
용
은
Amazon
FreeRTOS
보
안
소
켓
라
이
브
러
리 (p. 224)
단
원
을
참
조
하
실
시
오.

Amazon
FreeRTOS
라이브러리

FreeRTOS

+TCP

API

Amazon FreeRTOS

를

위한 확장 가능한 오픈 소스 및 스레드 세이프

TCP/IP

스택입니다.

자세한 내용은

FreeRTOS

+TCP를

참조하십시오.

Amazon FreeRTOS 이식 라이브러리 (p. 228)
단원을 참조하십시오.

Amazon
FreeRTOS
라이브러리

PKCS
Amazon
FreeRTOS
PKCS
#11
라
이
브
러
리
는
프
로
비
저
닝
및
TLS
클
라이
언
트
인
종
을
지
원
하
기
위
한
퍼
블
리
키
암
호
화
표
준
#11
의
참
조
구
현
입
니
다.

Amazon FreeRTOS
퍼블릭 키 암호화 표준 (PKCS) #11 라이브러리 (p. 221)
자세한 내용은 [Amazon FreeRTOS](#) 문서를 참조하십시오.

Amazon
FreeRTOS
라이
브러

Amazon
FreeRTOS
전송계층보
안 단원을 참조하
시오. (p. 227)

Amazon
FreeRTOS
라이
브러

Amazon
FreeRTOS
공통
API
단원을 참조하
시오. (p. 232)

Amazon FreeRTOS 애플리케이션 라이브러리

필요할 경우 클라우드의 AWS IoT 서비스와 상호 작용하기 위해 Amazon FreeRTOS 구성에 다음 독립 실행형 애플리케이션 라이브러리를 포함시킬 수 있습니다.

Note

일부 애플리케이션 라이브러리는 Embedded C용 AWS IoT 디바이스 SDK의 라이브러리와 동일한 API를 사용합니다. 이러한 라이브러리의 경우 [AWS IoT 디바이스 SDK C API 참조](#)를 참조하십시오. Embedded C용 AWS IoT 디바이스 SDK에 대한 자세한 내용은 [Embedded C용 AWS IoT 디바이스 SDK \(p. 59\)](#) 단원을 참조하십시오.

Amazon FreeRTOS 애플리케이션 라이브러리

제작
정형
주
러
리

AWS
FreeRTOS
Device
Defender
DeviceDefender
Defender
SDK
API
본
권
리
는
Amazon
FreeRTOS
디
바
이
스
를
AWS
IoT
Device
Defender
에
연
결
합
니
다.

자
세
한
내
용
은

Amazon

라이
브
러
리

FreeRTOS
AWS
IoT
Device
Defender
라
이
브
러
리 (p. 207)

단원
을 참조
하십시오.

API
정적
주
러
리

AWS
Greengrass
FreeRTOS
AWS
Greengrass
IoT
Greengrass
라
이
브
러
리
는
Amazon
FreeRTOS
디
바
이
스
를
AWS
IoT
Greengrass
에
연
결
합
니
다.

자
세
한
내
용
이

Amazon
FreeRTOS
AWS
IoT
Greengrass
Discovery
라
이
브
러

리 (p. 212)

단
원
을
참
조
하

설명
주
리
리
실시
오.

Amazon FreeRTOS 애플리케이션 라이브러리를 사용하기 위해 사용하는 프로토콜입니다.

Amazon
FreeRTOS

MQTT

라이
브
러
리
버
전

1.0.0

에
대
한
자
세
한
내
용
은

Amazon
FreeRTOS

MQTT

라이
브
러
리,
버
전

1.0.0 (p. 216)

Amazon FreeRTOS MQTT 라이브러리 단원을 참조하십시오.

Amazon
FreeRTOS

MQTT

라이브러리 버전

2.0.0

에 대한 자세한 내용은

Amazon
FreeRTOS

MQTT

라이브러리, 버전

2.0.0 (p. 214)

단원을 참조하십시오.

Amazon FreeRTOS 애플리케이션 라이브러리
AWS IoT 디바이스 샘플 코드도 우와상호작용할 수 있도록 지원합니다.
자세한 내용

제작
정형
주
리
리
용은
Amazon
FreeRTOS
AWS
IoT
디
바
이
스
섀
도
우
라
이
브
러
리 (p. 210)
단원
을
참조
하
실
시
오.

Amazon FreeRTOS 공통 라이브러리

다음의 공통 라이브러리는 내장형 애플리케이션 개발을 위한 추가 데이터 구조 및 함수로 커널 기능을 확장합니다. 이러한 라이브러리는 종종 Amazon FreeRTOS 이식 및 애플리케이션 라이브러리의 종속 항목입니다.

Note

Embedded C용 AWS IoT 디바이스 SDK에는 API가 있는 공통 라이브러리 및 이러한 라이브러리와 동일한 기능이 포함되어 있습니다. API 참조의 경우 [AWS IoT 디바이스 SDK C API 참조](#)를 참조하십시오.

Amazon FreeRTOS 공통 라이브러리

제작
정형
주
리
리
작
자

제작
주
리
리
Logging
제
한
SDK
총
운
Amazon
FreeRTOS
로
깅
라이
브
러
리 (p. 193)
단
원
을
참
조
하
십
시
오.

한국전통문화재단은 전통문화재단으로서 전통문화의 보존과 활성화를 목표로 합니다.



Amazon FreeRTOS 라이브러리 구성

Amazon FreeRTOS 및 Embedded C용 AWS IoT 디바이스 SDK의 구성 설정은 C 프리프로세서 상수로 정의됩니다. 전역 구성 파일을 사용하거나 gcc의 -D와 같은 컴파일러 옵션을 사용하여 구성 설정을 지정할 수 있습니다. 구성 설정은 컴파일 시간 상수로 정의되어 있기 때문에 구성 설정이 변경되면 라이브러리를 다시 빌드해야 합니다.

전역 구성 파일을 사용하여 구성 옵션을 설정하려면 이름이 `iot_config.h`인 파일을 만들고 저장한 다음 이를 포함 경로에 추가합니다. 파일 내에서 `#define` 지시문을 사용하여 Amazon FreeRTOS 라이브러리, 데모 및 테스트를 구성합니다.

지워지는 전역 구성 옵션에 대한 자세한 내용은 [전역 구성 파일 참조](#)를 참조하십시오.

공통 라이브러리

Amazon FreeRTOS에는 내장형 애플리케이션 개발을 위한 추가 데이터 구조 및 함수로 커널 기능을 확장하는 몇 가지 공통 라이브러리가 포함되어 있습니다. 이러한 라이브러리는 종종 다른 Amazon FreeRTOS 라이브러리의 총속 라이브러리이기도 합니다.

주제

- Amazon FreeRTOS 원자성 작업 (p. 193)

- Amazon FreeRTOS 선형 컨테이너 라이브러리 (p. 193)
- Amazon FreeRTOS 로깅 라이브러리 (p. 193)
- Amazon FreeRTOS 정적 메모리 라이브러리 (p. 194)
- Amazon FreeRTOS 작업 풀 라이브러리 (p. 194)

Amazon FreeRTOS 원자성 작업 개요

원자성 작업은 동시 프로그래밍에서 비 차단 동기화를 보장합니다. 원자성 작업을 사용하여 공유 메모리에서 작동하는 비동기 작업으로 인해 발생하는 성능 문제를 해결할 수 있습니다. Amazon FreeRTOS는 `iot_atomic.h` 헤더 파일에 구현된 것처럼 원자성 작업을 지원합니다.

`iot_atomic.h` 헤더 파일에는 원자성 작업을 위한 두 가지 구현이 포함됩니다.

- 전역적 인터럽트 비활성화.
이 구현은 모든 Amazon FreeRTOS 플랫폼에서 사용할 수 있습니다.
- ISA 네이티브 원자성 지원.

이 구현은 GCC, 버전 4.7.0 이상으로 컴파일되고 ISA 원자성 지원이 있는 플랫폼에서만 사용할 수 있습니다. GCC 기본 함수에 대한 자세한 내용은 [Built-in Functions for Memory Model Aware Atomic Operations](#)를 참조하십시오.

초기화

Amazon FreeRTOS 원자성 작업을 사용하기 전에, 사용하려는 원자성 작업의 구현을 선택해야 합니다.

1. 편집할 `FreeRTOSConfig.h` (p. 7) 구성 파일을 업니다.
2. ISA 네이티브 원자성 지원 구현을 위해서는 `configUSE_ATOMIC_INSTRUCTION` 변수를 정의하고 1로 설정합니다.

전역 인터럽트 구현을 비활성화하려면 `configUSE_ATOMIC_INSTRUCTION`의 정의를 해제하거나 지웁니다.

API 참조

전체 API 참조는 [원자성 작업 C SDK API 참조](#)를 참조하십시오.

Amazon FreeRTOS 선형 컨테이너 라이브러리

Amazon FreeRTOS 선형 컨테이너 라이브러리는 내장형 애플리케이션을 개발할 때 사용할 수 있는 목록 및 대기열을 포함한 선형 데이터 구조를 정의합니다.

API 참조

전체 API 참조는 [선형 컨테이너 C SDK API 참조](#)를 참조하십시오.

Amazon FreeRTOS 로깅 라이브러리

Amazon FreeRTOS 로깅 라이브러리는 Amazon FreeRTOS 라이브러리가 디버깅을 위해 로그에 메시지를 인쇄할 수 있게 합니다.

API 참조

전체 API 참조는 [로깅 C SDK API 참조](#)를 참조하십시오.

Amazon FreeRTOS 정적 메모리 라이브러리

Amazon FreeRTOS 정적 메모리 라이브러리는 정적 버퍼를 관리하기 위한 몇 가지 기능을 정의합니다. 이 라이브러리를 사용하면 정적 메모리 구성 요소를 사용하여 동적 메모리 할당 대신 정적으로 할당된 버퍼를 제공할 수 있습니다.

API 참조

전체 API 참조는 [정적 메모리 C SDK API 참조](#)를 참조하십시오.

Amazon FreeRTOS 작업 풀 라이브러리

개요

Amazon FreeRTOS는 Amazon FreeRTOS 작업 풀 라이브러리를 사용한 작업 관리를 지원합니다. 작업 풀 라이브러리를 사용하여 백그라운드 작업을 예약하고 안전하고 비동기적인 작업 예약 및 취소를 허용할 수 있습니다. 작업 풀 API로 애플리케이션의 작업을 구성하여 성능과 메모리 사용량 간의 균형을 최적화할 수 있습니다.

작업 풀 라이브러리는 작업 풀 및 작업 풀 작업의 두 가지 기본 데이터 구조로 빌드됩니다.

작업 풀(IotTaskPool_t)

작업 풀은 실행을 위해 작업 대기열을 관리하고 작업을 실행하는 작업자 스레드를 관리하는 디스패치 대기열을 포함합니다.

작업 풀 작업(IotTaskPoolJob_t)

작업 풀 작업은 백그라운드 작업 또는 시간이 지정된 백그라운드 작업으로 실행될 수 있습니다. 백그라운드 작업은 FIFO(선입선출) 방식으로 시작되며 시간 제한이 없습니다. 시간이 지정된 작업은 타이머에 따라 백그라운드 실행이 예약됩니다.

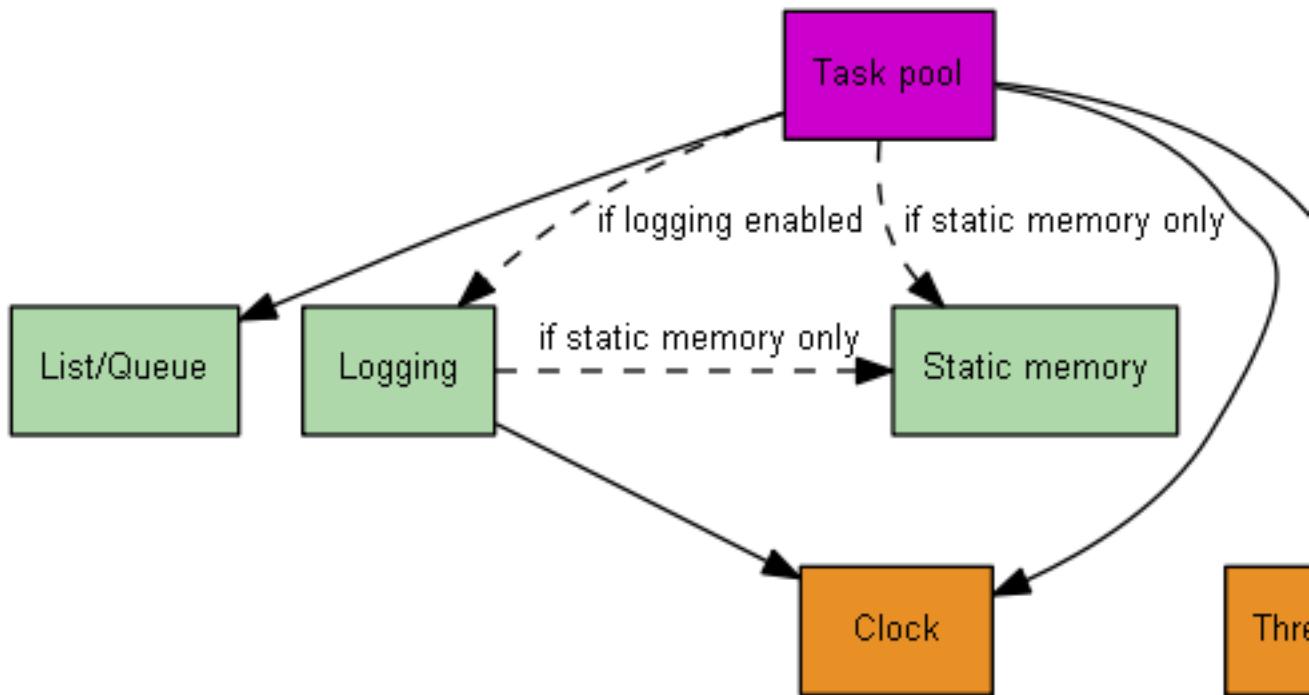
Note

작업 풀은 시간이 지정된 작업이 특정 시간대가 아니라 제한 시간이 경과한 후에만 실행되도록 보장 할 수 있습니다.

종속성 및 요구 사항

작업 풀 라이브러리에는 다음 종속성이 있습니다.

- 예약되거나 진행 중인 작업 풀 운영에 대한 데이터 구조를 유지 관리하는 선형 컨테이너(목록/대기열) 라이브러리
- 로깅 라이브러리(IOT_LOG_LEVEL_TASKPOOL 구성 설정이 IOT_LOG_NONE이 아닌 경우)
- 스레드 관리, 타이머, 클록 기능 등을 위해 운영 체제와의 인터페이스를 제공하는 플랫폼 계층입니다.



기능

작업 풀 라이브러리 API를 사용하여 다음을 수행할 수 있습니다.

- 라이브러리의 비 차단 API 함수로 즉시 작업과 지연 작업을 예약합니다.
- 정적 및 동적으로 할당된 작업을 만듭니다.
- 시스템 리소스를 기반으로 성능 및 공간을 확장하도록 라이브러리 설정을 구성합니다.
- 작업을 동적으로 생성할 때 캐싱을 사용자 지정하여 메모리 오버헤드를 낮춥니다.

문제 해결

작업 풀 라이브러리 함수는 오류 코드를 `IotTaskPoolError_t` 열거형 값으로 반환합니다. 각 오류 코드에 대한 자세한 내용은 [작업 풀 C SDK API 참조](#)의 `IotTaskPoolError_t` 열거형 데이터 형식에 대한 참조 설명서를 참조하십시오.

사용 제한

인터럽트 서비스 루틴(ISR)에서는 작업 풀 라이브러리를 사용할 수 없습니다.

차단 작업(특히 무기한 차단 작업)을 수행하는 작업 풀 사용자 콜백은 지양할 것을 강력히 권장합니다. 차단 작업이 오래 지속될 경우 작업 풀 스레드가 스탤드되고 교착 또는 결핍이 발생할 가능성이 있습니다.

초기화

애플리케이션은 작업 풀을 사용하기 전에 `IotTaskPool_CreateSystemTaskPool`을 호출하여 시스템 작업 풀의 인스턴스를 초기화해야 합니다. 애플리케이션은 부팅 시퀀스에서 시스템 수준 작업 풀이 라이브러리가 작업 풀을 사용하기 전에 충분히 일찍, 그리고 애플리케이션 코드가 작업 풀에 작업을 게시하기 전에 초기화되었는지 확인해야 합니다. 부팅 직후 시스템은 모든 라이브러리가 공유할 단일 시스템 수준 작업 풀을 초기화합니다. 초기화 후에, `IOT_SYSTEM_TASKPOOL` API와 함께 사용할 작업 풀 핸들을 검색할 수 있습니다.

Note

IotTaskPool_CreateSystemTaskPool을 호출하면 작업 풀 데이터 구조 및 상태를 보유할 메모리가 할당되지 않지만 작업 풀의 스레드와 같은 종속 엔터티 및 데이터 구조를 보유하기 위해 메모리를 할당할 수 있습니다.

API 참조

전체 API 참조는 [작업 풀 C SDK API 참조](#)를 참조하십시오.

사용 예

반복적인 AWS IoT Device Defender 지표 모음을 예약해야 하고, 타이머를 사용하여 MQTT 연결, 구독 및 게시 API 호출을 사용하여 모음을 예약하기로 결정했다고 가정합니다. 다음 코드는 MQTT 연결에서 연결을 끊는 연결 해제 콜백을 사용하여 MQTT에서 AWS IoT Device Defender 지표를 승인하기 위한 콜백 함수를 정의합니다.

```
/* An example of a user context to pass to a callback through a task pool thread. */
typedef struct JobUserContext
{
    uint32_t counter;
} JobUserContext_t;

/* An example of a user callback to invoke through a task pool thread. */
static void ExecutionCb( IotTaskPool_t * pTaskPool, IotTaskPoolJob_t * pJob, void * context )
{
    ( void )pTaskPool;
    ( void )pJob;
    JobUserContext_t * pUserContext = ( JobUserContext_t * )context;
    pUserContext->counter++;
}

void TaskPoolExample( )
{
    JobUserContext_t userContext = { 0 };
    IotTaskPoolJob_t job;
    IotTaskPool_t * pTaskPool;
    IotTaskPoolError_t errorSchedule;

    /* Configure the task pool to hold at least two threads and three at the maximum. */
    /* Provide proper stack size and priority per the application needs. */
    const IotTaskPoolInfo_t tpInfo = { .minThreads = 2, .maxThreads = 3, .stackSize =
512, .priority = 0 };

    /* Create a task pool. */
    IotTaskPool_Create( &tpInfo, &pTaskPool );

    /* Statically allocate one job, then schedule it. */
    IotTaskPool_CreateJob( &ExecutionCb, &userContext, &job );
    errorSchedule = IotTaskPool_Schedule( pTaskPool, &job, 0 );

    switch ( errorSchedule )
    {
        case IOT_TASKPOOL_SUCCESS:
            break;
        case IOT_TASKPOOL_BAD_PARAMETER:           // Invalid parameters, such as a NULL handle,
can trigger this error.
            case IOT_TASKPOOL_ILLEGAL_OPERATION:     // Scheduling a job that was previously
scheduled or destroyed could trigger this error.
            case IOT_TASKPOOL_NO_MEMORY:             // Scheduling a with flag
#IOT_TASKPOOL_JOB_HIGH_PRIORITY could trigger this error.
    }
}
```

```
        case IOT_TASKPOOL_SHUTDOWN_IN_PROGRESS:    // Scheduling a job after trying to destroy
            the task pool could trigger this error.
            // ASSERT
            break;
        default:
            // ASSERT*/
    }

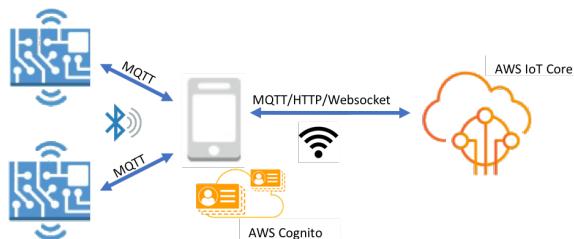
    /* .....
    /* ... Perform other operations ... */
    /* .....
```

IotTaskPool_Destroy(pTaskPool);
}

Amazon FreeRTOS Bluetooth Low Energy 라이브러리

개요

Amazon FreeRTOS는 프록시 디바이스(예: 휴대폰)를 통해 Bluetooth Low Energy를 이용한 MQTT 주제 게시 및 구독을 지원합니다. 마이크로 컨트롤러는 Amazon FreeRTOS Bluetooth Low Energy 라이브러리를 통해 AWS IoT MQTT 브로커와 안전하게 통신할 수 있습니다.



Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로 컨트롤러의 내장형 애플리케이션과 통신하는 네이티브 모바일 애플리케이션을 작성할 수 있습니다. 모바일 SDK에 대한 자세한 내용은 [Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK \(p. 206\)](#)을 참조하십시오.

Amazon FreeRTOS Bluetooth Low Energy 라이브러리에는 Wi-Fi 네트워크를 구성하고 대용량 데이터를 전송하며 Bluetooth Low Energy를 통해 네트워크 추상화를 제공하는 서비스가 포함됩니다. 또한 Amazon FreeRTOS Bluetooth Low Energy 라이브러리에는 Bluetooth Low Energy 스택을 통해 더 직접적으로 제어하기 위한 하위 수준 API와 미들웨어도 포함됩니다.

아키텍처

Amazon FreeRTOS Bluetooth Low Energy 라이브러리는 서비스, 미들웨어 및 하위 수준 래퍼 계층으로 구성되어 있습니다.

서비스

Amazon FreeRTOS Bluetooth Low Energy 서비스 계층은 미들웨어 API를 활용하는 네 가지 일반 속성 (GATT) 서비스인 디바이스 정보, Wi-Fi 프로비저닝, 네트워크 추상화 및 대용량 객체 전송으로 구성됩니다.

디바이스 정보

디바이스 정보 서비스는 마이크로컨트롤러에 대한 다음 정보를 수집합니다.

- 디바이스가 사용 중인 Amazon FreeRTOS의 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 애플리케이션 이 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

Amazon 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

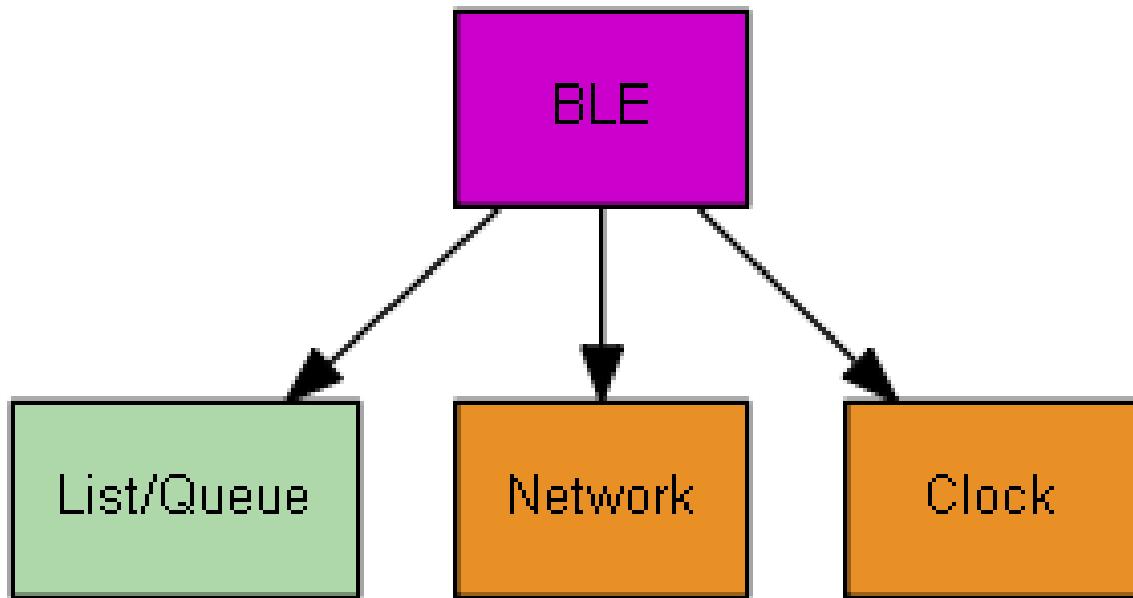
하위 수준 Amazon FreeRTOS Bluetooth Low Energy 래퍼는 제조업체의 Bluetooth Low Energy 스택의 추상화입니다. 하위 수준 래퍼는 하드웨어를 통한 직접 제어를 위해 공통 API 세트를 제공합니다. 하위 수준 API는 RAM 사용량을 최적화하지만 기능이 제한됩니다.

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.

- Amazon FreeRTOS 선행 컨테이너 라이브러리 (p. 193)
- 스레드 관리, 타이머, 클록 기능 및 네트워크 액세스를 위해 운영 체제와 인터페이스하는 플랫폼 계층입니다.



Wi-Fi 프로비저닝 서비스에만 Amazon FreeRTOS 라이브러리 종속성이 있습니다.

GATT 서비스	종속성
Wi-Fi 프로비저닝	Amazon FreeRTOS Wi-Fi 라이브러리 (p. 228)

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.

Amazon FreeRTOS Bluetooth Low Energy는 모바일 디바이스의 사용자 인증에 Amazon Cognito를 사용합니다. MQTT 프록시 서비스를 사용하려면 Amazon Cognito 자격 증명과 사용자 풀을 만들어야 합니다. 각 Amazon Cognito 자격 증명에는 적절한 정책이 연결되어 있어야 합니다. 자세한 내용은 [Amazon Cognito 개발자 안내서](#)를 참조하십시오.

라이브러리 구성 파일

Amazon FreeRTOS MQTT over Bluetooth Low Energy 서비스를 사용하는 애플리케이션은 구성 파라미터가 정의된 `iot_ble_config.h` 헤더 파일을 제공해야 합니다. 정의되지 않은 구성 파라미터는 `iot_ble_config_defaults.h`에 지정된 기본값을 사용합니다.

중요한 구성 파라미터는 다음과 같습니다.

`IOT_BLE_ADD_CUSTOM_SERVICES`

사용자가 고유한 서비스를 생성할 수 있습니다.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

사용자가 광고를 사용자 지정하고 응답 메시지를 검색할 수 있습니다.

자세한 내용은 [Bluetooth Low Energy API 참조](#)를 참조하십시오.

최적화

보드의 성능을 최적화 할 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 Amazon FreeRTOS Bluetooth Low Energy 라이브러리는 `eBTpropertySecureConnectionOnly` 속성을 TRUE로 설정하여 디바이스를 Secure Connections Only(보안 연결만 허용) 모드로 설정합니다. [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.

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

이 초기화 단계는 AWS 데모를 실행하는 경우에는 필요하지 않습니다. 데모 초기화는 `<amazon-freeertos>/demos/network_manager`에 있는 Network Manager에 의해 처리됩니다.

하위 수준 API

Amazon 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 * pxInterface = BTGetBluetoothInterface();
xStatus = pxInterface->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. 전체 Amazon FreeRTOS Bluetooth Low Energy 데모 애플리케이션은 [Bluetooth Low Energy 데모 애플리케이션](#)을 참조하십시오.

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.h`을 열고 `IOT_BLE_SET_CUSTOMADVERTISEMENT_MSG`를 1로 설정합니다. 이렇게 하면 `IotBle_SetCustomAdvCb` 콜백이 트리거됩니다.

새 서비스 추가

서비스의 전체 예는 `<amazon-freeertos>/.../ble/services` 단원을 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

Amazon FreeRTOS를 새 대상으로 이식하려면 Wi-Fi 프로비저닝 서비스와 Bluetooth Low Energy 기능을 위한 몇 가지 API를 구현해야 합니다.

Bluetooth Low Energy API

Amazon 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 Amazon FreeRTOS Bluetooth Low Energy 라이브러리를 이식하는 방법에 대한 자세한 내용은 Amazon FreeRTOS 이식 안내서의 [Bluetooth Low Energy 라이브러리 이식](#)을 참조하십시오.

Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK

Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로 컨트롤러와 상호 작용하는 모바일 애플리케이션을 만들 수 있습니다. Mobile SDK는 사용자 인증을 위해 Amazon Cognito를 사용하여 AWS 서비스와도 통신할 수 있습니다.

Amazon FreeRTOS Bluetooth 디바이스용 Android SDK

Amazon FreeRTOS Bluetooth 디바이스용 Android SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로 컨트롤러와 상호 작용하는 Android 모바일 애플리케이션을 빌드할 수 있습니다. SDK는 [GitHub](#)에서 사용 가능합니다.

Amazon FreeRTOS Bluetooth 디바이스용 Android SDK를 설치하려면

1. [GitHub](#)에서 SDK를 다운로드합니다.
2. Android Studio를 열고 `amazon-freertos-ble-android-sdk/amazonfreertosdk/` 디렉터리를 앱 프로젝트로 가져옵니다. [Android Studio 사용 설명서](#)에서 Android Studio 사용에 대한 자세한 정보를 알 수 있습니다.
3. 앱의 `gradle` 파일에서 다음 종속 항목을 추가합니다.

```
dependencies {
```

```
    implementation project(":amazonfreertosdk")  
}
```

4. 앱의 `AndroidManifest.xml` 파일에서 다음 권한을 추가합니다.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>  
    <!-- initiate device discovery and manipulate bluetooth settings -->  
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>  
    <!-- allow scan Bluetooth Low Energy -->  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
  
    <!-- AWS Mobile SDK -->  
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

SDK에 포함된 데모 모바일 애플리케이션 설정 및 실행에 대한 정보는 [사전 조건 \(p. 234\)](#) 및 [Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션 \(p. 236\)](#)을 참조하십시오.

Amazon FreeRTOS Bluetooth 디바이스용 iOS SDK

Amazon FreeRTOS Bluetooth 디바이스용 iOS SDK를 사용하여 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 'AmazonFreeRTOS', :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 [Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션 \(p. 236\)](#)을 참조하십시오.

Amazon FreeRTOS AWS IoT Device Defender 라이브러리

개요

AWS IoT Device Defender는 연결된 디바이스를 모니터링하여 비정상적인 동작을 감지하고 보안 위험을 완화할 수 있게 해주는 AWS IoT 서비스입니다. AWS IoT Device Defender를 사용하면 AWS IoT 디바이스 플랫폼에 대해 일관성 있는 IoT 구성을 적용하고, 디바이스가 손상된 경우 신속하게 대응하는 기능을 제공할 수 있습니다.

Amazon FreeRTOS는 Amazon FreeRTOS 기반 디바이스를 AWS IoT Device Defender와 함께 사용할 수 있도록 해주는 라이브러리를 제공합니다. Device Defender 라이브러리를 소프트웨어 구성에 추가하여 [Amazon FreeRTOS 콘솔](#)에서 Device Defender 라이브러리를 사용하여 Amazon FreeRTOS를 다운로드할 수 있습니다. 또한 모든 Amazon FreeRTOS 라이브러리를 포함하는 Amazon FreeRTOS GitHub 리포지토리를 복제할 수 있습니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.

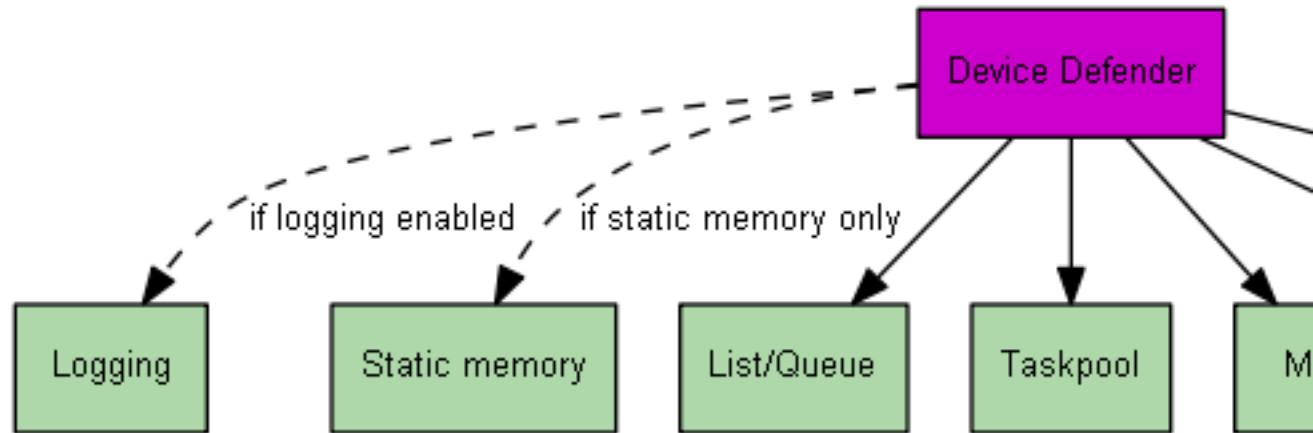
Note

Amazon FreeRTOS AWS IoT Device Defender 라이브러리는 연결 지표와 관련된 [디바이스 측 AWS IoT Device Defender 지표](#)의 하위 집합만 지원합니다. 자세한 내용은 [사용 제한 \(p. 209\)](#) 단원을 참조하십시오.

종속성 및 요구 사항

Device Defender 라이브러리에는 다음 종속 항목이 있습니다.

- [Amazon FreeRTOS 선형 컨테이너 라이브러리 \(p. 193\)](#).
- [Amazon FreeRTOS 로깅 라이브러리 \(p. 193\)](#)(구성 파라미터 `AWS_IOT_LOG_LEVEL_DEFENDER`가 `IOT_LOG_NONE`으로 설정되지 않은 경우).
- [Amazon FreeRTOS 정적 메모리 라이브러리 \(p. 194\)](#)(정적 메모리만 해당).
- 스레드 관리, 타이머, 클록 기능 등을 위해 운영 체제와의 인터페이스를 제공하는 플랫폼 계층입니다.
- [Amazon FreeRTOS 작업 풀 라이브러리 \(p. 194\)](#).
- [Amazon FreeRTOS MQTT 라이브러리, 버전 2.0.0 \(p. 214\)](#).



문제 해결

Amazon FreeRTOS Device Defender 오류 코드

Device Defender 라이브러리는 오류 코드를 양수 값으로 반환합니다. 각 오류 코드에 대한 자세한 내용은 [Device Defender C SDK API 참조](#)의 `AwsIotDefenderError_t`를 참조하십시오.

Amazon FreeRTOS Device Defender 이벤트

Device Defender 라이브러리에는 성공 또는 실패를 나타내는 "이벤트"라고 하는 양수의 열거형 값을 반환하는 `AwsIotDefenderCallback_t` 콜백 함수가 포함되어 있습니다. 이벤트 유형에 대한 자세한 내용은 [Device Defender C SDK API 참조](#)의 `AwsIotDefenderEventType_t`를 참조하십시오.

Amazon FreeRTOS Device Defender 디버깅

Device Defender 라이브러리에 대한 디버깅을 활성화하려면 [전역 구성 파일 \(p. 192\)](#)에서 Device Defender의 로그 수준을 디버깅 모드로 설정하십시오.

```
#define AWS_IOT_LOG_LEVEL_DEFENDER IOT_LOG_DEBUG
```

자세한 내용은 [전역 구성 파일 참조](#)를 참조하십시오.

개발자 지원

Device Defender 라이브러리에는 함수에 제공한 오류를 설명하는 문자열을 반환하는 `AwsIotDefender_strerror` 헬퍼 함수가 포함되어 있습니다.

```
const char * AwsIotDefender_strerror( AwsIotDefenderError_t error );
```

사용 제한

AWS IoT Device Defender 서비스는 데이터 직렬화를 위해 JSON 및 CBOR 형식을 모두 지원하지만 Amazon FreeRTOS Device Defender 라이브러리는 현재 구성 옵션 `AWS_IOT_DEFENDER_FORMAT`에 의해 제어되는 CBOR만 지원합니다.

또한 Amazon FreeRTOS AWS IoT Device Defender 라이브러리는 [디바이스 측 AWS IoT Device Defender 지표](#)의 하위 집합만 지원합니다.

긴 이름	짧은 이름	상위 요소	설명
<code>remote_addr</code>	<code>rad</code>	<code>connections</code>	TCP 연결의 원격 주소를 나열합니다.
<code>total</code>	<code>t</code>	<code>established_connections</code>	설정된 TCP 연결 수를 나열합니다.

예:

```
"tcp_connections": {
    "established_connections": {
        "connections": [
            {
                "remote_addr": "192.168.0.1:8000"
            },
            {
                "remote_addr": "192.168.0.2:8000"
            }
        ],
        "total": 2
    }
}
```

Amazon FreeRTOS Device Defender 라이브러리는 JSON 형식의 지표를 지원하지 않으므로 이 JSON 문서는 예제 전용입니다.

초기화

보안 소켓 지표를 사용하려면 AWS_IOT_SECURE_SOCKETS_METRICS_ENABLED 매크로를 정의해야 합니다. 이 매크로를 정의되지 않은 채로 두면 예기치 않은 동작이 발생할 수 있습니다.

Amazon FreeRTOS Device Defender API

전체 API 참조는 [Device Defender C SDK API 참조](#)를 참조하십시오.

사용 예

사용 중인 Device Defender 라이브러리의 전체 예제는 [AWS IoT Device Defender 데모 \(p. 250\)](#) 단원을 참조하십시오.

Amazon FreeRTOS AWS IoT 디바이스 샘플러 라이브러리

개요

Amazon FreeRTOS 디바이스 샘플러 API는 AWS IoT 디바이스 샘플러를 생성, 업데이트, 삭제하는 기능을 정의합니다. AWS IoT 디바이스 샘플러에 대한 자세한 내용은 [AWS IoT용 디바이스 샘플러 서비스](#)를 참조하십시오. MQTT 프로토콜을 사용하여 디바이스 샘플러 서비스에 액세스합니다. Amazon FreeRTOS 디바이스 샘플러 API는 MQTT API와 함께 작동하여 MQTT 프로토콜 작업에 대한 세부 정보를 처리합니다.

종속성 및 요구 사항

Amazon FreeRTOS에서 AWS IoT 디바이스 샘플러를 사용하려면 디바이스를 AWS IoT 사물로 등록해야 합니다. 사물에는 디바이스의 샘플러 액세스를 허용하는 정책을 따르는 인증서가 있어야 합니다. 자세한 내용은 [AWS IoT 시작하기](#)를 참조하십시오. Amazon FreeRTOS에 대한 예제 정책은 [AWS IoT 디바이스 샘플러 데모 애플리케이션 \(p. 263\)](#) 단원을 참조하십시오.

예제 클라이언트 자격 증명 헤더 파일은 `<amazon-freertos>/demos/include/aws_clientcredential.h`에 있습니다. 헤더 파일에서 다음 상수의 값을 설정했는지 확인합니다.

`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`

네트워크에서 사용되는 Wi-Fi 보안 유형입니다.

keyCLIENT_CERTIFICATE_PEM

IoT 사물과 연결된 인증서 PEM입니다.

keyCLIENT_PRIVATE_KEY_PEM

IoT 사물과 연결된 프라이빗 키 PEM입니다.

이 파일은 `aws_shadow_lightbulb_on_off.c`([디바이스 색도우 데모 애플리케이션 \(p. 263\)](#))에 포함되어 있습니다.

자체 애플리케이션을 개발 중인 경우 애플리케이션에 `aws_client_credentials.h` 헤더 파일을 포함한 다음 자격 증명을 `MQTTAgentConnectParams`로 `SHADOW_ClientConnect`에 전달하여 MQTT를 통해 AWS IoT에 연결해야 합니다. `MQTTAgentConnectParams`의 `pucClientId` 필드에 디바이스의 등록된 AWS IoT 사물 이름을 지정해야 합니다. 그렇지 않으면 디바이스 색도우 클라이언트가 연결되지 않습니다.

애플리케이션을 실행하기 전에 디바이스에 Amazon FreeRTOS MQTT 라이브러리가 설치되어 있는지 확인합니다. 자세한 내용은 [MQTT 라이브러리 \(p. 214\)](#)를 참조하십시오.

또한 MQTT 버퍼가 색도우 JSON 파일을 포함하기에 충분한 크기인지 확인합니다. 디바이스 색도우 문서의 최대 크기는 8KB입니다. 디바이스 색도우 API의 모든 기본 설정을 `aws_shadow_config_defaults.h` 파일에서 설정할 수 있습니다. `<amazon-freeertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_shadow_config.h` 파일에서 이러한 설정을 수정할 수 있습니다.

Important

디바이스 색도우 작업에 대해 정의한 JSON 형식이 `clientToken` 필드를 포함해야 합니다. `clientToken`은 고유한 값을 가질 수 있습니다. 예를 들어 `aws_shadow_lightbulb_on_off.c` 데모 애플리케이션은 `token-%d`를 사용합니다. 여기서 `%d`는 JSON 문서가 생성되었을 때 RTOS 틱 개수입니다.

JSON 형식이 `clientToken` 필드를 포함하지 않을 경우 `SHADOW_Delete()`, `SHADOW_Get()` 및 `SHADOW_Update()` 호출이 시간 초과됩니다.

API 참조

전체 API 참조는 [디바이스 색도우 C SDK API 참조](#)를 참조하십시오.

사용 예

1. `SHADOW_ClientCreate` API를 사용하여 색도우 클라이언트를 생성합니다. 대부분 애플리케이션의 경우 `xCreateParams.xMQTTClientType = eDedicatedMQTTClient` 필드만 채우면 됩니다.
2. `SHADOW_ClientConnect` API를 호출하고 `SHADOW_ClientCreate`에 의해 반환되는 클라이언트 핸들을 전달하여 MQTT 연결을 설정합니다.
3. `SHADOW_RegisterCallbacks` API를 호출하여 색도우 업데이트, 가져오기 및 삭제를 위한 콜백을 구성합니다.

연결이 설정된 후 다음 API를 사용하여 디바이스 색도우 작업을 수행할 수 있습니다.

`SHADOW_Delete`

디바이스 색도우를 삭제합니다.

`SHADOW_Get`

현재 디바이스 색도우를 가져옵니다.

`SHADOW_Update`

디바이스 색도우를 업데이트합니다.

Note

디바이스 셜도우 작업이 완료되면 `SHADOW_ClientDisconnect`를 호출하여 셜도우 클라이언트의 연결을 끊고 시스템 리소스를 확보합니다.

Amazon FreeRTOS AWS IoT Greengrass Discovery 라이브러리

개요

AWS IoT Greengrass Discovery 라이브러리는 마이크로 컨트롤러 디바이스가 네트워크에서 Greengrass 코어를 검색하는 데 사용됩니다. AWS IoT Greengrass Discovery API를 사용하면 디바이스가 코어의 엔드포인트를 찾은 후 Greengrass 코어에 메시지를 보낼 수 있습니다.

종속성 및 요구 사항

Greengrass Discovery 라이브러리를 사용하려면 AWS IoT에서 인증서, 정책 등과 같은 사물을 생성해야 합니다. 자세한 내용은 [AWS IoT 시작하기](#)를 참조하십시오.

`<amazon-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 `<amazon-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](#)를 참조하십시오.

MQTT 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.

Amazon FreeRTOS MQTT 라이브러리, 버전 2.0.0

개요

Amazon FreeRTOS MQTT 라이브러리를 사용하여 네트워크의 MQTT 클라이언트처럼 MQTT 주제를 게시 및 구독하는 애플리케이션을 만들 수 있습니다. Amazon FreeRTOS MQTT 라이브러리는 [AWS IoT MQTT 서버](#)와의 호환성을 위해 MQTT 3.1.1 표준을 구현합니다. 이 라이브러리는 다른 MQTT 서버와도 호환됩니다.

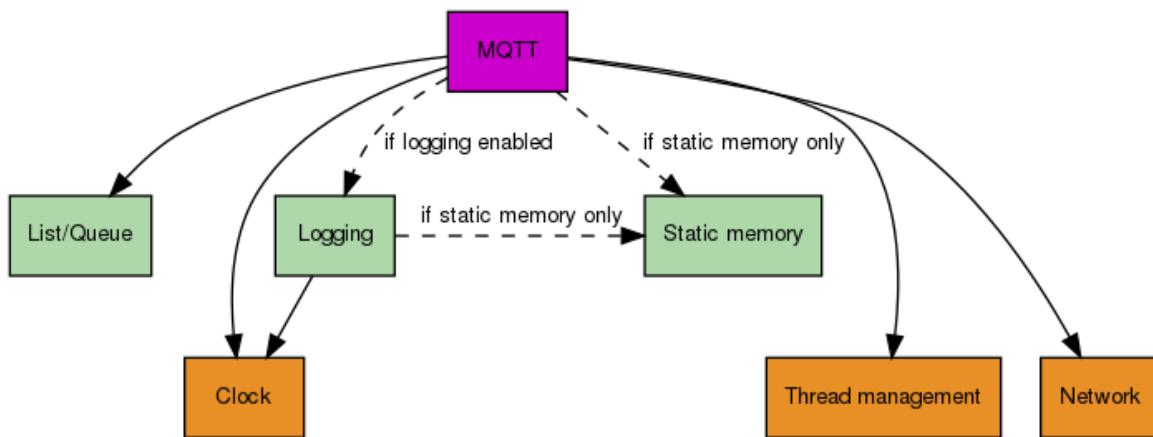
Amazon FreeRTOS MQTT 라이브러리의 소스 파일은 [`<amazon-freertos>/.../mqtt`](#)에 있습니다. 이러한 파일은 Amazon FreeRTOS MQTT 라이브러리 버전 2.0.0을 구현합니다. Amazon FreeRTOS에는 Amazon FreeRTOS MQTT 라이브러리 버전 1.0.0에 대한 이전 버전과의 호환성 계층도 포함되어 있습니다. Amazon FreeRTOS MQTT 버전 1.0.0에 대한 자세한 내용은 [Amazon FreeRTOS MQTT 라이브러리, 버전 1.0.0 \(p. 216\)](#) 단원을 참조하십시오.

종속성 및 요구 사항

Amazon FreeRTOS MQTT 라이브러리에는 다음 종속 항목이 있습니다.

- [Amazon FreeRTOS 선형 컨테이너 라이브러리 \(p. 193\)](#).
- [Amazon FreeRTOS 로깅 라이브러리 \(p. 193\)](#)(구성 파라미터 `AWS_IOT_MQTT_LOG_LEVEL`이 `AWS_IOT_LOG_NONE`으로 설정되지 않은 경우).
- [Amazon FreeRTOS 정적 메모리 라이브러리 \(p. 194\)](#)(정적 메모리만 해당).
- [Amazon FreeRTOS 작업 풀 라이브러리 \(p. 194\)](#).
- 스레드 관리, 클록 기능, 네트워킹 및 기타 플랫폼 수준 기능을 위해 운영 체제에 대한 인터페이스를 제공하는 플랫폼 계층
- C 표준 라이브러리 헤더

아래 다이어그램은 이러한 종속 항목을 보여 줍니다.



기능

The Amazon FreeRTOS MQTT 라이브러리에는 다음과 같은 기능이 있습니다.

- 기본적으로 이 라이브러리에는 완전 비동기식 MQTT API가 있습니다. `AwsIotMqtt_Wait` 함수를 사용하여 이 라이브러리를 동기식으로 사용할 수 있습니다.
- 이 라이브러리는 많은 처리량을 위해 병렬화와 스레드 인식이 가능합니다.
- 이 라이브러리는 확장 가능한 성능 및 공간이 특징입니다. 구성 설정을 사용하여 라이브러리를 시스템의 리소스에 맞게 조정할 수 있습니다.

API 참조

Amazon FreeRTOS 및 Embedded C용 AWS IoT 디바이스 SDK에는 동일한 MQTT v2.0.0 라이브러리 API가 있습니다. 전체 API 참조는 [MQTT\(v2.0.0\) C SDK API 참조](#)를 참조하십시오.

사용 예

Amazon FreeRTOS MQTT 라이브러리 사용 예제는 `iot_demo_mqtt.c`에 정의된 MQTT 데모 애플리케이션을 참조하십시오.

MQTT 데모에서는 MQTT의 구독-게시 워크플로를 보여 줍니다. 이 애플리케이션은 여러 주제 필터를 구독한 후 데이터 버스트를 다양한 주제 이름에 게시합니다. 각 메시지가 도착하면 데모는 확인 메시지를 다시 MQTT 서버에 게시합니다.

MQTT 데모를 사용 하려면 AWS IoT에서 인증서 및 정책과 같은 사물을 생성해야 합니다. 자세한 내용은 [AWS IoT 시작하기](#)를 참조하십시오.

전역 데모 구성 파라미터

<*amazon-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 <*amazon-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.

MQTT 데모 구성 파라미터

이러한 구성 파라미터는 MQTT 데모에 적용됩니다.

AWS_IOT_DEMO_MQTT_PUBLISH_BURST_SIZE

각 버스트에서 게시할 메시지 수입니다.

AWS_IOT_DEMO_MQTT_PUBLISH_BURST_COUNT

이 데모에서 게시 버스트의 수입니다.

Amazon FreeRTOS MQTT 라이브러리, 버전 1.0.0 개요

Amazon FreeRTOS에는 네트워크의 MQTT 클라이언트처럼 MQTT 주제를 게시 및 구독하는 애플리케이션을 만들 때 사용할 수 있는 오픈 소스 MQTT 클라이언트 라이브러리가 포함되어 있습니다.

MQTT 라이브러리 버전 2.0.0은 201960.00 이후 버전의 Amazon FreeRTOS에서 사용 가능합니다. 이 최신 라이브러리는 모든 전송 유형과 호환되므로 Bluetooth Low Energy 및 TCP/IP와 함께 사용할 수 있습니다. 자세한 내용은 [Amazon FreeRTOS MQTT 라이브러리, 버전 2.0.0 \(p. 214\)](#) 단원을 참조하십시오.

FreeRTOS MQTT 에이전트

Amazon FreeRTOS에는 MQTT 라이브러리를 관리하는 FreeRTOS MQTT 에이전트라고 하는 오픈 소스 데몬도 포함되어 있습니다. MQTT 에이전트는 MQTT 주제를 기본 MQTT 라이브러리와 연결, 게시 및 구독하기 위한 간단한 인터페이스를 제공합니다.

MQTT 에이전트는 별도 FreeRTOS 작업에서 실행되며 MQTT 프로토콜 사양에 설명된 대로 정기 연결 유지 메시지를 자동으로 전송합니다. 모든 MQTT API가 차단되고 시간 제한 파라미터를 사용합니다. 따라서 해당 작업이 완료되는 동안 API가 대기하는 시간이 가장 길니다. 제공된 시간 내에 작업이 완료되지 않을 경우 API는 시간 초과 오류 코드를 반환합니다.

종속성 및 요구 사항

Amazon FreeRTOS MQTT 라이브러리는 [Amazon FreeRTOS 보안 소켓 라이브러리 \(p. 224\)](#) 및 [Amazon FreeRTOS 버퍼 풀 라이브러리](#)를 사용합니다. MQTT 에이전트가 보안 MQTT 브로커에 연결하면 라이브러리도 [Amazon FreeRTOS 전송 계층 보안 \(p. 227\)](#)을 사용합니다.

기능

콜백

MQTT 에이전트와 브로커의 연결이 끊어지거나 브로커로부터 게시 메시지를 수신할 때마다 호출되는 선택적 콜백을 지정할 수 있습니다. 수신된 게시 메시지는 중앙 버퍼풀에서 가져온 버퍼에 저장됩니다. 이 메시지는 콜백에 전달됩니다. 이 콜백은 MQTT 작업의 컨텍스트에서 실행되므로 빨라야 합니다. 더 오래 처리해야 하는 경우 콜백에서 pdTRUE를 반환하여 버퍼에 대한 소유권을 획득해야 합니다. 그런 다음 `FreeRTOS_Agent_ReturnBuffer`를 호출하여 작업이 완료될 때마다 버퍼를 풀에 다시 반환해야 합니다.

구독 관리

구독 관리를 사용하여 구독 필터당 하나의 콜백을 등록할 수 있습니다. 구독하는 동안 이 콜백을 공급합니다. 주제에 대해 수신된 게시 메시지가 구독된 주제 필터와 일치할 때마다 이 콜백이 호출됩니다. 버퍼 소유권은 일반 콜백 사례에 설명된 것과 동일한 방법으로 작동합니다.

MQTT 작업 절전 모드 해제

사용자가 API를 호출하여 작업을 수행하거나 브로커로부터 게시 메시지가 수신될 때마다 MQTT 작업 절전 모드가 해제됩니다. 게시 메시지가 수신될 때 수행되는 이 비동기 절전 모드 해제는 연결된 소켓에 수신되는 데이터에 대해 호스트 MCU에 알리는 기능이 있는 플랫폼에서 지원됩니다. 이 기능이 없는 플랫폼에서는 연결된 소켓에 수신되는 데이터에 대해 MQTT 작업을 지속적으로 폴링해야 합니다. 게시 메시지가 수신되는 시간과 콜백을 호출하는 시간 사이의 지연을 최소화하기 위해 `mqttconfigMQTT_TASK_MAX_BLOCK_TICKS` 매크로는 MQTT 작업을 차단 상태로 유지할 수 있는 최대 시간을 제어합니다. 연결된 소켓에 수신되는 데이터에 대해 호스트 MCU에 알리는 기능이 부족한 플랫폼에서는 이 값이 짧아야 합니다.

주요 구성

MQTT 연결 요청 중에 다음과 같은 플래그를 지정할 수 있습니다.

- `mqttconfigKEEP_ALIVE_ACTUAL_INTERVAL_TICKS`: 전송되는 연결 유지 메시지의 빈도
- `mqttconfigENABLE_SUBSCRIPTION_MANAGEMENT`: 구독 관리 사용
- `mqttconfigMAX_BROKERS`: 최대 동시 MQTT 클라이언트 수
- `mqttconfigMQTT_TASK_STACK_DEPTH`: 작업 스택 깊이
- `mqttconfigMQTT_TASK_PRIORITY`: MQTT 작업의 우선 순위
- `mqttconfigRX_BUFFER_SIZE`: 데이터를 수신하는데 사용되는 버퍼의 길이
- `mqttagentURL_IS_IP_ADDRESS`: 제공된 URL이 IP 주소인 경우 `xFlags`에서 이 비트를 설정합니다.
- `mqttagentREQUIRE_TLS`: TLS를 사용하려면 `xFlags`에서 이 비트를 설정합니다.
- `mqttagentUSE_AWS_IOT_ALPN_443`: MQTT over TLS 포트 443에 대한 AWS IoT 지원을 사용하려면 `xFlags`에서 이 비트를 설정합니다.

ALPN에 대한 자세한 내용은 AWS IoT 개발자 안내서의 [AWS IoT 프로토콜 및 AWS 블로그의 사물 인터넷\(IoT\)에 관한 MQTT with TLS Client Authentication on Port 443: Why It Is Useful and How It Works\(포트 443에서의 TLS 클라이언트 인증을 통한 MQTT: 유용한 이유 및 작동 방식\)](#) 블로그 게시물을 참조하십시오.

최적화

지연 없이 수신한 패킷 처리

MQTT 에이전트를 구현하는 작업은 이벤트가 처리되기를 기다리는 Blocked 상태(따라서 CPU 주기를 사용하지 않음)에서 대부분의 시간을 보냅니다. MQTT 처리량은 네트워크에서 MQTT 패킷을 수신하자마자 에이전트 작업을 차단 해제하며 최대화됩니다. 완료되면 수신된 패킷이 가능한 한 빨리 처리됩니다. 완료되지 않은 경우 MQTT 에이전트가 다른 이유로 Blocked 상태에서 벗어날 때까지 수신된 패킷이 처리되지 않습니다.

MQTT 에이전트는 `IOptionName` 파라미터가 `SOCKETS_SO_WAKEUP_CALLBACK`으로 설정된 `SOCKETS_SetSockOpt()`를 호출하는 MQTT 에이전트가 설치한 콜백 실행으로 Blocked 상태에서 제거됩니다. 보안 소켓 설명서 링크가 여기에서 필요합니다. FreeRTOS+TCP TCP/IP 스택을 사용하는 경우, `FreeRTOSIPConfig.h`(TCP/IP 스택의 구성 파일)에서 `ipconfigSOCKET_HAS_USER_WAKE_CALLBACK`이 1로 설정되었다고 가정하고 콜백이 올바른 시간에 실행됩니다. FreeRTOS+TCP TCP/IP 스택을 사용하지 않는 경우, 보안 소켓은 이 기능이 사용 중인 스택의 보안 소켓 추상화 계층의 구현에 포함되도록 합니다.

TCP/IP 스택이 데이터가 수신되자마자 MQTT 에이전트를 차단 해제할 수 없는 경우, 수신되는 패킷과 처리되는 패킷 간의 최대 시간이 `mqttconfigMQTT_TASK_MAX_BLOCK_TICKS` 상수로 설정됩니다.

RAM 사용 최소화

다음 구성 상수는 MQTT 에이전트에 필요한 RAM 양에 직접적으로 영향을 줍니다.

- `mqttconfigMQTT_TASK_STACK_DEPTH`

- mqttconfigMQTT_TASK_STACK_DEPTH
- mqttconfigMAX_BROKERS
- mqttconfigMAX_PARALLEL_OPS
- mqttconfigRX_BUFFER_SIZE

이러한 상수를 가능한 최솟값으로 설정해야 합니다.

요구 사항 및 사용 제한

MQTT 에이전트 작업은 xTaskCreateStatic() API 함수를 사용하여 생성되므로 작업의 스택과 제어 블록은 컴파일 시간에 정적으로 할당됩니다. 따라서 MQTT 에이전트는 동적 메모리 할당을 허용하지 않는 애플리케이션에서 사용할 수 있지만, [FreeRTOSConfig.h \(p. 7\)](#)에서 1로 설정된 configSUPPORT_STATIC_ALLOCATION에 대한 종속성이 있음을 뜻합니다.

MQTT 에이전트는 FreeRTOS direct to task 알림 기능을 사용합니다. MQTT 에이전트 API 함수를 호출하면 호출 작업의 알림 값과 상태가 변경될 수 있습니다.

MQTT 패킷은 버퍼풀 모듈에서 제공하는 버퍼에 저장됩니다. 풀의 버퍼 수는 임의의 한 시점에 처리될 MQTT 트랜잭션 수의 최소 두 배로 하는 것이 좋습니다.

개발자 지원

mqttconfigASSERT

mqttconfigASSERT()는 FreeRTOS configASSERT() 매크로와 동일하며 정확히 동일한 방법으로 사용됩니다. MQTT 에이전트에서 문을 어설션하려는 경우 mqttconfigASSERT()를 정의합니다. MQTT 에이전트에서 문을 어설션하지 않으려면 mqttconfigASSERT()를 정의하지 않은 채로 둡니다. 아래와 같이 mqttconfigASSERT()를 정의하여 FreeRTOS configASSERT()를 호출하는 경우, FreeRTOS configASSERT()가 정의되었다면 MQTT 에이전트에 어설션 문만 포함됩니다.

```
#define mqttconfigASSERT( x ) configASSERT( x )
```

mqttconfigENABLE_DEBUG_LOGS

mqttconfigENABLE_DEBUG_LOGS를 1로 설정하여 vLoggingPrintf()로의 호출을 통해 디버그 로그를 인쇄합니다.

Initialization(초기화)

MQTT 통신을 시도하기 전에 아래와 같이 MQTT 에이전트와 종속 라이브러리를 모두 초기화해야 합니다. 네트워크 연결이 설정된 후 라이브러리를 초기화합니다.

```
BaseType_t SYSTEM_Init() { BaseType_t xResult = pdPASS; /* The bufferpool libraries provides the buffers use to store MQTT packets.*/
    xResult = BUFFERPOOL_Init();
    if( xResult == pdPASS ) { /* Create the MQTT agent task. */
        xResult = MQTT_AGENT_Init();
        if( xResult == pdPASS ) { /* Initialize the secure sockets abstraction layer. */
            xResult = SOCKETS_Init();
        }
    }
    return xResult;
}
```

API 참조

전체 API 참조는 [MQTT\(v1.0.0\) 라이브러리 API 참조](#) 및 [MQTT\(v1\) 에이전트 API 참조](#)를 참조하십시오.

이식

MQTT 에이전트가 호출하는 보안 소켓 추상화 계층은 특정 아키텍처로 이식해야 합니다. 자세한 내용은 Amazon FreeRTOS 이식 안내서의 [보안 소켓 라이브러리 이식](#)을 참조하십시오.

Amazon FreeRTOS HTTPS 클라이언트 라이브러리 개요

Amazon FreeRTOS HTTPS 클라이언트 라이브러리를 사용하면 HTTP 서버와 상호 작용하여 TLS를 통해 HTTP 요청을 전송하고 HTTP 응답을 수신하는 애플리케이션을 생성할 수 있습니다. Amazon FreeRTOS HTTPS 클라이언트 라이브러리는 TLS를 통해 HTTP/1.1 표준을 구현합니다.

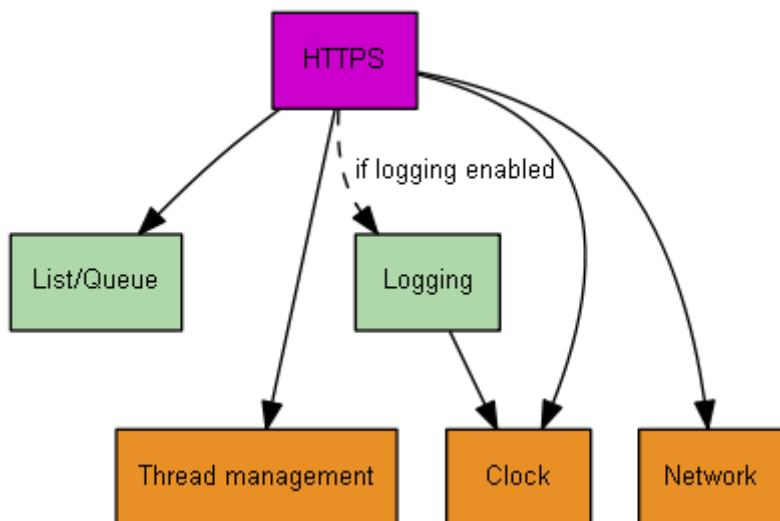
Amazon FreeRTOS HTTPS 클라이언트 라이브러리의 소스 파일은 <amazon-freertos>/libraries/csdk/standard/https에 있습니다.

종속성 및 요구 사항

Amazon FreeRTOS HTTPS 클라이언트 라이브러리에는 다음과 같은 종속성이 있습니다.

- Amazon FreeRTOS 선행 컨테이너 라이브러리 (p. 193).
- Amazon FreeRTOS 로깅 라이브러리 (p. 193)(구성 파라미터 AWS_IOT_HTTPS_LOG_LEVEL이 AWS_IOT_LOG_NONE으로 설정되지 않은 경우).
- Amazon FreeRTOS 정적 메모리 라이브러리 (p. 194)(정적 메모리만 해당).
- Amazon FreeRTOS 작업 풀 라이브러리 (p. 194).
- 스레드 관리, 클럭 함수, 네트워킹 및 기타 플랫폼 수준 기능을 위해 운영 체제에 대한 인터페이스를 제공하는 플랫폼 계층.
- C 표준 라이브러리 헤더.

아래 다이어그램은 이러한 종속 항목을 보여 줍니다.



기능

Amazon FreeRTOS HTTPS 클라이언트 라이브러리에는 다음 기능이 있습니다.

- 완전 비동기 및 동기(차단) API 함수.
- 내부 컨텍스트 및 HTTP 형식 헤더를 위한 애플리케이션 관리형 메모리.
- 스레드 인식 및 병렬화된 연결.

API 참조

전체 API 참조는 [HTTPS 클라이언트 API 참조](#)를 참조하십시오.

Amazon FreeRTOS OTA 에이전트 라이브러리 개요

무선(OTA) 에이전트를 사용하면 HTTP 또는 MQTT를 프로토콜로 사용하는 Amazon 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 룰백하기 위한 애플리케이션별로 직을 정의할 수 있습니다.

Amazon FreeRTOS를 통한 OTA 업데이트 설정에 대한 자세한 내용은 [Amazon FreeRTOS 무선\(OTA\) 업데이트 \(p. 8\)](#) 단원을 참조하십시오.

기능

전체 OTA 에이전트 인터페이스는 다음과 같습니다.

OTA_AgentInit

OTA 에이전트를 초기화합니다. 호출자가 메시징 프로토콜 컨텍스트, 선택적 콜백 및 제한 시간을 제공합니다.

OTA_AgentShutdown

OTA 에이전트를 사용한 이후에 리소스를 정리합니다.

OTA_GetAgentState

OTA 에이전트의 현재 상태를 가져옵니다.

OTA_ActivateNewImage

OTA를 통해 수신된 최신 마이크로 컨트롤러 펌웨어 이미지를 활성화합니다. 이제 상세 작업 상태가 자체 테스트됩니다.

OTA_SetImageState

현재 실행 중인 마이크로 컨트롤러 펌웨어 이미지의 확인 상태(테스트 중, 수락됨, 거부됨)를 설정합니다.

OTA_GetImageState

현재 실행 중인 마이크로 컨트롤러 펌웨어 이미지의 상태(테스트 중, 수락됨, 거부됨)를 가져옵니다.

OTA_CheckForUpdate

OTA 업데이트 서비스에서 사용 가능한 다음 OTA 업데이트를 요청합니다.

API 참조

자세한 내용은 [OTA 에이전트 API 참조](#)를 참조하십시오.

사용 예

MQTT 프로토콜을 사용하는 일반적인 OTA 지원 디바이스 애플리케이션은 다음과 같은 API 호출 시퀀스를 사용하여 OTA 에이전트를 구동합니다.

1. AWS IoT MQTT 브로커에 연결합니다. 자세한 내용은 [Amazon FreeRTOS MQTT 라이브러리, 버전 1.0.0 \(p. 216\)](#) 단원을 참조하십시오.
 2. OTA_AgentInit을 호출하여 OTA 에이전트를 초기화합니다. 애플리케이션에서 사용자 지정 OTA 콜백 함수를 정의하거나 NULL 콜백 함수 포인터를 지정하여 기본 콜백을 사용할 수 있습니다. 또한 초기화 시간 제한을 입력해야 합니다.
- 콜백은 OTA 업데이트 작업을 완료한 이후에 실행되는 애플리케이션별로 직렬화됩니다. 시간 제한은 초기화를 완료하는 동안 대기하는 시간을 정의합니다.
3. 에이전트가 준비되기 이전에 OTA_AgentInit이 시간 초과된 경우 OTA_GetAgentState를 호출하여 에이전트가 초기화되고 예상한 대로 작동하는지 확인할 수 있습니다.
 4. OTA 업데이트가 완료되면 Amazon FreeRTOS는 accepted, rejected, self test 이벤트 중 하나를 사용하여 작업 완료 콜백을 호출합니다.
 5. 유효성 검사 오류 등으로 인해 새 펌웨어 이미지가 거부된 경우 애플리케이션에서는 일반적으로 알림을 무시하고 다음 업데이트를 대기할 수 있습니다.
 6. 업데이트가 유효하고 수락됨으로 표시된 경우 OTA_ActivateNewImage를 호출하여 디바이스를 재설정하고 새 펌웨어 이미지를 부팅합니다.

이식

OTA 기능을 플랫폼에 이식하는 방법에 대한 자세한 내용은 Amazon FreeRTOS 이식 안내서의 [OTA 라이브러리 이식](#) 단원을 참조하십시오.

Amazon FreeRTOS 퍼블릭 키 암호화 표준(PKCS) #11 라이브러리

개요

퍼블릭 키 암호화 표준 #11(PKCS#11)은 키 스토리지, 암호화 객체에 대한 get/set 속성 및 세션 의미 체계를 추출하는 암호화 API입니다. Amazon FreeRTOS 소스 코드 리포지토리에서 OASIS 표준 기구에서 가져온 pkcs11.h를 참조하십시오. Amazon FreeRTOS 참조 구현에서는 TLS 헬퍼 인터페이스에서 SOCKETS_Connect 중에 TLS 클라이언트 인증을 위해 PKCS#11 API를 호출합니다. 1회 개발자 프로비저닝 워크플로우에서 인증을 위한 프라이빗 키와 TLS 클라이언트 인증서를 AWS IoT MQTT 브로커로 가져오기 위해서도 PKCS#11 API를 호출합니다. 이 두 사용 사례(프로비저닝 및 TLS 클라이언트 인증)에서는 PKCS#11 인터페이스 표준의 작은 하위 집합만 구현해야 합니다.

기능

PKCS#11의 다음 하위 집합이 사용됩니다. 이 목록은 프로비저닝, TLS 클라이언트 인증 및 정리를 지원하기 위해 루틴을 호출하는 순서와 거의 동일합니다. 함수에 대한 자세한 내용은 표준 위원회에서 제공하는 PKCS#11 문서를 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

Amazon FreeRTOS PKCS#11 참조 구현은 NIST P-256 곡선을 가진 ECDSA 및 2048비트 RSA(서명 전용)를 지원합니다. 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 단계는 aws configure를 사용하여 AWS CLI를 구성했다는 가정 하에 작성되었습니다.

P-256 클라이언트 인증서를 기반으로 AWS IoT 사물 생성

1. aws iot create-thing --thing-name dcgecc를 실행하여 AWS IoT 사물을 생성합니다.
2. openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt ec_param_enc:named_curve -outform PEM -out dcgecc.key를 실행하여 OpenSSL로 P-256 키를 생성합니다.
3. openssl req -new -nodes -days 365 -key dcgecc.key -out dcgecc.req를 실행하여 2 단계에서 생성한 키로 서명된 인증서 등록 요청을 생성합니다.
4. aws iot create-certificate-from-csr --certificate-signing-request file://dcgecc.req --set-as-active --certificate-pem-outfile dcgecc.crt를 실행하여 인증서 등록 요청을 AWS IoT에 제출합니다.
5. aws iot attach-thing-principal --thing-name dcgecc --principal "arn:aws:iot:us-east-1:123456789012:cert/86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de" 실행하여 이전 명령에서 출력된 ARN에서 참조되는 인증서를 사물에 연결합니다.
6. aws iot create-policy --policy-name FullControl --policy-document file://policy.json을 실행하여 정책을 생성합니다. 이 정책은 지나치게 허용적이므로, 개발 목적으로만 사용해야 합니다.

다음은 create-policy 명령에 지정되는 policy.json 파일 목록입니다. Greengrass 연결 및 검색을 위해 Amazon FreeRTOS 데모를 실행하지 않으려면 greengrass:* 작업을 생략할 수 있습니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iot:*",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "greengrass:*",  
            "Resource": "*"  
        }  
    ]  
}
```

7. aws iot attach-principal-policy --policy-name FullControl --principal "arn:aws:iot:us-east-1:785484208847:cert/86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de" 실행하여 보안 주체(인증서) 및 정책을 사물에 연결합니다.

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.h에 복사합니다.

이식

PKCS #11 라이브러리를 플랫폼에 이식하는 방법에 대한 자세한 내용은 Amazon FreeRTOS 이식 안내서의 [PKCS #11 라이브러리 이식](#) 단원을 참조하십시오.

Amazon FreeRTOS 보안 소켓 라이브러리

개요

Amazon FreeRTOS 보안 소켓 라이브러리를 사용하여 안전하게 통신하는 내장형 애플리케이션을 만들 수 있습니다. 이 라이브러리는 다양한 네트워크 프로그래밍 배경의 소프트웨어 개발자가 쉽게 온보딩할 수 있도록 설계되었습니다.

Amazon FreeRTOS 보안 소켓 라이브러리는 Berkeley 소켓 인터페이스를 기반으로 하며, TLS 프로토콜을 사용한 추가 보안 통신 옵션도 있습니다. Amazon FreeRTOS 보안 소켓 라이브러리와 Berkeley 소켓 인터페이스 간의 차이점에 대한 자세한 내용은 [보안 소켓 API 참조](#)의 `SOCKETS_SetSockOpt`를 참조하십시오.

Note

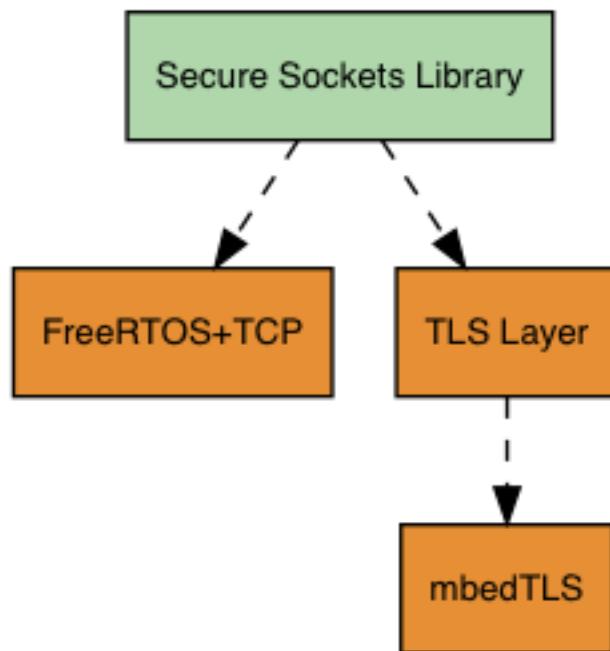
현재 Amazon FreeRTOS 보안 소켓에는 클라이언트 API만 지원됩니다.

종속성 및 요구 사항

Amazon FreeRTOS 보안 소켓 라이브러리는 TCP/IP 스택 및 TLS 구현에 종속됩니다. Amazon FreeRTOS의 포트는 다음 세 가지 방법 중 하나로 이러한 종속성을 충족합니다.

- TCP/IP 및 TLS의 사용자 지정 구현
- TCP/IP의 사용자 지정 구현과 [mbedtls](#)가 있는 Amazon FreeRTOS TLS 계층
- [FreeRTOS+TCP](#)와 [mbedtls](#)가 있는 Amazon FreeRTOS TLS 계층

아래 종속성 다이어그램은 Amazon 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. Amazon FreeRTOS TLS 계층에 대한 자세한 내용은 [Amazon FreeRTOS 전송 계층 보안 \(p. 227\)](#) 단원을 참조하십시오.



기능

Amazon 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

Amazon 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

Amazon FreeRTOS 보안 소켓 API가 오류 코드를 반환하는 경우, Amazon FreeRTOS 보안 소켓 라이브러리에 종속된 [Amazon FreeRTOS MQTT 라이브러리, 버전 1.0.0 \(p. 216\)](#)는 AWS_IOT_MQTT_SEND_ERROR 오류 코드를 반환합니다.

개발자 지원

Amazon 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 Amazon FreeRTOS 보안 소켓 라이브러리에서 지원됩니다. UDP 소켓은 지원되지 않습니다.

클라이언트 API만 Amazon FreeRTOS 보안 소켓 라이브러리에서 지원됩니다. Bind, Accept, Listen 등의 서버 API는 지원되지 않습니다.

Initialization(초기화)

Amazon 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"
"MIIBtjCCAvgAwIBAgIBmyf1XSXNmY/Owu2eiedgPySjAKBggqhkJOPQODAjA5\n"
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"
"Um9vdCBDSQAzMB4XDTE1MDUyNjAwMDAwMFQxDTQwMDUyNjAwMDAwMFowOTELMAkG\n"
"A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"
"Q0EgMzBZMBMByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG81Kl\n"
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcV0ho6AF2hiRVd9RFgdszf1ZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkJOPQODAgNJADBAiEA4IWSoxe3jfkr\n"
"BqWTrBqYAgFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xIJyz1vnrxir4tiz+OpAUFeM\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_SNDTIMEO, &xSendTimeOut,  
sizeof( xSendTimeOut ) );  
  
/* Set the socket to use TLS. */  
SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_REQUIRE_TLS, NULL, ( size_t ) 0 );  
SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE,  
cTlsECHO_SERVER_CERTIFICATE_PEM, ulTlsECHO_SERVER_CERTIFICATE_LENGTH );  
  
if( SOCKETS_Connect( xSocket, &xEchoServerAddress, sizeof( xEchoServerAddress ) ) ==  
0 )  
{  
    /* Send the string to the socket. */  
    xTransmitted = SOCKETS_Send( xSocket, /* The socket  
receiving. */  
                                ( void * )"some message", /* The data being  
sent. */  
                                12, /* The length of the  
data being sent. */  
                                0 ); /* No flags. */  
  
    if( xTransmitted < 0 )  
    {  
        /* Error while sending data */  
        return;  
    }  
  
    SOCKETS_Shutdown( xSocket, SOCKETS_SHUT_RDWR );  
}  
else  
{  
    //failed to connect to server  
}  
  
SOCKETS_Close( xSocket );  
}
```

전체 예제는 보안 소켓 에코 클라이언트 데모 (p. 264) 단원을 참조하십시오.

이식

Amazon FreeRTOS 보안 소켓은 TCP/IP 스택 및 TLS 구현에 종속됩니다. 스택에 따라 보안 소켓 라이브러리를 이식하려면 다음 중 일부를 이식해야 할 수 있습니다.

- [FreeRTOS+TCP TCP/IP 스택](#)
- [Amazon FreeRTOS 퍼블릭 키 암호화 표준\(PKCS\) #11 라이브러리 \(p. 221\)](#)
- [Amazon FreeRTOS 전송 계층 보안 \(p. 227\)](#)

이식에 대한 자세한 내용은 Amazon FreeRTOS 이식 안내서의 [보안 소켓 라이브러리 이식](#) 단원을 참조하십시오.

Amazon FreeRTOS 전송 계층 보안

Amazon FreeRTOS 전송 계층 보안(TLS) 인터페이스는 프로토콜 스택의 상위에 있는 Secure Sockets Layer(SSL) 인터페이스로부터 암호화 구현 세부 정보를 추출하는 데 사용되는 선택적 씬 래퍼입니다. TLS 인터페이스의 목적은 현재 소프트웨어 암호화 라이브러리인 mbed TLS를 TLS 프로토콜 협상 및 암호화 프리미티브를 위한 대체 구현과 쉽게 교체할 수 있도록 만드는 것입니다. SSL 인터페이스를 변경할 필요 없이 TLS 인터페이스를 교체할 수 있습니다. Amazon FreeRTOS 소스 코드 리포지토리의 `iot_tls.h`를 참조하십시오.

SSL에서 암호화 라이브러리로 직접 인터페이스할 수 있으므로 TLS 인터페이스는 선택 사항입니다. 이 인터페이스는 네트워크 전송 및 TLS의 전체 스택 오프로드 구현을 포함하는 MCU 솔루션에는 사용되지 않습니다.

TLS 인터페이스 포팅에 대한 자세한 내용은 Amazon FreeRTOS 포팅 안내서의 [TLS 라이브러리 포팅](#) 단원을 참조하십시오.

Amazon FreeRTOS Wi-Fi 라이브러리

개요

Amazon FreeRTOS Wi-Fi 라이브러리는 포트별 Wi-Fi 구현을 공통 API로 추상화하여 Wi-Fi 기능을 갖춘 모든 Amazon 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

Amazon FreeRTOS Wi-Fi 라이브러리에는 [FreeRTOS+TCP](#) 코어가 필요합니다.

기능

Wi-Fi 라이브러리에는 다음 기능이 있습니다.

- WEP, WPA, WPA2 인증 지원
- 액세스 포인트 검색
- 전력 관리
- 네트워크 프로파일링

Wi-Fi 라이브러리의 기능에 대한 자세한 내용은 아래를 참조하십시오.

Wi-Fi 모드

Wi-Fi 디바이스는 스테이션, 액세스 포인트 또는 P2P 등의 세 가지 모드 중 하나일 수 있습니다. `WIFI_GetMode`를 호출하여 Wi-Fi 디바이스의 현재 모드를 가져올 수 있습니다. `WIFI_SetMode`를 호출하여 디바이스의 Wi-Fi 모드를 설정할 수 있습니다. `WIFI_SetMode`를 호출하여 모드를 전환하면 네트워크에 이미 연결된 디바이스의 연결이 해제됩니다.

스테이션 모드

보드를 기준 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.

P2P 모드

액세스 포인트 없이 여러 디바이스가 서로 직접 연결할 수 있게 하려면 디바이스를 P2P 모드로 설정합니다.

보안

Wi-Fi API는 WEP, WPA, WPA2 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`

검색 및 연결

인근의 액세스 포인트를 검색하려면 디바이스를 스테이션 모드로 설정하고 `WIFI_Scan` 함수를 호출합니다. 스캔에서 원하는 네트워크를 찾은 경우 `WIFI_ConnectAP`를 호출하고 네트워크 자격 증명을 제공하여 네트워크에 연결할 수 있습니다. `WIFI_Disconnect`를 호출하여 네트워크에서 Wi-Fi 디바이스를 연결 해제할 수 있습니다. 검색 및 연결에 대한 자세한 내용은 [사용 예 \(p. 230\)](#) 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_NetworkAdd`는 네트워크 프로필을 추가하고, `WIFI_NetworkGet`은 네트워크 프로필을 검색하고, `WIFI_NetworkDel`은 네트워크 프로필을 삭제합니다. 저장할 수 있는 프로필 수는 플랫폼에 따라 다릅니다.

구성

Wi-Fi 라이브러리를 사용하려면 구성 파일에서 여러 식별자를 정의해야 합니다. 이러한 식별자에 대한 자세한 내용은 [API 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 포함 시켜야 합니다.

Initialization(초기화)

Wi-Fi 라이브러리를 사용하기 전에 FreeRTOS 구성 요소 외에도 일부 보드 관련 구성 요소를 초기화해야 합니다. `vendors/<vendor>/boards/<board>/aws_demos/application_code/main.c` 파일을 초기화를 위한 템플릿으로 사용하여 다음을 수행합니다.

1. 애플리케이션이 Wi-Fi 연결을 처리하는 경우 `main.c`에서 샘플 Wi-Fi 연결 로직을 제거합니다. 다음 `DEMO_RUNNER_RunDemos()` 함수 호출을

```
if( SYSTEM_Init() == pdPASS )
{
    ...
    DEMO_RUNNER_RunDemos();
    ...
}
```

자체 애플리케이션 호출로 바꿉니다.

```
if( SYSTEM_Init() == pdPASS )
{
    ...
    // This function should create any tasks
    // that your application requires to run.
    YOUR_APP_FUNCTION();
    ...
}
```

2. WIFI_On()을 호출하여 Wi-Fi 칩을 초기화하고 전원을 켭니다.

Note

일부 보드에는 추가 하드웨어 초기화가 필요할 수 있습니다.

3. 구성된 WFINetworkParams_t 구조를 WIFI_ConnectAP()로 전달하여 보드를 사용 가능한 Wi-Fi 네트워크에 연결합니다. WFINetworkParams_t 구조에 대한 자세한 내용은 [사용 예 \(p. 230\)](#) 및 [API 참조 \(p. 230\)](#)를 참조하십시오.

API 참조

전체 API 참조는 [Wi-Fi API 참조](#)를 참조하십시오.

사용 예

알려진 AP에 연결

```
#define clientcredentialWIFI_SSID      "MyNetwork"
#define clientcredentialWIFI_PASSWORD    "hunter2"

INetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

xWifiStatus = WIFI_On(); // Turn on Wi-Fi module

// Check that Wi-Fi initialization was successful
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi library initialized.\n" ) );
}
else
{
    configPRINT( ( "WiFi library failed to initialize.\n" ) );
    // Handle module init failure
}

/* Setup parameters. */
xNetworkParams.pcSSID = clientcredentialWIFI_SSID;
xNetworkParams.ucSSIDLength = sizeof( clientcredentialWIFI_SSID );
xNetworkParams.pcPassword = clientcredentialWIFI_PASSWORD;
```

```
xNetworkParams.ucPasswordLength = sizeof( clientcredentialWIFI_PASSWORD );
xNetworkParams.xSecurity = eWiFiSecurityWPA2;

// Connect!
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );

if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi Connected to AP.\n" ) );
    // IP Stack will receive a network-up event on success
}
else
{
    configPRINT( ( "WiFi failed to connect to AP.\n" ) );
    // Handle connection failure
}
```

인근 AP 검색

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

configPRINT( ("Turning on wifi...\n" ) );
xWifiStatus = WIFI_On();

configPRINT( ("Checking status...\n" ) );
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ("WiFi module initialized.\n" ) );
}
else
{
    configPRINTF( ("WiFi module failed to initialize.\n" ) );
    // Handle module init failure
}

WIFI_SetMode(eWiFiModeStation);

/* Some boards might require additional initialization steps to use the Wi-Fi library. */

while (1)
{
    configPRINT( ("Starting scan\n" ) );
    const uint8_t ucNumNetworks = 12; //Get 12 scan results
    WiFiScanResult_t xScanResults[ ucNumNetworks ];
    xWifiStatus = WIFI_Scan( xScanResults, ucNumNetworks ); // Initiate scan

    configPRINT( ("Scan started\n" ) );

    // For each scan result, print out the SSID and RSSI
    if ( xWifiStatus == eWiFiSuccess )
    {
        configPRINT( ("Scan success\n" ) );
        for ( uint8_t i=0; i<ucNumNetworks; i++ )
        {
            configPRINTF( ("%s : %d \n", xScanResults[i].cSSID, xScanResults[i].cRSSI ) );
        }
    } else {
        configPRINTF( ("Scan failed, status code: %d\n", (int)xWifiStatus ) );
    }

    vTaskDelay(200);
}
```

이식

`aws_wifi.c` 구현은 `aws_wifi.h`에 정의된 함수를 구현해야 합니다. 적어도 이 구현은 필수적이지 않거나 지원되지 않는 함수에 대해 `eWiFiNotSupported`을 반환해야 합니다.

Wi-Fi 라이브러리 이식에 대한 자세한 내용은 Amazon FreeRTOS 이식 안내서의 [Wi-Fi 라이브러리 이식](#) 단원을 참조하십시오.

Amazon FreeRTOS 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.

Amazon FreeRTOS 공통 I/O는 이 하드웨어 추상화 계층 역할을 수행합니다. 지원되는 참조 보드에서 공통 직렬 디바이스에 액세스하기 위한 표준 API 세트를 제공합니다. 이러한 공통 API를 사용하면 이러한 주변 장치와 통신하고 상호 작용하며 코드가 플랫폼 간에 작동하도록 활성화됩니다. 공통 I/O가 없는 경우 저수준 디바이스로 작업하도록 코드를 작성하는 방법은 실리콘 공급업체에 따라 다릅니다.

지원되는 주변 장치

- UART
- SPI
- I2C

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 [Amazon FreeRTOS 포팅 안내서](#)를 참조하십시오.

Amazon FreeRTOS 데모

Amazon FreeRTOS는 주 Amazon FreeRTOS 디렉터리의 `demos` 폴더에 몇 개의 데모 애플리케이션을 포함하고 있습니다. Amazon FreeRTOS에서 실행할 수 있는 모든 예제는 `demos`의 `common` 폴더에 있습니다. 또한 `demos` 폴더에는 Amazon FreeRTOS 적격 플랫폼별로 폴더가 있습니다. `demos 콘솔`을 사용할 경우, 선택한 대상 플랫폼만 Amazon FreeRTOS 아래에 하위 디렉터리가 있습니다.

데모 애플리케이션을 사용하기 전에 [Amazon FreeRTOS 시작하기 \(p. 61\)](#)에서 자습서를 완료하는 것이 좋습니다. Hello World MQTT 데모를 설정 및 실행하는 방법을 설명합니다.

Amazon FreeRTOS 데모 실행

다음 주제에서는 Amazon FreeRTOS 데모를 설정하고 실행하는 방법을 설명합니다.

- [Bluetooth Low Energy 데모 애플리케이션 \(p. 233\)](#)
- [Microchip Curiosity PIC32MZEF용 데모 부트로더 \(p. 245\)](#)
- [AWS IoT Device Defender 데모 \(p. 250\)](#)
- [AWS IoT Greengrass Discovery 데모 애플리케이션 \(p. 251\)](#)
- [OTA\(Over-the-Air\) 업데이트 데모 애플리케이션 \(p. 253\)](#)
- [보안 소켓 에코 클라이언트 데모 \(p. 264\)](#)
- [AWS IoT 디바이스 샘플러 데모 애플리케이션 \(p. 263\)](#)

`<amazon-freertos>/demos/demo_runner/iot_demo_runner.c`에 있는 `DEMO_RUNNER_RunDemos()` 함수는 단일 데모 애플리케이션이 실행되는 분리된 스레드를 초기화 합니다. 기본적으로 `DEMO_RUNNER_RunDemos()`는 Hello World MQTT 데모 시작만 호출합니다. Amazon FreeRTOS를 다운로드할 때 선택한 구성에 따라 또는 Amazon FreeRTOS를 다운로드한 위치에 따라 다른 예제 실행 함수가 기본적으로 시작될 수 있습니다. 데모 애플리케이션을 활성화하려면 `<amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_demo_config.h`를 열고 실행할 데모를 정의합니다.

Note

모든 예제 조합이 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_files`에서 찾을 수 있습니다.

Bluetooth Low Energy 데모 애플리케이션 개요

Amazon FreeRTOS Bluetooth Low Energy에는 세 개의 데모 애플리케이션이 포함되어 있습니다.

MQTT over Bluetooth Low Energy (p. 240) 데모

이 애플리케이션은 MQTT over Bluetooth Low Energy 서비스를 사용하는 방법을 보여 줍니다.

Wi-Fi 프로비저닝 (p. 242) 데모

이 애플리케이션은 Bluetooth Low Energy Wi-Fi 프로비저닝 서비스를 사용하는 방법을 보여 줍니다.

일반 속성 서버 (p. 244) 데모

이 애플리케이션은 Amazon FreeRTOS Bluetooth Low Energy 미들웨어 API를 사용하여 간단한 GATT 서버를 생성하는 방법을 보여 줍니다.

사전 조건

이 데모를 따라 수행하려면 Bluetooth Low Energy 기능을 지원하는 마이크로 컨트롤러가 필요합니다. 또한 [Amazon FreeRTOS Bluetooth 디바이스용 iOS SDK \(p. 207\)](#) 또는 [Amazon FreeRTOS Bluetooth 디바이스용 Android SDK \(p. 206\)](#)이 필요합니다.

Amazon FreeRTOS Bluetooth Low Energy용 AWS IoT 및 Amazon Cognito 설정

MQTT 전반에 대해 디바이스를 AWS IoT에 연결하려면 AWS IoT 및 Amazon Cognito를 설정해야 합니다.

AWS IoT를 설정하려면

1. <https://aws.amazon.com>에서 AWS 계정을 설정합니다.
2. AWS IoT 콘솔을 열고 탐색 창에서 관리를 선택한 후 사물을 선택합니다.
3. 생성을 선택한 후 단일 사물 생성을 선택합니다.
4. 디바이스에 대해 기억하기 쉬운 이름을 입력한 후 다음을 선택합니다.
5. 모바일 디바이스를 통해 클라우드에 마이크로 컨트롤러를 연결하는 경우 인증서없이 사물을 선택합니다. Mobile SDK는 디바이스 인증에 Amazon Cognito를 사용하기 때문에 Bluetooth Low Energy를 사용하는 데모용 디바이스 인증서를 생성할 필요가 없습니다.

WiFi를 통해 마이크로 컨트롤러를 클라우드에 직접 연결하는 경우 인증서 생성을 선택하고 활성화를 선택한 후 사물 인증서, 퍼블릭 키, 프라이빗 키를 다운로드합니다.

6. 등록된 사물 목록에서 방금 생성한 사물을 선택한 다음 사물 페이지에서 상호 작용을 선택합니다. AWS IoT REST API 엔드포인트를 기록해둡니다.

설정에 대한 자세한 내용은 [AWS IoT 시작하기](#)를 참조하십시오.

Amazon Cognito 사용자 풀을 생성하려면

1. Amazon Cognito 콘솔을 열고 Manage User Pools(사용자 풀 관리)를 선택합니다.
2. [Create a user pool]을 선택합니다.
3. 사용자 풀에 이름을 지정한 다음 기본값 검토를 선택합니다.
4. 왼쪽 탐색 창에서 앱 클라이언트를 선택한 다음 앱 클라이언트 추가를 선택합니다.
5. 앱 클라이언트의 이름을 입력한 후 앱 클라이언트 생성을 선택합니다.
6. 왼쪽 탐색 창에서 검토를 선택한 후 풀 생성을 선택합니다.

사용자 풀의 일반 설정 페이지에 나타나는 풀 ID를 기록해둡니다.

7. 왼쪽 탐색 창에서 앱 클라이언트를 선택한 다음 세부 정보 표시를 선택합니다. 앱 클라이언트 ID와 앱 클라이언트 암호를 메모합니다.

Amazon Cognito 자격 증명 풀을 생성하려면

1. Amazon Cognito 콘솔을 열고 Manage Identity Pools(자격 증명 풀 관리)를 선택합니다.
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 보기를 확장하고 2개의 IAM 역할 이름을 적어둡니다. 허용을 선택하여 인증된 자격 증명 및 인증되지 않은 자격 증명에 대한 IAM 역할을 만들고 Amazon Cognito를 액세스합니다.
6. 자격 증명 풀 편집을 선택합니다. 자격 증명 풀 ID를 메모합니다. us-west-2:12345678-1234-1234-1234-123456789012 형식이어야 합니다.

Amazon Cognito 설정에 대한 자세한 내용은 [Amazon Cognito 시작하기](#)를 참조하십시오.

인증된 자격 증명에 IAM 정책을 생성 및 연결하려면

1. IAM 콘솔을 열고 탐색 창에서 역할을 선택합니다.
2. 인증된 자격 증명의 역할을 검색 및 선택하고 Attach policies(정책 연결)를 선택한 후 Add inline policy(인라인 정책 추가)를 선택합니다.
3. JSON 탭을 선택하고 다음 JSON을 붙여 넣습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

4. 정책 검토를 선택하고 정책 이름을 입력한 후 정책 생성을 선택합니다.

AWS IoT 및 Amazon Cognito 정보를 수중에 보관합니다. AWS 클라우드를 사용하여 모바일 애플리케이션을 인증하려면 앤드포인트와 ID가 필요합니다.

Bluetooth Low Energy용 Amazon FreeRTOS 환경 설정

환경을 설정하려면 마이크로컨트롤러에 [Amazon FreeRTOS Bluetooth Low Energy 라이브러리 \(p. 197\)](#)와 함께 Amazon FreeRTOS를 다운로드해야 하며 모바일 디바이스에 Amazon FreeRTOS Bluetooth 디바이스용 Mobile SDK를 다운로드하고 구성해야 합니다.

Amazon FreeRTOS Bluetooth Low Energy로 마이크로 컨트롤러 환경을 설정하려면

1. [GitHub](#)에서 Amazon FreeRTOS를 다운로드하거나 복제합니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오.
2. 마이크로컨트롤러에 Amazon FreeRTOS를 설정합니다.

Amazon FreeRTOS 인증 마이크로 컨트롤러에서 Amazon FreeRTOS를 시작하는 방법에 대한 자세한 내용은 [Amazon FreeRTOS 시작하기](#)의 지침을 참조하십시오.

Note

Amazon FreeRTOS 및 이식된 Amazon FreeRTOS Bluetooth Low Energy 라이브러리를 지원하는 Bluetooth Low Energy 사용 마이크로 컨트롤러에서 데모를 실행할 수 있습니다. 현재 Amazon FreeRTOS [MQTT over Bluetooth Low Energy \(p. 240\)](#) 데모 프로젝트는 다음 Bluetooth Low Energy 사용 디바이스에 완벽하게 이식됩니다.

- [Espressif ESP32-DevKitC 및 ESP-WROVER-KIT](#)
- [Nordic nRF52840-DK](#)

공통 구성 요소

Amazon FreeRTOS 데모 애플리케이션에는 두 개의 공통 요소가 있습니다.

- Network Manager
- Bluetooth Low Energy Mobile SDK 데모 애플리케이션

Network Manager

Network Manager는 마이크로 컨트롤러의 네트워크 연결을 관리하며, `demos/network_manager/aws_iot_network_manager.c`의 Amazon FreeRTOS 디렉터리에 있습니다. Network Manager를 Wi-Fi와 Bluetooth Low Energy 모두에 대해 활성화한 경우 데모는 기본적으로 Bluetooth Low Energy로 시작합니다. Bluetooth Low Energy 연결이 끊어지고 보드가 Wi-Fi를 지원하는 경우 Network Manager는 사용 가능한 Wi-Fi 연결로 전환하여 네트워크 연결이 끊어지지 않도록 보호합니다.

Network Manager를 사용하여 네트워크 연결 유형을 활성화하려면 `vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_iot_network_config.h`의 `configENABLED_NETWORKS` 파라미터에 네트워크 연결 유형을 추가합니다. 여기서 `vendor`는 공급업체의 이름이고 `board`는 데모를 실행하는 데 사용하는 보드 이름입니다.

예를 들어, Bluetooth Low Energy와 Wi-Fi를 모두 활성화한 경우 `aws_iot_network_config.h`에서 `#define configENABLED_NETWORKS`로 시작하는 행은 다음과 같습니다.

```
#define configENABLED_NETWORKS ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

현재 지원되는 네트워크 연결 유형의 목록을 가져오려면 `aws_iot_network.h`에서 `#define AWSIOT_NETWORK_TYPE`으로 시작하는 줄을 참조하십시오.

Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션

Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션은 `amazon-freertos-ble-android-sdk/app` 아래의 [Amazon FreeRTOS Bluetooth 디바이스용 Android SDK](#) 및 `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo` 아래의 [Amazon FreeRTOS Bluetooth 디바이스용 iOS SDK](#)에 있습니다. 이 예에서는 iOS 버전의 데모 모바일 애플리케이션의 스크린샷을 사용합니다.

Note

iOS 디바이스를 사용하는 경우 데모 모바일 애플리케이션을 빌드하려면 Xcode가 필요합니다.
Android 디바이스를 사용하는 경우 Android Studio를 사용하여 데모 모바일 애플리케이션을 빌드할 수 있습니다.

iOS SDK 데모 애플리케이션을 구성하려면

구성 변수를 정의할 경우 구성 파일에 제공된 자리 표시자 값의 형식을 사용합니다.

1. [Amazon FreeRTOS Bluetooth 디바이스용 iOS SDK \(p. 207\)](#)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 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. <aws-region> 및 <aws-account>를 해당 AWS 리전 및 계정 ID로 바꿉니다.

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

- c. 생성을 선택합니다.

5. amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift를 열고 다음 변수를 재정의합니다.

- region: 해당 AWS 리전.
 - iotPolicyName: 해당 AWS IoT 정책 이름.
 - mqttCustomTopic: 게시하려는 MQTT 주제.
6. Open amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json.

CognitoIdentity 아래에서 다음 변수를 재정의합니다.

- PoolId: 해당 Amazon Cognito 자격 증명 풀 ID.
- Region: 해당 AWS 리전.

CognitoUserPool 아래에서 다음 변수를 재정의합니다.

- PoolId: 해당 Amazon Cognito 사용자 풀 ID.
- 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. [Amazon FreeRTOS Bluetooth 디바이스용 Android SDK \(p. 206\)](#)이 설치되어 있는지 확인합니다.
2. 해당 리전에서 AWS IoT 정책을 생성합니다(아직 설정하지 않은 경우).

Note

이 정책은 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. <aws-region> 및 <aws-account>를 해당 AWS 리전 및 계정 ID로 바꿉니다.

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

```
        ]  
    }
```

c. 생성을 선택합니다.

3. <https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants.java> 를 열고 다음 변수를 재정의합니다.
 - AWS_IOT_POLICY_NAME: 해당 AWS IoT 정책 이름.
 - AWS_IOT_REGION: 해당 AWS 리전.
4. <https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/res/raw/awsconfiguration.json>을 업니다.

CognitoIdentity 아래에서 다음 변수를 재정의합니다.

- PoolId: 해당 Amazon Cognito 자격 증명 풀 ID.
- Region: 해당 AWS 리전.

CognitoUserPool 아래에서 다음 변수를 재정의합니다.

- PoolId: 해당 Amazon Cognito 사용자 풀 ID.
- 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)가 필요합니다. [터미널 에뮬레이터 설치 \(p. 69\)](#)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.



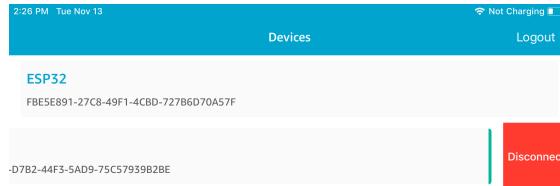
Note

Amazon FreeRTOS를 지원하는 모든 디바이스와 범위 내에 있는 디바이스 정보 서비스 ([`<amazon-freertos>/.../device_information`](#))가 목록에 나타납니다.

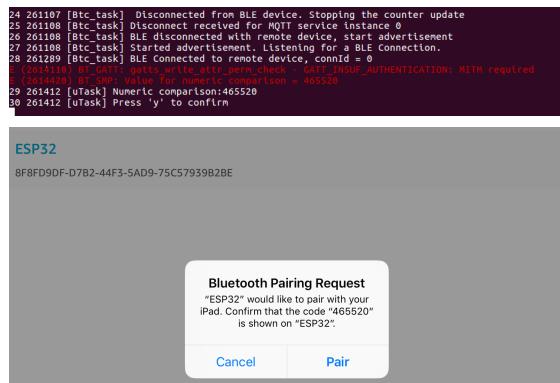
5. 디바이스 목록에서 마이크로 컨트롤러를 선택합니다. 애플리케이션에서 보드와의 연결을 설정하고 연결된 디바이스 옆에 녹색 선이 나타납니다.



선을 왼쪽으로 끌어서 마이크로 컨트롤러를 연결 해제할 수 있습니다.



6. 메시지가 표시되면 마이크로컨트롤러와 모바일 디바이스를 페어링합니다.



두 디바이스의 수 비교를 위한 코드가 동일한 경우 디바이스를 연결합니다.

Note

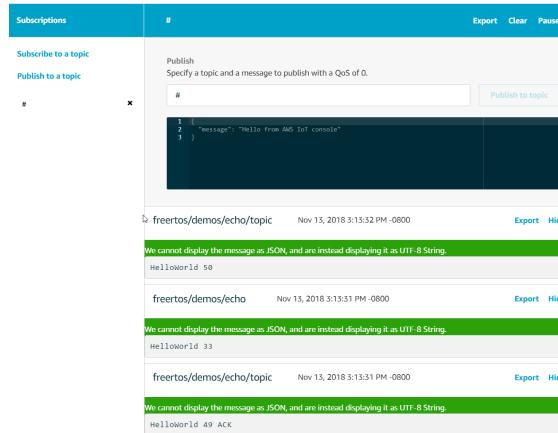
Bluetooth Low Energy Mobile SDK 데모 애플리케이션에서는 사용자 인증을 위해 Amazon Cognito 를 사용합니다. Amazon Cognito 사용자 및 자격 증명 풀을 설정하고 IAM 정책을 인증된 자격 증명에 연결했는지 확인합니다.

MQTT over Bluetooth Low Energy

MQTT over Bluetooth Low Energy 데모에서, 마이크로 컨트롤러는 MQTT 프록시를 통해 AWS 클라우드에 메시지를 게시합니다.

데모 MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. Subscription topic(구독 주제)에 `iotdemo/#`를 입력한 다음 Subscribe to topic(주제 구독)을 선택합니다.



Bluetooth Low Energy 또는 Wi-Fi 연결을 통해 MQTT 데모를 실행할 수 있습니다. [Network Manager \(p. 236\)](#)의 구성에 따라 사용되는 연결 유형이 결정됩니다.

Bluetooth Low Energy를 사용하여 마이크로 컨트롤러를 모바일 디바이스와 연결하는 경우 MQTT 메시지는 모바일 디바이스의 Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 통해 라우팅됩니다.

Wi-Fi를 사용하는 경우 데모는 `demos/mqtt/aws_hello_world.c`에 있는 MQTT Hello World 데모 프로젝트와 동일합니다. 이 데모가 대부분의 [Amazon FreeRTOS 시작하기](#) 데모 프로젝트에서 사용됩니다.

Bluetooth Low Energy 또는 Wi-Fi를 통해 데모를 활성화하려면

`vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_demo_config.h`를 열고 `CONFIG_MQTT_DEMO_ENABLED`을 정의합니다.

- `<amazon-freertos>/aws_demos/config_files/aws_demo_config.h`를 열고 네트워크 유형을 구성합니다.

구성에는 다음이 포함됩니다.

```
#define democonfigNETWORK_TYPES ( AWSIOT_NETWORK_TYPE_BLE )
```

Bluetooth Low Energy가 연결될 때 실행됩니다.

```
#define democonfigNETWORK_TYPES ( AWSIOT_NETWORK_TYPE_WIFI )
```

WIFI가 AP에 연결될 때 실행됩니다.

```
#define democonfigNETWORK_TYPES ( AWSIOT_NETWORK_TYPE_WIFI | AWSIOT_NETWORK_TYPE_BLE )
```

첫 번째 사용 가능한 네트워크 유형에 연결됩니다. 두 유형 모두 사용할 수 있는 경우 Wi-Fi가 사용됩니다.

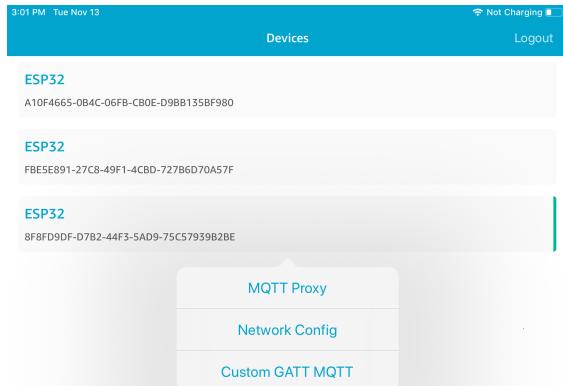
데모를 실행하려면

Network Manager가 Wi-Fi에 대해서만 구성되어 있는 경우 보드에서 데모 프로젝트를 빌드한 후 실행하면 됩니다.

Network Manager가 Bluetooth Low Energy에 대해 구성되어 있는 경우 다음을 수행합니다.

1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
2. [Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션 \(p. 236\)](#)을 사용하여 보드와 모바일 디바이스를 연결했는지 확인합니다.

- 데모 모바일 앱의 디바이스 목록에서 마이크로 컨트롤러를 선택한 다음 MQTT Proxy(MQTT 프록시)를 선택하여 MQTT 프록시 설정을 엽니다.



- MQTT 프록시를 활성화한 후에는 MQTT 메시지가 `iotdemo/#` 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 Amazon FreeRTOS Bluetooth Low Energy 서비스입니다. Wi-Fi 프로비저닝 서비스를 위한 소스 코드는 [amazon-freertos](#)/.../wifi_provisioning에서 찾을 수 있습니다.

Note

Wi-Fi 프로비저닝 데모는 현재 Espressif ESP32-DevKitC에서 지원됩니다.

Android 버전 데모 모바일 애플리케이션에서는 현재 Wi-Fi 프로비저닝을 지원하지 않습니다.

데모를 활성화하려면

- 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

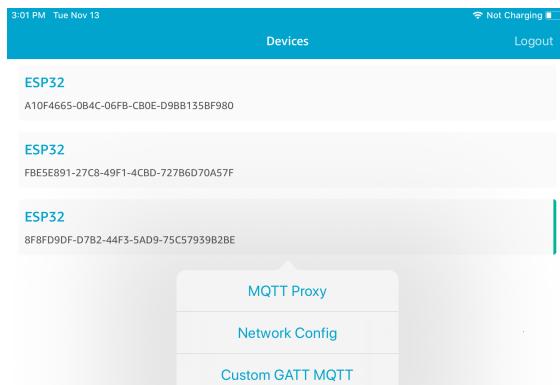
Wi-Fi 프로비저닝 서비스는 기본적으로 비활성화되어 있습니다.

- Bluetooth Low Energy와 Wi-Fi를 모두 활성화하도록 [Network Manager \(p. 236\)](#)를 구성합니다.

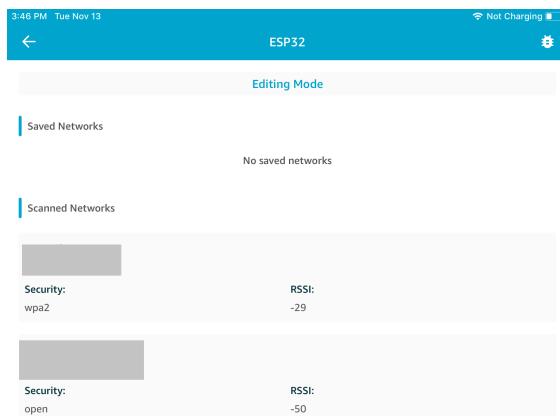
데모를 실행하려면

- 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
- [Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션 \(p. 236\)](#)을 사용하여 마이크로 컨트롤러와 모바일 디바이스를 연결했는지 확인합니다.
- 데모 모바일 앱의 디바이스 목록에서 마이크로 컨트롤러를 선택한 다음 Network Config(네트워크 구성)를 선택하여 네트워크 구성 설정을 엽니다.

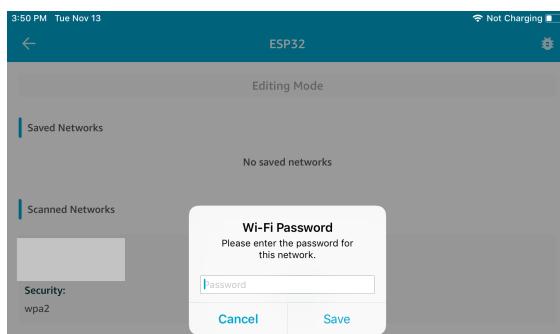
Amazon FreeRTOS 사용 설명서 Wi-Fi 프로비저닝



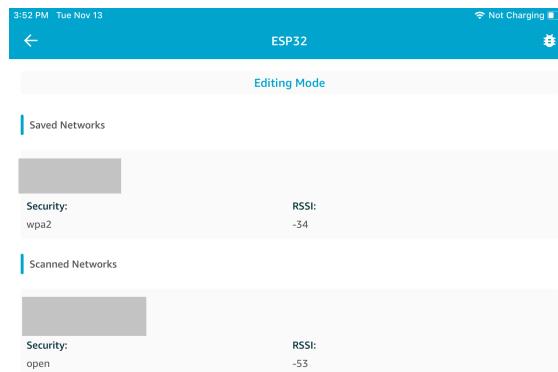
- 보드에 대한 Network Config(네트워크 구성)를 선택하면 마이크로 컨트롤러는 근처에 있는 네트워크 목록을 모바일 디바이스로 전송합니다. 사용 가능한 Wi-Fi 네트워크가 Scanned Networks(스캔된 네트워크) 아래 목록에 표시됩니다.



Scanned Networks(스캔된 네트워크) 목록에서 네트워크를 선택한 후 필요한 경우 SSID 및 암호를 입력합니다.

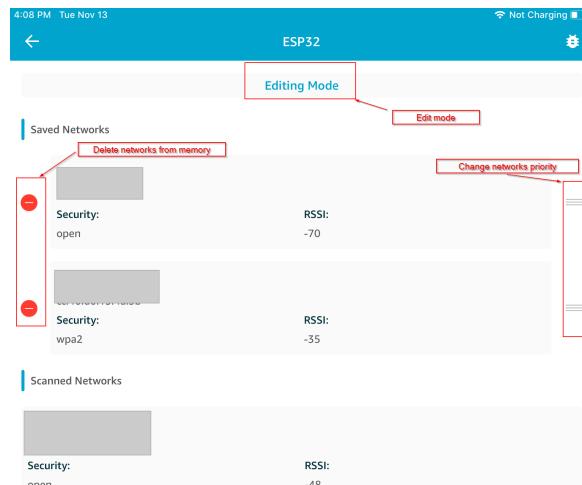


마이크로 컨트롤러가 네트워크에 연결되고 네트워크를 저장합니다. 네트워크가 Saved Networks(저장된 네트워크) 아래에 나타납니다.



데모 모바일 앱에서 여러 네트워크를 저장할 수 있습니다. 애플리케이션 및 데모를 재시작하면 마이크로 컨트롤러가 Saved Networks(저장된 네트워크) 목록의 맨 위에서부터 첫 번째 사용 가능한 저장 네트워크에 연결됩니다.

네트워크 우선 순위를 변경하거나 네트워크를 삭제하려면 Network Configuration(네트워크 구성) 페이지에서 Editing Mode(모드 편집)를 선택합니다. 네트워크 우선 순위를 변경하려면 우선 순위를 조정하려는 네트워크의 오른쪽을 선택하고 네트워크를 위 또는 아래로 끌어 놓습니다. 네트워크를 삭제하려면 삭제할 네트워크의 왼쪽에 있는 빨간색 버튼을 선택합니다.



일반 속성 서버

이 예에서 마이크로 컨트롤러에 대한 데모 일반 속성(GATT) 서버 애플리케이션은 단순 카운터 값을 [Amazon FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션 \(p. 236\)](#)에 전송합니다.

Bluetooth Low Energy Mobile SDK를 사용하여 마이크로 컨트롤러의 GATT 서버에 연결되고 데모 모바일 앱 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 보드 이름임) `#define IOT_BLE_ADD_CUSTOM_SERVICES (1)`를 정의문 목록에 추가합니다.

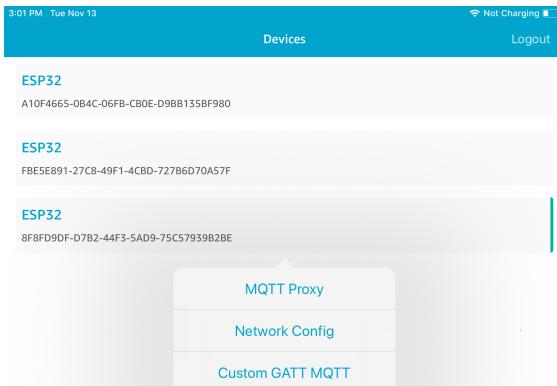
Note

Bluetooth Low Energy GATT 데모는 기본적으로 비활성화되어 있습니다.

2. <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_BLE_GATT_SERVER_DEMO_ENABLED를 정의합니다.

데모를 실행하려면

1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
2. Amazon FreeRTOS Bluetooth Low Energy Mobile 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 iotdemo/# MQTT 주제에 대한 데이터 게시를 시작합니다.

MQTT 프록시를 활성화한 후에는 Hello World 및 종분 카운터 메시지가 iotdemo/# 주제에 나타납니다.

Microchip Curiosity PIC32MZEF용 데모 부트로더

이 데모 부트로더는 펌웨어 버전 확인, 암호화 서명 검증, 애플리케이션 셀프 테스트를 수행합니다. 이들 기능은 Amazon FreeRTOS에 대한 OTA(Over-the-Air) 펌웨어 업데이트를 지원합니다.

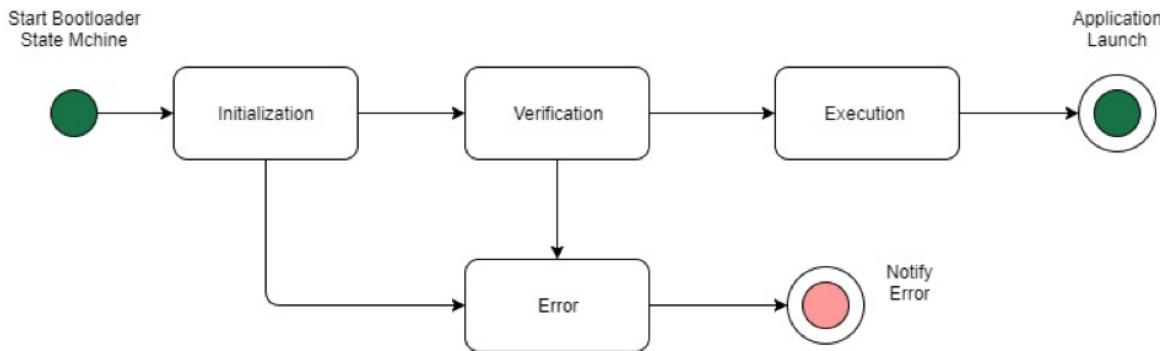
펌웨어 검증에는 원격(OTA)으로 수신한 새 펌웨어의 신뢰성 및 무결성 검증이 포함됩니다. 부트로더는 부팅 전에 애플리케이션의 암호화 서명을 검증합니다. 데모에서는 SHA-256을 통한 ECDSA(Elliptic-Curve Digital Signature Algorithm)를 사용합니다. 제공된 유ти리티를 사용하여 디바이스에 플래시할 수 있는 서명된 애플리케이션을 생성할 수 있습니다.

부트로더는 OTA에 필요한 다음 기능을 지원합니다.

- 디바이스에 애플리케이션 이미지를 유지하여 이미지 간에 전환합니다.
- 수신된 OTA 이미지의 셀프 테스트 실행 및 실패 시 룰백을 허용합니다.
- OTA 업데이트 이미지의 서명과 버전을 확인합니다.

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.
Verification(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.
Error(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

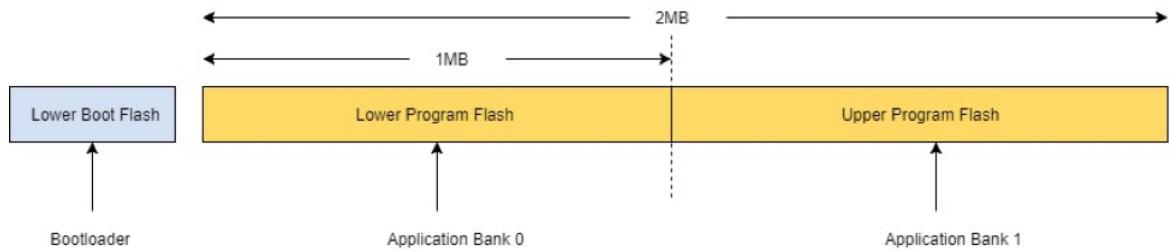
기본 이미지를 실행하는 구성 옵션이 활성화된 경우 부트로더는 기본 실행 주소에서 애플리케이션을 시작합니다. 이 옵션은 디버깅할 때를 제외하고는 비활성화해야 합니다.

부트로더 오류 상태

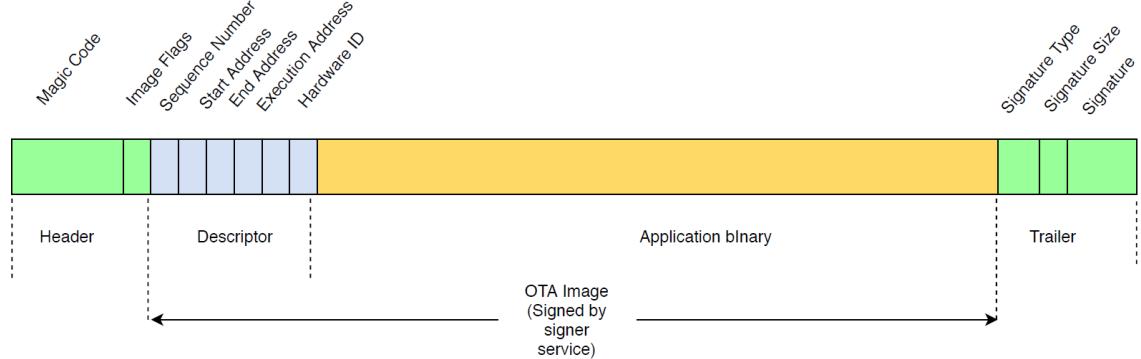
부트로더가 오류 상태이고, 디바이스에 유효한 이미지가 없습니다. 부트로더가 사용자에게 알려야 합니다. 기본 구현은 콘솔에 로그 메시지를 전송하고, 보드의 LED를 계속 빠르게 깜박이는 것입니다.

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	8바이트
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	4바이트
시작 주소	4bytes
종료 주소	4바이트
실행 주소	4바이트
하드웨어 ID	4바이트
예약	4바이트

시퀀스 번호

시퀀스 번호가 증가한 뒤에 새 OTA 이미지를 빌드해야 합니다. `ota-descriptor.config` 파일을 참조하십시오. 부트로더는 이 번호로 부팅할 이미지를 결정합니다. 유효한 값은 1~4294967295입니다.

시작 주소

디바이스에 있는 애플리케이션 이미지의 시작 주소입니다. 이미지 설명자가 애플리케이션 이진 파일에 추가되기 때문에 이 주소는 이미지 설명자의 시작입니다.

종료 주소

디바이스에 있는 애플리케이션 이미지의 종료 주소입니다(이미지 트레일러 제외).

실행 주소

이미지의 실행 주소입니다.

하드웨어 ID

OTA 이미지가 올바른 플랫폼용으로 빌드되었는지 확인하기 위해 부트로더가 사용하는 고유한 하드웨어 ID입니다.

예약

나중에 사용하기 위해 예약되어 있습니다.

이미지 트레일러

이미지 트레일러는 애플리케이션 이진 파일에 추가됩니다. 서명 유형 문자열, 서명 크기, 이미지의 서명 등이 들어 있습니다.

트레일러 필드	크기(바이트)
서명 유형	32바이트
서명 크기	4바이트
서명	256바이트

서명 유형

이 서명 유형은 사용되는 암호화 알고리즘을 나타내며, 트레일러를 위한 마커 역할을 합니다. 부트로더는 ECDSA(Elliptic-Curve Digital Signature Algorithm)를 지원합니다. 기본값은 sig-sha256-ecdsa입니다.

서명 크기

암호화 서명의 크기(바이트)입니다.

서명

이미지 설명자와 함께 추가된 애플리케이션 이진 파일의 암호화 서명입니다.

부트로더 구성

기본 부트로더 구성 옵션은 `<amazon-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 비활성화할 수 있습니다.

Enable Address Verification(주소 검증 활성화)

OTA 이미지의 설명자에 있는 시작, 종료, 실행 주소를 검증합니다. 이 옵션을 항상 활성화해 놓는 것이 좋습니다.

부트로더 빌드

데모 부트로더는 Amazon FreeRTOS 소스 코드 리포지토리의 [`amazon-freertos`/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab`](#)에 있는 aws_demos 프로젝트에 로드 가능한 프로젝트로 포함되어 있습니다. 빌드된 aws_demos 프로젝트는 먼저 부트로더를 빌드한 다음 애플리케이션을 빌드합니다. 최종 출력은 부트로더와 애플리케이션을 포함하여 통합 16진 이미지입니다. 암호화 서명이 있는 통합 16진 이미지를 생성할 수 있도록 factory_image_generator.py 유ти리티가 제공됩니다. 부트로더 유ти리티 스크립트는 [`amazon-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_demos를 사용하지 않고 부트로더 프로젝트를 빌드할 수도 있으며 일반 애플리케이션처럼 디버깅할 수 있습니다.

AWS IoT Device Defender 데모

Amazon FreeRTOS에는 디바이스에 대한 일부 AWS IoT Device Defender 지표를 수집하고 이를 MQTT 주제에 게시하는 단일 스레드 데모 애플리케이션이 포함되어 있습니다. 이 데모는 [`amazon-freertos/demos/defender/aws_iot_defender_demo.c`](#)에 정의되어 있습니다.

Device Defender 데모를 실행하려면 디바이스가 AWS 클라우드와 통신할 수 있도록 첫 번째 단계 (p. 62) 시작하기를 완료하여 AWS IoT 및 Amazon FreeRTOS를 설정해야 합니다.

[`amazon-freertos/vendors/<vendor>/boards/<board>/aws_demos/config_files/`](#) aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_DEFENDER_DEMO_ENABLED를 정의합니다.

Device Defender 데모가 활성화된 상태로 디바이스에서 Amazon FreeRTOS를 빌드, 플래시 및 실행하면 다음과 같은 출력이 나타납니다.

```
12 343 [iot_thread] [INFO ][DEMO][343] ----Device Defender Demo Start----  
13 343 [iot_thread] [INFO ][MQTT][343] MQTT library successfully initialized.  
14 343 [iot_thread] [INFO ][Defender][343] Metrics are successfully updated.  
15 343 [iot_thread] [INFO ][Defender][343] Period has been set to 300 seconds successfully.  
16 343 [iot_thread] [INFO ][DEMO][343] Defender Thing Name is Thing-1 (length 7).  
17 711 [iot_thread] [INFO ][MQTT][711] Establishing new MQTT connection.  
18 711 [iot_thread] [INFO ][MQTT][711] Anonymous metrics (SDK language, SDK version) will  
be provided to AWS IoT. Recompile with AWS_IOT_ENABLE_METRICS set to 0 to disable.
```

```
19 711 [iot_thread] [INFO ][MQTT][711] (MQTT connection 00530B30, CONNECT operation
00530CC0) Waiting for operation completion.
20 771 [iot_thread] [INFO ][MQTT][771] (MQTT connection 00530B30, CONNECT operation
00530CC0) Wait complete with result SUCCESS.
21 771 [iot_thread] [INFO ][MQTT][771] New MQTT connection 0203FA0C established.
22 771 [iot_thread] [INFO ][MQTT][771] (MQTT connection 00530B30) SUBSCRIBE operation
scheduled.
23 771 [iot_thread] [INFO ][MQTT][771] (MQTT connection 00530B30, SUBSCRIBE operation
00530E30) Waiting for operation completion.
24 811 [iot_thread] [INFO ][MQTT][811] (MQTT connection 00530B30, SUBSCRIBE operation
00530E30) Wait complete with result SUCCESS.
25 811 [iot_thread] [INFO ][Defender][811] Defender agent has successfully started.
26 812 [iot_thread] [INFO ][MQTT][812] (MQTT connection 00530B30) MQTT PUBLISH operation
queued.
27 932 [iot_thread] [INFO ][Defender][932] Metrics report was accepted by defender service.
28 932 [iot_thread] [INFO ][DEMO][932] User's callback is invoked on event: Defender
Metrics accepted.
29 932 [iot_thread] [INFO ][DEMO][932] Published metrics report.
30 932 [iot_thread] [INFO ][DEMO][932] Received MQTT message.
31 8811 [iot_thread] [INFO ][Defender][8811] Unsubscribing from MQTT topics
32 8811 [iot_thread] [INFO ][MQTT][8811] (MQTT connection 00530B30) UNSUBSCRIBE operation
scheduled.
33 8811 [iot_thread] [INFO ][MQTT][8811] (MQTT connection 00530B30, UNSUBSCRIBE operation
00530F40) Waiting for operation completion.
34 8891 [iot_thread] [INFO ][MQTT][8891] (MQTT connection 00530B30, UNSUBSCRIBE operation
00530F40) Wait complete with result SUCCESS.
35 8891 [iot_thread] [INFO ][Defender][8891] Defender agent has stopped.
35 8891 [iot_thread] [INFO ][MQTT][8891] (MQTT connection 00530B30) Disconnecting
connection.
37 8892 [iot_thread] [INFO ][MQTT][8892] (MQTT connection 00530B30, DISCONNECT operation
00530CEO) Waiting for operation completion.
38 8892 [iot_thread] [INFO ][MQTT][8892] (MQTT connection 00530B30, DISCONNECT operation
00530CEO) Wait complete with result SUCCESS.
39 8892 [iot_thread] [INFO ][MQTT][8892] (MQTT connection 00530B30) Connection
disconnected.
40 8893 [iot_thread] [INFO ][MQTT][8893] (MQTT connection 00530B30) Network connection
closed.
41 8931 [iot_thread] [INFO ][MQTT][8931] (MQTT connection 00530B30) Network connection
destroyed.
42 8931 [iot_thread] [INFO ][MQTT][8931] MQTT library cleanup done.
43 8931 [iot_thread] [INFO ][DEMO][8931] ----Device Defender Demo End. Status:
SUCCESS----
```

AWS IoT Greengrass Discovery 데모 애플리케이션

Amazon FreeRTOS용 AWS IoT Greengrass Discovery 데모를 실행하려면 먼저 AWS, AWS IoT Greengrass, AWS IoT를 설정해야 합니다. AWS를 설정하려면 [AWS 계정 및 권한 설정 \(p. 62\)](#)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

1. [IAM 콘솔](#)로 이동합니다.
2. 탐색 창에서 역할을 선택한 후 Greengrass_ServiceRole을 찾아 선택합니다.
3. 정책 연결을 선택하고 AmazonS3FullAccess와 AWSIoTFullAccess를 선택한 후 정책 연결을 선택합니다.
4. [AWS IoT 콘솔](#)로 이동합니다.

5. 탐색 창에서 Greengrass를 선택하고 그룹을 선택한 후 앞에서 만든 Greengrass 그룹을 선택합니다.
6. 설정을 선택한 후 역할 추가를 선택합니다.
7. Greengrass_ServiceRole을 선택한 후 저장을 선택합니다.

Amazon FreeRTOS 콘솔에서 빠른 연결 위크플로우를 사용하여 보드를 AWS IoT에 빠르게 연결하고 데모를 실행할 수 있습니다. 다음 보드에는 현재 Amazon FreeRTOS 구성은 사용할 수 없습니다.

- Cypress CYW943907AEVAL1F 개발 키트
- Cypress CYW954907AEVAL1F 개발 키트
- Espressif ESP-WROVER-KIT
- Espressif ESP32-DevKitC
- Nordic nRF52840-DK

보드를 AWS IoT에 연결하고 Amazon FreeRTOS 데모를 직접 구성할 수도 있습니다.

1. AWS IoT에 MCU 보드 등록 (p. 63)

보드를 등록한 후에는 새로운 Greengrass 정책을 생성하여 디바이스 인증서에 연결해야 합니다.

새로운 AWS IoT Greengrass 정책을 만들려면

1. AWS IoT 콘솔로 이동합니다.
2. 탐색 창에서 Secure(보안)를 선택하고 Policies(정책)를 선택한 다음 Create(생성)를 선택합니다.
3. 정책을 식별할 이름을 입력합니다.
4. Add statements(설명문 추가) 섹션에서 Advanced mode(고급 모드)를 선택합니다. 다음 JSON을 복사하여 정책 편집기 창에 붙여 넣습니다.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "greengrass:*"  
    ],  
    "Resource": "*"  
}
```

이 정책은 모든 리소스에 대한 AWS IoT Greengrass 권한을 부여합니다.

5. Create를 선택합니다.

AWS IoT Greengrass 정책을 디바이스의 인증서에 연결하려면

1. AWS IoT 콘솔로 이동합니다.
2. 탐색 창에서 관리를 선택하고 사물을 선택한 후 앞에서 만든 사물을 선택합니다.
3. 보안을 선택한 후 디바이스에 연결된 인증서를 선택합니다.
4. 정책을 선택하고 작업을 선택한 후 정책 연결을 선택합니다.
5. 앞에서 만든 Greengrass 정책을 선택한 후 연결을 선택합니다.

2. Amazon FreeRTOS 다운로드 (p. 65)

Note

Amazon FreeRTOS 콘솔에서 Amazon FreeRTOS를 다운로드하는 경우 AWS IoT- #####에 연결 대신 AWS IoT Greengrass- #####에 연결을 선택합니다.

3. Amazon FreeRTOS 데모 구성 (p. 66)를 선택하십시오.

<amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/
aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음
CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED를 정의합니다.

AWS IoT 및 AWS IoT Greengrass를 설정하고 Amazon FreeRTOS를 다운로드하고 구성하면 디바이스에서 Greengrass를 빌드, 플래시 및 실행할 수 있습니다. 보드의 하드웨어 및 소프트웨어 개발 환경을 설정하려면 [보드별 시작 안내서 \(p. 76\)](#)의 지침을 수행해야 합니다.

Greengrass 데모는 코어에 Greengrass 코어 및 AWS IoT MQTT 클라이언트에 일련의 메시지를 게시합니다. AWS IoT MQTT 클라이언트에서 메시지를 보려면 [AWS IoT 콘솔](#)을 열고 테스트를 선택한 후 freertos/demos/ggd에 구독을 추가합니다.

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
```

OTA(Over-the-Air) 업데이트 데모 애플리케이션

Amazon FreeRTOS에는 무선(OTA) 라이브러리의 기능을 시연하는 데모 애플리케이션이 포함되어 있습니다. OTA 데모 애플리케이션은 <amazon-freertos>/demos/ota 하위 디렉터리에 있습니다.

OTA 데모 애플리케이션:

1. FreeRTOS 네트워크 스택과 MQTT 버퍼 풀을 초기화합니다. ([main.c](#)를 참조하십시오.)
2. OTA 라이브러리 연습을 위한 작업을 생성합니다. ([aws_ota_update_demo.c](#)의 vOTAUpdateDemoTask를 참조하십시오.)
3. [MQTT_AGENT_Create](#)를 사용하여 MQTT 클라이언트를 생성합니다.
4. [MQTT_AGENT_Connect](#)를 사용하여 AWS IoT MQTT 브로커에 연결합니다.
5. [OTA_AgentInit](#)를 호출하여 OTA 작업을 생성하고, OTA 작업이 완료될 때 사용할 콜백을 등록합니다.

OTA 업데이트를 사용하기 전에 [Amazon FreeRTOS 무선\(OTA\) 업데이트 \(p. 8\)](#) 단원을 읽고 명시된 모든 사전 조건을 갖추십시오.

OTA 업데이트를 위한 설정을 완료한 경우 OTA 기능을 지원하는 플랫폼에서 Amazon FreeRTOS OTA 데모를 다운로드, 빌드, 플래시 및 실행합니다. 디바이스별 데모 지침은 다음 Amazon FreeRTOS 적격 디바이스에서 사용할 수 있습니다.

- Texas Instruments CC3220SF-LAUNCHXL (p. 256)
- Microchip Curiosity PIC32MZEF (p. 258)
- Espressif ESP32 (p. 261)

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 종에 발생한 오류를 기록해둡니다.

OTA 업데이트 작업이 성공하면 다음과 같은 출력이 표시됩니다. 간결하게 하기 위해 이 예제에서는 일부 행을 목록에서 제거했습니다.

```
313 267848 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
```

```
314 268733 [OTA Task] [OTA] Set job doc parameter [ jobId:  
fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]  
315 268734 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]  
316 268734 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]  
317 268734 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]  
318 268735 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]  
319 268735 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]  
320 268735 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]  
321 268737 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:  
Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]  
322 268737 [OTA Task] [OTA] Job was accepted. Attempting to start transfer.  
323 268737 [OTA Task] Sending command to MQTT task.  
324 268737 [MQTT] Received message 50000 from queue.  
325 268848 [OTA] Queued: 2 Processed: 1 Dropped: 0  
326 269039 [MQTT] MQTT Subscribe was accepted. Subscribed.  
327 269039 [MQTT] Notifying task.  
328 269040 [OTA Task] Command sent to MQTT task passed.  
329 269041 [OTA Task] Subscribed to topic: $aws/things/TI-LaunchPad/streams/327  
  
330 269848 [OTA] Queued: 2 Processed: 1 Dropped: 0  
... // Output removed for brevity  
346 284909 [OTA Task] [OTA] file token: 74594452  
.. // Output removed for brevity  
363 301327 [OTA Task] [OTA] file ready for access.  
364 301327 [OTA Task] [OTA] Returned buffer to MQTT Client.  
365 301328 [OTA Task] Sending command to MQTT task.  
366 301328 [MQTT] Received message 60000 from queue.  
367 301328 [MQTT] Notifying task.  
368 301329 [OTA Task] Command sent to MQTT task passed.  
369 301329 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/  
streams/327/get  
370 301632 [OTA Task] [OTA] Received file block 0, size 1024  
371 301647 [OTA Task] [OTA] Remaining: 127  
... // Output removed for brevity  
508 304622 [OTA Task] Sending command to MQTT task.  
509 304622 [MQTT] Received message 70000 from queue.  
510 304622 [MQTT] Notifying task.  
511 304623 [OTA Task] Command sent to MQTT task passed.  
512 304623 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/  
streams/327/get  
513 304860 [OTA] Queued: 47 Processed: 47 Dropped: 83  
514 304926 [OTA Task] [OTA] Received file block 4, size 1024  
515 304941 [OTA Task] [OTA] Remaining: 82  
... // Output removed for brevity  
797 315047 [MQTT] MQTT Publish was successful.  
798 315048 [MQTT] Notifying task.  
799 315048 [OTA Task] Command sent to MQTT task passed.  
800 315049 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/  
jobs/fe18c7ec_8c31_4438_b0b9_ad55acd9561801 315049 [OTA Task] Sending command to MQTT task.  
802 315049 [MQTT] Received message d0000 from queue.  
803 315150 [MQTT] MQTT Unsubscribe was successful.  
804 315150 [MQTT] Notifying task.  
805 315151 [OTA Task] Command sent to MQTT task passed.  
806 315152 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/streams/327  
  
807 315172 [OTA Task] Sending command to MQTT task.  
808 315172 [MQTT] Received message e0000 from queue.  
809 315273 [MQTT] MQTT Unsubscribe was successful.  
810 315273 [MQTT] Notifying task.  
811 315274 [OTA Task] Command sent to MQTT task passed.  
812 315274 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/streams/327  
  
813 315275 [OTA Task] [OTA] Resetting MCU to activate new image.  
0 0 [Tmr Svc] Starting Wi-Fi Module ...  
1 0 [Tmr Svc] Simple Link task created
```

```
Device came up in Station mode

2 137 [Tmr Svc] Wi-Fi module initialized.
3 137 [Tmr Svc] Starting key provisioning...
4 137 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 339 [Tmr Svc] Write device certificate...
7 436 [Tmr Svc] Key provisioning done...
Device disconnected from the AP on an ERROR!!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 44:48:c1:ba:b2:c3

[NETAPP EVENT] IP acquired by the device

Device has connected to Guest

Device IP Address is 192.168.3.72

8 1443 [Tmr Svc] Wi-Fi connected to AP Guest.
9 1444 [Tmr Svc] IP Address acquired 192.168.3.72
10 1444 [OTA] OTA demo version 0.9.1
11 1445 [OTA] Creating MQTT Client...
12 1445 [OTA] Connecting to broker...
13 1445 [OTA] Sending command to MQTT task.
14 1445 [MQTT] Received message 10000 from queue.
15 2910 [MQTT] MQTT Connect was accepted. Connection established.
16 2910 [MQTT] Notifying task.
17 2911 [OTA] Command sent to MQTT task passed.
18 2912 [OTA] Connected to broker.
19 2913 [OTA Task] Sending command to MQTT task.
20 2913 [MQTT] Received message 20000 from queue.
21 3014 [MQTT] MQTT Subscribe was accepted. Subscribed.
22 3014 [MQTT] Notifying task.
23 3015 [OTA Task] Command sent to MQTT task passed.
24 3015 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/get/
accepted

25 3028 [OTA Task] Sending command to MQTT task.
26 3028 [MQTT] Received message 30000 from queue.
27 3129 [MQTT] MQTT Subscribe was accepted. Subscribed.
28 3129 [MQTT] Notifying task.
29 3130 [OTA Task] Command sent to MQTT task passed.
30 3138 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-next

31 3138 [OTA Task] [OTA] Check For Update #0
32 3138 [OTA Task] Sending command to MQTT task.
33 3138 [MQTT] Received message 40000 from queue.
34 3241 [MQTT] MQTT Publish was successful.
35 3241 [MQTT] Notifying task.
36 3243 [OTA Task] Command sent to MQTT task passed.
37 3245 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
38 3245 [OTA Task] [OTA] Set job doc parameter [ jobId:
fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]
39 3245 [OTA Task] [OTA] Identified job doc parameter [ self_test ]
40 3246 [OTA Task] [OTA] Set job doc parameter [ updatedBy: 589827 ]
41 3246 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]
42 3246 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]
43 3247 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]
44 3247 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]
45 3247 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]
46 3247 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]
47 3249 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:
Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]
48 3249 [OTA Task] [OTA] Job is ready for self test.
```

```
49 3250 [OTA Task] Sending command to MQTT task.  
51 3351 [MQTT] MQTT Publish was successful.  
52 3352 [MQTT] Notifying task.  
53 3352 [OTA Task] Command sent to MQTT task passed.  
54 3353 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/jobs/  
fe18c7ec_8c31_4438_b0b9_ad55acd95610/u55 3353 [OTA Task] Sending command to MQTT task.  
56 3353 [MQTT] Received message 60000 from queue.  
57 3455 [MQTT] MQTT Unsubscribe was successful.  
58 3455 [MQTT] Notifying task.  
59 3456 [OTA Task] Command sent to MQTT task passed.  
60 3456 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPadstreams/327  
  
61 3456 [OTA Task] [OTA] Accepted final image. Commit.  
62 3578 [OTA Task] Sending command to MQTT task.  
63 3578 [MQTT] Received message 70000 from queue.  
64 3779 [MQTT] MQTT Publish was successful.  
65 3780 [MQTT] Notifying task.  
66 3780 [OTA Task] Command sent to MQTT task passed.  
67 3781 [OTA Task] [OTA] Published 'SUCCEEDED' status to $aws/things/TI-LaunchPad/jobs/  
fe18c7ec_8c31_4438_b0b9_ad55acd95610/upd68 3781 [OTA Task] [OTA] Returned buffer to MQTT  
Client.  
69 4251 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0  
70 4381 [OTA Task] [OTA] Missing job parameter: execution  
71 4382 [OTA Task] [OTA] Missing job parameter: jobId  
72 4382 [OTA Task] [OTA] Missing job parameter: jobDocument  
73 4382 [OTA Task] [OTA] Missing job parameter: ts_ota  
74 4382 [OTA Task] [OTA] Missing job parameter: files  
75 4382 [OTA Task] [OTA] Missing job parameter: streamname  
76 4382 [OTA Task] [OTA] Missing job parameter: certfile  
77 4382 [OTA Task] [OTA] Missing job parameter: filepath  
78 4383 [OTA Task] [OTA] Missing job parameter: filesize  
79 4383 [OTA Task] [OTA] Missing job parameter: sig-shal-rsa  
80 4383 [OTA Task] [OTA] Missing job parameter: fileid  
81 4383 [OTA Task] [OTA] Missing job parameter: attr  
82 4383 [OTA Task] [OTA] Returned buffer to MQTT Client.  
83 5251 [OTA] [OTA] Queued: 2 Processed: 2 Dropped: 0
```

Texas Instruments CC3220SF-LAUNCHXL에 대해 Amazon FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

Amazon FreeRTOS 및 OTA 데모 코드를 다운로드하려면

1. AWS IoT 콘솔로 이동하여 탐색 창에서 소프트웨어를 선택합니다.
2. Amazon FreeRTOS 디바이스 소프트웨어에서 구성 다운로드를 선택합니다.
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.

Amazon FreeRTOS 및 OTA 데모 코드가 포함된 zip 파일을 컴퓨터에 저장합니다.

데모 애플리케이션을 빌드하려면

1. .zip 파일의 압축을 풉니다.

2. Amazon FreeRTOS 시작하기 (p. 61)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. <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/ aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
4. 솔루션을 빌드하고 오류 없이 빌드되는지 확인합니다.
5. 터미널 에뮬레이터를 시작하고 다음 설정을 사용하여 보드에 연결합니다.
 - 전송 속도: 115200
 - 데이터 비트: 8
 - 패리티: 없음
 - 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 0 [Tmr Svc] Starting Wi-Fi Module ...
1 0 [Tmr Svc] Simple Link task created
Device came up in Station mode
2 142 [Tmr Svc] Wi-Fi module initialized.
3 142 [Tmr Svc] Starting key provisioning...
4 142 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 340 [Tmr Svc] Write device certificate...
7 433 [Tmr Svc] Key provisioning done...
[WLAN EVENT] STA Connected to the AP: Mobile , BSSID: 24:de:c6:5d:32:a4
[NETAPP EVENT] IP acquired by the device

Device has connected to Mobile
Device IP Address is 192.168.111.12

8 2666 [Tmr Svc] Wi-Fi connected to AP Mobile.
9 2666 [Tmr Svc] IP Address acquired 192.168.111.12
10 2667 [OTA] OTA demo version 0.9.2
11 2667 [OTA] Creating MQTT Client...
12 2667 [OTA] Connecting to broker...
13 3512 [OTA] Connected to broker.
14 3715 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/$next/
get/accepted
15 4018 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/notify-
next
16 4027 [OTA Task] [prvPAL_GetPlatformImageState] xFileInfo.Flags = 0250
17 4027 [OTA Task] [prvPAL_GetPlatformImageState] eOTA_PAL_ImageState_Valid
18 4034 [OTA Task] [OTA_CheckForUpdate] Request #0
19 4248 [OTA] [OTA_AgentInit] Ready.
20 4249 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:OtaGA ]
21 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
22 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
23 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
24 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
25 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
26 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: files
27 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
28 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
29 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
30 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
31 4251 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha1-rsa
32 4251 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
33 4251 [OTA Task] [prvOTA_Close] Context->0x2001b2c4
```

```
34 5248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
35 6248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
36 7248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
37 8248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
38 9248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

Microchip Curiosity PIC32MZEF에 대해 Amazon FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

Amazon FreeRTOS OTA 데모 코드를 다운로드하려면

1. AWS IoT 콘솔로 이동하여 탐색 창에서 소프트웨어를 선택합니다.
2. Amazon FreeRTOS 디바이스 소프트웨어에서 구성 다운로드를 선택합니다.
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. 데모 프로젝트에서 OTA Update(OTA 업데이트)를 선택합니다.
6. 이름 필수 아래에 사용자 지정 Amazon FreeRTOS 소프트웨어 구성에 대한 이름을 입력합니다.
7. 생성 후 다운로드를 선택합니다.

OTA 업데이트 데모 애플리케이션을 빌드하려면

1. 방금 다운로드한 .zip 파일의 압축을 풉니다.
2. [Amazon FreeRTOS 시작하기 \(p. 61\)](#)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. demos/include/aws_ota_codesigner_certificate.h를 업니다.
4. 코드 서명 인증서의 내용을 static const char signingcredentialSIGNING_CERTIFICATE_PEM 변수에 붙여넣습니다.
aws_clientcredential_keys.h와 동일한 형식에 따라 각 줄은 줄 바꿈 문자('\n')로 끝나고 인용 부호로 둑여 있어야 합니다.

예를 들어, 인증서는 다음과 비슷합니다.

```
"-----BEGIN CERTIFICATE-----\n"
"MIIBXTCCAQOgAwIBAgIJJAM4DeybZcTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n"
"FnRlc3Rf621nbmVyQGtYXpbvi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\n"
"MTkxODM2WjAhMR8wHQYDVQBZZ0ZXNpZ251ckBhbWF6b24uY29tMFkwEwYH\n"
"KoZIZj0CAQYIKoZIZj0DAQcDQgAERavZfvwL1X+E4dIF7dbkVMuN4IrJ1CasFkc8\n"
"gzXZpzn683H40XMKltDZPEwr9ng78w9+QYQg7ygnr2stz8yhh06MkMC1wCwYDVROP\n"
"BAQDAgeAMBMGAAUdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIF0R\n"
"r5cb7rEUNtW0vGd05MacrgOABfSoVYvBOK9fp63WAqt5h3BaS123coKSGg84twlq\n"
"TkO/pV/xEmyZmZdV+HxV/OM=\n"
"-----END CERTIFICATE-----\n";
```

5. Python 3 이상을 설치합니다.
6. pip install pyopenssl을 실행하여 pyOpenSSL을 설치합니다.
7. 코드 서명 인증서를 demos/ota/bootloader/utility/codesigner_cert_utility/ 경로에 .pem 형식으로 복사합니다. 인증서 파일의 이름을 aws_ota_codesigner_certificate.pem으로 바꿉니다.
8. <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/ aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
9. 솔루션을 빌드하고 오류 없이 빌드되는지 확인합니다.

10. 터미널 에뮬레이터를 시작하고 다음 설정을 사용하여 보드에 연결합니다.

- 전송 속도: 115200
- 데이터 비트: 8
- 패리티: 없음
- 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] MOTT 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 ESP32에 대해 Amazon FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

- GitHub에서 Amazon FreeRTOS 소스를 다운로드합니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오. 모든 필요한 소스와 라이브러리가 포함된 IDE에서 프로젝트를 생성합니다.
- [Espressif 시작하기](#)의 지침에 따라 필요한 GCC 기반 도구 체인을 설정합니다.

3. <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
4. vendors/espressif/boards/esp32/aws_demos/make 디렉터리에서 make를 실행하여 데모 프로젝트를 빌드합니다. [Espressif 시작하기](#)에 설명된 대로 make flash monitor를 실행하여 데모 프로그램을 플래시하고 출력을 확인할 수 있습니다.
5. OTA 업데이트 데모를 실행하기 전에:
 - <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
 - SHA-256/ECDSA 코드 서명 인증서가 demos/include/aws_ota_codesigner_certificate.h에 복사되는지 확인합니다.

HTTPS 데모 애플리케이션

개요

Amazon FreeRTOS HTTPS 를 사용하는 예제는 iot_demo_https_s3_download_async.c 및 iot_demo_https_s3_download_sync.c에 정의된 HTTPS 클라이언트 데모 애플리케이션을 참조하십시오.

HTTPS 클라이언트 데모는 마리 서명된 URL을 사용하여 Amazon S3에서 파일을 다운로드하는 방법을 보여 줍니다. 파일은 HTTP 부분 콘텐츠 헤더를 사용하여 충분 방식으로 다운로드됩니다. 응답 본문 버퍼 크기의 바이트 범위는 각 충분 요청에서 지정됩니다. HTTPS 클라이언트 라이브러리는 다른 웹 서버에서 파일을 다운로드하는 데도 사용할 수 있는 일반 라이브러리입니다. 일부 HTTP 서버는 바이트 범위를 사용하는 부분 콘텐츠 다운로드를 지원하지 않을 수 있다는 점에 주의하십시오!

HTTPS 클라이언트 데모를 사용하려면 <amazon-freertos>/demos/include/aws_clientcredential_keys.h 파일에서 다음 상수의 값도 설정해야 합니다.

keyCLIENT_CERTIFICATE_PEM

TLS 연결에 필요한 인증서 PEM입니다.

keyCLIENT_PRIVATE_KEY_PEM

TLS 연결에 필요한 프라이빗 키 PEM입니다.

구성 파라미터

이러한 구성 파라미터는 HTTPS 클라이언트 데모에 적용되며 정의해야 합니다.

IOT_DEMO_HTTPS_PRESIGNED_GET_URL

특정 객체에 대해 Amazon S3에 대한 GET 요청의 미리 서명된 URL입니다. 이 항목은 다음 형식이어야 합니다.

```
https://my-bucket.s3.amazonaws.com/object-key.txt?  
AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1560555644&Signature=SomeHash12345UrlABCdEFgfhijk  
%3D
```

동일한 폴더에서 Python 스크립트를 사용하여 미리 서명된 URL을 생성하는 방법에 대한 자침은 <amazon-freertos>/demos/https/README.md 단원을 참조하십시오.

이러한 구성 파라미터는 HTTPS 클라이언트 데모에 적용되며 정의할 필요가 없습니다. 이러한 구성 파라미터는 추가 사용자 지정 옵션입니다.

IOT_DEMO_HTTPS_PORT

연결할 HTTPS 서버 TCP 포트입니다. 기본값은 443입니다.

IOT_DEMO_HTTPS_TRUSTED_ROOT

HTTPS 서버에 연결할 신뢰할 수 있는 ROOT CA입니다. HTTPS 서버는

IOT_DEMO_HTTPS_PRESIGNED_GET_URL에서 호스트 이름에 따라 정의됩니다. 기본값은 Baltimore Cybertrust 루트 인증 기관입니다.

기타 구성은 [HTTPS 클라이언트 API 참조 데모 구성](#)을 참조하십시오.

사용량 지침

자세한 내용과 출력 예제는 [HTTPS 클라이언트 API 참조 데모 사용 지침](#)을 참조하십시오.

AWS IoT 디바이스 샘플 디바이스 데모 애플리케이션

Amazon FreeRTOS에는 프로그래밍 방식으로 AWS IoT 디바이스 샘플 디바이스의 변경 사항에 대응하고 업데이트하는 방법을 보여주는 데모 애플리케이션이 포함되어 있습니다. 이 데모 애플리케이션은 Amazon FreeRTOS/demos/shadow/aws_iot_demo_shadow.c에 정의되어 있습니다. 이 시나리오에서 디바이스는 빨간색 또는 녹색으로 색상을 설정할 수 있는 전구입니다.

데모 애플리케이션은 다음 세 작업을 생성합니다.

- prvShadowMainTask를 호출하는 기본 데모 작업
- prvUpdateTask를 호출하는 디바이스 업데이트 작업
- prvShadowUpdateTasks를 호출하는 샘플 디바이스 업데이트 작업의 수

prvShadowMainTask는 디바이스 샘플 디바이스 클라이언트를 시작하고 demos/include/aws_clientcredential.h에 지정된 클라이언트 자격 증명을 사용하여 AWS IoT에 대한 MQTT 연결을 시작합니다. 애플리케이션은 성공적으로 AWS 클라우드에 연결하려면 디바이스의 AWS IoT 사물 이름, MQTT 브로커 엔드포인트 및 포트 등 aws_clientcredential.h에 지정된 정보가 정확해야 합니다.

MQTT 연결이 설정되면 애플리케이션이 디바이스 업데이트 작업을 생성합니다. 마지막으로 샘플 디바이스 업데이트 작업을 생성한 후 종료합니다. aws_iot_demo_shadow.c에 정의된 democonfigSHADOW_DEMO_NUM_TASKS 상수는 생성되는 샘플 디바이스 업데이트 작업의 수를 제어합니다.

prvShadowUpdateTasks는 첫 번째 사물 샘플 디바이스를 생성하고 그 문서로 샘플 디바이스를 업데이트합니다. 그런 다음, 원하는 사물 샘플 디바이스 상태를 주기적으로 업데이트하는 무한 루프를 실행하고 전구의 색 변경(빨간색과 녹색)을 요청합니다.

prvUpdateTask는 원하는 샘플 디바이스 상태가 달라졌을 때 대응합니다. 원하는 상태가 변경되면 이 작업은 원하는 샘플 디바이스 상태를 반영하도록 보고된 샘플 디바이스 상태를 업데이트합니다.

디바이스 샘플 디바이스 데모를 실행하려면 디바이스가 AWS 클라우드와 통신할 수 있도록 [첫 번째 단계 \(p. 62\)](#) 시작하기를 완료하여 AWS IoT 및 Amazon FreeRTOS를 설정해야 합니다.

AWS IoT 및 Amazon FreeRTOS를 설정한 후 다음을 수행합니다.

1. 다음 정책을 디바이스 인증서에 추가합니다.

```
{  
    "Version": "2012-10-17",
```

```
"Statement": [
    {
        "Effect": "Allow",
        "Action": "iot:Connect",
        "Resource": "arn:aws:iot:us-west-2:123456789012:client/<yourClientId>"
    },
    {
        "Effect": "Allow",
        "Action": "iot:Subscribe",
        "Resource": "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/things/thingName/shadow/*"
    },
    {
        "Effect": "Allow",
        "Action": "iot:Receive",
        "Resource":
            "arn:aws:iot:us-west-2:123456789012:topic/$aws/things/thingName/shadow/*"
    },
    {
        "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource":
            "arn:aws:iot:us-west-2:123456789012:topic/$aws/things/thingName/shadow/*"
    }
]
```

2. <[amazon-freertos](#)>/vendors/<[vendor](#)>/boards/<[board](#)>/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_SHADOW_DEMO_ENABLED를 정의합니다.
3. Amazon FreeRTOS를 빌드 및 플래시하고 디바이스에 실행합니다.

AWS IoT 콘솔에서 디바이스의 새도우를 보고, 원하는 상태와 보고되는 상태가 주기적으로 달라지는지 확인할 수 있습니다.

1. AWS IoT 콘솔의 왼쪽 탐색 창에서 관리를 선택합니다.
2. 관리에서 사물을 선택한 후 새도우를 보려는 사물을 선택합니다.
3. 사물 세부 정보 페이지의 왼쪽 탐색 창에서 새도우를 선택하여 사물 새도우를 표시합니다.

디바이스와 새도우의 상호 작용 방식에 대한 자세한 내용은 [디바이스 새도우 서비스 데이터 흐름](#)을 참조하십시오.

보안 소켓 에코 클라이언트 데모

다음 예제에서는 단일 RTOS 작업을 사용합니다. 이 예제의 소스 코드는 demos/tcp/aws_tcp_echo_client_single_task.c에서 찾을 수 있습니다.

시작하기 전에 마이크로 컨트롤러에 Amazon FreeRTOS를 다운로드하고 Amazon FreeRTOS 데모 프로젝트를 빌드 및 실행했는지 확인합니다. [GitHub](#)에서 Amazon FreeRTOS를 복제하거나 다운로드할 수 있습니다. 자세한 내용은 [README.md](#) 파일을 참조하십시오. Amazon FreeRTOS 인증 보드 설정에 대한 자세한 내용은 [Amazon FreeRTOS 시작하기](#)를 참조하십시오.

데모를 실행하려면

Note

TCP 서버와 클라이언트 데모는 현재 Cypress CYW943907AEVAL1F 및 CYW954907AEVAL1F 개발 키트에서 지원되지 않습니다.

1. Amazon FreeRTOS 이식 안내서에 있는 [TLS 에코 서버 설정](#)의 지침을 따르십시오.

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.h에 복사합니다.
3. FreeRTOSConfig.h 파일을 엽니다.
4. configECHO_SERVER_ADDR0, configECHO_SERVER_ADDR1, configECHO_SERVER_ADDR2, configECHO_SERVER_ADDR3 변수를 TLS Echo Server가 실행되는 IP 주소를 구성하는 정수 네 개로 설정합니다.
5. configTCP_ECHO_CLIENT_PORT 변수를 TLS Echo Server가 수신 대기하는 포트인 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. <amazon-freertos>/vendors/<vendor>/boards/<board>/aws_demos/config_files/ aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_TCP_ECHO_CLIENT_DEMO_ENABLED를 정의합니다.

マイクロ컨트롤러와 TLS Echo Server는 동일한 네트워크에 있어야 합니다. 데모가 (main.c)를 시작하면 Received correct string from echo server라는 로그 메시지가 표시됩니다.

Amazon FreeRTOS용 AWS IoT 디바이스 테스터 사용

Amazon FreeRTOS용 AWS IoT Device Tester(IDT)를 사용하면 해당 디바이스에서 Amazon FreeRTOS 운영 체제가 로컬로 작동하고 AWS IoT 클라우드와 통신할 수 있는지 확인할 수 있습니다. 특히, Amazon FreeRTOS 라이브러리의 이식 계층 인터페이스가 올바르게 구현되었는지 확인합니다. 또한 이 IDT는 AWS IoT Core와의 엔드 투 엔드 테스트를 수행합니다. 예를 들어 보드에서 MQTT 메시지를 송수신하고 올바르게 처리할 수 있는지 확인합니다. IDT에서 Amazon FreeRTOS에 대해 실행하는 테스트는 [Amazon FreeRTOS GitHub 리포지토리](#)에서 정의됩니다.

테스트는 내장형 애플리케이션으로 실행되어 보드에 플래시됩니다. 애플리케이션 바이너리 이미지에는 Amazon FreeRTOS, 반도체 공급업체의 이식된 Amazon FreeRTOS 인터페이스, 보드 디바이스 드라이버가 포함되어 있습니다. 테스트의 용도는 이식된 Amazon FreeRTOS 인터페이스가 디바이스 드라이버 위에서 올바르게 작동하는지 확인하는 것입니다.

Amazon FreeRTOS용 IDT는 AWS Partner Device Catalog에 하드웨어를 추가하기 위해 AWS IoT에 제출할 수 있는 테스트 보고서를 생성합니다. 자세한 내용은 [AWS Device Qualification Program](#)을 참조하십시오.

Amazon 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.

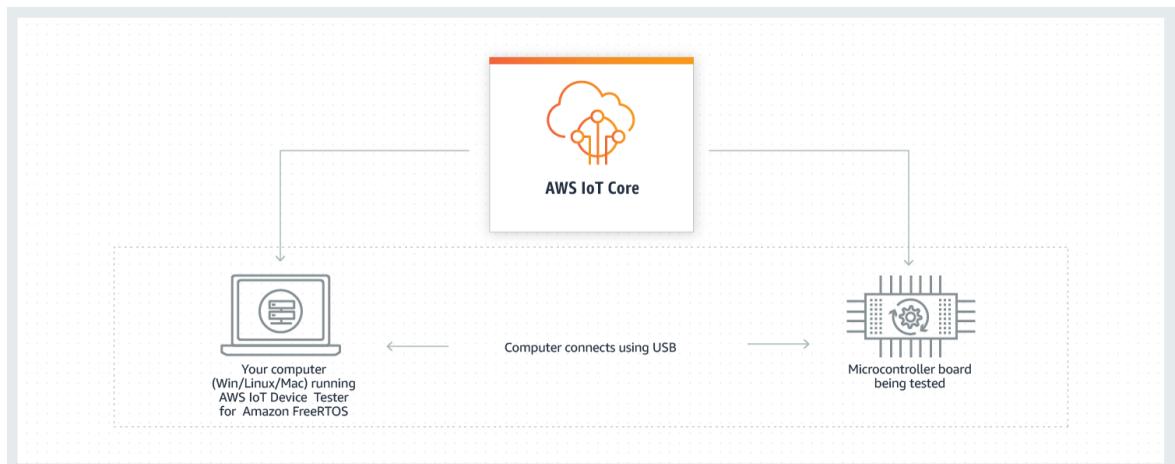
Amazon FreeRTOS용 IDT는 디바이스 테스트 외에 검증 프로세스를 쉽게 진행하기 위한 리소스(예: AWS IoT 사물, Amazon FreeRTOS 그룹, Lambda 함수 등)를 생성합니다.

이러한 리소스를 생성하기 위해 Amazon FreeRTOS용 IDT는 config.json에서 구성된 AWS 자격 증명을 사용하여 사용자를 대신해서 API를 호출합니다. 이러한 리소스는 테스트 중 다양한 시점에서 프로비저닝됩니다.

Amazon FreeRTOS용 IDT를 호스트 컴퓨터에서 실행하면 IDT에서 다음 단계를 수행합니다.

1. 디바이스 및 자격 증명 구성 파일을 로드하고 검증합니다.
2. 필수 로컬 및 클라우드 리소스를 사용하여 선택한 테스트를 수행합니다.
3. 로컬 및 클라우드 리소스를 정리합니다.
4. 보드에서 검증에 필요한 테스트를 통과했는지를 나타내는 테스트 보고서를 생성합니다.

다음 다이어그램은 테스트 인프라 설정을 보여 줍니다.



Amazon FreeRTOS 201912.00용 IDT v1.6.0 다운로드

다음 링크를 사용하여 최신 버전의 Amazon FreeRTOS용 IDT를 다운로드합니다. 소프트웨어를 다운로드하면 Amazon FreeRTOS용 IDT 라이선스 계약에 동의하는 것입니다. Amazon FreeRTOS용 IDT의 각 버전에는 Amazon FreeRTOS의 해당 버전이 하나 이상 있습니다.

Amazon FreeRTOS 201912.00용 IDT v1.6.0

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

릴리스 정보

- Amazon FreeRTOS 201912.00을 지원합니다.
- HTTPS를 통한 OTA에 대한 선택적 테스트를 지원하여 Amazon FreeRTOS 개발 보드를 검증합니다.
- 테스트 시 AWS IoT ATS 앤드포인트를 지원합니다.
- 테스트 제품군 시작 전에 사용자에게 최신 IDT 버전을 알리는 기능을 지원합니다.

이전 버전의 Amazon FreeRTOS를 테스트해야 하는 경우 [이전 Amazon FreeRTOS용 IDT 버전 \(p. 267\)](#) 단원을 참조하십시오.

이전 Amazon FreeRTOS용 IDT 버전

이 단원에는 Amazon FreeRTOS용 IDT의 이전 버전에 대한 다운로드 링크가 포함되어 있습니다.

Amazon FreeRTOS 201910.00용 IDT v1.5.1

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

릴리스 정보

- 보안 요소(온보드 키)가 있는 Amazon 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.

Amazon FreeRTOS 201908.00용 IDT v1.4.0

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

릴리스 정보

- 새 PKCS11 라이브러리와 테스트 사례 업데이트에 대한 지원을 추가했습니다.
- 실행 가능한 오류 코드를 도입했습니다. 자세한 내용은 [IDT 오류 코드 \(p. 286\)](#) 단원을 참조하십시오.
- IDT를 실행하는 데 사용된 IAM 정책이 업데이트되었습니다. 최신 정책에 대한 업데이트는 [권한 정책 템플릿 \(p. 270\)](#)에서 사용할 수 있습니다.

Amazon FreeRTOS 201906.00 메이저용 IDT v1.3.1

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

릴리스 정보

- Bluetooth Low Energy(BLE) 테스트에 대한 지원이 추가되었습니다.
- IDT CLI 명령에 대한 사용자 경험이 개선되었습니다.
- IDT를 실행하는 데 사용된 IAM 정책이 업데이트되었습니다. 최신 정책에 대한 업데이트는 [권한 정책 템플릿 \(p. 270\)](#)에서 사용할 수 있습니다.

Amazon FreeRTOS v1.4.8 및 v1.4.9용 IDT

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

릴리스 정보

- Amazon FreeRTOS v1.4.8 및 v1.4.9에 대한 지원이 추가되었습니다.
- CMAKE 빌드 시스템으로 Amazon FreeRTOS 디바이스를 테스트할 수 있도록 지원이 추가됨

Amazon FreeRTOS v1.4.7용 IDT v1.1

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

Amazon FreeRTOS v1.4.6용 IDT v1.0

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

Amazon FreeRTOS v1.4.5용 IDT v1.0

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

Amazon FreeRTOS v1.4.4용 IDT v1.0

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

Amazon FreeRTOS v1.4.3용 IDT v1.0

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

Amazon FreeRTOS v1.4.2용 IDT v1.0

- Amazon FreeRTOS용 IDT: [Linux](#)
- Amazon FreeRTOS용 IDT: [macOS](#)
- Amazon FreeRTOS용 IDT: [Windows](#)

사전 조건

이 단원에서는 AWS IoT 디바이스 테스트를 사용하여 마이크로 컨트롤러를 테스트하기 위한 사전 조건을 설명합니다.

Amazon FreeRTOS 다운로드

테스트하려는 Amazon FreeRTOS 버전을 [GitHub](#)에서 다운로드할 수 있습니다. Windows의 경우 260자 의 경로 길이 제한이 있습니다. Amazon FreeRTOS의 경로 구조는 여러 수준으로 깊게 되어 있기 때문에 Windows를 사용하는 경우 파일 경로를 260자 제한 미만으로 유지하십시오. 예를 들어 Amazon FreeRTOS 를 C:\Users\username\programs\projects\AmazonFreeRTOS\ 대신 C:\AFreeRTOS로 복제하십시오.

Amazon FreeRTOS용 IDT 다운로드

Amazon FreeRTOS의 모든 버전에는 검증 테스트를 수행하기 위한 해당 버전의 Amazon FreeRTOS용 IDT 가 있습니다. [Amazon FreeRTOS 201912.00용 IDT v1.6.0 다운로드 \(p. 267\)](#)에서 적절한 버전의 Amazon FreeRTOS용 IDT를 다운로드합니다.

읽기 및 쓰기 권한이 있는 파일 시스템의 위치에 Amazon FreeRTOS용 IDT의 압축을 풉니다. 경로 길이 제한 때문에 Microsoft Windows에서는 c:\ 또는 d:\와 같은 루트 디렉터리에 Amazon FreeRTOS용 IDT의 압축 을 풁니다.

AWS 계정 생성 및 구성

다음 단계에 따라 테스트를 실행하는 동안 사용자를 대신하여 리소스에 액세스할 수 있는 Amazon FreeRTOS 권한을 IDT에 부여하는 IAM 정책, AWS 계정 및 IAM 사용자를 생성하고 구성합니다.

1. [AWS 계정을 생성합니다.](#)
2. 테스트를 실행하고 IDT 사용량 데이터를 수집하는데 필요한 권한을 Amazon FreeRTOS용 IDT에 부여 하는 [IAM 정책을 생성합니다.](#) [권한 정책 템플릿 \(p. 270\)](#)에서 설명하는 정책 템플릿을 사용합니다.

3. AWS 계정에서 IAM 사용자를 생성하고 방금 생성한 정책을 연결합니다.

권한 정책 템플릿

다음은 Amazon FreeRTOS용 IDT에서 테스트를 실행하는 데 필요한 권한을 부여하는 정책 템플릿입니다.

Important

다음 정책 템플릿은 역할을 생성하고, 정책을 생성하고, 역할에 정책을 연결할 수 있는 권한을 부여합니다. Amazon FreeRTOS용 IDT는 역할을 생성하는 데 필요한 테스트에 이러한 권한을 사용합니다. 정책 템플릿은 사용자에게 관리자 권한을 제공하지 않지만, AWS 계정에 대한 관리자 액세스 권한을 얻기 위해 이러한 권한을 잠재적으로 사용할 수 있습니다.

```
    "arn:aws:iot::*:cert/*",
    "arn:aws:iot::*:job/*",
    "arn:aws:iot::*:stream/*",
    "arn:aws:iam::*:role/idt*",
    "arn:aws:s3:::afr-ota/*",
    "arn:aws:s3:::idt/*",
    "arn:aws:iam:::role/idt*"
]
},
{
    "Sid": "VisualEditor2",
    "Effect": "Allow",
    "Action": [
        "iot:DetachThingPrincipal",
        "iot:AttachThingPrincipal",
        "s3>ListBucketVersions",
        "iot>CreatePolicy",
        "iam>ListRoles",
        "freertos>ListHardwarePlatforms",
        "signer>DescribeSigningJob",
        "s3>ListBucket",
        "signer:*",
        "iot>DescribeEndpoint",
        "iot>CreateStream",
        "signer>StartSigningJob",
        "s3>ListAllMyBuckets",
        "signer>ListSigningJobs",
        "acm>GetCertificate",
        "acm>ListCertificates",
        "acm>ImportCertificate",
        "freertos>DescribeHardwarePlatform",
        "iot>CreateKeysAndCertificate",
        "iot>CreateCertificateFromCsr",
        "s3>GetBucketLocation",
        "iot>GetRegistrationCode",
        "iot>RegisterCACertificate",
        "iot>RegisterCertificate",
        "iot>UpdateCACertificate",
        "iot>DeleteCACertificate",
        "iot>DeleteCertificate",
        "iot>UpdateCertificate"
],
    "Resource": "*"
},
{
    "Sid": "VisualEditor3",
    "Effect": "Allow",
    "Action": [
        "s3>PutObject",
        "s3>GetObject"
],
    "Resource": [
        "arn:aws:s3:::afr/*",
        "arn:aws:s3:::idt/*"
    ]
},
{
    "Sid": "VisualEditor4",
    "Effect": "Allow",
    "Action": [
        "iot>CreateOTAUpdate",
        "iot>CreateThing",
        "iot>DeleteThing"
],
    "Resource": "*"
},
```

```
{  
    "Sid": "VisualEditor5",  
    "Effect": "Allow",  
    "Action": [  
        "execute-api:Invoke"  
    ],  
    "Resource": [  
        "arn:aws:execute-api:*:098862408343:/*"  
    ]  
}  
}
```

(선택 사항) AWS 명령줄 인터페이스(CLI) 설치

몇 가지 작업은 CLI를 사용하여 수행할 수도 있습니다. CLI가 설치되지 않은 경우 [AWS CLI 설치](#)의 지침을 따릅니다.

명령줄에서 `aws configure`를 실행하여 사용할 AWS 리전에 대해 CLI를 구성합니다. Amazon FreeRTOS용 IDT를 지원하는 AWS 리전에 대한 자세한 내용은 [AWS 리전 및 엔드포인트](#)를 참조하십시오.

처음으로 마이크로 컨트롤러 보드의 테스트 준비

Amazon FreeRTOS 인터페이스를 이식할 때 Amazon FreeRTOS용 IDT를 사용하여 테스트 할 수 있습니다. 보드의 디바이스 드라이버에 대한 Amazon FreeRTOS 인터페이스를 이식한 후 AWS IoT Device Tester를 사용하여 마이크로 컨트롤러 보드에 대한 검증 테스트를 실행합니다.

라이브러리 이식 계층 추가

Amazon FreeRTOS를 디바이스에 이식하려면 [Amazon FreeRTOS 이식 안내서](#)의 지침을 따르십시오.

AWS 자격 증명 구성

Device Tester가 AWS 클라우드와 통신하도록 AWS 자격 증명을 구성해야 합니다. 자세한 내용은 [개발을 위한 AWS 자격 증명 및 리전 설정](#)을 참조하십시오. 유효한 AWS 자격 증명을 `<devicetester_extract_location> /devicetester_afreertos_[win/mac/linux]/configs/config.json` 구성 파일에 지정해야 합니다.

Amazon FreeRTOS용 IDT에서 디바이스 폴 만들기

테스트할 디바이스는 디바이스 폴에서 구성됩니다. 각 디바이스 폴은 하나 이상의 동일한 디바이스로 구성됩니다. 폴에서 단일 디바이스를 테스트하거나 폴에서 여러 디바이스를 테스트하도록 Amazon FreeRTOS용 IDT를 구성할 수 있습니다. 검증 프로세스를 가속화하기 위해 Amazon FreeRTOS용 IDT는 동일한 사양이 있는 디바이스를 별별로 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 별별 빌드를 활성화하기 위해 Amazon FreeRTOS용 IDT는 Amazon FreeRTOS용 IDT에서 앱죽이 폴린 폴더 내부의 결과 폴더에 소스 코드를 복사합니다. 빌드 또는 플래시 명령

에 있는 소스 코드 경로는 `testdata.sourcePath` 변수를 사용하여 참조해야 합니다. Amazon FreeRTOS 용 IDT는 이 변수를 복사된 소스 코드의 임시 경로로 바꿉니다. 자세한 내용은 [Amazon FreeRTOS용 IDT 변수 \(p. 279\)](#) 단원을 참조하십시오.

다음은 여러 디바이스가 있는 디바이스 풀을 생성하는 데 사용되는 예제 `device.json` 파일입니다.

```
[{
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
        {
            "name": "WIFI",
            "value": "Yes | No"
        },
        {
            "name": "OTA",
            "value": "Yes | No",
            "configs": [
                {
                    "name": "OTADataPlaneProtocol",
                    "value": "HTTP | MQTT | Both"
                }
            ]
        },
        {
            "name": "TCP/IP",
            "value": "On-chip | Offloaded | No"
        },
        {
            "name": "TLS",
            "value": "On-chip | Offloaded | No"
        },
        {
            "name": "BLE",
            "value": "Yes | No"
        }
    ],
    "devices": [
        {
            "id": "<device-id1>",
            "connectivity": {
                "protocol": "uart",
                "serialPort": "<computer_serial_port_1>"
            },
            "secureElementConfig": {
                "publicKeyAsciiHexFilePath": "<absolute-path-to-public-key>",
                "secureElementSerialNumber": "<optional: serialnumber-of-secure-element>"
            },
            "identifiers": [
                {
                    "name": "serialNo",
                    "value": "<serialNo-value>"
                }
            ]
        },
        {
            "id": "<device-id2>",
            "connectivity": {
                "protocol": "uart",
                "serialPort": "<computer_serial_port_2>"
            },
            "secureElementConfig": {
                "publicKeyAsciiHexFilePath": "<absolute-path-to-public-key>",
                "secureElementSerialNumber": "<optional: serialnumber-of-secure-element>"
            },
            "identifiers": [
                {
                    "name": "serialNo",
                    "value": "<serialNo-value>"
                }
            ]
        }
    ]
}]
```

```
    }]  
}]
```

device.json 파일에서 사용되는 속성은 다음과 같습니다.

id

디바이스 풀을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 디바이스는 동일한 유형이어야 합니다. 테스트 제품군을 실행할 때 풀에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다.

sku

테스트하는 보드를 고유하게 식별하는 영숫자 값입니다. SKU는 정규화된 보드를 추적하는 데 사용됩니다.

Note

AWS Partner Device Catalog에 보드를 등록하려면 여기서 지정하는 SKU가 목록 등록 프로세스에 사용하는 SKU와 일치해야 합니다.

features

디바이스의 지원되는 기능이 포함된 배열입니다. 디바이스 테스터는 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.

TLS

보드가 TLS를 지원하는지 여부를 나타냅니다. 검증에는 TLS가 필요합니다.

PKCS11

보드가 지원하는 퍼블릭 키 암호화 알고리즘을 나타냅니다. 검증에는 PKCS11이 필요합니다. 지원되는 값은 ECC, RSA, Both 및 No입니다.

KeyProvisioning

신뢰할 수 있는 X.509 클라이언트 인증서를 보드에 작성하는 방법을 나타냅니다. 유효 값은 Import, Onboard 및 No입니다. 검증에는 키 프로비저닝이 필요합니다.

보드가 프라이빗 키 가져오기를 지원하는 경우 Import를 사용합니다. 보드가 온보드 프라이빗 키 생성을 지원하는 경우 Onboard를 사용합니다. 보드가 키 프로비저닝을 지원하지 않는 경우 No를 사용합니다.

Onboard를 사용하는 경우 각 device 섹션에 secureElementConfig 요소를 추가하고 퍼블릭 키 파일에 대한 절대 경로를 publicKeyAsciiHexFilePath 필드에 입력합니다.

OTA

보드가 무선(Over-the-Air) 업데이트 기능을 지원하는지 여부를 나타냅니다.

OtaDataPlaneProtocol 속성은 디바이스가 지원하는 OTA 데이터 영역 프로토콜을 나타냅니다. 디바이스에서 OTA 기능을 지원하지 않으면 이 속성은 무시됩니다. "Both"를 선택하면 MQTT, HTTP 둘 다와 혼합된 테스트가 실행되어 OTA 테스트 실행 시간이 길어집니다.

BLE

보드가 Bluetooth Low Energy(BLE)를 지원하는지 여부를 나타냅니다.

devices.id

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

devices.connectivity.protocol

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 지원되는 값: uart.

devices.connectivity.serialPort

테스트할 디바이스에 연결하는 데 사용되는 호스트 컴퓨터의 직렬 포트입니다.

devices.secureElementConfig.PublicKeyAsciiHexFilePath

온보드 프라이빗 키에서 추출한 16진수 바이트 퍼블릭 키를 포함하는 파일에 대한 절대 경로입니다.

devices.secureElementConfig.SecureElementSerialNumber

선택 사항입니다. 보안 요소의 일련 번호입니다. Amazon FreeRTOS 데모/테스트 프로젝트를 실행할 때 디바이스 퍼블릭 키와 더불어 일련 번호가 출력되는 경우 이 필드를 제공합니다.

identifiers

선택. 임의 이름-값 페어의 배열입니다. 다음 단원에서 설명하는 빌드 및 플래시 명령에 이러한 값을 사용할 수 있습니다.

빌드, 플래시 및 테스트 설정 구성

Amazon FreeRTOS용 IDT를 사용하여 보드에서 자동으로 테스트를 빌드 및 플래시하는 경우, 하드웨어에 대한 빌드 및 플래시 명령을 실행하도록 IDT를 구성해야 합니다. 빌드 및 플래시 명령 설정은 config 폴더에 있는 `userdata.json` 템플릿 파일에서 구성됩니다.

디바이스 테스트를 위한 설정 구성

빌드, 플래시 및 테스트 설정은 `userdata.json` 파일에서 수행됩니다. 다음 JSON 예제에서는 여러 디바이스를 테스트하기 위해 Amazon FreeRTOS용 IDT를 구성하는 방법을 보여 줍니다.

```
{  
    "sourcePath": "<absolute-path-to/amazon-freertos>",  
    "buildTool": {  
        "name": "<your-build-tool-name>",  
        "version": "<your-build-tool-version>",  
        "command": [  
            "<absolute-path-to/build-parallel-script> {{testData.sourcePath}}"  
        ],  
        "enableTests": [  
            "buildImageInfo" : {  
                "testsImageName": "<tests-image-file-name>",  
                "demosImageName": "<demos-image-file-name>"  
            },  
            "flashTool": {  
                "name": "<your-flash-tool-name>",  
                "version": "<your-flash-tool-version>",  
                "command": [  
                    "<absolute-path-to/flash-parallel-script> {{testData.sourcePath}}"  
                ]  
            },  
            "otaTool": {  
                "name": "<your-ota-tool-name>",  
                "version": "<your-ota-tool-version>",  
                "command": [  
                    "<absolute-path-to/ota-parallel-script> {{testData.sourcePath}}"  
                ]  
            }  
        ]  
    }  
}
```

```

    "clientWifiConfig": {
        "wifiSSID": "<ssid>",
        "wifiPassword": "<password>",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2"
    },
    "testWifiConfig": {
        "wifiSSID": "<ssid>",
        "wifiPassword": "<password>",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2"
    },
    "echoServerConfiguration": {
        "securePortForSecureSocket": 33333,
        "insecurePortForSecureSocket": 33334,
        "insecurePortForWifi": 33335
    },
    "otaConfiguration": {
        "otaFirmwareFilePath": "{ testData.sourcePath }<relative-path-to/ota-image-generated-in-build-process>",
        "deviceFirmwareFileName": "<absolute-path-to/ota-image-on-device>",
        "awsSignerHashingAlgorithm": "SHA1 | SHA256",
        "awsSignerSigningAlgorithm": "RSA | ECDSA",
        "awsSignerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
        "awsSignerCertificateArn": "arn:aws:acm:<region>:<account-id>.certificate:<certificate-id>",
        "awsUntrustedSignerCertificateArn": "arn:aws:acm:<region>:<account-id>.certificate:<certificate-id>",
        "awsSignerCertificateFileName": "<awsSignerCertificate-file-name>",
        "compileCodesignerCertificate": true | false,
        "otaDemoConfigFilePath": "<full path to aws_demo_config.h>"
    },
    "cmakeConfiguration": {
        "boardName": "<board-name>",
        "vendorName": "<vendor-name>",
        "compilerName": "<compiler-name>",
        "afrToolchainPath": "</path/to/afr/toolchain>",
        "cmakeToolchainPath": "</path/to/cmake/toolchain>"
    }
}
}

```

다음 목록에서는 `userdata.json`에서 사용되는 속성을 나열합니다.

`sourcePath`

이식된 Amazon FreeRTOS 소스 코드의 루트에 대한 경로입니다.

`vendorPath`

공급업체별 Amazon FreeRTOS 코드에 대한 경로입니다. 직렬 테스트의 경우 `vendorPath`를 절대 경로로 설정할 수 있습니다. 예:

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

병렬 테스트의 경우 `vendorPath`을 `{ testData.sourcePath }` 자리 표시자를 사용하여 설정할 수 있습니다. 예:

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

Note

테스트를 병렬로 실행하는 경우 vendorPath, buildTool, flashTool 필드에 {{testdata.sourcePath}} 자리 표시자를 사용해야 합니다. 단일 디바이스로 테스트를 실행하는 경우 vendorPath, buildTool, flashTool 필드에 절대 경로를 사용해야 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 디바이스 테스터 변수 {{testdata.sourcePath}}로 바꿔야 합니다.

- buildImageInfo

- testsImageName: <afr-source>/tests 폴더에서 테스트를 빌드할 때 빌드 명령으로 인해 생성되는 파일의 이름입니다.
- demosImageName: <afr-source>/demos 폴더에서 테스트를 빌드할 때 빌드 명령으로 인해 생성되는 파일의 이름입니다.

buildTool.buildImageInfo.testsImageName

<afr-source-code>/tests 폴더에서 테스트를 빌드할 때 빌드 명령에 따라 출력되는 파일의 이름입니다.

buildTool.buildImageInfo.demosImageName

<afr-source-code>/demos 폴더에서 데모를 빌드할 때 빌드 명령에 따라 출력되는 파일의 이름입니다.

flashTool

디바이스에 대한 플래시 명령이 포함된 플래시 스크립트(.sh 또는 .bat)의 전체 경로입니다. 플래시 명령에 있는 소스 코드 경로에 대한 모든 참조는 Amazon FreeRTOS용 IDT 변수 {{testdata.sourcePath}}로 바꿔야 합니다.

clientWifiConfig

클라이언트 Wi-Fi 구성입니다. Wi-Fi 라이브러리 테스트를 수행하려면 MCU 보드를 두 개의 액세스 포인트에 연결해야 합니다. 이 속성은 첫 번째 액세스 포인트에 대한 Wi-Fi 설정을 구성합니다. 클라이언트 Wi-Fi 설정은 <amazon-freertos>/test/include/aws_clientcredential.h에서 구성됩니다. 다음 매크로는 aws_clientcredential.h에 있는 값을 사용하여 설정됩니다. 일부 Wi-Fi 테스트 사례에서는 액세스 포인트가 보안을 갖추고 있고 열리지 않아야 합니다.

- wifi_ssid: Wi-Fi SSID입니다.
- wifi_password: Wi-Fi 암호입니다.
- wifiSecurityType: 사용되는 Wi-Fi 보안의 유형입니다.

testWifiConfig

Wi-Fi 구성을 테스트합니다. Wi-Fi 라이브러리 테스트를 수행하려면 보드를 두 개의 액세스 포인트에 연결해야 합니다. 이 속성은 두 번째 액세스 포인트를 구성합니다. 테스트 Wi-Fi 설정은 <amazon-freertos>/.../wifi/test/aws_test_wifi.h에서 구성됩니다. 다음 매크로는 aws_test_wifi.c에 있는 값을 사용하여 설정됩니다. 일부 Wi-Fi 테스트 사례에서는 액세스 포인트가 보안을 갖추고 있고 열리지 않아야 합니다.

Note

보드가 Wi-Fi를 지원하지 않는 경우에도 clientWifiConfig 및 testWifiConfig 섹션을 device.json 파일에 포함시켜야 하지만 이러한 속성의 값을 생략할 수 있습니다.

- testwifiWIFI_SSID: Wi-Fi SSID입니다.
- testwifiWIFI_PASSWORD: Wi-Fi 암호입니다.
- testwifiWIFI_SECURITY: 사용되는 Wi-Fi 보안의 유형입니다. 다음 값 중 하나입니다.
 - eWiFiSecurityOpen
 - eWiFiSecurityWEP

- eWiFiSecurityWPA
- eWiFiSecurityWPA2

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.

awsSignerHashingAlgorithm

디바이스에서 지원되는 해싱 알고리즘입니다. 가능한 값은 SHA1 또는 SHA256입니다.

awsSignerSigningAlgorithm

디바이스에서 지원되는 서명 알고리즘입니다. 가능한 값은 RSA 또는 ECDSA입니다.

awsSignerPlatform

AWS Code Signer에서 OTA 업데이트 작업을 생성하는 동안 사용되는 서명 및 해싱 알고리즘입니다. 현재 이 필드에 가능한 값은 AmazonFreeRTOS-TI-CC3220SF 및 AmazonFreeRTOS-Default입니다. SHA1 및 RSA인 경우 AmazonFreeRTOS-Default를 사용하고 SHA256 및 ECDSA인 경우 AmazonFreeRTOS-TI-CC3220SF를 선택합니다. 구성에 SHA256 | RSA 또는 SHA1 | ECDSA가 필요한 경우 AWS에 문의하여 추가 지원을 요청하십시오.

awsSignerCertificateArn

AWS Certificate Manager(ACM)에 업로드된 신뢰할 수 있는 인증서의 Amazon 리소스 이름(ARN)입니다. 신뢰할 수 있는 인증서 생성에 대한 자세한 내용은 [코드 서명 인증서 생성](#)을 참조하십시오.

awsUntrustedSignerCertificateArn

ACM에 업로드된 코드 서명 인증서의 ARN입니다.

compileCodesignerCertificate

코드 서명자 서명 확인 인증서를 프로비저닝하거나 플래시하지 않는 경우 true로 설정합니다. 따라서 이 항목을 프로젝트로 컴파일해야 합니다. AWS IoT Device Tester는 신뢰할 수 있는 인증서를 ACM에서 가져오고 aws_codesigner_certificate.h로 컴파일합니다.

otaDemoConfigFilePath

aws_demo_config.h에 대한 전체 경로이며 <afr-source>/vendors/vendor/boards/
board/ aws_demos/config_files/ 내에 있습니다. 이러한 파일은 Amazon FreeRTOS에서 제
공한 이식 코드 템플릿에 포함되어 있습니다.

cmakeConfiguration

CMake 구성[선택 사항]

boardName

테스트 중인 보드의 이름입니다. 보드 이름은 *path/to/afr/source/code/vendors/<vendor>/
boards/<board>*의 폴더 이름과 같아야 합니다.

vendorName

테스트 중인 보드의 공급업체 이름입니다. 공급업체 이름은 *path/to/afr/source/code/
vendors/<vendor>* 아래의 폴더 이름과 같아야 합니다.

compilerName

컴파일러 이름입니다.

afrToolchainPath

컴파일러 도구 체인의 정규화된 경로입니다.

cmakeToolchainPath

CMake 도구 체인의 정규화된 경로입니다. 이 필드는 선택 사항입니다.

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.

Amazon FreeRTOS용 IDT 변수

코드를 빌드하고 디바이스를 플래시하는 명령을 성공적으로 실행하려면 연결 및 디바이스에 대한 기타 정보
가 필요할 수 있습니다. AWS IoT Device Tester에서는 JsonPath를 사용하여 플래시 및 빌드 명령에 있는 디
바이스 정보를 참조할 수 있습니다. 단순 JsonPath 표현식을 사용하여 device.json 파일에 지정된 필수
정보를 가져올 수 있습니다.

경로 변수

Amazon FreeRTOS용 IDT는 명령줄과 구성 파일에 사용할 수 있는 다음 경로 변수를 정의합니다.

`{{testData.sourcePath}}`

소스 코드 경로로 확장됩니다. 이 변수를 사용하는 경우 플래시 명령과 빌드 명령에서 이 변수를 모두 사
용해야 합니다.

`{{device.connectivity.serialPort}}`

직렬 포트로 확장됩니다.

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

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.

Bluetooth Low Energy 테스트 실행

이 단원에서는 Amazon FreeRTOS용 AWS IoT Device Tester를 사용하여 Bluetooth 테스트를 설정하고 실행하는 방법을 설명합니다. 코어 검증에는 Bluetooth 테스트가 필요하지 않습니다. Amazon FreeRTOS Bluetooth를 지원하는 디바이스를 테스트하지 않으려는 경우 이 설정을 건너뛸 수 있으며 device.json의 BLE 기능을 No로 설정된 상태로 그대로 두어야 합니다.

사전 조건

- 처음으로 마이크로 컨트롤러 보드의 테스트 준비 (p. 272)의 지침을 따르십시오.
- Raspberry Pi 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 3B+가 있어야 합니다.

BLE 테스트를 실행하도록 Raspberry Pi를 설정하려면

- 테스트를 수행하는 데 필요한 소프트웨어가 포함된 사용자 지정 [Yocto 이미지](#)를 다운로드합니다.
- yocto 이미지를 Raspberry Pi용 SD 카드로 플래시합니다.
 - Etcher와 같은 SD 카드 쓰기 도구를 사용하여 다운로드된 <image-name>.rpi-sdimg 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.
- Raspberry Pi를 구성합니다.
 - 처음으로 부팅하는 경우 Raspberry Pi를 모니터, 키보드 및 마우스에 연결하는 것이 좋습니다.
 - Raspberry Pi를 마이크로 USB 전원에 연결합니다.
 - 기본 자격 증명을 사용하여 로그인합니다. 사용자 ID에 **root**를 입력합니다. 암호에 **idtafr**를 입력합니다.
 - 이더넷 또는 Wi-Fi 연결을 사용하여 Raspberry Pi를 네트워크에 연결합니다.
 - Wi-Fi를 통해 Raspberry Pi를 연결하려면 Raspberry Pi에서 /etc/wpa_supplicant.conf를 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"
}
```

- ifup wlan0을 실행하여 Wi-Fi 연결을 시작합니다. Wi-Fi 네트워크에 연결하려면 1분 정도 걸릴 수 있습니다.
- 이더넷 연결의 경우 ifconfig eth0를 실행합니다. Wi-Fi 연결의 경우 ifconfig wlan0을 실행합니다. IP 주소를 기록해둡니다. 이 주소는 명령 출력에 inet addr로 나타납니다. 이 절차의 뒷 부분에서 IP 주소가 필요합니다.
- (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 SSK 키 페어가 이미 있는 경우 새 키 페어를 생성하여 Amazon FreeRTOS용 AWS IoT Device Tester가 Raspberry Pi에 로그인하도록 허용하는 것이 가장 좋습니다.

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 Enter를 눌러 계속합니다.
- iv. Amazon FreeRTOS용 AWS IoT Device Tester가 디바이스에 로그인할 수 있도록 SSH 키를 Raspberry Pi에 추가하려면 호스트 컴퓨터에서 ssh-copy-id 명령을 사용합니다. 이 명령은 Raspberry Pi에서 ~/.ssh/authorized_keys 파일에 퍼블릭 키를 추가합니다.

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

- v. 암호를 묻는 화면이 나타나면 `idtafr`을 입력합니다. 이 값은 yocto 이미지의 기본 암호입니다.

Note

ssh-copy-id 명령은 퍼블릭 키에 id_rsa.pub라는 이름이 지정된다고 가정합니다. macOS와 Linux에서 기본 위치는 ~/.ssh/입니다. Windows에서 기본 위치는 C:\Users\<user-name>\.ssh입니다. 퍼블릭 키에 다른 이름을 지정하거나 퍼블릭 키를 다른 위치에 저장한 경우 ssh-copy-id에 -i 옵션을 사용하여 SSH 퍼블릭 키의 정규화된 경로를 지정해야 합니다(예: ssh-copy-id -i ~/my/path/myKey.pub). SSH 키 생성 및 퍼블릭 키 복사에 대한 자세한 내용은 [SSH-COPY-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 no로 설정합니다.
- C. sshd_config 파일을 저장하고 닫습니다.
- D. /etc/init.d/sshd reload를 실행하여 SSH 서버를 다시 로드합니다.

- g. resource.json 파일을 생성합니다.

- i. AWS IoT Device Tester를 추출한 디렉터리에서 resource.json이라는 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",  
        "name": "BLE Test Device",  
        "type": "BLE",  
        "ip": "192.168.1.100",  
        "port": 1883,  
        "cert": "ble-test-raspberry-pi-1.cert",  
        "key": "ble-test-raspberry-pi-1.key",  
        "region": "us-east-1",  
        "aws_iot_endpoint": "https://iot.us-east-1.amazonaws.com"  
      }  
    ]  
  }  
]
```

```
        "connectivity": {  
            "protocol": "ssh",  
            "ip": "<rasp-pi-id-address>"  
        }  
    }  
}
```

- iii. (선택 사항) SSH에 퍼블릭 키 인증을 사용하도록 선택한 경우 `resource.json` 파일의 `connectivity` 섹션에 다음을 추가합니다.

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

Amazon 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

`device.json`에서 BLE 기능을 활성화한 후 `group-id`를 지정하지 않고 `devicetester_[linux | mac | win_x86-64] run-suite`를 실행하면 BLE 테스트가 실행됩니다.

BLE 테스트를 별도로 실행하려면 BLE의 그룹 ID를 지정할 수 있습니다. `devicetester_[linux | mac | win_x86-64] run-suite --userdata <path-to-userdata>/userdata.json --group-id FullBLE`

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. 272)의 단계를 따랐는지 확인합니다. BLE 이외의 다른 테스트가 실패하면 문제가 Bluetooth 구성으로 인한 것이 아닐 가능성이 높습니다.

Amazon FreeRTOS Qualification Suite 실행

Amazon FreeRTOS용 IDT 실행 파일을 사용하여 Amazon FreeRTOS용 IDT와 상호 작용할 수 있습니다. 다른 명령줄에서는 디바이스 풀(동일한 디바이스의 세트)에 대한 검증 테스트를 실행하는 방법을 보여 줍니다.

```
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 있어야 합니다.

Note

Windows에서 Amazon FreeRTOS용 IDT를 실행하는 경우 슬래시(/)를 사용하여 userdata.json 파일의 경로를 지정합니다.

다음 명령을 사용하여 특정 테스트 그룹을 실행합니다.

```
devicetester_[linux | mac | win] run-suite --suite-id AFQ_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.

Amazon FreeRTOS용 IDT 명령줄 옵션

suite-id

선택. 실행할 테스트 제품군을 지정합니다.

pool-id

테스트할 디바이스 폴을 지정합니다. 디바이스 폴이 하나만 있는 경우 이 옵션을 생략할 수 있습니다.

Amazon FreeRTOS용 IDT 명령

Amazon FreeRTOS용 IDT 명령은 다음과 같은 작업을 지원합니다.

help

지정된 명령에 대한 정보를 나열합니다.

list-groups

지정된 제품군에 있는 그룹을 나열합니다.

list-suites

사용 가능한 제품군을 나열합니다.

run-suite

디바이스의 폴에 대해 테스트 제품군을 실행합니다.

재검증을 위한 테스트

새로운 버전의 Amazon FreeRTOS용 IDT 검증 테스트가 릴리스되거나 보드별 패키지 또는 디바이스 드라이버를 업데이트하면 Amazon FreeRTOS용 IDT를 사용하여 마이크로 컨트롤러 보드를 테스트할 수 있습니다. 이후 검증에서는 최신 버전의 Amazon FreeRTOS 및 Amazon FreeRTOS용 IDT가 있는지 확인하고 검증 테스트를 다시 실행해야 합니다.

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`은 AWS Partner Device Catalog에 디바이스를 등록하기 위해 AWS에 제출하는 자격 테스트 보고서입니다. 보고서에는 다음 요소가 포함됩니다.

- Amazon FreeRTOS용 IDT 버전
- 테스트된 Amazon FreeRTOS 버전
- 테스트 통과를 기반으로 디바이스에서 지원되는 Amazon FreeRTOS 기능
- `device.json` 파일에 지정된 SKU 및 디바이스 이름
- `device.json` 파일에 지정된 디바이스의 기능.
- 테스트 사례 결과의 집계 요약입니다.
- 디바이스 기능(예: FullWiFi, FullMQTT 등)을 기반으로 테스트된 라이브러리별 테스트 사례 결과의 분석입니다.

`AFO_Report.xml`은 표준 [JUnit XML 형식](#)의 보고서입니다. [Jenkins](#), [Bamboo](#) 등의 CI/CD 플랫폼에 이 보고서를 통합할 수 있습니다. 보고서에는 다음 요소가 포함됩니다.

- 테스트 사례 결과의 집계 요약
- 디바이스 기능을 기반으로 테스트된 라이브러리별 테스트 사례 결과의 분석

Amazon FreeRTOS용 IDT 결과 해석

`awsiotdevicetester_report.xml` 또는 `AFO_Report.xml`의 보고서 섹션에는 실행된 테스트와 테스트의 결과가 나열됩니다.

첫 번째 XML 태그 `<testsuites>`에는 테스트 실행의 전체 요약이 포함됩니다. 예:

```
<testsuites name="AFO 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`

Amazon FreeRTOS용 IDT에서 실행할 수 없는 테스트 사례의 수

`disabled`

이 속성은 사용되지 않으므로 무시해도 좋습니다.

테스트 사례 실패 또는 오류가 없는 경우 디바이스는 Amazon FreeRTOS를 실행하기 위한 기술 요구 사항을 충족하며 AWS IoT 서비스와 상호 작용할 수 있습니다. AWS Partner Device Catalog에 디바이스를 등록하도록 선택하는 경우 이 보고서를 검증의 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. 예:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1"></testcase>
```

<testcase> 태그에 사용되는 속성

`name`

테스트 사례의 이름입니다.

`attempts`

Amazon FreeRTOS용 IDT에서 테스트 사례를 실행한 횟수

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase">
<failure type="Failure">Reason for the test case failure</failure>
<error>Reason for the test case execution error</error> </testcase>
```

자세한 내용은 [문제 해결 \(p. 285\)](#) 단원을 참조하십시오.

로그 보기

Amazon FreeRTOS용 IDT가 테스트 실행에서 생성하는 로그는 <`devicetester-extract-location`>/`results/<execution-id>/logs`에서 확인할 수 있습니다. 두 개의 로그 세트가 생성됩니다.

`test_manager.log`

Amazon FreeRTOS용 IDT에서 생성된 로그를 포함합니다(예: 로그 관련 구성 및 보고서 생성).

`<test_group_name>.log` (for example, `Full_MQTT.log`)

테스트 대상 디바이스의 로그를 비롯한 테스트 그룹의 로그입니다.

문제 해결

각 테스트 실행에는 `uuid` 디렉터리를 생성하는 데 사용되는 고유의 테스트 ID가 있습니다. 개별 테스트 그룹 로그는 `uuid` 디렉터리에 있습니다. AWS IoT 콘솔을 사용하여 실패한 테스트 그룹을 조회한 다음 /`results/<uuid>` 디렉터리에서 해당 그룹에 대한 로그 파일을 업니다. 이 파일에는 있는 정보는 다음과 같습니다.

- 전체 빌드 및 플래시 명령 출력.
- 테스트 실행 결과
- Amazon FreeRTOS 콘솔 출력을 위한 자세한 IDT

다음 문제 해결 워크플로우를 사용하는 것이 좋습니다.

1. 실행 UUID 및 현재 실행 중인 작업과 같은 정보를 찾으려면 콘솔 출력을 읽습니다.
2. AFO_Report.xml 파일에서 각 테스트의 오류 문장을 찾습니다. 이 디렉터리에는 각 테스트 그룹의 실행 로그가 포함되어 있습니다.
3. /results/<uuid>/logs 아래에 있는 로그 파일을 살펴봅니다.
4. 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 Amazon FreeRTOS 소스 코드가 있는지 확인합니다. Amazon FreeRTOS 릴리스는 Amazon FreeRTOS 버전에 따라 태그가 지정됩니다. 특정 코드 버전을 복제하려면 git clone --branch <version-number> https://github.com/aws/amazon-freertos.git를 사용하십시오.

디바이스 구성 문제 해결

Amazon FreeRTOS용 IDT를 사용할 때는 바이너리를 실행하기 전에 올바른 구성 파일을 준비해야 합니다. 구문 문법 및 구성 오류가 발생할 경우 첫 번째 단계는 환경에 적합한 구성 템플릿을 찾아서 사용하는 것입니다. 이러한 템플릿은 <IDT_ROOT>/configs 디렉터리에 있습니다.

그래도 문제가 발생할 경우 다음 디버깅 프로세스를 참조하십시오.

어디를 살펴봐야 합니까?

먼저 콘솔 출력을 읽고 이 설명서에서 uuid로 참조되는 실행 UUID와 같은 정보를 찾습니다.

그런 다음 /results/<uuid> 디렉터리에서 AFO_Report.xml 파일을 찾습니다. 이 파일에는 실행된 모든 테스트 사례와 각 실패에 대한 오류 코드 조각이 포함되어 있습니다. 모든 실행 로그를 가져오려면 각 테스트 그룹의 /results/<uuid>/<test-case-id>.log 아래를 살펴봅니다.

IDT 오류 코드

다음 표에서는 Amazon 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 내용은 처음으로 마이크로 컨트롤러 보드의 테스트 준비 (p. 272) 단원을 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">• 잘못된 AWS 리전 - config.json 파일에서 유효한 AWS 리전을 지정합니다. AWS

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			<p>리전에 대한 자세한 내용은 리전 및 엔드포인트를 참조하십시오.</p> <ul style="list-style-type: none"> 잘못된 AWS 자격 증명 - 테스트 시스템에서 유효한 AWS 자격 증명을 설정합니다(환경 변수 또는 자격 증명 파일을 통해). 인증 필드가 올바르게 구성되었는지 확인합니다. 자세한 내용은 AWS 계정 생성 및 구성 (p. 269) 단원을 참조하십시오.
203	CopySourceCodeError	Amazon FreeRTOS 소스 코드를 지정된 디렉터리에 복사할 수 없습니다.	<p>다음 항목을 확인합니다.</p> <ul style="list-style-type: none"> <code>userdata.json</code> 파일에서 유효한 <code>sourcePath</code>가 지정되었는지 확인합니다. Amazon FreeRTOS 소스 코드 디렉터리에서 <code>build</code> 폴더를 삭제합니다(해당 폴더가 있는 경우). 자세한 내용은 빌드, 플래시 및 테스트 설정 구성 (p. 275) 단원을 참조하십시오.
204	BuildSourceError	Amazon FreeRTOS 소스 코드를 컴파일할 수 없습니다.	<p>다음 항목을 확인합니다.</p> <ul style="list-style-type: none"> <code>userdata.json</code> 파일에서 <code>buildTool</code> 아래 있는 정보가 올바른지 확인합니다. <code>cmake</code>을 빌드 도구로 사용하는 경우 <code>buildTool</code> 명령에서 <code>enableTests</code>가 지정되었는지 확인합니다. 자세한 내용은 빌드, 플래시 및 테스트 설정 구성 (p. 275) 단원을 참조하십시오.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
205	FlashOrRunTestError	IDT Amazon FreeRTOS 가 DUT에서 Amazon FreeRTOS를 플래시하거나 실행할 수 없습니다.	userdata.json 파일에서 flashTool 아래 있는 정보가 올바른지 확인합니다. 자세한 내용은 빌드, 플래시 및 테스트 설정 구성 (p. 275) 단원을 참조하십시오.
206	StartEchoServerError	IDT Amazon FreeRTOS는 WiFi 또는 SecureSocket 테스트를 위해 에코 서버를 시작할 수 없습니다.	userdata.json 파일의 echoServerConfiguration에 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. Amazon FreeRTOS용 IDT는 JSON 확인을 수행하고 디버깅 정보를 인쇄합니다. IDT는 구문 오류가 발생한 줄, 줄 번호 및 열 번호를 인쇄합니다. 이 정보만 있으면 오류를 수정할 수 있지만, 그래도 오류를 찾는 데 문제가 있는 경우 IDE, Atom이나 Sublime과 같은 텍스트 편집기에서 또는 JSONLint와 같은 온라인 도구를 통해 수동으로 확인을 수행할 수 있습니다.

필수 파라미터 누락 오류 디버깅

Amazon FreeRTOS용 IDT에 새로운 기능이 추가되고 있으므로 구성 파일에 대한 변경 사항이 도입될 수 있습니다. 기존 구성 파일을 사용하면 구성이 손상될 수 있습니다. 이 문제가 발생할 경우 /results/<uuid>/logs 아래의 <test_case_id>.log 파일에 누락된 모든 파라미터가 명시적으로 나열됩니다. 또한 Amazon FreeRTOS용 IDT는 JSON 구성 파일 스키마를 검사하여 지원되는 최신 버전이 사용되었는지 확인합니다.

테스트를 시작할 수 없음 오류 디버깅

테스트 시작 중에 실패를 가리키는 오류가 발생할 수 있습니다. 이 오류에는 여러 원인이 있을 수 있으므로 다음 영역이 정확한지 확인하십시오.

- 실행 명령에 포함된 폴 이름이 실제로 존재하는지 확인합니다. 이 이름은 device.json 파일에서 직접 참조됩니다.
- 폴에 있는 디바이스에 올바른 구성 파라미터가 있는지 확인합니다.

네트워크 테스트 오류

네트워크 기반 테스트의 경우 IDT는 호스트 시스템의 비 예약 포트에 바인딩하는 에코 서버를 시작합니다. WiFi 또는 SecureSocket 테스트에서 제한 시간 또는 사용할 수 없는 연결로 인해 오류가 발생하는 경우, 네트워크가 1024 - 49151 범위로 구성된 포트에 대한 트래픽을 허용하도록 구성되어 있는지 확인하십시오.

SecureSocket 테스트는 기본적으로 포트 33333 및 33334를 사용합니다. WiFi 테스트는 기본적으로 포트 33335를 사용합니다. 이 3개의 포트가 사용 중이거나 방화벽 또는 네트워크에 의해 차단된 경우 userdata.json의 다른 포트를 테스트에 사용하도록 선택할 수 있습니다. 자세한 내용은 [빌드, 플래시 및 테스트 설정 구성 \(p. 275\)](#) 단원을 참조하십시오. 다음 명령을 사용하여 특정 포트가 사용 중인지 확인할 수 있습니다.

- Windows: netsh advfirewall firewall show rule name=all | grep port
- Linux: sudo netstat -pan | grep port
- macOS: netstat -nat | grep port

동일한 버전 페이로드로 인해 OTA 업데이트 실패

OTA가 수행된 후 디바이스에 동일한 버전이 있어 OTA 테스트 사례가 실패한 경우 빌드 시스템(예: cmake)이 Amazon 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 사례에 대한 OTA 테스트 실패

이 테스트의 한 가지 전제 조건은 OTA 업데이트 시간이 60초 이상이어야 한다는 것입니다. 그렇지 않으면 테스트가 실패합니다. 이 경우 로그에 "Test takes less than 60 seconds (url expired time) to finish. Please reach out to us." 오류 메시지가 표시됩니다.

디바이스 인터페이스 및 포트 오류 디버깅

이 단원에는 IDT가 디바이스에 연결하기 위해 사용하는 디바이스 인터페이스에 대한 정보가 포함되어 있습니다.

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 확인하는데 도움이 되는 디바이스 드라이버 정보를 인쇄합니다.
- macOS의 경우 Launchpad를 열고 System Information을 검색합니다.
- Windows의 경우 Device Manager를 열고 직렬 디바이스 그룹을 확장합니다.

디바이스 인터페이스

포함된 각 디바이스가 다르므로 하나 이상의 직렬 포트가 있을 수 있습니다. 일반적으로 시스템에 연결될 때 디바이스에는 두 개의 포트가 있습니다.

- 디바이스를 플래시하기 위한 데이터 포트
- 출력을 읽기 위한 읽기 포트.

device.json 파일에서 올바른 읽기 포트를 설정해야 합니다. 그렇지 않으면 디바이스에서 출력을 읽지 못할 수 있습니다.

여러 포트가 있는 경우 device.json 파일에서 디바이스의 읽기 포트를 사용하는지 확인합니다. 예를 들어, Espressif WROver 디바이스를 플러그인하고 이 디바이스에 할당된 두 개의 포트가 /dev/ttyUSB0 및 /dev/ttyUSB1인 경우 device.json 파일에서 /dev/ttyUSB1을 사용합니다.

Windows의 경우 동일한 로직을 따릅니다.

디바이스 데이터 읽기

Amazon FreeRTOS용 IDT는 개별 디바이스 빌드 및 플래시 도구를 사용하여 포트 구성을 지정합니다. 디바이스를 테스트하는 동안 출력을 얻을 수 없는 경우 다음 기본 설정을 사용해 봅니다.

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

이러한 설정은 Amazon FreeRTOS용 IDT에서 처리됩니다. 직접 설정할 필요가 없습니다. 하지만 동일한 방법을 사용하여 디바이스 출력을 수동으로 읽을 수 있습니다. Linux 또는 macOS에서는 screen 명령을 사용하여 이 작업을 수행할 수 있습니다. Windows에서는 TeraTerm과 같은 프로그램을 사용할 수 있습니다.

Screen: screen /dev/cu.usbserial 115200

TeraTerm: Use the above-provided settings to set the fields explicitly in the GUI.

개발 도구 체인 문제

이 단원에서는 도구 체인에 발생할 수 있는 문제를 설명합니다.

Ubuntu의 Code Composer Studio

최신 버전의 Ubuntu(17.10 및 18.04)에는 Code Composer Studio 7.x 버전과 호환되지 않는 glibc 패키지 버전이 있습니다. Code Composer Studio 버전 8.2 이상을 설치하는 것이 좋습니다.

비호환성을 나타내는 증상에는 다음이 포함될 수 있습니다.

- Amazon FreeRTOS가 디바이스에 빌드하거나 플래시하지 못합니다.
- Code Composer Studio 설치 관리자가 동결될 수 있습니다.
- 빌드 또는 플래시 프로세스 중에 로그 출력이 콘솔에 표시되지 않습니다.
- 헤드리스로 호출하더라도 빌드 명령이 GUI 모드에서 시작하려고 시도합니다.

로깅

Amazon FreeRTOS용 IDT 로그는 단일 위치에 배치됩니다. 루트 IDT 디렉터리에서 사용 가능한 파일은 다음과 같습니다.

- ./results/<uuid>
- AFO_Report.xml
- awsiotdevicetester_report.xml

- `/logs/<test_group_id>.log`

검사할 가장 중요한 로그는 `<test_group_id>.log`와 `results.xml`입니다. 후자에는 어떤 테스트가 특정 오류 메시지와 함께 실패했는지에 대한 정보가 포함되어 있습니다. 그런 다음 컨텍스트를 더 잘 이해하기 위해 전자를 사용하여 문제를 더 깊이 분석할 수 있습니다.

콘솔 오류

AWS IoT 디바이스 테스터가 실행될 때 실패는 간략한 메시지와 함께 콘솔에 보고됩니다. 오류에 대해 자세히 알아보려면 `<test_group_id>.log`를 살펴봅니다.

오류 로그

`<test_group_id>.log` 파일은 `/results/<uuid>` 디렉터리에 있습니다. 각 테스트 실행에는 `<uuid>` 디렉터리를 생성하는 데 사용되는 고유의 테스트 ID가 있습니다. 개별 테스트 그룹 로그는 `<uuid>` 디렉터리에 있습니다. AWS IoT 콘솔을 사용하여 실패한 테스트 그룹을 조회한 다음 `/results/<uuid>` 디렉터리에서 해당 그룹에 대한 로그 파일을 엽니다. 이 파일의 정보에는 전체 빌드 및 플래시 명령 출력, 테스트 실행 출력 및 더 상세한 AWS IoT Device Tester 콘솔 출력도 포함됩니다.

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. 예:

```
./devicetester_linux run-suite --suite-id AFQ_1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

AWS의 보안

AWS에서 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 매우 보안에 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 귀하의 공동 책임입니다. [책임 분담 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- **클라우드의 보안** – AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. 또한 AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사원은 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 항목에서는 보안 및 규정 준수 목적에 맞게 AWS를 구성하는 방법을 보여줍니다. 또한 AWS 리소스를 모니터링하고 보호하는 데 도움이 될 수 있는 AWS 서비스를 사용하는 방법도 배웁니다.

AWS IoT 보안에 대한 자세한 내용은 [AWS IoT의 보안 및 자격 증명](#)을 참조하십시오.

주제

- [AWS 리소스에 대한 자격 증명 및 액세스 관리](#) (p. 292)
- [규정 준수 확인](#) (p. 302)
- [AWS의 복원성](#) (p. 303)
- [Amazon FreeRTOS의 인프라 보안](#) (p. 303)

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 AWS 리소스를 사용하기 위해 인증(로그인) 및 권한 부여(권한 있음)할 수 있는 사람을 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

- [대상](#) (p. 292)
- [자격 증명을 통한 인증](#) (p. 293)
- [정책을 이용한 액세스 관리](#) (p. 294)
- [자세히 알아보기](#) (p. 296)
- [AWS 서비스에서 IAM로 작업하는 방식](#) (p. 296)
- [자격 증명 기반 정책 예제](#) (p. 298)
- [자격 증명 및 액세스 문제 해결](#) (p. 300)

대상

AWS Identity and Access Management(IAM) 사용 방법은 AWS에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 – AWS 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 추가 기능을 사용하여 작업을 수행할 때는 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식

을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. AWS의 기능에 액세스할 수 없는 경우 [자격 증명 및 액세스 문제 해결 \(p. 300\)](#) 단원을 참조하십시오.

서비스 관리자 – 회사에서 AWS 리소스를 책임지고 있는 경우 사용하는 서비스에 대한 완전한 액세스 권한을 갖고 있을 것입니다. 직원이 어떤 기능과 리소스에 액세스해야 하는지를 결정하는 작업은 서비스 관리자의 책임입니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사가 AWS에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 [AWS 서비스에서 IAM로 작업하는 방식 \(p. 296\)](#) 단원을 참조하십시오.

IAM 관리자 – IAM 관리자인 경우 AWS에 대한 액세스를 관리하기 위한 정책을 작성하는 방법에 대해 자세히 알아보고 싶을 수 있습니다. IAM에서 사용할 수 있는 AWS 자격 증명 기반 정책에 대한 자세한 내용은 AWS 자격 증명 및 액세스 관리 사용 설명서의 [정책 및 권한](#)을 참조하십시오.

자격 증명을 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. AWS Management 콘솔을 사용하여 로그인하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 콘솔 및 로그인 페이지](#)를 참조하십시오.

AWS 계정 루트 사용자, `class="non-printable-char non-printable-space"> IAM class="non-printable-char non-printable-space">` 사용자로서 `class="non-printable-char non-printable-space"> IAM class="non-printable-char non-printable-space">` 또는 `class="non-printable-char non-printable-space"> IAM class="non-printable-char non-printable-space">` 역할을 `class="non-printable-char non-printable-space"> 수임하여 class="non-printable-char non-printable-space">` 인증(AWS에 `class="non-printable-char non-printable-space"> 로그인)되어야 class="non-printable-char non-printable-space">` 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 콘솔](#)에 직접 로그인하려면 루트 사용자 이메일이나 IAM 사용자 이름과 비밀번호를 사용하십시오. 루트 사용자 또는 IAM 사용자 액세스 키를 사용하여 프로그래밍 방식으로 AWS에 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 정보는 AWS General Reference의 [서명 버전 4 서명 프로세스](#) 단원을 참조하십시오.

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 IAM 사용 설명서의 [AWS에서 멀티 팩터 인증\(MFA\) 사용](#)을 참조하십시오.

AWS 계정 루트 사용자

AWS 계정을 처음 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 SSO(Single Sign-In) ID로 시작합니다. 이 자격 증명은 AWS 계정 루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업은 물론 관리 작업에도 루트 사용자를 사용하지 않는 것이 좋습니다. 대신 [IAM 사용자를 처음 생성할 때만 루트 사용자를 사용하는 모범 사례](#)를 준수하십시오. 그런 다음 루트 사용자 자격 증명을 안전하게 보관해 두고 몇 가지 계정 및 서비스 관리 작업을 수행할 때만 해당 자격 증명을 사용합니다.

IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. IAM 사용자에게는 사용자 이름과 암호 또는 액세스 키 세트와 같은 장기 자격 증명이 있을 수 있습니다. 액세스 키를 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 사용자의 액세스 키 관리](#)를 참조하십시오. 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 ID입니다. 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관

리할 수 있습니다. 예를 들어, IAMAdmins이라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 자격 증명만 제공합니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자\(역할 대신\)를 생성하는 경우](#)를 참조하십시오.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. 이 역할은 IAM 사용자와 비슷하지만, 특정 개인과 연결되지 않습니다. [역할을 전환](#)하여 AWS Management 콘솔에서 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 IAM 사용 설명서의 [IAM 역할 사용](#)을 참조하십시오.

임시 자격 증명이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- **임시 IAM 사용자 권한** – IAM 사용자는 IAM 역할을 수임하여 특정 작업에 대한 다른 권한을 임시로 받을 수 있습니다.
- **연합된 사용자 액세스** – IAM 사용자를 생성하는 대신 AWS Directory Service의 기존 ID, 엔터프라이즈 사용자 디렉터리 또는 웹 ID 공급자를 사용할 수 있습니다. 이 사용자를 연합된 사용자라고 합니다. AWS에서는 [ID 공급자](#)를 통해 액세스가 요청되면 연합된 사용자에게 역할을 할당합니다. 연합된 사용자에 대한 자세한 내용은 IAM 사용 설명서의 [연합된 사용자 및 역할](#)을 참조하십시오.
- **교차 계정 액세스** – IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 할 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 정책의 차이점을 알아보려면 IAM 사용 설명서의 [IAM 역할과 리소스 기반 정책의 차이](#)를 참조하십시오.
- **AWS 서비스 액세스** – 서비스 역할은 서비스가 사용자를 대신하여 사용자 계정에서 작업을 수행하기 위해 수임하는 IAM 역할입니다. 일부 AWS 서비스 환경을 설정할 때 서비스에서 맙을 역할을 정의해야 합니다. 이 서비스 역할에는 서비스가 AWS 리소스에 액세스하는 데 필요한 모든 권한이 포함되어야 합니다. 서비스 역할은 서비스마다 다르지만 해당 서비스에 대한 문서화된 요구 사항을 충족하는 한 대부분의 경우 권한을 선택할 수 있습니다. 서비스 역할은 해당 계정 내 액세스 권한만 제공하며 다른 계정의 서비스에 대한 액세스 권한을 부여하는 데 사용될 수 없습니다. IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 예를 들어 Amazon Redshift에서 사용자 대신 Amazon S3 버킷에 액세스하도록 허용하는 역할을 생성한 후 해당 버킷에 있는 데이터를 Amazon Redshift 클러스터로 로드할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성](#)을 참조하십시오.
- **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 참조하십시오.

IAM 역할을 사용할지 여부를 알아보려면 IAM 사용 설명서의 [사용자 대신 IAM 역할을 생성해야 하는 경우](#)를 참조하십시오.

정책을 이용한 액세스 관리

정책을 생성하고 IAM 자격 증명 또는 AWS 리소스에 연결하여 AWS 액세스를 제어합니다. 정책은 자격 증명이나 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 IAM 사용 설명서의 [JSON 정책 개요](#)를 참조하십시오.

IAM 관리자는 AWS 리소스에 액세스할 수 있는 사람과 해당 리소스에 대해 수행할 수 있는 작업을 지정할 수 있습니다. 모든 IAM 엔터티(사용자 또는 역할)는 처음에는 권한이 없습니다. 다시 말해, 기본적으로 사용자는 아무 작업도 수행할 수 없으며, 자신의 암호를 변경할 수도 없습니다. 사용자에게 작업을 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 작업을 실행하기 위한 방법과 상관없이 작업을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management 콘솔, 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 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하십시오.

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 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책과 인라인 정책의 선택](#)을 참조하십시오.

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.

ACL(액세스 제어 목록)

액세스 제어 정책(ACL)은 리소스에 액세스할 수 있는 권한을 가진 주체(계정 멤버, 사용자 또는 역할)를 제어합니다. ACL은 리소스 기반 정책과 비슷합니다. 다만 JSON 정책 문서 형식을 사용하지 않은 유일한 정책 유형입니다. Amazon S3, AWS WAF, Amazon VPC는 ACL을 지원하는 서비스의 예입니다. ACL에 대한 자세한 정보는 Amazon Simple Storage Service 개발자 가이드의 [ACL\(액세스 제어 목록\) 개요](#) 단원을 참조하십시오.

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 엔터티의 자격 증명 기반 정책의 교차와 그 권한 경계입니다. 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 IAM 사용 설명서의 [IAM 엔터티에 대한 권한 경계](#)를 참조하십시오.
- 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 종양에서 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. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 [SCP의 작동 방식](#) 단원을 참조하십시오.
- 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 IAM 사용 설명서의 [세션 정책](#)을 참조하십시오.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지 여부를 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하십시오.

자세히 알아보기

AWS 리소스의 자격 증명 및 액세스 관리에 대해 자세히 알아보려면 다음 페이지로 진행하십시오.

- [AWS 서비스에서 IAM로 작업하는 방식](#) (p. 296)
- [자격 증명 기반 정책 예제](#) (p. 298)
- [자격 증명 및 액세스 문제 해결](#) (p. 300)

AWS 서비스에서 IAM로 작업하는 방식

IAM을 사용하여 AWS 서비스에 대한 액세스를 관리하려면 먼저 어떤 IAM 기능을 사용할 수 있는지를 이해해야 합니다. AWS 서비스에서 IAM으로 작업하는 방식에 대한 상위 수준 보기를 얻으려면 IAM 사용 설명서의 [IAM으로 작업하는 AWS 서비스](#)를 참조하십시오.

주제

- [자격 증명 기반 정책](#) (p. 296)
- [AWS 리소스 기반 정책](#) (p. 297)
- [태그 기반 권한 부여](#) (p. 298)
- [IAM 역할](#) (p. 298)

자격 증명 기반 정책

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하십시오.

Actions

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 설명에는 Action 또는 NotAction 요소가 반드시 추가되어야 합니다. 각 서비스는 서비스에서 수행할 수 있는 작업을 설명하는 고유의 작업 세트를 정의합니다.

명령문 하나에 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [  
    "service-prefix:action1",  
    "service-prefix:action2"
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "service-prefix:Describe*"
```

AWS 작업 목록을 보려면 IAM 사용 설명서의 [Actions, Resources, and Condition Keys for AWS Services](#) 단원을 참조하십시오.

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 참조하십시오.

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 [Actions, Resources, and Condition Keys for AWS Services](#) 단원을 참조하십시오.

조건 키

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 선택 사항입니다. 같음, 미만 등 [조건 연산자](#)를 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 작업을 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우 AWS는 논리적 OR 작업을 사용하여 조건을 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 정책 요소: 변수 및 태그](#) 단원을 참조하십시오.

모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 [AWS 전역 조건 컨텍스트 키](#)를 참조하십시오.

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 엔터티를 [리소스 기반 정책의 보안 주체](#)로 지정할 수 있습니다. 리소스 기반 정책에 교차 계정 보안 주체를 추가하는 것은 신뢰 관계 설정의 절

반밖에 되지 않는다는 것을 유념하십시오. 보안 주체와 리소스가 다른 AWS 계정에 있는 경우 보안 주체 엔터티가 리소스에 액세스할 권한도 부여해야 합니다. 엔터티에 자격 증명 기반 정책을 연결하여 권한을 부여합니다. 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 요소](#)에 태그 정보를 제공하십시오.

리소스에서 태그를 기반으로 해당 리소스에 대한 액세스를 제한하기 위한 자격 증명 기반 정책 예제를 보려면 [태그를 기반으로 리소스 보기 \(p. 300\)](#) 단원을 참조하십시오.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 내 개체입니다.

임시 자격 증명 사용

임시 자격 증명을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 수입하거나, 교차 계정 역할을 수입할 수 있습니다. `AssumeRole` 또는 `GetFederationToken`과 같은 AWS Security Token Service(AWS STS) API 작업을 호출하여 임시 보안 자격 증명을 획득합니다.

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 제품이 다른 서비스의 리소스에 액세스하여 사용자 대신 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신해 서비스 역할을 맡을 수 있습니다. 이 역할을 사용하면 서비스는 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 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 콘솔, AWS CLI 또는 AWS API를 사용해 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 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 템플릿에서 정책 생성](#)을 참조하십시오.

주제

- [정책 모범 사례 \(p. 299\)](#)
- [AWS 콘솔 사용 \(p. 299\)](#)
- [사용자가 자신이 권한을 볼 수 있도록 허용 \(p. 299\)](#)
- [태그를 기반으로 리소스 보기 \(p. 300\)](#)

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.

- AWS 관리형 정책을 사용하여 시작하기 – AWS 서비스 사용을 빠르게 시작하려면 AWS 관리형 정책을 사용하여 필요한 권한을 직원에게 부여합니다. 이 정책은 이미 계정에서 사용할 수 있으며 AWS에 의해 유지 관리 및 업데이트됩니다. 자세한 내용은 IAM 사용 설명서의 [AWS 관리형 정책으로 권한 사용 시작하기](#)를 참조하십시오.
- 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 설명서의 [최소 권한 부여](#)를 참조하십시오.
- 중요한 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 Multi-Factor Authentication\(MFA\) 사용하기](#)를 참조하십시오.
- 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 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하십시오.

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 IAM 사용 설명서의 [사용자에게 권한 추가](#)를 참조하십시오.

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 보여 줍니다. 이 정책에는 콘솔에서 또는 AWS CLI나 AWS API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ViewOwnUserInfo",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetUserPolicy",  
                "iam>ListGroupsForUser",  
                "iam>ListAttachedUserPolicies",  
                "iam>ListUserPolicies",  
                "iam:GetUser"  
            ],  
            "Resource": [  
                "arn:aws:iam::*:user/${aws:username}"  
            ]  
        },  
    ]  
},
```

```
{  
    "Sid": "NavigateInConsole",  
    "Effect": "Allow",  
    "Action": [  
        "iam:GetGroupPolicy",  
        "iam:GetPolicyVersion",  
        "iam:GetPolicy",  
        "iam>ListAttachedGroupPolicies",  
        "iam>ListGroupPolicies",  
        "iam>ListPolicyVersions",  
        "iam>ListPolicies",  
        "iam>ListUsers"  
    ],  
    "Resource": "*"  
}  
}  
]
```

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 작업을 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 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하십시오.

자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 IAM 사용 시 공통적으로 발생할 수 있는 문제를 진단 및 수정하십시오.

주제

- [작업을 수행할 권한이 없음 \(p. 301\)](#)
- [iam:PassRole을 수행할 권한이 없음 \(p. 301\)](#)

- 액세스 키를 보기로 원함 (p. 301)
- 관리자이며 다른 사용자가 AWS 리소스에 액세스하도록 허용하려고 함 (p. 302)
- AWS 계정 외부의 사용자가 리소스에 액세스하도록 허용하려고 함 (p. 302)

작업을 수행할 권한이 없음

AWS Management 콘솔에서 작업을 수행할 권한이 없다는 메시지가 나타나는 경우 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 mateojackson IAM 사용자가 콘솔을 사용하여 *my-example-resource*에 대한 세부 정보를 보려고 하지만 *prefix:Action* 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
perform: prefix:Action on resource: my-example-resource
```

이 경우 Mateo는 *prefix:Action*을 사용하여 *my-example-resource*에 액세스하는 것을 허용하도록 정책을 업데이트하라고 관리자에게 요청합니다.

iam:PassRole을 수행할 권한이 없음

iam:PassRole 작업을 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 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 서비스에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 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 작업을 수행하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

액세스 키를 보기로 원함

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/K7MDENG/bPxRficiYEXAMPLEKEY)의 2가지 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 제3자에게 제공하지 마십시오. 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 추가할 수 있습니다. 최대 두 개의 액세스 키를 가질 수 있습니다. 이미 두 개가 있

는 경우 새로 생성하려면 먼저 키 페어 하나를 삭제해야 합니다. 지침을 보려면 IAM 사용 설명서의 [액세스 키 관리](#)를 참조하십시오.

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 만들어야 합니다. 다른 사용자들은 해당 엔티티에 대한 자격 증명을 사용해 AWS에 액세스합니다. 그런 다음 AWS에서 올바른 권한을 부여하는 정책을 개체에 연결해야 합니다.

즉시 시작하려면 IAM 사용 설명서의 [첫 번째 IAM 위임 사용자 및 그룹 생성](#)을 참조하십시오.

AWS 계정 외부의 사용자가 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 참조하십시오.

- 서비스가 이러한 기능을 지원하는지 여부를 알아보려면 [AWS 서비스에서 IAM로 작업하는 방식](#) (p. 296) 단원을 참조하십시오.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [자신이 소유한 다른 AWS 계정의 IAM 사용자에 대한 액세스 권한 제공](#)을 참조하십시오.
- 리소스에 대한 액세스 권한을 제3자 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 [제3자가 소유한 AWS 계정에게 액세스 권한 제공](#)을 참조하십시오.
- 자격 증명 연동을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(자격 증명 연동\)](#)을 참조하십시오.
- 교차 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 [IAM 역할과 리소스 기반 정책의 차이](#)를 참조하십시오.

규정 준수 확인

Amazon FreeRTOS은 AWS 규정 준수 프로그램의 범위 내에 있지 않습니다. 특정 규정 준수 프로그램 범위에 속하는 AWS 제품의 목록은 [규정 준수 프로그램 제공 범위 내 AWS 제품](#)을 참조하십시오. 일반 정보는 [AWS 규정 준수 프로그램](#)을 참조하십시오.

AWS Artifact를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 [AWS 아티팩트의 보고서 다운로드](#)를 참조하십시오.

Amazon FreeRTOS 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS에서는 규정 준수에 도움이 되는 다음과 같은 리소스를 제공합니다.

- [보안 및 규정 준수 빠른 시작 안내서](#) – 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- [HIPAA 보안 및 규정 준수 기술 백서 설계](#) – 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 만드는 방법을 설명합니다.
- [AWS 규정 준수 리소스](#) – 이 워크북 및 안내서는 귀사의 산업 및 위치에 적용될 수 있습니다.
- [AWS Config](#) – 이 AWS 제품으로 리소스 구성이 내부 관행, 업계 지침 및 규정을 준수하는 정도를 평가할 수 있습니다.

- [AWS Security Hub](#) – 이 AWS 제품으로 보안 업계 표준 및 모범 사례 규정 준수 여부를 확인하는 데 도움이 되는 AWS 내 보안 상태에 대한 포괄적인 관점을 제공합니다.

AWS의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전에서는 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 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하십시오.

Amazon FreeRTOS의 인프라 보안

AWS 관리형 서비스는 [Amazon Web Services: 보안 프로세스 개요](#) 백서에 설명된 AWS 글로벌 네트워크 보안 절차를 통해 보호됩니다.

AWS에서 개시한 API 호출을 사용하여 네트워크를 통해 AWS 서비스에 액세스합니다. 클라이언트가 TLS(전송 계층 보안) 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.