aws

이식 안내서

# FreeRTOS

# FreeRTOS: 이식 안내서

# Table of Contents

# FreeRTOS 이식

## FreerTOS란 무엇인가요?

20여 년 동안 세계 유수의 칩 회사와 협력하여 개발되어 현재 170초마다 다운로드되는 FreeRTOS는 마이크로컨트롤러 및 소형 마이크로프로세서를 위한 시장을 선도하는 실시간 운영 체제입니다. MIT 오픈 소스 라이선스에 따라 자유롭게 배포되는 FreeRTOS는 모든 산업 분야에서 사용하기에 적합한 커널과 증가하는 라이브러리 세트가 포함되어 있습니다. FreeRTOS는 신뢰성과 사용 편의성에 중점을 두고 빌드되었습니다. FreeRTOS에는 연결, 보안 및 무선 업데이트(OTA)용 라이브러리가 포함되어 있으며, 적격 보드에서 FreeRTOS 기능을 보여주는 데모 애플리케이션도 포함되어 있습니다.

자세한 내용을 알아보려면 FreeRTOS.org를 방문하세요.

## FreeRTOS를 IoT 보드에 이식

기능 및 애플리케이션에 따라 FreeRTOS 소프트웨어 라이브러리를 마이크로컨트롤러 기반 보드에 이식해야 합니다.

FreeRTOS를 디바이스로 이식하려면

1. 이식을 위해 FreeRTOS 다운로드의 지침에 따라 이식을 위해 최신 버전의 FreeRTOS를 다운로드합니다.
2. 이식을 위한 작업 영역 및 프로젝트 설정의 지침에 따라 이식 및 테스트를 위해 FreeRTOS 다운로드의 파일 및 폴더를 구성합니다.
3. FreeRTOS 라이브러리 이식의 지침에 따라 FreeRTOS 라이브러리를 디바이스로 이식합니다. 각 이식 주제에는 포트 테스트에 대한 지침이 포함되어 있습니다.

### 이식 FAQ

FreeRTOS 포트란 무엇인가요?

> FreeRTOS 포트는 필수 FreeRTOS 라이브러리 및 플랫폼에서 지원하는 FreeRTOS 커널에 대한 API의 보드별 구현입니다. 이 포트를 사용하면 API가 보드에서 작동하며, 필요에 따라 플랫폼 공급업체가 제공하는 디바이스 드라이버 및 BSP와의 통합을 구현합니다. 또한 포트에는 보드에서 요구하는 구성 조정(예: 클럭 속도, 스택 크기, 힙 크기)이 포함되어야 합니다.

이 페이지 또는 FreeRTOS 이식 안내서의 나머지 부분에서 답변을 얻지 못한 이식에 대해 질문이 있는 경우 사용 가능한 FreeRTOS 지원 옵션을 참조하세요.

# 이식을 위해 FreeRTOS 다운로드

최신 FreeRTOS 또는 장기 지원(LTS) 버전을 freertos.org에서 다운로드하거나 GitHub (FreeRTOS-LTS) 또는 (FreeRTOS)에서 복제합니다.

> **Note**
>
> 리포지토리를 복제하는 것이 좋습니다. 복제 기능을 사용하면 리포지토리에 푸시될 때 메인 브랜치에 대한 업데이트를 쉽게 선택할 수 있습니다.

FreeRTOS 또는 Freertos-LTS 리포지토리에서 개별 라이브러리를 하위 모듈로 생성할 수도 있습니다. 그러나 라이브러리 버전이 FreeRTOS 또는 Freertos-LTS 리포지토리의 `manifest.yml` 파일에 나열된 조합과 일치하는지 확인해야 합니다.

FreeRTOS를 다운로드 또는 복제한 후 FreeRTOS 라이브러리를 보드로 이식하기 시작할 수 있습니다. 지침은 이식을 위한 작업 영역 및 프로젝트 설정 및 FreeRTOS 라이브러리 이식 단원을 참조하십시오.

# 이식을 위한 작업 영역 및 프로젝트 설정

아래 단계에 따라 작업 영역 및 프로젝트를 설정합니다.

- 선택한 프로젝트 구조 및 빌드 시스템을 사용하여 FreeRTOS 라이브러리를 가져옵니다.
- 보드에서 지원하는 통합 개발 환경(IDE) 및 도구 체인을 사용하여 프로젝트를 생성합니다.
- 보드 지원 패키지(BSP) 및 보드별 드라이버를 프로젝트에 포함합니다.

작업 영역이 설정되면 개별 FreeRTOS 라이브러리 이식을 시작할 수 있습니다.

# FreeRTOS 라이브러리 이식

이식을 시작하기 전에 이식을 위한 작업 영역 및 프로젝트 설정의 지침을 따르세요.

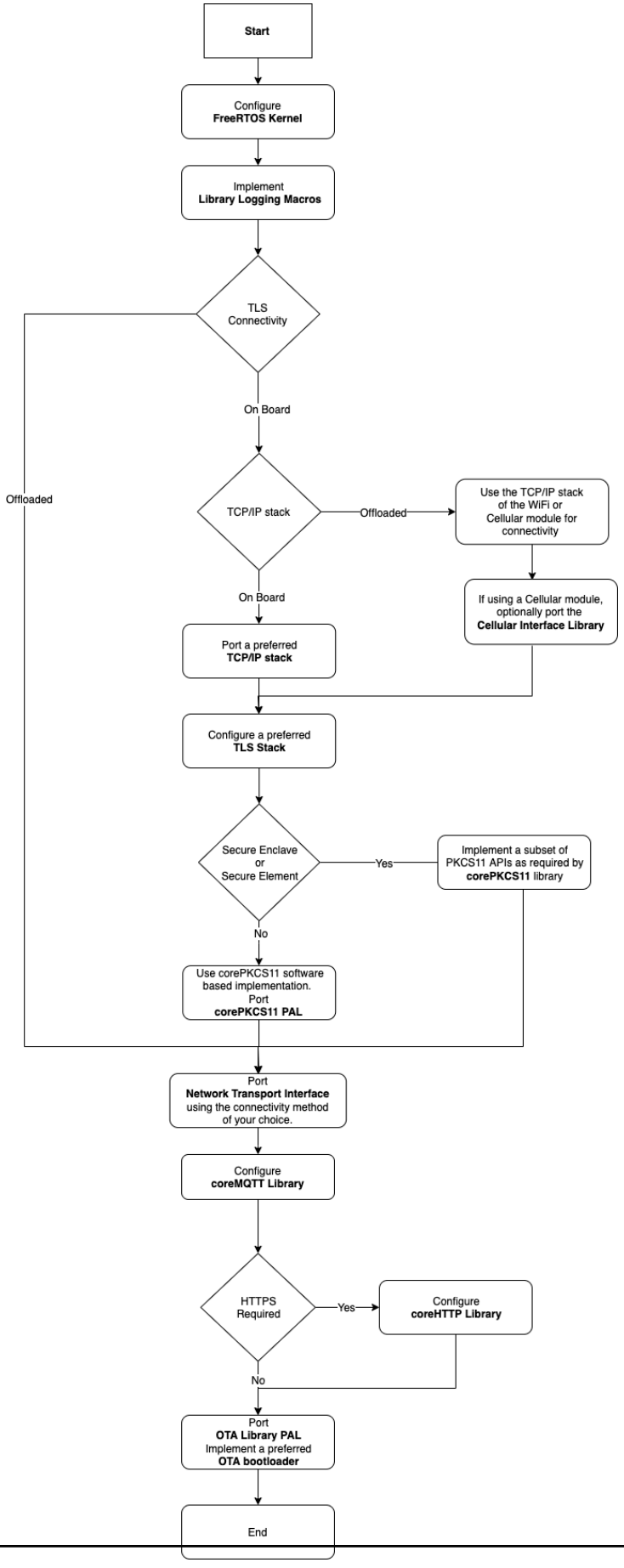
FreeRTOS 이식 순서도에서는 이식에 필요한 라이브러리에 대해 설명합니다.

FreeRTOS를 디바이스에 이식하려면 아래 주제의 지침을 따릅니다.

1. FreeRTOS 커널 포트 구성
2. 라이브러리 로깅 매크로 구현
3. TCP/IP 스택 이식
4. 네트워크 전송 인터페이스 이식
5. corePKCS11 라이브러리 이식
6. coreMQTT 라이브러리 구성
7. coreHTTP 라이브러리 구성
8. AWS IoT over-the-air(OTA) 업데이트 라이브러리 이식
9. 셀룰러 인터페이스 라이브러리 이식

## FreeRTOS 이식 순서도

FreeRTOS를 보드로 이식할 때 아래 이식 순서도를 시각적 보조 자료로 사용합니다.


Start
Configure
FreeRTOS Kernel
Implement
Library Logging Macros
TLS
Connectivity
On Board
Offloaded
TCP/IP stack
Offloaded
Use the TCP/IP stack
of the WiFi or
Cellular module for
connectivity
If using a Cellular module,
optionally port the
Cellular Interface Library
On Board
Port a preferred
TCP/IP stack
Configure a preferred
TLS Stack
Secure Enclave
or
Secure Element
Yes
Implement a subset of
PKCS11 APIs as required by
corePKCS11 library
No
Use corePKCS11 software
based implementation.
Port
corePKCS11 PAL
Port
Network Transport Interface
using the connectivity method
of your choice.
Configure
coreMQTT Library
HTTPS
Required
Yes
Configure
coreHTTP Library
No
Port
OTA Library PAL
Implement a preferred
OTA bootloader
End

# FreeRTOS 커널 포트 구성

이 섹션에서는 FreeRTOS 커널의 포트를 FreeRTOS 포트 테스트 프로젝트에 통합하는 방법을 설명합니다. 사용 가능한 커널 포트 목록은 FreeRTOS 커널 포트를 참조하십시오.

FreeRTOS는 멀티태스킹 및 태스크 간 통신에 FreeRTOS 커널을 사용합니다. 자세한 내용은 FreeRTOS 사용자 안내서의 FreeRTOS 커널 기본 사항 및 FreeRTOS.org를 참조하세요.

> **Note**
>
> FreeRTOS 커널을 새로운 아키텍처로 이식하는 방법은 이 설명서에서 다루지 않습니다. 관심이 있으시면 FreeRTOS 엔지니어링 팀에 문의하세요.
> FreeRTOS 검증 프로그램의 경우 기존 FreeRTOS 커널만 지원됩니다. 이러한 포트에 대한 수정은 검증 프로그램 내에서 허용되지 않습니다. 자세한 내용은 FreeRTOS kernel port policy를 참조하세요.

## 사전 조건

이식을 위해 FreeRTOS 커널을 설정하려면 다음이 필요합니다.

- 공식 FreeRTOS 커널 포트, 또는 대상 플랫폼에서 지원되는 FreeRTOS.
- 대상 플랫폼 및 컴파일러에 대한 올바른 FreeRTOS 커널 포트 파일을 포함하는 IDE 프로젝트. 테스트 프로젝트 설정에 대한 자세한 내용은 이식을 위한 작업 영역 및 프로젝트 설정 단원을 참조하십시오.

## FreeRTOS 커널 구성

FreeRTOS 커널은 `FreeRTOSConfig.h`라는 구성 파일을 사용하여 사용자 지정됩니다. 이 파일은 커널에 대한 애플리케이션별 구성 설정을 지정합니다. 각 구성 옵션에 대한 설명은 FreeRTOS.org의 Customisation을 참조하세요.

FreeRTOS 커널을 디바이스에서 작동하도록 구성하려면 `FreeRTOSConfig.h`를 포함하고 추가로 FreeRTOS 구성을 수정합니다.

각 구성 옵션에 대한 설명은 FreeRTOS.org의 Customisation을 참조하세요.

### 테스트

- 간단한 FreeRTOS 태스크를 실행하여 메시지를 직렬 출력 콘솔에 로깅합니다.
- 메시지가 예상대로 콘솔에 출력되는지 확인합니다.

## 라이브러리 로깅 매크로 구현

FreeRTOS 라이브러리는 다음과 같은 로깅 매크로를 사용합니다(세부 수준 오름차순으로 나열됨).

- `LogError`
- `LogWarn`
- `LogInfo`
- `LogDebug`

모든 매크로에 대한 정의를 제공해야 합니다. 권장 사항은 다음과 같습니다.

- 매크로는 C89 스타일 로깅을 지원해야 합니다.
- 로깅은 스레드 안전이어야 합니다. 여러 태스크의 로그 줄이 인터리브되지 않아야 합니다.
- 로깅 API는 I/O에서 차단되지 않아야 하고 애플리케이션 태스크가 I/O에서 차단되지 않도록 해야 합니다.

구현 세부 사항은 FreeRTOS.org의 Logging Functionality를 참조하세요. 이 예제에서 구현을 확인할 수 있습니다.

### 테스트

- 여러 태스크가 포함된 테스트를 실행하여 로그가 인터리브되지 않는지 확인합니다.
- 테스트를 실행하여 로깅 API가 I/O에서 차단되지 않는지 확인합니다.
- C89, C99 스타일 로깅과 같은 다양한 표준으로 로깅 매크로를 테스트합니다.
- `Debug`, `Info`, `Error`, `Warning` 등 다양한 로그 수준을 설정하여 로깅 매크로를 테스트합니다.

# TCP/IP 스택 이식

이 섹션에서는 온보드 TCP/IP 스택을 이식하고 테스트하기 위한 지침을 제공합니다. 플랫폼이 TCP/IP 및 TLS 기능을 별도의 네트워크 프로세서 또는 모듈로 오프로드하는 경우 이 이식 섹션을 건너뛰고 네트워크 전송 인터페이스 이식 섹션으로 이동하세요.

FreeRTOS+TCP는 FreeRTOS 커널의 네이티브 TCP/IP 스택입니다. FreeRTOS+TCP는 FreeRTOS 엔지니어링 팀에서 개발 및 유지 관리하며 FreeRTOS와 함께 사용하도록 권장되는 TCP/IP 스택입니다. 자세한 내용은 FreeRTOS+TCP 이식 단원을 참조하십시오. 타사 TCP/IP 스택 lwIP를 사용할 수도 있습니다. 이 섹션에 제공된 테스트 지침은 TCP 일반 텍스트용 전송 인터페이스 테스트를 사용하며 구현된 특정 TCP/IP 스택에 종속되지 않습니다.

## FreeRTOS+TCP 이식

FreeRTOS+TCP는 FreeRTOS 커널의 네이티브 TCP/IP 스택입니다. 자세한 내용은 FreeRTOS.org를 참조하십시오.

### 사전 조건

FreeRTOS+TCP 라이브러리를 이식하려면 다음이 필요합니다.

- 공급업체에서 제공하는 이더넷 또는 Wi-Fi 드라이버가 포함된 IDE 프로젝트.

  테스트 프로젝트 설정에 대한 자세한 내용은 이식을 위한 작업 영역 및 프로젝트 설정 단원을 참조하십시오.
- FreeRTOS 커널의 검증된 구성.

  플랫폼에 맞는 FreeRTOS 커널 구성에 대한 정보는 FreeRTOS 커널 포트 구성 단원을 참조하십시오.

### 이식

FreeRTOS-TCP 라이브러리 이식을 시작하기 전에 GitHub 디렉터리를 확인하여 보드에 대한 포트가 이미 존재하는지 확인합니다.

포트가 존재하지 않으면 다음을 수행하십시오.

1. FreeRTOS+TCP를 디바이스에 이식하는 방법은 FreeRTOS.org의 Porting FreeRTOS+TCP to a Different Microcontroller를 참조하십시오.

2. 필요한 경우 FreeRTOS+TCP를 새 컴파일러에 이식하는 방법은 FreeRTOS.org의 Porting FreeRTOS+TCP to a New Embedded C Compiler를 참조하십시오.
3. 공급업체가 제공한 이더넷 또는 Wi-Fi 드라이버를 사용하는 새 포트를 `NetworkInterface.c`라는 파일에 구현합니다. 템플릿을 보려면 GitHub 리포지토리를 방문하세요.

포트를 생성한 후 또는 포트가 이미 있는 경우 `FreeRTOSIPConfig.h`를 생성하고 사용 중인 플랫폼에 맞게 구성 옵션을 편집합니다. 구성 옵션에 대한 자세한 내용은 FreeRTOS.org의 FreeRTOS+TCP Configuration을 참조하십시오.

### 테스트

FreeRTOS+TCP 라이브러리를 사용하든 타사 라이브러리를 사용하든 아래 단계에 따라 테스트합니다.

- 전송 인터페이스 테스트에서 API에 대한 `connect/disconnect/send/receive` 구현을 제공합니다.
- 에코 서버를 일반 텍스트 TCP 연결 모드로 설정하고 전송 인터페이스 테스트를 실행합니다.

> **Note**
>
> FreeRTOS용 디바이스를 공식적으로 검증하려면 아키텍처에서 TCP/IP 소프트웨어 스택을 이식해야 하는 경우 일반 텍스트 TCP 연결 모드에서 전송 인터페이스 테스트와 비교하여 디바이스의 이식된 소스 코드를 검증해야 합니다 AWS IoT Device Tester. FreeRTOS 사용 설명서의 FreeRTOS AWS IoT Device Tester 용 사용의 지침에 따라 포트 검증 AWS IoT Device Tester 을 설정합니다. FreeRTOS 특정 라이브러리의 포트를 테스트하려면 Device Tester `configs` 폴더의 `device.json` 파일에서 올바른 테스트 그룹을 활성화해야 합니다.

## corePKCS11 라이브러리 이식

퍼블릭 키 암호화 표준 #11은 암호화 토큰을 관리하고 사용하기 위한 플랫폼 독립적 API를 정의합니다. PKCS 11은 표준 자체 및 표준에서 정의한 API를 의미합니다. PKCS #11 암호화 API는 키 스토리지, 암호화 객체에 대한 get/set 속성 및 세션 의미 체계를 추상화합니다. 일반적인 암호화 객체를 조작하는 데 널리 사용됩니다. 해당 함수를 통해 애플리케이션 소프트웨어는 암호화 객체를 애플리케이션 메모리에 노출하지 않고도 암호화 객체를 사용, 생성, 수정 및 삭제할 수 있습니다.

FreeRTOS 라이브러리 및 참조 통합은 비대칭 키, 난수 생성 및 해싱 관련 작업 중심으로 PCKS #11 인터페이스 표준의 일부만 사용합니다. 아래 표에는 사용 사례 및 지원해야 하는 필수 PKCS #11 API가 나와 있습니다.

사용 사례

| 사용 사례 | 필수 PKCS #11 API 제품군 |
| --- | --- |
| 모두 | Initialize, Finalize, Open/Close Session, GetSlotList, Login |
| 프로비저닝 | GenerateKeyPair, CreateObject, DestroyObject, InitToken, GetTokenInfo |
| TLS | Random, Sign, FindObject, GetAttributeValue |
| FreeRTOS+TCP | Random, Digest |
| OTA | Verify, Digest, FindObject, GetAttributeValue |

## 전체 PKCS #11 모듈을 구현해야 하는 시기

범용 플래시 메모리에 프라이빗 키를 저장하면 평가 및 신속한 프로토타입 생성 시나리오에서 편리할 수 있습니다. 프로덕션 시나리오에서는 데이터 도용 및 디바이스 복제의 위협을 줄이기 위해 전용 암호화 하드웨어를 사용하는 것이 좋습니다. 암호화 하드웨어에는 암호화 비밀 키의 내보내기를 방지하는 기능이 있는 구성 요소가 포함됩니다. 이를 지원하려면 위 표에 정의된 대로 FreeRTOS 라이브러리와 함께 작동하는 데 필요한 PKCS #11의 하위 집합을 구현해야 합니다.

## FreeRTOS corePKCS11을 사용해야 하는 시기

corePKCS11 라이브러리에는 Mbed TLS에서 제공하는 암호화 기능을 사용하는 PKCS #11 인터페이스(API)의 소프트웨어 기반 구현이 포함되어 있습니다. 이 구현은 하드웨어에 전용 암호화 하드웨어가 없는 경우 신속한 프로토타입 생성 및 평가 시나리오를 위해 제공됩니다. 이 경우, corePKCS11 소프트웨어 기반 구현이 하드웨어 플랫폼에서 작동하도록 하려면 corePKCS11 PAL만 구현하면 됩니다.

## corePKCS11 이식

온보드 플래시 메모리와 같은 비휘발성 메모리(NVM)에 암호화 객체를 읽고 쓸 수 있는 구현이 있어야 합니다. 암호화 객체는 초기화되지 않고 디바이스 재프로그래밍 시에도 지워지지 않는 NVM 섹션

에 저장해야 합니다. corePKCS11 라이브러리 사용자는 디바이스에 보안 인증 정보를 제공한 다음, corePKCS11 인터페이스를 통해 이러한 보안 인증 정보에 액세스하는 새 애플리케이션으로 디바이스를 다시 프로그래밍합니다. corePKCS11 PAL 포트는 다음 항목의 저장 위치를 제공해야 합니다.

- 디바이스 클라이언트 인증서
- 디바이스 클라이언트 프라이빗 키
- 디바이스 클라이언트 퍼블릭 키
- 신뢰할 수 있는 루트 CA
- 보안 부트 로더 및 무선 업데이트(OTA)를 위한 코드 확인 퍼블릭 키(또는 코드 확인 퍼블릭 키를 포함하는 인증서)
- JIT 프로비저닝 인증서

헤더 파일을 포함하고 정의된 PAL API를 구현합니다.

PAL API

| 함수 | 설명 |
| --- | --- |
| PKCS11_PAL_Initialize | PAL 계층을 초기화합니다. 초기화 시퀀스의 시작 부분에서 corePKCS11 라이브러리에 의해 호출됩니다. |
| PKCS11_PAL_SaveObject | 비휘발성 스토리지에 데이터를 씁니다. |
| PKCS11_PAL_FindObject | PKCS #11 `CKA_LABEL` 을 사용하여 비휘발성 스토리지에서 해당 PKCS #11 객체를 검색하고 해당 객체의 핸들이 있으면 반환합니다. |
| PKCS11_PAL_GetObjectValue | 핸들이 지정된 객체의 값을 가져옵니다. |
| PKCS11_PAL_GetObjectValueCleanup | `PKCS11_PAL_GetObjectValue` 호출을 위해 정리합니다. `PKCS11_PAL_GetObjectValue` 호출에 할당된 여유 메모리를 해제하는 데 사용할 수 있습니다. |

# 테스트

FreeRTOS corePKCS11 라이브러리를 사용하거나 PKCS11 API 중 필요한 하위 집합을 구현하는 경우 FreeRTOS PKCS11 테스트를 통과해야 합니다. 이를 통해 FreeRTOS 라이브러리의 필수 함수가 예상대로 작동하는지 테스트됩니다.

또한 이 섹션에서는 검증 테스트를 사용하여 로컬에서 FreeRTOS PKCS11 테스트를 실행하는 방법을 설명합니다.

## 사전 조건

FreeRTOS PKCS11 테스트를 설정하려면 다음을 구현해야 합니다.

- 지원되는 PKCS11 API 포트.
- 다음을 포함하는 FreeRTOS 검증 테스트 플랫폼 함수의 구현:
  - `FRTest_ThreadCreate`
  - `FRTest_ThreadTimedJoin`
  - `FRTest_MemoryAlloc`
  - `FRTest_MemoryFree`

(GitHub에서 PKCS #11 FreeRTOS 라이브러리 통합 테스트에 대한 README.md 파일을 참조하세요.)

## 이식 테스트

- FreeRTOS-Libraries-Integration-Tests를 하위 모델로 프로젝트에 추가합니다. 하위 모듈은 빌드가 가능하다면 프로젝트의 어느 디렉터리에든 배치할 수 있습니다.
- `config_template/test_execution_config_template.h` 및 `config_template/test_param_config_template.h`를 빌드 경로의 프로젝트 위치에 복사하고 이름을 `test_execution_config.h` 및 `test_param_config.h`로 바꿉니다.
- 관련 파일을 빌드 시스템에 포함합니다. `CMake`를 사용하는 경우 관련 파일을 포함하는 데 `src/pkcs11_tests.cmake` 및 `qualification_test.cmake`를 사용할 수 있습니다.
- 테스트 출력 로그와 디바이스 로그가 인터리브되지 않도록 `UNITY_OUTPUT_CHAR`을 구현합니다.
- cryptoki 작업 결과를 검증하는 MbedTLS를 통합합니다.
- 애플리케이션에서 `RunQualificationTest()`를 직접 호출합니다.

## 테스트 구성

PKCS11 테스트 제품군은 PKCS11 구현에 따라 구성되어야 합니다. 다음 표에는 `test_param_config.h` 헤더 파일에 있는 PKCS11 테스트에 필요한 구성이 나와 있습니다.

PKSC11 테스트 구성

| 구성 | 설명 |
| --- | --- |
| PKCS11_TEST_RSA_KEY_SUPPORT | 이식이 RSA 키 기능을 지원합니다. |
| PKCS11_TEST_EC_KEY_SUPPORT | 이식이 EC 키 기능을 지원합니다. |
| PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT | 이식이 프라이빗 키 가져오기를 지원합니다. 지원되는 키 기능이 활성화된 경우 테스트에서 RSA 및 EC 키 가져오기가 검증됩니다. |
| PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT | 이식이 키 페어 생성을 지원합니다. 지원되는 키 기능이 활성화된 경우 테스트에서 EC 키 페어 생성이 검증됩니다. |
| PKCS11_TEST_PREPROVISIONED_SUPPORT | 이식에 보안 인증 정보가 사전 프로비저닝 되어 있습니다. `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` 및 `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` 는 보안 인증 정보의 예입니다. |
| PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS | 테스트에 사용되는 프라이빗 키의 레이블입니다. |
| PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS | 테스트에 사용되는 퍼블릭 키의 레이블입니다. |
| PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS | 테스트에 사용되는 인증서의 레이블입니다. |

| 구성 | 설명 |
| --- | --- |
| PKCS11_TEST_JITP_CODEVERIFY_ROOT_CERT_SUPPORTED | 이식이 JITP용 스토리지를 지원합니다. JITP `codeverify` 테스트를 활성화하려면 이 값을 1로 설정합니다. |
| PKCS11_TEST_LABEL_CODE_VERIFICATION_KEY | JITP `codeverify` 테스트에 사용되는 코드 확인 키의 레이블입니다. |
| PKCS11_TEST_LABEL_JITP_CERTIFICATE | JITP `codeverify` 테스트에 사용되는 JITP 인증서의 레이블입니다. |
| PKCS11_TEST_LABEL_ROOT_CERTIFICATE | JITP `codeverify` 테스트에 사용되는 루트 인증서의 레이블입니다. |

FreeRTOS 라이브러리 및 참조 통합은 RSA 또는 타원 곡선 키와 같은 최소 하나의 키 기능 구성과 PKCS11 API에서 지원하는 하나의 키 프로비저닝 메커니즘을 지원해야 합니다. 테스트는 다음과 같은 구성을 활성화해야 합니다.

- 다음 키 기능 구성 중 하나 이상:
  - PKCS11_TEST_RSA_KEY_SUPPORT
  - PKCS11_TEST_EC_KEY_SUPPORT
- 다음 키 프로비저닝 구성 중 하나 이상:
  - PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT
  - PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT
  - PKCS11_TEST_PREPROVISIONED_SUPPORT

사전 프로비저닝된 디바이스 보안 인증 정보 테스트는 다음 조건에서 실행되어야 합니다.

- `PKCS11_TEST_PREPROVISIONED_SUPPORT`는 활성화되고 다른 프로비저닝 메커니즘은 비활성화해야 합니다.
- 키 기능이 `PKCS11_TEST_RSA_KEY_SUPPORT` 또는 `PKCS11_TEST_EC_KEY_SUPPORT` 중 하나만 활성화됩니다.
- `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS`,

PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS를 포함하여 키 기능에 따라 사전 프로비저닝된 키 레이블을 설정합니다. 테스트를 실행하기 전에 이러한 보안 인증 정보가 있어야 합니다.

구현이 사전 프로비저닝된 보안 인증 정보 및 기타 프로비저닝 메커니즘을 지원하는 경우 테스트를 다른 구성으로 여러 번 실행해야 할 수 있습니다.

> **Note**
>
> PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT 또는 PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT가 활성화되면 테스트 중에 PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS, PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS 및 PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS 레이블을 가진 객체가 삭제됩니다.

## 테스트 실행

이 섹션에서는 검증 테스트를 사용하여 로컬에서 PKCS11 인터페이스를 테스트하는 방법을 설명합니다. 또는 IDT를 사용하여 실행을 자동화할 수도 있습니다. 자세한 내용은 FreeRTOS 사용 설명서의 FreeRTOS용AWS IoT Device Tester를 참조하세요.

다음 지침에서는 테스트 실행 방법을 설명합니다.

- test_execution_config.h를 열고 CORE_PKCS11_TEST_ENABLED를 1로 정의합니다.
- 애플리케이션을 빌드하고 디바이스에 플래시하여 실행합니다. 테스트 결과가 직렬 포트로 출력됩니다.

다음은 출력된 테스트 결과의 예입니다.

```
TEST(Full_PKCS11_StartFinish, PKCS11_StartFinish_FirstTest) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_GetFunctionList) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_InitializeFinalize) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_GetSlotList) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_OpenSessionCloseSession) PASS
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities) PASS
TEST(Full_PKCS11_NoObject, PKCS11_Digest) PASS
```

```
TEST(Full_PKCS11_NoObject, PKCS11_Digest_ErrorConditions) PASS
TEST(Full_PKCS11_NoObject, PKCS11_GenerateRandom) PASS
TEST(Full_PKCS11_NoObject, PKCS11_GenerateRandomMultiThread) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_CreateObject) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_FindObject) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_GetAttributeValue) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_Sign) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_FindObjectMultiThread) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_GetAttributeValueMultiThread) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_DestroyObject) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_GenerateKeyPair) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_CreateObject) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_FindObject) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_GetAttributeValue) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_Sign) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_Verify) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_FindObjectMultiThread) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_GetAttributeValueMultiThread) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_SignVerifyMultiThread) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_DestroyObject) PASS

-----------------------
27 Tests 0 Failures 0 Ignored
OK
```

모든 테스트가 통과되면 테스트가 완료됩니다.

> **Note**
>
> FreeRTOS용 디바이스를 공식적으로 검증하려면 디바이스의 이식된 소스 코드를 검증해야 합니다 AWS IoT Device Tester. FreeRTOS 사용 설명서의 FreeRTOS AWS IoT Device Tester 용 사용의 지침에 따라 포트 검증 AWS IoT Device Tester 을 설정합니다. FreeRTOS 특정 라이브러리의 포트를 테스트하려면 `configs` 폴더의 `device.json` 파일 AWS IoT Device Tester 에서 올바른 테스트 그룹을 활성화해야 합니다.

# 네트워크 전송 인터페이스 이식

## TLS 라이브러리 통합

전송 계층 보안(TLS) 인증의 경우 선호하는 TLS 스택을 사용합니다. FreeRTOS 라이브러리로 테스트되었으므로 Mbed TLS를 사용하는 것이 좋습니다. 이 GitHub 리포지토리에서 예제를 찾을 수 있습니다.

디바이스에서 사용하는 TLS 구현에 관계없이 TCP/IP 스택을 사용하여 TLS 스택의 기본 전송 후크를 구현해야 합니다. AWS IoT에서 지원하는 TLS 암호 제품군을 지원해야 합니다.

## 네트워크 전송 인터페이스 라이브러리 이식

coreMQTT 및 CoreHTTP를 사용하려면 네트워크 전송 인터페이스를 구현해야 합니다. 네트워크 전송 인터페이스에는 단일 네트워크 연결에서 데이터를 보내고 받는 데 필요한 함수 포인터와 컨텍스트 데이터가 포함되어 있습니다. 자세한 내용은 전송 인터페이스를 참조하세요. FreeRTOS는 이러한 구현을 검증하기 위한 일련의 네트워크 전송 인터페이스 테스트를 기본 제공합니다. 다음 섹션에서는 이러한 테스트를 실행하도록 프로젝트를 설정하는 방법을 안내합니다.

## 사전 조건

이 테스트를 이식하려면 다음이 필요합니다.

- 검증된 FreeRTOS 커널 포트로 FreeRTOS를 빌드할 수 있는 빌드 시스템이 포함된 프로젝트.
- 정상 작동하는 네트워크 드라이버 구현.

## 이식

- FreeRTOS-Libraries-Integration-Tests를 하위 모델로 프로젝트에 추가합니다. 하위 모듈은 빌드가 가능하다면 프로젝트 내 어디에 배치하든 상관이 없습니다.
- `config_template/test_execution_config_template.h` 및 `config_template/test_param_config_template.h`를 빌드 경로의 프로젝트 위치에 복사하고 이름을 `test_execution_config.h` 및 `test_param_config.h`로 바꿉니다.
- 관련 파일을 빌드 시스템에 포함합니다. `CMake`를 사용하는 경우 `src/transport_interface_tests.cmake` 및 `qualification_test.cmake`가 관련 파일을 포함하는 데 사용됩니다.
- 적절한 프로젝트 위치에서 다음 함수를 구현합니다.

- • `network connect function`: 서명은 `src/common/network_connection.h`에서 `NetworkConnectFunc`에 의해 정의됩니다. 이 함수는 네트워크 컨텍스트에 대한 포인터, 호스트 정보에 대한 포인터, 네트워크 보안 인증 정보에 대한 포인터를 받습니다. 제공된 네트워크 보안 인증 정보를 사용하여 호스트 정보에 지정된 서버와의 연결을 설정합니다.
  - `network disconnect function`: 서명은 `src/common/network_connection.h`에서 `NetworkDisconnectFunc`에 의해 정의됩니다. 이 함수는 네트워크 컨텍스트에 대한 포인터를 받습니다. 네트워크 컨텍스트에 저장된 이전에 설정된 연결을 해제합니다.
  - `setupTransportInterfaceTestParam()`: 이 함수는 `src/transport_interface/transport_interface_tests.h`에 정의되어 있습니다. 구현은 `transport_interface_tests.h`에 정의된 것과 정확히 같은 이름 및 서명을 가져야 합니다. 이 함수는 TransportInterfaceTestParam 구조에 대한 포인터를 받습니다. 전송 인터페이스 테스트에서 사용되는 TransportInterfaceTestParam 구조의 필드를 채웁니다.
- 테스트 출력 로그가 디바이스 로그와 인터리브되지 않도록 UNITY_OUTPUT_CHAR을 구현합니다.
- 애플리케이션에서 `runQualificationTest()`를 직접 호출합니다. 직접 호출 전에 디바이스 하드웨어를 제대로 초기화하고 네트워크를 연결해야 합니다.

### 보안 인증 정보 관리(디바이스 내 생성 키)

`test_param_config.h`에서 FORCE_GENERATE_NEW_KEY_PAIR을 1로 설정한 경우 디바이스 애플리케이션은 새 디바이스 내 키 페어를 생성하고 퍼블릭 키를 출력합니다. 디바이스 애플리케이션은 에코 서버와 TLS 연결을 설정할 때 ECHO_SERVER_ROOT_CA 및 TRANSPORT_CLIENT_CERTIFICATE를 에코 서버 루트 CA 및 클라이언트 인증서로 사용합니다. IDT는 검증 실행 중에 이러한 파라미터를 설정합니다.

### 보안 인증 정보 관리(키 가져오기)

디바이스 애플리케이션은 에코 서버와 TLS 연결을 설정할 때 `test_param_config.h`의 ECHO_SERVER_ROOT_CA, TRANSPORT_CLIENT_CERTIFICATE 및 TRANSPORT_CLIENT_PRIVATE_KEY를 에코 서버 루트 CA, 클라이언트 인증서 및 클라이언트 프라이빗 키로 사용합니다. IDT는 검증 실행 중에 이러한 파라미터를 설정합니다.

## 테스트

이 섹션에서는 검증 테스트를 사용하여 로컬에서 전송 인터페이스를 테스트하는 방법을 설명합니다. 추가 세부 정보는 GitHub에서 FreeRTOS-Libraries-Integration-Tests의 transport_interface 섹션에 제공된 README.md 파일을 통해 확인할 수 있습니다.

또는 IDT를 사용하여 실행을 자동화할 수도 있습니다. 자세한 내용은 FreeRTOS 사용 설명서의 FreeRTOS용AWS IoT Device Tester를 참조하세요.

### 테스트 활성화

`test_execution_config.h`를 열고 TRANSPORT_INTERFACE_TEST_ENABLED를 1로 정의합니다.

### 테스트를 위해 에코 서버를 설정합니다.

로컬 테스트에는 테스트를 실행하는 디바이스에서 액세스할 수 있는 에코 서버가 필요합니다. 전송 인터페이스 구현이 TLS를 지원하는 경우 에코 서버는 TLS를 지원해야 합니다. 아직 이 구현이 없다면 FreeRTOS-Libraries-Integration-Tests GitHub 리포지토리에 에코 서버 구현이 있습니다.

### 테스트를 위한 프로젝트 구성

`test_param_config.h`에서 ECHO_SERVER_ENDPOINT 및 ECHO_SERVER_PORT를 이전 단계의 엔드포인트 및 서버 설정으로 업데이트합니다.

### 보안 인증 정보 설정(디바이스 내 생성 키)

- ECHO_SERVER_ROOT_CA를 에코 서버의 서버 인증서로 설정합니다.
- FORCE_GENERATE_NEW_KEY_PAIR를 1로 설정하여 키 페어를 생성하고 퍼블릭 키를 가져옵니다.
- 키 생성 후 FORCE_GENERATE_NEW_KEY_PAIR를 다시 0으로 설정합니다.
- 퍼블릭 키 및 서버 키와 인증서를 사용하여 클라이언트 인증서를 생성합니다.
- TRANSPORT_CLIENT_CERTIFICATE를 생성된 클라이언트 인증서로 설정합니다.

### 보안 인증 정보 설정(키 가져오기)

- ECHO_SERVER_ROOT_CA를 에코 서버의 서버 인증서로 설정합니다.
- TRANSPORT_CLIENT_CERTIFICATE를 사전 생성된 클라이언트 인증서로 설정합니다.
- TRANSPORT_CLIENT_PRIVATE_KEY를 사전 생성된 클라이언트 프라이빗 키로 설정합니다.

### 애플리케이션 빌드 및 플래시

원하는 도구 체인을 사용하여 애플리케이션을 빌드하고 플래시합니다.
`runQualificationTest()`가 간접 호출되면 전송 인터페이스 테스트가 실행됩니다. 테스트 결과가 직렬 포트로 출력됩니다.

> **Note**
>
> FreeRTOS용 디바이스를 공식적으로 검증하려면 디바이스의 이식된 소스 코드를 OTA PAL 및 OTA E2E 테스트 그룹과 비교하여 검증해야 합니다 AWS IoT Device Tester. FreeRTOS 사용 설명서의 FreeRTOS AWS IoT Device Tester 용 사용의 지침에 따라 포트 검증 AWS IoT Device Tester 을 설정합니다. FreeRTOS  특정 라이브러리의 포트를 테스트하려면 폴더의 `device.json` 파일 AWS IoT Device Tester `configs`에서 올바른 테스트 그룹을 활성화해야 합니다.

## coreMQTT 라이브러리 구성

엣지의 디바이스는 MQTT 프로토콜을 사용하여 AWS 클라우드와 통신할 수 있습니다. AWS IoT 는 엣지의 연결된 디바이스와 메시지를 주고받는 MQTT 브로커를 호스팅합니다.

coreMQTT 라이브러리는 FreeRTOS를 실행하는 디바이스에 대해 MQTT 프로토콜을 구현합니다. coreMQTT 라이브러리를 이식할 필요는 없지만, 검증을 위해서는 디바이스의 테스트 프로젝트가 모든 MQTT 테스트를 통과해야 합니다. 자세한 내용은 FreeRTOS 사용 설명서의 coreMQTT 라이브러리를 참조하세요.

### 사전 조건

coreMQTT 라이브러리 테스트를 설정하려면 네트워크 전송 인터페이스 포트가 필요합니다. 자세한 내용은 네트워크 전송 인터페이스 이식 섹션을 참조하세요.

### 테스트

coreMQTT 통합 테스트 실행:

- MQTT 브로커에 클라이언트 인증서를 등록합니다.
- `config`에서 브로커 엔드포인트를 설정하고 통합 테스트를 실행합니다.

# 참조 MQTT 데모 생성

coreMQTT 에이전트를 사용하여 모든 MQTT 작업에 대한 스레드 안전성을 처리하는 것이 좋습니다. 또한 사용자는 게시 및 구독 태스크와 애플리케이션이 TLS, MQTT 및 기타 FreeRTOS 라이브러리를 효과적으로 통합하는지 검증하기 위한 Device Advisor 테스트가 필요합니다.

FreeRTOS용 디바이스를 공식적으로 검증하려면 AWS IoT Device Tester MQTT 테스트 사례를 사용하여 통합 프로젝트를 검증합니다. 설정 및 테스트 지침은 AWS IoT Device Advisor 워크플로를 참조하세요. TLS 및 MQTT에 대한 필수 테스트 사례는 다음과 같습니다.

TLS 테스트 사례

| 테스트 사례 | 테스트 사례 | 필수 테스트 |
|---|---|---|
| TLS | TLS 연결 | 예 |
| TLS | TLS 지원 AWS IoT 암호 제품군 | 권장 암호 제품군 |
| TLS | TLS 비보안 서버 인증서 | 예 |
| TLS | TLS Incorrect Subject Name Servr Cert | 예 |

MQTT 테스트 사례

| 테스트 사례 | 테스트 사례 | 필수 테스트 |
|---|---|---|
| MQTT | MQTT Connect | 예 |
| MQTT | MQTT Connect Jitter Retries | 예, 경고 없음 |
| MQTT | MQTT Subscribe | 예 |
| MQTT | MQTT Publish | 예 |
| MQTT | MQTT ClientPuback QoS1 | 예 |
| MQTT | MQTT No Ack PingResp | 예 |

# coreHTTP 라이브러리 구성

엣지의 디바이스는 HTTP 프로토콜을 사용하여 엣지의 연결된 디바이스와 메시지를 주고 받는 HTTP 서버를 AWS Cloud. AWS IoT services 호스트와 통신할 수 있습니다.

## 테스트

테스트는 아래 단계를 따릅니다.

- AWS 또는 HTTP 서버를 사용하여 TLS 상호 인증을 위한 PKI를 설정합니다.
- CoreHTTP 통합 테스트를 실행합니다.

# AWS IoT over-the-air(OTA) 업데이트 라이브러리 이식

FreeRTOS 무선 업데이트(OTA)를 통해 다음을 수행할 수 있습니다.

- 새 펌웨어 이미지를 단일 디바이스, 디바이스 그룹 또는 전체 플릿에 배포합니다.
- 그룹에 추가되거나, 재설정되거나, 다시 프로비저닝되는 디바이스에 펌웨어를 배포합니다.
- 디바이스에 배포된 이후에 새 펌웨어의 신뢰성과 무결성을 확인합니다.
- 배포 진행 상황을 모니터링합니다.
- 실패한 배포를 디버깅합니다.
- 코드 서명을 사용하여 펌웨어에 디지털 서명합니다 AWS IoT.

자세한 내용은 AWS IoT Over-the-air Update Documentation과 함께 FreeRTOS 사용 설명서의 FreeRTOS 무선(OTA) 업데이트를 참조하세요.

OTA 에이전트 라이브러리를 사용하여 OTA 기능을 FreeRTOS 애플리케이션에 통합할 수 있습니다. 자세한 내용은 FreeRTOS 사용 설명서의 FreeRTOS OTA 업데이트 라이브러리를 참조하세요.

FreeRTOS 디바이스는 수신한 OTA 펌웨어 이미지에 대해 암호화 코드 서명 확인을 수행해야 합니다. 다음 알고리즘이 권장됩니다.

- ECDSA(Elliptic-Curve Digital Signature Algorithm)
- NIST P256 곡선
- SHA-256 해시

## 사전 조건

- 이식을 위한 작업 영역 및 프로젝트 설정의 지침을 완료합니다.
- 네트워크 전송 인터페이스 포트를 생성합니다.

  자세한 내용은 네트워크 전송 인터페이스 이식 단원을 참조하세요.
- coreMQTT 라이브러리를 통합합니다. FreeRTOS 사용 설명서에서 coreMQTT 라이브러리를 참조하세요.
- OTA 업데이트를 지원할 수 있는 부트 로더를 생성합니다.

## 플랫폼 이식

OTA 라이브러리를 새 디바이스로 이식하려면 OTA 이식 가능 추상화 계층(PAL) 구현을 제공해야 합니다. PAL API는 ota_platform_interface.h 파일에 정의되어 있으며, 이 파일에 대한 구현별 세부 정보를 제공해야 합니다.

| 함수 이름 | 설명 |
| --- | --- |
| `otaPal_Abort` | OTA 업데이트를 중지합니다. |
| `otaPal_CreateFileForRx` | 수신한 데이터 청크를 저장할 파일을 생성합니다. |
| `otaPal_CloseFile` | 지정된 파일을 닫습니다. 암호화 보호를 구현하는 스토리지를 사용 중인 경우 파일을 인증할 수 있습니다. |
| `otaPal_WriteBlock` | 지정된 파일의 지정된 오프셋에 데이터 블록을 씁니다. 성공 시, 함수가 기록된 바이트 수를 반환합니다. 그렇지 않으면 함수는 음수 오류 코드를 반환합니다. 블록 크기는 항상 2의 거듭제곱이며 정렬됩니다. 자세한 내용은 OTA 라이브러리 구성을 참조하세요. |
| `otaPal_ActivateNewImage` | 새 펌웨어 이미지를 활성화하거나 실행합니다. 일부 포트의 경우 디바이스가 프로그래밍 방식 |

| 함수 이름 | 설명 |
| --- | --- |
| | 으로 동기적으로 재설정되면 이 함수가 반환되지 않습니다. |
| otaPal_SetPlatformImageState | 플랫폼이 최신 OTA 펌웨어 이미지(또는 번들)를 수락하거나 거부하는 데 필요한 작업을 수행합니다. 이 함수를 구현하려면 보드(플랫폼) 세부 정보 및 아키텍처 설명서를 참조하세요. |
| otaPal_GetPlatformImageState | OTA 업데이트 이미지의 상태를 가져옵니다. |

디바이스에 내장된 지원 기능이 있는 경우 이 표에 있는 함수를 구현하십시오.

| 함수 이름 | 설명 |
| --- | --- |
| otaPal_CheckFileSignature | 지정된 파일의 서명을 검증합니다. |
| otaPal_ReadAndAssumeCertificate | 파일 시스템에서 지정된 서명자 인증서를 읽고 호출자에게 반환합니다. |
| otaPal_ResetDevice | 디바이스를 재설정합니다. |

> **Note**
>
> OTA 업데이트를 지원할 수 있는 부트로더가 있는지 확인합니다. AWS IoT 디바이스 부트 로더를 생성하기 위한 지침은 IoT 디바이스 부트로더 섹션을 참조하세요.

## E2E 및 PAL 테스트

OTA PAL 및 E2E 테스트를 실행합니다.

### E2E 테스트

OTA 엔드투엔드(E2E) 테스트는 디바이스의 OTA 기능을 확인하고 실제 시나리오를 시뮬레이션하는 데 사용됩니다. 이 테스트에는 오류 처리가 포함됩니다.

### 사전 조건

이 테스트를 이식하려면 다음이 필요합니다.

- AWS OTA 라이브러리가 통합된 프로젝트입니다. 자세한 내용을 알아보려면 OTA Library Porting Guide를 참조하세요.
- OTA 라이브러리를 사용하여 데모 애플리케이션을 이식하고 AWS IoT Core 와 상호 작용하여 OTA 업데이트를 수행합니다. OTA 데모 애플리케이션 이식을(를) 참조하세요.
- IDT 도구를 설정합니다. 그러면 OTA E2E 호스트 애플리케이션이 실행되어 다양한 구성으로 디바이스를 빌드, 플래시 및 모니터링하고 OTA 라이브러리 통합을 검증합니다.

### OTA 데모 애플리케이션 이식

OTA E2E 테스트에는 OTA 라이브러리 통합을 검증하기 위한 OTA 데모 애플리케이션이 있어야 합니다. 데모 애플리케이션에는 OTA 펌웨어 업데이트를 수행할 수 있는 기능이 있어야 합니다. FreeRTOS OTA 데모 애플리케이션은 FreeRTOS GitHub 리포지토리에서 찾을 수 있습니다. 데모 애플리케이션을 참조로 사용하고 사양에 따라 수정하는 것이 좋습니다.

### 이식 단계

1. OTA 에이전트를 초기화합니다.
2. OTA 애플리케이션 콜백 함수를 구현합니다.
3. OTA 에이전트 이벤트 처리 태스크를 생성합니다.
4. OTA 에이전트를 시작합니다.
5. OTA 에이전트 통계를 모니터링합니다.
6. OTA 에이전트를 종료합니다.

자세한 지침을 알아보려면 FreeRTOS OTA over MQTT - Entry point of the demo를 방문하세요.

### 구성

와 상호 작용하려면 다음 구성이 필요합니다. AWS IoT Core

- AWS IoT Core 클라이언트 자격 증명
  - Amazon Trust Services 엔드포인트를 사용하여 `Ota_Over_Mqtt_Demo/demo_config.h`에 democonfigROOT_CA_PEM을 설정합니다. 자세한 내용은 AWS 서버 인증을 참조하세요.

  - AWS IoT 클라이언트 자격 증명을 `Ota_Over_Mqtt_Demo/demo_config.h` 사용하여에서 democonfigCLIENT_CERTIFICATE_PEM 및 democonfigCLIENT_PRIVATE_KEY_PEM을 설정합니다. 클라이언트 인증서 및 프라이빗 키에 대해 알아보려면 AWS 클라이언트 인증 세부 정보를 참조하세요.
- 애플리케이션 버전
- OTA 제어 프로토콜
- OTA 데이터 프로토콜
- 코드 서명 보안 인증 정보
- 기타 OTA 라이브러리 구성

위의 정보는 FreeRTOS OTA 데모 애플리케이션의 `demo_config.h` 및 `ota_config.h`에서 찾을 수 있습니다. 자세한 내용을 알아보려면 FreeRTOS OTA over MQTT - Setting up the device를 방문하세요.

빌드 확인

데모 애플리케이션을 실행하여 OTA 작업을 실행합니다. 성공적으로 완료되면 OTA E2E 테스트를 계속 실행할 수 있습니다.

FreeRTOS OTA 데모에서는 FreeRTOS Windows 시뮬레이터에서 OTA 클라이언트 및 AWS IoT Core OTA 작업을 설정하는 방법에 대한 자세한 정보를 제공합니다. AWS OTA는 MQTT 및 HTTP 프로토콜을 모두 지원합니다. 자세한 내용은 다음 예제를 참조하세요.

- OTA over MQTT Demo on Windows Simulator
- OTA over HTTP Demo on Windows Simulator

IDT 도구를 사용한 테스트 실행

OTA E2E 테스트를 실행하려면 AWS IoT Device Tester (IDT)를 사용하여 실행을 자동화해야 합니다. 자세한 내용은 FreeRTOS 사용 설명서의 FreeRTOS용AWS IoT Device Tester를 참조하세요.

E2E 테스트 사례

| 테스트 사례 | 설명 |
| --- | --- |
| OTAE2EGreaterVersion | 정기 OTA 업데이트를 위한 해피 패스 테스트입니다. 새 버전으로 업데이트를 생성하여 디바이스가 성공적으로 업데이트합니다. |
| OTAE2EBackToBackDownloads | 이 테스트는 3개의 연속 OTA 업데이트를 생성합니다. 디바이스가 3회 연속 업데이트될 것으로 예상됩니다. |
| OTAE2ERollbackIfUnableToConnectAfterUpdate | 이 테스트에서는 새 펌웨어로 네트워크에 연결할 수 없는 경우 디바이스가 이전 펌웨어로 롤백되는지 확인합니다. |
| OTAE2ESameVersion | 이 테스트를 통해 버전이 동일하게 유지되는 경우 디바이스가 들어오는 펌웨어를 거부하는지 확인할 수 있습니다. |
| OTAE2EUnsignedImage | 이 테스트는 이미지가 서명되지 않은 경우 디바이스가 업데이트를 거부하는지 확인합니다. |
| OTAE2EUntrustedCertificate | 이 테스트는 펌웨어가 신뢰할 수 없는 인증서로 서명된 경우 디바이스가 업데이트를 거부하는지 확인합니다. |
| OTAE2EPreviousVersion | 이 테스트는 디바이스가 이전 업데이트 버전을 거부하는지 확인합니다. |
| OTAE2EIncorrectSigningAlgorithm | 디바이스마다 다른 서명 및 해싱 알고리즘을 지원합니다. 이 테스트는 OTA 업데이트가 지원되지 않는 알고리즘으로 생성된 경우 디바이스가 업데이트에 실패하는지 확인합니다. |
| OTAE2EDisconnectResume | 일시 중지 및 재개 기능에 대한 해피 패스 테스트입니다. 이 테스트는 OTA 업데이트를 생성하고 업데이트를 시작합니다. 그런 다음 동일한 클라이언트 ID(사물 이름)와 자격 증명을 AWS IoT |

| 테스트 사례 | 설명 |
| --- | --- |
| | Core 사용하여에 연결합니다. AWS IoT Core 그런 다음 디바이스의 연결을 해제합니다. 디바이스는 연결이 끊어졌음을 감지 AWS IoT Core하고 일정 기간이 지나면 일시 중지 상태로 전환한 후에 다시 연결하고 다운로드를 AWS IoT Core 재개합니다. |
| `OTAE2EDisconnectCancelUpdate` | 이 테스트는 OTA 작업이 일시 중지 상태에서 취소될 경우 디바이스가 스스로 복구될 수 있는지 확인합니다. 디바이스 연결을 AWS IoT Core끊은 후 OTA 업데이트를 취소한다는 점을 제외하고 `OTAE2EDisconnectResume` 테스트와 동일한 작업을 수행합니다. 새 업데이트가 생성됩니다. 디바이스는에 다시 연결하고 AWS IoT Core, 현재 업데이트를 중단하고, 대기 상태로 돌아가고, 다음 업데이트를 수락하고 완료해야 합니다. |
| `OTAE2EPresignedUrlExpired` | OTA 업데이트가 생성되면 S3 사전 서명된 URL의 수명을 구성할 수 있습니다. 이 테스트는 URL이 만료되어 다운로드를 완료할 수 없더라도 디바이스가 OTA를 수행할 수 있는지 확인합니다. 디바이스는 다운로드를 재개하기 위한 새 URL이 포함된 새 작업 문서를 요청해야 합니다. |
| `OTAE2E2UpdatesCancel1st` | 이 테스트에서는 두 개의 OTA 업데이트가 연속으로 생성됩니다. 디바이스가 첫 번째 업데이트를 다운로드 중이라고 보고하면 테스트는 첫 번째 업데이트를 강제 취소합니다. 디바이스는 현재 업데이트를 중단하고 두 번째 업데이트를 다운로드하여 완료해야 합니다. |

| 테스트 사례 | 설명 |
| --- | --- |
| `OTAE2ECancelThenUpdate` | 이 테스트에서는 두 개의 OTA 업데이트가 연속으로 생성됩니다. 디바이스가 첫 번째 업데이트를 다운로드 중이라고 보고하면 테스트는 첫 번째 업데이트를 강제 취소합니다. 디바이스는 현재 업데이트를 중단하고 두 번째 업데이트를 다운로드하여 완료해야 합니다. |
| `OTAE2EImageCrashed` | 이 테스트에서는 이미지가 충돌할 때 디바이스가 업데이트를 거부할 수 있는지 확인합니다. |

## PAL 테스트

### 사전 조건

네트워크 전송 인터페이스 테스트를 이식하려면 다음이 필요합니다.

- 유효한 FreeRTOS 커널 포트로 FreeRTOS를 빌드할 수 있는 프로젝트.
- 정상 작동하는 OTA PAL 구현.

### 이식

- FreeRTOS-Libraries-Integration-Tests를 하위 모델로 프로젝트에 추가합니다. 프로젝트에서 하위 모듈의 위치는 빌드가 가능한 위치여야 합니다.
- `config_template/test_execution_config_template.h` 및 `config_template/test_param_config_template.h`를 빌드 경로 내의 위치에 복사하고 이름을 `test_execution_config.h` 및 `test_param_config.h`로 바꿉니다.
- 관련 파일을 빌드 시스템에 포함합니다. `CMake`를 사용하는 경우 관련 파일을 포함하는 데 `src/ota_pal_tests.cmake` 및 `qualification_test.cmake`를 사용할 수 있습니다.
- 다음 함수를 구현하여 테스트를 구성합니다.
  - `SetupOtaPalTestParam()`: `src/ota/ota_pal_test.h`에 정의되어 있습니다. 구현은 `ota_pal_test.h`에 정의된 것과 동일한 이름 및 서명을 가져야 합니다. 현재는 이 함수를 구성할 필요가 없습니다.
- 테스트 출력 로그가 디바이스 로그와 인터리브되지 않도록 UNITY_OUTPUT_CHAR을 구현합니다.

- 애플리케이션에서 `RunQualificationTest()`를 직접 호출합니다. 직접 호출 전에 디바이스 하드웨어를 제대로 초기화하고 네트워크를 연결해야 합니다.

테스트

이 섹션에서는 OTA PAL 검증 테스트의 로컬 테스트에 대해 설명합니다.

테스트 활성화

`test_execution_config.h`를 열고 OTA_PAL_TEST_ENABLED를 1로 정의합니다.

`test_param_config.h`에서 다음 옵션을 업데이트합니다.

- OTA_PAL_TEST_CERT_TYPE: 사용할 인증서 유형을 선택합니다.
- OTA_PAL_CERTIFICATE_FILE: 해당하는 경우 디바이스 인증서의 경로입니다.
- OTA_PAL_FIRMWARE_FILE: 해당하는 경우 펌웨어 파일의 이름입니다.
- OTA_PAL_USE_FILE_SYSTEM: OTA PAL이 파일 시스템 추상화를 사용하는 경우 1로 설정합니다.

원하는 도구 체인을 사용하여 애플리케이션을 빌드하고 플래시합니다.
`RunQualificationTest()`가 호출되면 OTA PAL 테스트가 실행됩니다. 테스트 결과가 직렬 포트로 출력됩니다.

## OTA 태스크 통합

- OTA 에이전트를 현재 MQTT 데모에 추가합니다.
- 를 사용하여 OTA E2E(End to End) 테스트를 실행합니다 AWS IoT. 이렇게 하면 통합이 예상대로 작동하는지 확인할 수 있습니다.

> **Note**
>
> FreeRTOS용 디바이스를 공식적으로 검증하려면 디바이스의 이식된 소스 코드를 OTA PAL 및 OTA E2E 테스트 그룹과 비교하여 검증해야 합니다 AWS IoT Device Tester. FreeRTOS 사용 설명서의 FreeRTOS AWS IoT Device Tester 용 사용의 지침에 따라 포트 검증 AWS IoT Device Tester 을 설정합니다. FreeRTOS  특정 라이브러리의 포트를 테스트하려면 폴더의 `device.json` 파일 AWS IoT Device Tester `configs`에서 올바른 테스트 그룹을 활성화해야 합니다.

# IoT 디바이스 부트로더

자체 보안 부트 로더 애플리케이션을 제공해야 합니다. 설계 및 구현을 통해 보안 위협을 적절히 완화할 수 있는지 확인하십시오. 다음은 참조용 위협 모델링입니다.

## IoT 디바이스 부트로더에 대한 위협 모델링

### 배경

이 위협 모델에서 참조하는 임베디드 AWS IoT 디바이스는 클라우드 서비스와 상호 작용하는 마이크로컨트롤러 기반 제품입니다. 소비자, 상업용 또는 산업용 환경에서 이러한 디바이스를 배포할 수 있습니다. IoT 디바이스는 사용자, 환자, 기계 또는 환경에 대한 데이터를 수집할 수 있으며 공장 기계의 조명 전구 및 도어 잠금 기능을 제어할 수 있습니다.

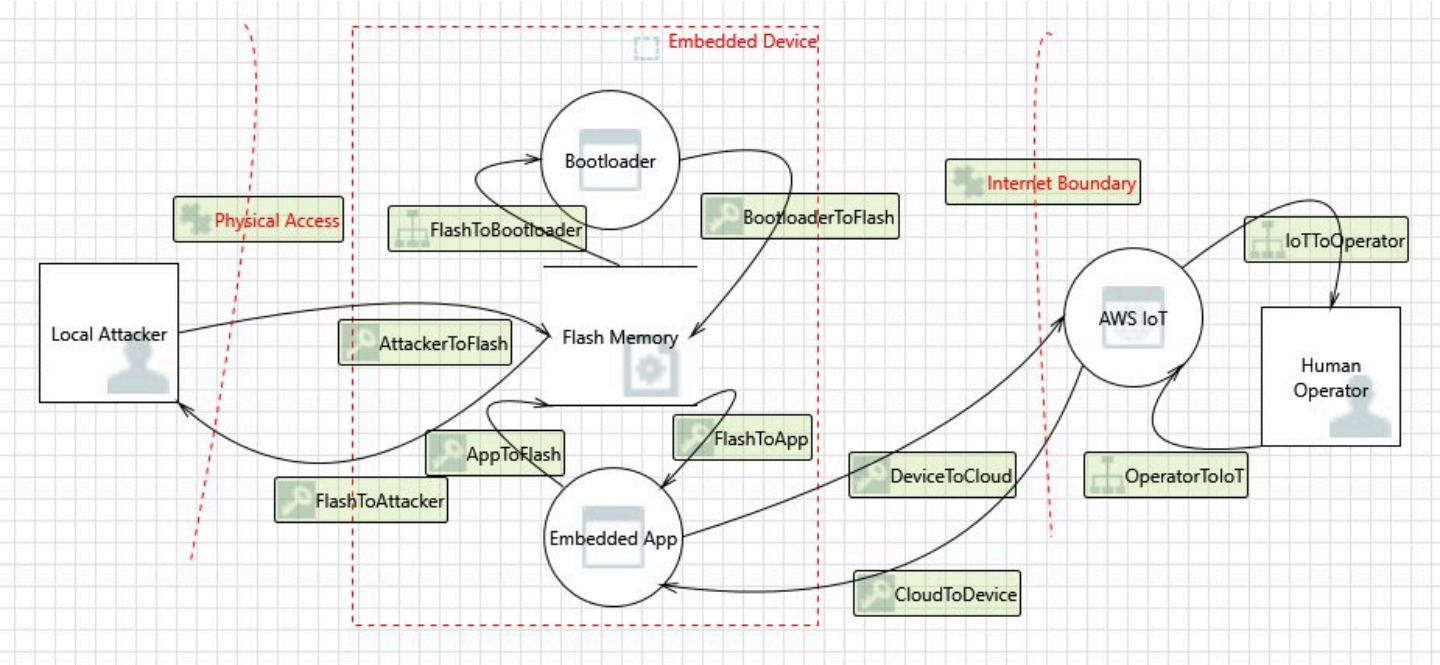
위협 모델링은 가상 악의적 사용자의 시점에서 보안에 대한 접근 방식입니다. 악의적 사용자의 목표와 방법을 고려하여 위협 목록이 생성됩니다. 위협은 악의적 사용자가 수행하는 리소스 또는 자산에 대한 공격입니다. 이 목록을 우선 순위 지정하고 사용하여 완화 솔루션을 식별하거나 생성합니다. 완화 솔루션을 선택할 때 해당 솔루션을 구현 및 유지 관리하는 비용은 해당 솔루션이 제공하는 보안 가치와 균형을 이루어야 합니다. 여러 가지 위협 모델 방법론이 있습니다. 각는 안전하고 성공적인 AWS IoT 제품 개발을 지원할 수 있습니다.

FreeRTOS는 AWS IoT 디바이스에 OTA(over-the-air) 소프트웨어 업데이트를 제공합니다. 시설 업데이트는 클라우드 서비스를 온디바이스 소프트웨어 라이브러리 및 파트너가 제공하는 부트로더와 결합합니다. 이 위협 모델은 특히 부트로더에 대한 위협에 중점을 둡니다.

### 부트로더 사용 사례

- 배포하기 전에 펌웨어를 디지털 방식으로 서명하고 암호화합니다.
- 새 펌웨어 이미지를 단일 디바이스, 디바이스 그룹 또는 전체 플릿에 배포합니다.
- 디바이스에 배포된 이후에 새 펌웨어의 신뢰성과 무결성을 확인합니다.
- 디바이스는 신뢰할 수 있는 소스의 수정되지 않은 소프트웨어만 실행합니다.
- 디바이스는 OTA를 통해 수신된 오류 소프트웨어에 대해 복원력이 있습니다.

### 데이터 흐름 다이어그램




## Threats

일부 공격에는 여러 가지 완화 모델이 있습니다. 예를 들어, 악성 펌웨어 이미지를 전달하는 네트워크 중간자 공격은 TLS 서버가 제공하는 인증서와 새 펌웨어 이미지의 코드 서명자 인증서 모두에 대한 신뢰를 검증하여 완화됩니다. 부트 로더 보안을 최대화하기 위해 부트 로더가 아닌 모든 완화 솔루션은 신뢰할 수 없는 것으로 간주됩니다. 부트 로더에는 각 공격에 대한 내장 완화 솔루션이 있어야 합니다. 계층적 완화 솔루션을 갖추는 것은 심층적 방어라고 알려져 있습니다.

위협:

- 공격자는 서버에 대한 디바이스의 연결을 하이재킹하여 악성 펌웨어 이미지를 전달합니다.

  완화 예

  - 부팅 시 부트로더가 알려진 인증서를 사용하여 이미지의 암호화 서명을 검증합니다. 검증이 실패하면 부트로더가 이전의 이미지로 롤백됩니다.
- 공격자는 버퍼 오버플로를 이용하여 플래시에 저장된 기존 펌웨어 이미지에 악성 동작을 도입합니다.

  완화 예

  - 앞의 설명과 같이, 부팅 시 부트로더가 검증합니다. 사용 가능한 이전 이미지가 없는 상태에서 검증이 실패하면 부트로더가 중지됩니다.

  - 앞의 설명과 같이, 부팅 시 부트로더가 검증합니다. 사용 가능한 이전 이미지가 없는 상태에서 검증이 실패하면 부트 로더가 페일세이프 OTA 전용 모드로 전환됩니다.
- 공격자는 이전에 저장된 악용 가능한 이미지로 디바이스를 부팅합니다.

  완화 예

  - 새 이미지를 성공적으로 설치하고 테스트하면 마지막 이미지를 저장하는 플래시 섹터가 지워집니다.
  - 성공적으로 업그레이드할 때마다 퓨즈가 버닝되고, 정확한 수의 퓨즈가 버닝되지 않으면 각 이미지가 실행을 거부합니다.
- OTA 업데이트가 디바이스를 구성하는 잘못된 이미지 또는 악성 이미지를 전달합니다.

  완화 예

  - 부트로더가 이전 이미지로 롤백을 트리거하는 하드웨어 watchdog 타이머를 시작합니다.
- 공격자는 디바이스가 서명되지 않은 이미지를 수락하도록 부트로더를 패치하여 이미지 검증을 우회합니다.

  완화 예

  - 부트로더가 ROM(읽기 전용 메모리)에 있으며 부트로더를 수정할 수 없습니다.
  - 부트로더가 OTP(일회성 프로그래밍 가능한 메모리)에 있으며 부트로더를 수정할 수 없습니다.
  - 부트로더가 ARM TrustZone의 보안 영역에 있으며 부트로더를 수정할 수 업습니다.
- 공격자는 디바이스가 악성 이미지를 수락하도록 검증 인증서를 교체합니다.

  완화 예

  - 인증서가 암호화 보조 프로세서에 있으며 인증서를 수정할 수 없습니다.
  - 인증서가 ROM(또는 OTP 또는 보안 영역)에 있으며 인증서를 수정할 수 없습니다.

### 추가 위협 모델링

이 위협 모델은 부트로더만 고려합니다. 추가 위협 모델링을 통해 전체 보안을 개선할 수 있습니다. 권장 방법은 악의적 사용자의 목표, 이러한 목표가 대상으로 하는 자산, 자산에 대한 진입점을 나열하는 것입니다. 자산을 제어하기 위해 진입점에 있는 공격자를 고려하여 위협 목록을 만들 수 있습니다. 다음은 IoT 디바이스에 대한 목표, 자산 및 진입점의 사례 목록입니다. 이러한 목록은 완전하지 않으며 더 많은 생각을 유도하기 위한 것입니다.

### 악의적 사용자의 목표

- 현금 갈취
- 평판 손상
- 데이터 위조
- 리소스 전용
- 대상에 대한 원격 염탐
- 사이트에 대한 물리적 액세스 획득
- 막대한 피해 초래
- 공포심 주입

### 핵심 자산

- 프라이빗 키
- 클라이언트 인증서
- CA 루트 인증서
- 보안 자격 증명 및 토큰
- 고객의 개인 식별 정보
- 기업 비밀 구현
- 센서 데이터
- 클라우드 분석 데이터 스토어
- 클라우드 인프라

### 진입점

- DHCP 응답
- DNS 응답
- MQTT over TLS
- HTTPS 응답
- OTA 소프트웨어 이미지
- 애플리케이션에 따라 결정되는 기타 항목(예: USB)
- 버스에 대한 물리적 액세스

- 캡슐화 해제된 IC

# 셀룰러 인터페이스 라이브러리 이식

FreeRTOS는 TCP 오프로드 셀룰러 추상화 계층의 AT 명령을 지원합니다. 자세한 내용은 freertos.org의 Cellular Interface Library 및 Porting the Cellular Interface Library를 참조하세요.

## 사전 조건

셀룰러 인터페이스 라이브러리에는 직접 종속성이 없습니다. 그러나 FreeRTOS 네트워크 스택에서는 이더넷, Wi-Fi 및 셀룰러가 공존할 수 없으므로 개발자는 둘 중 하나를 선택하여 네트워크 전송 인터페이스 이식와 통합해야 합니다.

> **Note**
>
> 셀룰러 모듈이 TLS 오프로드를 지원할 수 있거나 AT 명령을 지원하지 않는 경우 개발자는 자체 셀룰러 추상화를 구현하여 네트워크 전송 인터페이스 이식와 통합할 수 있습니다.

# MQTT 버전 3에서 coreMQTT로 마이그레이션

이 마이그레이션 가이드에서는 애플리케이션을 MQTT에서 coreMQTT로 마이그레이션하는 방법을 설명합니다.

# OTA 애플리케이션을 버전 1에서 버전 3으로 마이그레이션

이 가이드는 애플리케이션을 OTA 라이브러리 버전 1에서 버전 3으로 마이그레이션하는 데 도움이 됩니다.

> **Note**
>
> OTA 버전 2 API는 OTA v3 API와 동일하므로 애플리케이션이 API 버전 2를 사용하는 경우 API 호출을 변경할 필요는 없지만 라이브러리 버전 3을 통합하는 것이 좋습니다.

OTA 버전 3 데모는 다음에서 확인할 수 있습니다.

- ota_demo_core_mqtt.
- ota_demo_core_http.
- ota_ble.

## API 변경 사항 요약

OTA 라이브러리 버전 1과 버전 3 간의 API 변경 사항 요약

| OTAT 버전 1 API | OTA 버전 3 API | 변경 사항 설명 |
|---|---|---|
| OTA_AgentInit | OTA_Init | OTA v3의 구현 변경으로 인해 입력 파라미터와 함수에서 반환되는 값이 변경됩니다. 자세한 내용은 아래 OTA_Init 섹션을 참조하세요. |
| OTA_AgentShutdown | OTA_Shutdown | 선택적 MQTT 주제 구독 취소를 위한 추가 파라미터를 포함하여 입력 파라미터가 변경되었습니다. 자세한 내용은 아래 OTA_Shutdown 섹션을 참조하세요. |

| OTAT 버전 1 API | OTA 버전 3 API | 변경 사항 설명 |
| --- | --- | --- |
| OTA_GetAgentState | OTA_GetState | 입력 파라미터 변경 없이 API 이름이 변경되었습니다. 반환 값은 동일하지만 열거형 및 멤버의 이름이 변경되었습니다. 자세한 내용은 아래 OTA_GetState 섹션을 참조하세요. |
| 해당 사항 없음 | OTA_GetStatistics | 다음 API를 대체하는 새 API가 추가되었습니다. OTA_GetPacketsReceived, OTA_GetPacketsQueued, OTA_GetPacketsProcessed, OTA_GetPacketsDropped. 자세한 내용은 아래 OTA_GETStatistics 섹션을 참조하세요. |
| OTA_GetPacketsReceived | 해당 사항 없음 | 이 API는 버전 3에서 제거되었으며 OTA_GetStatistics로 대체되었습니다. |
| OTA_GetPacketsQueued | 해당 사항 없음 | 이 API는 버전 3에서 제거되었으며 OTA_GetStatistics로 대체되었습니다. |
| OTA_GetPacketsProcessed | 해당 사항 없음 | 이 API는 버전 3에서 제거되었으며 OTA_GetStatistics로 대체되었습니다. |
| OTA_GetPacketsDropped | 해당 사항 없음 | 이 API는 버전 3에서 제거되었으며 OTA_GetStatistics로 대체되었습니다. |

| OTAT 버전 1 API | OTA 버전 3 API | 변경 사항 설명 |
| --- | --- | --- |
| OTA_ActivateNewImage | OTA_ActivateNewImage | OTA 라이브러리 버전 3에서 입력 파라미터는 동일하지만 반환 OTA 오류 코드의 이름이 변경되고 새 오류 코드가 추가되었습니다. 자세한 내용은 OTA_ActivateNewImage 섹션을 참조하세요. |
| OTA_SetImageState | OTA_SetImageState | OTA 라이브러리 버전 3에서 입력 파라미터는 동일하지만 이름이 변경되고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다. 자세한 내용은 OTA_SetImageState 섹션을 참조하세요. |
| OTA_GetImageState | OTA_GetImageState | OTA 라이브러리 버전 3에서 입력 파라미터는 동일하고 반환 열거형의 이름이 변경되었습니다. 자세한 내용은 OTA_GetImageState 섹션을 참조하세요. |
| OTA_Suspend | OTA_Suspend | OTA 라이브러리 버전 3에서 입력 파라미터는 동일하고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다. 자세한 내용은 OTA_Suspend 섹션을 참조하세요. |

| OTAT 버전 1 API | OTA 버전 3 API | 변경 사항 설명 |
| --- | --- | --- |
| OTA_Resume | OTA_Resume | OTA 라이브러리 버전 3에서는 OTA 데모/애플리케이션이 연결을 처리하므로 연결을 위한 입력 파라미터가 제거되고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다. 자세한 내용은 OTA_Resume 섹션을 참조하세요. |
| OTA_CheckForUpdate | OTA_CheckForUpdate | OTA 라이브러리 버전 3에서 입력 파라미터는 동일하고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다. 자세한 내용은 OTA_CheckForUpdate 섹션을 참조하세요. |
| 해당 사항 없음 | OTA_EventProcessingTask | 새 API가 추가되었습니다. 이 API는 OTA 업데이트를 위한 이벤트를 처리하는 기본 이벤트 루프로, 애플리케이션 태스크에서 호출해야 합니다. 자세한 내용은 OTA_EventProcessingTask 섹션을 참조하세요. |
| 해당 사항 없음 | OTA_SignalEvent | 새 API가 추가되었습니다. 이 API는 OTA 이벤트 대기열 뒤에 이벤트를 추가하며, 내부 OTA 모듈이 에이전트 태스크에 신호를 보내는 데 사용됩니다. 자세한 내용은 OTA_SignalEvent 섹션을 참조하세요. |

| OTAT 버전 1 API | OTA 버전 3 API | 변경 사항 설명 |
| --- | --- | --- |
| 해당 사항 없음 | OTA_Err_strerror | OTA 오류의 오류 코드를 문자열로 변환하기 위한 새 API입니다. |
| 해당 사항 없음 | OTA_JobParse_strerror | 작업 파싱 오류의 오류 코드를 문자열로 변환하기 위한 새 API입니다. |
| 해당 사항 없음 | OTA_OsStatus_strerror | OTA OS 포트 상태의 상태 코드를 문자열로 변환하기 위한 새 API입니다. |
| 해당 사항 없음 | OTA_PalStatus_strerror | OTA PAL 포트 상태의 상태 코드를 문자열로 변환하기 위한 새 API입니다. |

# 필요한 변경에 대한 설명

## OTA_Init

v1에서 OTA 에이전트를 초기화할 때 연결 컨텍스트, 사물 이름, 전체 콜백 및 제한 시간에 대한 파라미터를 입력으로 받는 `OTA_AgentInit` API가 사용됩니다.

```
OTA_State_t OTA_AgentInit( void * pvConnectionContext,
                           const uint8_t * pucThingName,
                           pxOTACompleteCallback_t xFunc,
                           TickType_t xTicksToWait );
```

이 API는 이제 ota, ota 인터페이스, 사물 이름 및 애플리케이션 콜백에 필요한 버퍼의 파라미터를 사용하는 `OTA_Init`로 변경되었습니다.

```
OtaErr_t OTA_Init( OtaAppBuffer_t * pOtaBuffer,
                   OtaInterfaces_t * pOtaInterfaces,
                   const uint8_t * pThingName,
                   OtaAppCallback OtaAppCallback );
```

제거된 입력 파라미터 -

pvConnectionContext -

OTA 라이브러리 버전 3에서는 연결 컨텍스트를 전달할 필요가 없고 MQTT/HTTP 작업은 OTA 데모/애플리케이션의 해당 인터페이스에서 처리되므로 연결 컨텍스트가 제거되었습니다.

xTicksToWait -

OTA_Init를 호출하기 전에 OTA 데모/애플리케이션에서 태스크가 생성되므로 ticksToWait 파라미터도 제거되었습니다.

이름이 변경된 입력 파라미터 -

xFunc -

파라미터 이름이 OtaAppCallback으로 변경되고 유형이 OtaAppCallback_t로 변경되었습니다.

새 입력 파라미터 -

pOtaBuffer

애플리케이션은 초기화 중에 OtaAppBuffer_t 구조체를 사용하여 버퍼를 할당하고 OTA 라이브러리로 전달해야 합니다. 필요한 버퍼는 파일 다운로드에 사용되는 프로토콜에 따라 약간씩 다릅니다. MQTT 프로토콜의 경우 스트림 이름을 위한 버퍼가 필요하고 HTTP 프로토콜의 경우 사전 서명된 URL 및 권한 부여 체계를 위한 버퍼가 필요합니다.

MQTT를 파일 다운로드에 사용할 경우 필요한 버퍼 -

```
static OtaAppBuffer_t otaBuffer =
{
    .pUpdateFilePath    = updateFilePath,
    .updateFilePathsize = otaexampleMAX_FILE_PATH_SIZE,
    .pCertFilePath      = certFilePath,
    .certFilePathSize   = otaexampleMAX_FILE_PATH_SIZE,
    .pStreamName        = streamName,
    .streamNameSize     = otaexampleMAX_STREAM_NAME_SIZE,
    .pDecodeMemory      = decodeMem,
    .decodeMemorySize   = ( 1U << otaconfigLOG2_FILE_BLOCK_SIZE ),
    .pFileBitmap        = bitmap,
    .fileBitmapSize     = OTA_MAX_BLOCK_BITMAP_SIZE
};
```

HTTP를 파일 다운로드에 사용할 경우 필요한 버퍼 -

```
static OtaAppBuffer_t otaBuffer =
{
    .pUpdateFilePath    = updateFilePath,
    .updateFilePathsize = otaexampleMAX_FILE_PATH_SIZE,
    .pCertFilePath      = certFilePath,
    .certFilePathSize   = otaexampleMAX_FILE_PATH_SIZE,
    .pDecodeMemory      = decodeMem,
    .decodeMemorySize   = ( 1U << otaconfigLOG2_FILE_BLOCK_SIZE ),
    .pFileBitmap        = bitmap,
    .fileBitmapSize     = OTA_MAX_BLOCK_BITMAP_SIZE,
    .pUrl               = updateUrl,
    .urlSize            = OTA_MAX_URL_SIZE,
    .pAuthScheme        = authScheme,
    .authSchemeSize     = OTA_MAX_AUTH_SCHEME_SIZE
};
```

위치 -

```
    pUpdateFilePath    Path to store the files.
    updateFilePathsize Maximum size of the file path.
    pCertFilePath      Path to certificate file.
    certFilePathSize   Maximum size of the certificate file path.
    pStreamName        Name of stream to download the files.
    streamNameSize     Maximum size of the stream name.
    pDecodeMemory      Place to store the decoded files.
    decodeMemorySize   Maximum size of the decoded files buffer.
    pFileBitmap        Bitmap of the parameters received.
    fileBitmapSize     Maximum size of the bitmap.
    pUrl               Presigned url to download files from S3.
    urlSize            Maximum size of the URL.
    pAuthScheme        Authentication scheme used to validate download.
    authSchemeSize     Maximum size of the auth scheme.
```

pOtaInterfaces

OTA_Init에 대한 두 번째 입력 파라미터는 OtaInterfaces_t 유형의 OTA 인터페이스에 대한 참조입니다. 이 인터페이스 세트는 OTA 라이브러리로 전달되어야 하며 운영 체제 인터페이스에 MQTT 인터페이스, HTTP 인터페이스 및 플랫폼 추상화 계층 인터페이스를 포함해야 합니다.

OTAT OS 인터페이스

OTA OS 기능 인터페이스는 디바이스가 OTA 라이브러리를 사용할 수 있도록 구현해야 하는 API 세트입니다. 이 인터페이스의 기능 구현은 사용자 애플리케이션의 OTA 라이브러리

에 제공됩니다. OTA 라이브러리는 일반적으로 운영 체제에서 제공하는 기능을 수행하기 위해 기능 구현을 호출합니다. 여기에는 이벤트, 타이머, 메모리 할당 관리가 포함됩니다. FreeRTOS 및 POSIX의 구현이 OTA 라이브러리와 함께 제공됩니다.

제공된 FreeRTOS 포트를 사용하는 FreeRTOS의 예 -

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.os.event.init   = OtaInitEvent_FreeRTOS;
otaInterfaces.os.event.send   = OtaSendEvent_FreeRTOS;
otaInterfaces.os.event.recv   = OtaReceiveEvent_FreeRTOS;
otaInterfaces.os.event.deinit = OtaDeinitEvent_FreeRTOS;
otaInterfaces.os.timer.start  = OtaStartTimer_FreeRTOS;
otaInterfaces.os.timer.stop   = OtaStopTimer_FreeRTOS;
otaInterfaces.os.timer.delete = OtaDeleteTimer_FreeRTOS;
otaInterfaces.os.mem.malloc   = Malloc_FreeRTOS;
otaInterfaces.os.mem.free     = Free_FreeRTOS;
```

제공된 POSIX 포트를 사용하는 Linux의 예 -

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.os.event.init    = Posix_OtaInitEvent;
otaInterfaces.os.event.send    = Posix_OtaSendEvent;
otaInterfaces.os.event.recv    = Posix_OtaReceiveEvent;
otaInterfaces.os.event.deinit  = Posix_OtaDeinitEvent;
otaInterfaces.os.timer.start   = Posix_OtaStartTimer;
otaInterfaces.os.timer.stop    = Posix_OtaStopTimer;
otaInterfaces.os.timer.delete  = Posix_OtaDeleteTimer;
otaInterfaces.os.mem.malloc    = STDC_Malloc;
otaInterfaces.os.mem.free      = STDC_Free;
```

MQTT 인터페이스

OTA MQTT 인터페이스는 OTA 라이브러리가 스트리밍 서비스에서 파일 블록을 다운로드할 수 있도록 라이브러리에 구현해야 하는 API 세트입니다.

OTA over MQTT 데모의 coreMQTT 에이전트 API 사용 예제 -

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.mqtt.subscribe = prvMqttSubscribe;
otaInterfaces.mqtt.publish = prvMqttPublish;
otaInterfaces.mqtt.unsubscribe = prvMqttUnSubscribe;
```

HTTP 인터페이스

OTA HTTP 인터페이스는 OTA 라이브러리가 사전 서명된 URL에 연결하고 데이터 블록을 가져와서 파일 블록을 다운로드할 수 있도록 라이브러리에 구현해야 하는 API 세트입니다. 스트리밍 서비스 대신 사전 서명된 URL에서 다운로드하도록 OTA 라이브러리를 구성하지 않는 한 선택 사항입니다.

OTA over HTTP 데모의 coreMQTT API 사용 예제 -

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.http.init = httpInit;
otaInterfaces.http.request = httpRequest;
otaInterfaces.http.deinit = httpDeinit;
```

OTAA PAL 인터페이스

OTA PAL 인터페이스는 디바이스가 OTA 라이브러리를 사용할 수 있도록 구현해야 하는 API 세트입니다. OTA PAL용 디바이스별 구현은 사용자 애플리케이션의 라이브러리에 제공됩니다. 라이브러리는 이러한 기능을 사용하여 다운로드를 저장, 관리 및 인증합니다.

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.pal.getPlatformImageState = otaPal_GetPlatformImageState;
otaInterfaces.pal.setPlatformImageState = otaPal_SetPlatformImageState;
otaInterfaces.pal.writeBlock = otaPal_WriteBlock;
otaInterfaces.pal.activate = otaPal_ActivateNewImage;
otaInterfaces.pal.closeFile = otaPal_CloseFile;
otaInterfaces.pal.reset = otaPal_ResetDevice;
otaInterfaces.pal.abort = otaPal_Abort;
otaInterfaces.pal.createFile = otaPal_CreateFileForRx;
```

반환 변경 사항 -

반환이 OTA 에이전트 상태에서 OTA 오류 코드로 변경되었습니다. AWS IoT Over-the-air Update v3.0.0 : OtaErr_t를 참조하세요.

## OTA_Shutdown

OTA 라이브러리 버전 1에서 OTA 에이전트를 종료하는 데 사용되던 API OTA_AgentShutdown가 이제 OTA_Shutdown으로 변경되고 입력 파라미터도 변경되었습니다.

OTA 에이전트 종료(버전 1)

```
OTA_State_t OTA_AgentShutdown( TickType_t xTicksToWait );
```

OTA 에이전트 종료(버전 3)

```
OtaState_t OTA_Shutdown( uint32_t ticksToWait,
                         uint8_t unsubscribeFlag );
```

ticksToWait -

OTA 에이전트가 종료 프로세스를 완료할 때까지 기다려야 하는 틱 수입니다. 이 값을 0으로 설정하면 함수는 기다리지 않고 즉시 반환됩니다. 실제 상태가 호출자에게 반환됩니다. 이 동안에는 에이전트가 절전 모드로 전환되지 않고 비지 루핑에 사용됩니다.

새 입력 파라미터 -

unsubscribeFlag -

종료가 호출될 때 작업 주제에서 구독 취소 작업을 수행해야 하는지 여부를 나타내는 플래그입니다. 이 플래그가 0이면 작업 주제에 대한 구독 취소 작업이 호출되지 않습니다. 애플리케이션이 작업 주제 구독을 취소해야 하는 경우 Ota_Shutdown을 호출할 때 이 플래그를 1로 설정해야 합니다.

반환 변경 사항 -

OtaState_t -

OTA 에이전트 상태 및 해당 멤버의 열거형 이름이 변경되었습니다. AWS IoT Over-the-air Update v3.0.0을 참조하세요.

## OTA_GetState

API 이름이 OTA_AgentGetState에서 OTA_GetState로 변경되었습니다.

OTA 에이전트 종료(버전 1)

```
OTA_State_t OTA_GetAgentState( void );
```

OTA 에이전트 종료(버전 3)

```
OtaState_t OTA_GetState( void );
```

반환 변경 사항 -

OtaState_t -

OTA 에이전트 상태 및 해당 멤버의 열거형 이름이 변경되었습니다. AWS IoT Over-the-air Update v3.0.0을 참조하세요.

## OTA_GetStatistics

통계를 위한 새로운 단일 API가 추가되었습니다. 이 API는 다음 API를 대체합니다. OTA_GetPacketsReceived, OTA_GetPacketsQueued, OTA_GetPacketsProcessed, OTA_GetPacketsDropped. 또한 OTA 라이브러리 버전 3에서는 통계 수치가 현재 작업과만 관련이 있습니다.

OTAA 라이브러리 버전 1

```
uint32_t OTA_GetPacketsReceived( void );
uint32_t OTA_GetPacketsQueued( void );
uint32_t OTA_GetPacketsProcessed( void );
uint32_t OTA_GetPacketsDropped( void );
```

OTAA 라이브러리 버전 3

```
OtaErr_t OTA_GetStatistics( OtaAgentStatistics_t * pStatistics );
```

pStatistics -

현재 작업에 대해 수신, 삭제, 대기 및 처리된 패킷과 같은 통계 데이터에 대한 입력/출력 파라미터입니다.

출력 파라미터 -

OTA 오류 코드.

사용 예 -

```
OtaAgentStatistics_t otaStatistics = { 0 };
OTA_GetStatistics( &otaStatistics );
LogInfo( ( " Received: %u   Queued: %u   Processed: %u   Dropped: %u",
                            otaStatistics.otaPacketsReceived,
                            otaStatistics.otaPacketsQueued,
                            otaStatistics.otaPacketsProcessed,
```

```
                                otaStatistics.otaPacketsDropped ) );
```

## OTA_ActivateNewImage

OTA 라이브러리 버전 3에서 입력 파라미터는 동일하지만 반환 OTA 오류 코드의 이름이 변경되고 새 오류 코드가 추가되었습니다.

OTAA 라이브러리 버전 1

```
OTA_Err_t OTA_ActivateNewImage( void );
```

OTAA 라이브러리 버전 3

```
OtaErr_t OTA_ActivateNewImage( void );
```

반환 OTA 오류 코드 열거형이 변경되고 새 오류 코드가 추가되었습니다. AWS IoT Over-the-air Update v3.0.0 : OtaErr_t를 참조하세요.

사용 예 -

```
OtaErr_t otaErr = OtaErrNone;
otaErr = OTA_ActivateNewImage();
/* Handle error */
```

## OTA_SetImageState

OTA 라이브러리 버전 3에서 입력 파라미터는 동일하지만 이름이 변경되고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다.

OTAA 라이브러리 버전 1

```
OTA_Err_t OTA_SetImageState( OTA_ImageState_t eState );
```

OTAA 라이브러리 버전 3

```
OtaErr_t OTA_SetImageState( OtaImageState_t state );
```

입력 파라미터의 이름이 OtaImageState_t로 변경되었습니다. AWS IoT Over-the-air Update v3.0.0을 참조하세요.

반환 OTA 오류 코드 열거형이 변경되고 새 오류 코드가 추가되었습니다. AWS IoT Over-the-air Update v3.0.0 / OtaErr_t를 참조하세요.

사용 예 -

```
OtaErr_t otaErr = OtaErrNone;
otaErr = OTA_SetImageState( OtaImageStateAccepted );
/* Handle error */
```

## OTA_GetImageState

OTA 라이브러리 버전 3에서 입력 파라미터는 동일하고 반환 열거형의 이름이 변경되었습니다.

OTAA 라이브러리 버전 1

```
OTA_ImageState_t OTA_GetImageState( void );
```

OTAA 라이브러리 버전 3

```
OtaImageState_t OTA_GetImageState( void );
```

반환 열거형의 이름이 OtaImageState_t로 변경되었습니다. AWS IoT Over-the-air Update v3.0.0 : OtaImageState_t를 참조하세요.

사용 예 -

```
OtaImageState_t imageState;
imageState = OTA_GetImageState();
```

## OTA_Suspend

OTA 라이브러리 버전 3에서 입력 파라미터는 동일하고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다.

OTAA 라이브러리 버전 1

```
OTA_Err_t OTA_Suspend( void );
```

OTAA 라이브러리 버전 3

```
OtaErr_t OTA_Suspend( void );
```

반환 OTA 오류 코드 열거형이 변경되고 새 오류 코드가 추가되었습니다. AWS IoT Over-the-air Update v3.0.0 : OtaErr_t를 참조하세요.

사용 예 -

```
OtaErr_t xOtaError = OtaErrUninitialized;
xOtaError = OTA_Suspend();
/* Handle error */
```

## OTA_Resume

OTA 라이브러리 버전 3에서는 OTA 데모/애플리케이션이 연결을 처리하므로 연결을 위한 입력 파라미터가 제거되고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다.

OTAA 라이브러리 버전 1

```
OTA_Err_t OTA_Resume( void * pxConnection );
```

OTAA 라이브러리 버전 3

```
OtaErr_t OTA_Resume( void );
```

반환 OTA 오류 코드 열거형이 변경되고 새 오류 코드가 추가되었습니다. AWS IoT Over-the-air Update v3.0.0 : OtaErr_t를 참조하세요.

사용 예 -

```
OtaErr_t xOtaError = OtaErrUninitialized;
xOtaError = OTA_Resume();
/* Handle error */
```

## OTA_CheckForUpdate

OTA 라이브러리 버전 3에서 입력 파라미터는 동일하고, 반환 OTA 오류 코드의 이름이 변경되고, 새 오류 코드가 추가되었습니다.

OTAA 라이브러리 버전 1

```
OTA_Err_t OTA_CheckForUpdate( void );
```

OTAA 라이브러리 버전 3

```
OtaErr_t OTA_CheckForUpdate( void )
```

반환 OTA 오류 코드 열거형이 변경되고 새 오류 코드가 추가되었습니다. AWS IoT Over-the-air Update v3.0.0 : OtaErr_t를 참조하세요.

## OTA_EventProcessingTask

이 새 API는 OTA 업데이트 이벤트를 처리하는 기본 이벤트 루프입니다. 애플리케이션 태스크가 호출해야 합니다. 이 루프는 애플리케이션이 이 태스크를 종료할 때까지 OTA 업데이트에 대해 수신된 이벤트를 계속 처리하고 실행합니다.

OTAA 라이브러리 버전 3

```
void OTA_EventProcessingTask( void * pUnused );
```

FreeRTOS 예 -

```
/* Create FreeRTOS task*/
xTaskCreate( prvOTAAgentTask,
             "OTA Agent Task",
             otaexampleAGENT_TASK_STACK_SIZE,
             NULL,
             otaexampleAGENT_TASK_PRIORITY,
             NULL );

/* Call OTA_EventProcessingTask from the task */
static void prvOTAAgentTask( void * pParam )
{
    /* Calling OTA agent task. */
    OTA_EventProcessingTask( pParam );
    LogInfo( ( "OTA Agent stopped." ) );

    /* Delete the task as it is no longer required. */
    vTaskDelete( NULL );
```

```
}
```

POSIX 예 -

```
/* Create posix thread.*/
if( pthread_create( &threadHandle, NULL, otaThread, NULL ) != 0 )
{
    LogError( ( "Failed to create OTA thread: "
                ",errno=%s",
                strerror( errno ) ) );

   /* Handle error. */
}

/* Call OTA_EventProcessingTask from the thread.*/
static void * otaThread( void * pParam )
{
    /* Calling OTA agent task. */
    OTA_EventProcessingTask( pParam );
    LogInfo( ( "OTA Agent stopped." ) );

    return NULL;
}
```

## OTA_SignalEvent

이 새 API는 OTA 이벤트 대기열 뒤에 이벤트를 추가하며, 내부 OTA 모듈이 에이전트 태스크에 신호를 보내는 데 사용됩니다.

OTAA 라이브러리 버전 3

```
bool OTA_SignalEvent( const OtaEventMsg_t * const pEventMsg );
```

사용 예 -

```
OtaEventMsg_t xEventMsg = { 0 };
xEventMsg.eventId = OtaAgentEventStart;
( void ) OTA_SignalEvent( &xEventMsg );
```

# OTA 라이브러리를 애플리케이션의 하위 모듈로 통합

OTA 라이브러리를 자체 애플리케이션에 통합하려는 경우 git submodule 명령을 사용할 수 있습니다. Git 하위 모듈을 사용하면 Git 리포지토리를 다른 Git 리포지토리의 하위 디렉터리로 유지할 수 있습니다. OTA 라이브러리 버전 3은 ota-for-aws-iot-embedded-sdk 리포지토리에서 유지 관리됩니다.

```
git submodule add https://github.com/aws/ota-for-aws-iot-embedded-
sdk.git destination_folder
```

```
git commit -m "Added the OTA Library as submodule to the project."
```

```
git push
```

자세한 내용은 FreeRTOS 사용 설명서의 애플리케이션에 OTA 에이전트 통합을 참조하세요.

## 참조

- OTAv1.
- OTAv3.

# OTA PAL 포트의 버전 1에서 버전 3으로 마이그레이션

무선 업데이트(OTA) 라이브러리에서는 라이브러리 및 데모 애플리케이션에 필요한 폴더 구조와 구성 배치에 몇 가지 변경 사항을 도입했습니다. v1.2.0에서 작동하도록 설계된 OTA 애플리케이션을 라이브러리 v3.0.0으로 마이그레이션하려면 이 마이그레이션 가이드에 설명된 대로 PAL 포트 함수 서명을 업데이트하고 추가 구성 파일을 포함해야 합니다.

## OTA PAL 변경 사항

- OTA PAL 포트 디렉터리 이름이 `ota`에서 `ota_pal_for_aws`로 업데이트되었습니다. 이 폴더에는 두 개의 파일 `ota_pal.c` 및 `ota_pal.h`가 있어야 합니다. PAL 헤더 파일 `libraries/freertos_plus/aws/ota/src/aws_iot_ota_pal.h`가 OTA 라이브러리에서 삭제되었으므로 포트 내부에서 정의해야 합니다.
- 반환 코드(`OTA_Err_t`)가 열거형 `OTAMainStatus_t`로 변환됩니다. 변환된 반환 코드는 ota_platform_interface.h를 참조하세요. 또한 `OtaPalMainStatus` 및 `OtaPalSubStatus` 코드를 결합하고 `OtaPalStatus` 등에서 `OtaMainStatus`를 추출할 수 있는 헬퍼 매크로도 제공됩니다.
- PAL 로그인
  - `DEFINE_OTA_METHOD_NAME` 매크로를 제거했습니다.
  - 업데이트 전: `OTA_LOG_L1( "[%s] Receive file created.\r\n", OTA_METHOD_NAME );`.
  - 업데이트 후: `LogInfo(( "Receive file created."));` 적절한 로그를 위해 `LogDebug`, `LogWarn` 및 `LogError`를 사용합니다.
- 변수 `cOTA_JSON_FileSignatureKey`를 `OTA_JsonFileSignatureKey`로 변경했습니다.

### 함수

함수 서명은 `ota_pal.h`에서 정의되며 접두사 `prvPAL` 대신 `otaPal`로 시작합니다.

> **Note**
>
> PAL의 정확한 이름은 기술적으로 공개되어 있지만 검증 테스트와 호환되려면 아래에 지정된 이름을 따라야 합니다.

- 버전 1: `OTA_Err_t prvPAL_CreateFileForRx( OTA_FileContext_t * const *C* );`

버전 3: `OtaPalStatus_t otaPal_CreateFileForRx( OtaFileContext_t * const *pFileContext* );`

참고: 데이터 청크가 들어오는 대로 새 수신 파일을 생성합니다.

- 버전 1: `int16_t prvPAL_WriteBlock( OTA_FileContext_t * const C, uint32_t ulOffset, uint8_t * const pcData, uint32_t ulBlockSize );`

  버전 3: `int16_t otaPal_WriteBlock( OtaFileContext_t * const pFileContext, uint32_t ulOffset, uint8_t * const pData, uint32_t ulBlockSize );`

  참고: 지정된 파일의 지정된 오프셋에 데이터 블록을 씁니다.

- 버전 1: `OTA_Err_t prvPAL_ActivateNewImage( void );`

  버전 3: `OtaPalStatus_t otaPal_ActivateNewImage( OtaFileContext_t * const *pFileContext* );`

  참고: OTA를 통해 수신한 최신 MCU 이미지를 활성화합니다.

- 버전 1: `OTA_Err_t prvPAL_ResetDevice( void );`

  버전 3: `OtaPalStatus_t otaPal_ResetDevice( OtaFileContext_t * const *pFileContext* );`

  참고: 디바이스를 재설정합니다.

- 버전 1: `OTA_Err_t prvPAL_CloseFile( OTA_FileContext_t * const *C* );`

  버전 3: `OtaPalStatus_t otaPal_CloseFile( OtaFileContext_t * const *pFileContext* );`

  참고: 지정된 OTA 컨텍스트에서 기본 수신 파일을 인증하고 닫습니다.

- 버전 1: `OTA_Err_t prvPAL_Abort( OTA_FileContext_t * const *C* );`

  버전 3: `OtaPalStatus_t otaPal_Abort( OtaFileContext_t * const *pFileContext* );`

  참고: OTA 전송을 중지합니다.

- 버전 1: `OTA_Err_t prvPAL_SetPlatformImageState( OTA_ImageState_t *eState* );`

버전 3: `OtaPalStatus_t otaPal_SetPlatformImageState( OtaFileContext_t * const pFileContext, OtaImageState_t eState );`

참고: OTA 업데이트 이미지의 상태를 설정하려고 시도합니다.

- 버전 1: `OTA_PAL_ImageState_t prvPAL_GetPlatformImageState( void );`

  버전 3: `OtaPalImageState_t otaPal_GetPlatformImageState( OtaFileContext_t * const *pFileContext* );`

  참고: OTA 업데이트 이미지의 상태를 가져옵니다.

## 데이터 타입

- 버전 1: `OTA_PAL_ImageState_t`

  파일: `aws_iot_ota_agent.h`

  버전 3: `OtaPalImageState_t`

  파일: `ota_private.h`

  참고: 플랫폼 구현이 설정한 이미지 상태입니다.

- 버전 1: `OTA_Err_t`

  파일: `aws_iot_ota_agent.h`

  버전 3: `OtaErr_t OtaPalStatus_t (combination of OtaPalMainStatus_t and OtaPalSubStatus_t)`

  파일: `ota.h`, `ota_platform_interface.h`

  참고: v1: 32개의 부호 없는 정수를 정의하는 매크로였습니다. v3: 오류 유형을 나타내고 오류 코드와 연결된 특수 열거형입니다.

- 버전 1: `OTA_FileContext_t`

  파일: `aws_iot_ota_agent.h`

  버전 3: `OtaFileContext_t`

파일: ota_private.h

참고: v1: 데이터에 대한 열거형 및 버퍼를 포함합니다. v3: 추가 데이터 길이 변수를 포함합니다.

- 버전 1: OTA_ImageState_t

  파일: aws_iot_ota_agent.h

  버전 3: OtaImageState_t

  파일: ota_private.h

  참고: OTA 이미지 상태

## 구성 변경

파일 이름 aws_ota_agent_config.h가 ota_config.h로 변경되어 포함 가드가 _AWS_OTA_AGENT_CONFIG_H_에서 OTA_CONFIG_H_로 변경되었습니다.

- 파일 aws_ota_codesigner_certificate.h가 삭제되었습니다.
- 디버그 메시지를 인쇄하기 위한 새 로깅 스택이 포함되었습니다.

```
/************************************************/
/******* DO NOT CHANGE the following order ********/
/************************************************/

/* Logging related header files are required to be included in the following order:
 * 1. Include the header file "logging_levels.h".
 * 2. Define LIBRARY_LOG_NAME and  LIBRARY_LOG_LEVEL.
 * 3. Include the header file "logging_stack.h".
 */

/* Include header that defines log levels. */
#include "logging_levels.h"

/* Configure name and log level for the OTA library. */
#ifndef LIBRARY_LOG_NAME
    #define LIBRARY_LOG_NAME     "OTA"
#endif
#ifndef LIBRARY_LOG_LEVEL
    #define LIBRARY_LOG_LEVEL    LOG_INFO
```

```
#endif

#include "logging_stack.h"

/************ End of logging configuration ***************/
```

- 상수 구성 추가:

```
/** * @brief Size of the file data block message (excluding the header). */
#define otaconfigFILE_BLOCK_SIZE ( 1UL << otaconfigLOG2_FILE_BLOCK_SIZE )
```

새 파일: ota_demo_config.h가 코드 서명 인증서 및 애플리케이션 버전과 같이 OTA 데모에 필요한 구성을 포함합니다.

- demos/include/aws_ota_codesigner_certificate.h에서 정의된 ota_demo_config.h가 signingcredentialSIGNING_CERTIFICATE_PEM의 otapalconfigCODE_SIGNING_CERTIFICATE로 이동되었으며 PAL 파일에서 다음과 같이 액세스할 수 있습니다.

  ```
  static const char codeSigningCertificatePEM[] = otapalconfigCODE_SIGNING_CERTIFICATE;
  ```

  파일 aws_ota_codesigner_certificate.h가 삭제되었습니다.
- 매크로 APP_VERSION_BUILD, APP_VERSION_MINOR, APP_VERSION_MAJOR이 ota_demo_config.h에 추가되었습니다. 버전 정보가 들어 있는 이전 파일이 제거되었습니다(예: tests/include/aws_application_version.h, libraries/c_sdk/standard/common/include/iot_appversion32.h, demos/demo_runner/aws_demo_version.c).

## OTA PAL 테스트 변경 사항

- 'Full_OTA_Agent' 테스트 그룹 및 모든 관련 파일을 제거했습니다. 이전에는 검증에 이 테스트 그룹이 필요했습니다. 이들 테스트는 OTA 라이브러리를 대상으로 했으며 OTA PAL 포트에만 국한되지 않았습니다. 이제 OTA 라이브러리에는 OTA 리포지토리에서 호스팅되는 전체 테스트 범위가 제공되므로 이 테스트 그룹은 더 이상 필요하지 않습니다.
- 'Full_OTA_CBOR' 및 'Quarantine_OTA_CBOR" 테스트 그룹 및 모든 관련 파일을 제거했습니다. 이러한 테스트는 검증 테스트의 일부가 아닙니다. 이 테스트에서 다룬 기능은 현재 OTA 리포지토리에서 테스트되고 있습니다.

- 테스트 파일을 라이브러리 디렉터리에서 `tests/integration_tests/ota_pal` 디렉터리로 이동했습니다.
- OTA PAL 검증 테스트가 OTA 라이브러리 API v3.0.0을 사용하도록 업데이트되었습니다.
- OTA PAL 테스트가 테스트용 코드 서명 인증서에 액세스하는 방법을 업데이트했습니다. 이전에는 코드 서명 보안 인증을 위한 전용 헤더 파일이 있었습니다. 새 버전의 라이브러리에서는 더 이상 그렇지 않습니다. 테스트 코드에서는 이 변수가 `ota_pal.c`에 정의되어야 합니다. 값은 플랫폼별 OTA 구성 파일에 정의된 매크로에 할당됩니다.

## 체크리스트

이 체크리스트를 사용하여 마이그레이션에 필요한 단계를 따르고 있는지 확인합니다.

- ota pal port 폴더의 이름을 `ota`에서 `ota_pal_for_aws`로 업데이트합니다.
- 위에서 언급한 기능이 포함된 `ota_pal.h` 파일을 추가합니다. 예제 `ota_pal.h` 파일은 GitHub를 참조하세요.
- 구성 파일 추가:
  - 파일 이름을 `aws_ota_agent_config.h`에서 `ota_config.h`로 변경(또는 생성)합니다.
    - 추가:

      ```
      otaconfigFILE_BLOCK_SIZE ( 1UL << otaconfigLOG2_FILE_BLOCK_SIZE )
      ```

    - 포함:

      ```
      #include "ota_demo_config.h"
      ```

  - 위 파일을 `aws_test config` 폴더에 복사하고 `ota_demo_config.h`의 포함 항목을 `aws_test_ota_config.h`로 바꿉니다.
  - `ota_demo_config.h` 파일을 추가합니다.
  - `aws_test_ota_config.h` 파일을 추가합니다.
- `ota_pal.c`을 다음과 같이 변경합니다.
  - 포함 항목을 최신 OTA 라이브러리 파일 이름으로 업데이트합니다.
  - `DEFINE_OTA_METHOD_NAME` 매크로를 제거합니다.
  - OTA PAL 함수의 서명을 업데이트합니다.
  - 파일 컨텍스트 변수의 이름을 `C`에서 `pFileContext`로 업데이트합니다.
  - `OTA_FileContext_t` 구조체 및 모든 관련 변수를 업데이트합니다.

  - cOTA_JSON_FileSignatureKey를 OTA_JsonFileSignatureKey로 업데이트합니다.
  - OTA_PAL_ImageState_t 및 Ota_ImageState_t 유형을 업데이트합니다.
  - 오류 유형 및 값을 업데이트합니다.
  - 로깅 스택을 사용하도록 인쇄 매크로를 업데이트합니다.
  - signingcredentialSIGNING_CERTIFICATE_PEM를 otapalconfigCODE_SIGNING_CERTIFICATE로 업데이트합니다.
  - otaPal_CheckFileSignature 및 otaPal_ReadAndAssumeCertificate 함수 주석을 업데이트합니다.
- CMakeLists.txt 파일을 업데이트합니다.
- IDE 프로젝트를 업데이트합니다.

# 문서 이력

다음 표에서는 FreeRTOS 이식 안내서 및 FreeRTOS 검증 안내서의 문서 기록을 설명합니다.

| 날짜 | 문서 버전 | 변경 기록 | FreeRTOS 버전 |
| --- | --- | --- | --- |
| 2022년 5월 | FreeRTOS 이식 안내서<br><br>FreeRTOS 검증 안내서 | • 기존 테스트를 업데이트하고, 새 테스트를 추가하고, FreeRTOS 장기 지원(LTS) 라이브러리를 기반으로 하는 중복 테스트를 제거. 자세한 내용은 GitHub의 FreeRTOS Libraries Integration Tests 202205.00을 참조하세요.<br>• FreeRTOS 이식 순서도 업데이트됨<br>• 새 네트워크 전송 인터페이스 이식 추가됨<br>• AWS IoT over-the-air(OTA) 업데이트 라이브러리 이식은 이제 검증에 필요<br>• Wi-Fi 및 TLS 추상화 이식 안내서는 더 이상 필요하지 않으므로 제거됨<br>• FreeRTOS 검증에 대한 추가 업데이트 | 202012.04-LTS<br><br>202112.00 |

| 날짜 | 문서 버전 | 변경 기록 | FreeRTOS 버전 |
| --- | --- | --- | --- |
| | | 는 최신 변경 사항을 참조하세요. | |
| 2021년 7월 | 202107.00(이식 안내서)<br><br>202107.00(검증 안내서) | • 릴리스 202107.00<br>• AWS IoT over-the-air(OTA) 업데이트 라이브러리 이식 변경됨<br>• OTA 애플리케이션을 버전 1에서 버전 3으로 마이그레이션 추가됨<br>• OTA PAL 포트의 버전 1에서 버전 3으로 마이그레이션 추가됨 | 202107.00 |
| 2020년 12월 | 202012.00(이식 안내서)<br><br>202012.00(검증 안내서) | • 릴리스 202012.00<br>• coreHTTP 라이브러리 구성 추가됨<br>• 셀룰러 인터페이스 라이브러리 이식 추가됨 | 202012.00 |
| 2020년 11월 | 202011.00(이식 안내서)<br><br>202011.00(검증 안내서) | • 릴리스 202011.00<br>• coreMQTT 라이브러리 구성 추가됨 | 202011.00 |
| 2020년 7월 | 202007.00(이식 안내서)<br><br>202007.00(검증 안내서) | • 릴리스 202007.00 | 202007.00 |

| 날짜 | 문서 버전 | 변경 기록 | FreeRTOS 버전 |
| --- | --- | --- | --- |
| 2020년 2월 18일 | 202002.00(포팅 안내서)<br><br>202002.00(검증 안내서) | • 릴리스 202002.00<br>• Amazon FreeRTOS는 이제 FreeRTOS입니다. | 202002.00 |
| 2019년 12월 17일 | 201912.00(포팅 안내서)<br><br>201912.00(검증 안내서) | • 릴리스 201912.00<br>• 공통 I/O 라이브러리 이식 추가됨 | 201912.00 |
| 2019년 10월 29일 | 201910.00(포팅 안내서)<br><br>201910.00(검증 안내서) | • 릴리스 201910.00<br>• 난수 생성기 이식 정보가 업데이트되었습니다. | 201910.00 |
| 2019년 8월 26일 | 201908.00(포팅 안내서)<br><br>201908.00(검증 안내서) | • 릴리스 201908.00<br>• 테스트용 HTTPS 클라이언트 라이브러리 구성 추가됨<br><br>corePKCS11 라이브러리 이식 업데이트됨 | 201908.00 |
| 2019년 6월 17일 | 201906.00(이식 안내서)<br><br>201906.00(검증 안내서) | • 릴리스 201906.00<br>• 디렉터리 구조 업데이트 | 201906.00 메이저 |

| 날짜 | 문서 버전 | 변경 기록 | FreeRTOS 버전 |
| --- | --- | --- | --- |
| 2019년 5월 21일 | 1.4.8(이식 안내서)<br>1.4.8(검증 안내서) | • 이식 설명서를 FreeRTOS 이식 안내서로 이동<br>• 검증 설명서를 FreeRTOS 검증 안내서로 이동 | 1.4.8 |
| 2019년 2월 25일 | 1.1.6 | • 시작 안내서 템플릿 부록에서 다운로드 및 구성 지침 제거 (84페이지) | 1.4.5<br>1.4.6<br>1.4.7 |
| 2018년 12월 27일 | 1.1.5 | • CMake 요구 사항에 대한 검증 체크리스트 부록 업데이트 (70페이지) | 1.4.5<br>1.4.6 |
| 2018년 12월 12일 | 1.1.4 | • TCP/IP 이식 부록에 lwIP 이식 지침 추가 (31페이지) | 1.4.5 |
| 2018년 11월 26일 | 1.1.3 | • 블루투스 로우 에너지 이식 부록 추가 (52페이지)<br>• 문서 전체에서 FreeRTOS용 AWS IoT 디바이스 테스터 테스트 정보 추가<br>• FreeRTOS 콘솔 부록의 등록에 대한 정보에 CMake 링크 추가(85페이지) | 1.4.4 |

| 날짜 | 문서 버전 | 변경 기록 | FreeRTOS 버전 |
| --- | --- | --- | --- |
| 2018년 11월 7일 | 1.1.2 | • PKCS #11 이식 부록의 PKCS #11 PAL 인터페이스 이식 지침 업데이트 (38페이지)<br>• `CertificateConfigurator.html` 경로 업데이트(76페이지)<br>• 시작 안내서의 템플릿 부록 업데이트 (80페이지) | 1.4.3 |

| 날짜 | 문서 버전 | 변경 기록 | FreeRTOS 버전 |
| --- | --- | --- | --- |
| 2012년 10월 8일 | 1.1.1 | • aws_test_runner_config.h 테스트 구성 표에 새 “AFQP에 필요” 열 추가(16페이지)<br>• 테스트 프로젝트 만들기 섹션의 Unity 모듈 디렉터리 경로 업데이트(14페이지)<br>• "권장 이식 순서" 차트 업데이트(22페이지)<br>• TLS 부록, 테스트 설정의 클라이언트 인증서 및 키 변수 이름 업데이트(40페이지)<br>• 보안 소켓 이식 부록과 테스트 설정(34페이지), TLS 이식 부록과 테스트 설정(40페이지) 및 TLS 서버 설정 부록(57페이지)의 파일 경로 변경 | 1.4.2 |
| 2018년 8월 27일 | 1.1.0 | • OTA 업데이트 이식 부록 추가(47페이지)<br>• 부트로더 이식 부록 추가(51페이지) | 1.4.0<br>1.4.1 |

| 날짜 | 문서 버전 | 변경 기록 | FreeRTOS 버전 |
| --- | --- | --- | --- |
| 2018년 8월 9일 | 1.0.1 | • "권장 이식 순서" 차트 업데이트(22페이지)<br>• PKCS #11 이식 부록 업데이트(36페이지)<br>• TLS 이식 부록, 테스트 설정(40페이지) 및 TLS 서버 설정 부록, 9단계(51페이지)의 파일 경로 변경<br>• MQTT 이식 부록, 사전 조건의 하이퍼링크 수정(45페이지)<br>• BYOC 생성 지침 부록의 예제에 AWS CLI 구성 지침 추가(57페이지) | 1.3.1<br>1.3.2 |
| 2018년 7월 31일 | 1.0.0 | FreeRTOS 검증 프로그램 안내서의 초기 버전 | 1.3.0 |

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.