7.5

모바일 메시징과 *M2M*

참고

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 179 페이지의 『주의사항』에 있는 정보를 확인하십시오.

This edition applies to version 7 release 5 of IBM® WebSphere® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

목차

MQTT 소개

MQ 텔레메트리 전송을 사용하여 모바일 앱 간에 메시지를 전송하는 방법에 대해 학습하십시오 (MQTT). 이 프로토콜은 무선 및 저대역폭 네트워크에서 사용하기 위한 것입니다. MQTT를 사용하는 모바일 애플리케이션은 MQTT 라이브러리를 호출하여 메시지를 송수신합니다. 메시지는 MQTT 메시징 서버를 통해 교환됩니다. MQTT 클라이언트와 서버는 모바일 앱에 대한 메시지 전달의 복잡도를 신뢰성 있게 처리하고 네트워크 관리 비용을 낮게 유지합니다.

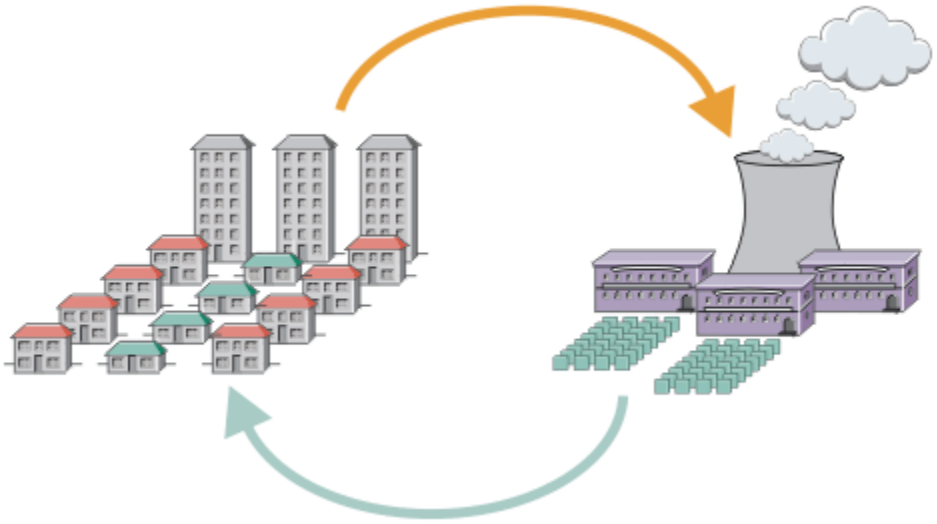
MQTT 애플리케이션은 스마트폰 및 태블릿과 같은 모바일 디바이스에서 실행됩니다. MQTT는 Telemetry가 센서에서 데이터를 수신하여 원격으로 제어하는 데도 사용됩니다. 모바일 디바이스 및 센서에 대해 MQTT는 전달이 보장된 확장성 높은 발행/구독 프로토콜을 제공합니다. MQTT 메시지를 송수신하려면 MQTT 클라이언트 라이브러리를 애플리케이션에 추가하십시오.

MQTT 클라이언트 라이브러리는 작습니다. 이 라이브러리는 편지함 역할을 수행하여 MQTT 서버에 연결된 다른 MQTT 애플리케이션과 메시지를 송수신합니다. 응답을 기다리는 서버에 연결된 상태를 유지하는 대신 메시지를 송신하여 MQTT 애플리케이션은 배터리 수명을 유지합니다. 이 라이브러리는 MQTT version 3.1 프로토콜을 실행 중인 MQTT 서버를 통해 다른 디바이스에 메시지를 송신합니다. 특정 클라이언트에 메시지를 송신하거나 발행/구독 메시징을 사용하여 다수의 디바이스에 연결할 수 있습니다.

MQTT 클라이언트 라이브러리는 MQTT 프로토콜을 사용하여 모바일 디바이스와 센서에 대한 애플리케이션을 MQTT 서버에 연결합니다.

IBM MessageSight 및 IBM WebSphere MQ 는 MQTT 서버입니다. 다량의 MQTT 클라이언트 애플리케이션을 연결할 수 있고 MQTT 및 IBM WebSphere MQ 네트워크를 함께 연결할 수 있습니다. 130 페이지의 『MQTT 서버 시작하기』 을(를) 참조하십시오. IBM WebSphere MQ와 IBM MessageSight 모두 모바일 디바이스와 센서로 실행되는 외부 웹 애플리케이션과, 엔터프라이즈 내에서 실행되는 다른 유형의 발행/구독 및 메시징 애플리케이션 사이에서 브릿지를 형성할 수 있습니다. 브릿지를 사용하면 모바일 디바이스와 센서를 통합하는 "스마트 솔루션"을 더 편리하게 빌드할 수 있습니다.

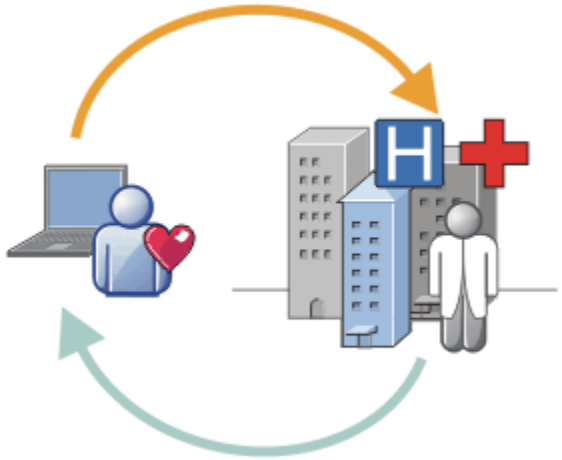
스마트 솔루션은 인터넷에서 사용 가능한 풍부한 정보를 모바일 및 센서 디바이스에서 실행 중인 애플리케이션에 제공합니다. 텔레메트리를 기반으로 하는 스마트 애플리케이션의 두 가지 예는 스마트 전기 및 스마트 건강 서비스입니다.



- 서비스 제공자에게 전송되는 에너지 사용 데이터가 들어 있는 MQTT 메시지입니다.
- 텔레메트리 애플리케이션은 에너지 사용량 데이터에 대한 분석을 기반으로 한 제어 명령을 송신합니다.
- 자세한 정보는 텔레메트리 시나리오: 가정 에너지 모니터링 및 제어를 참조하십시오.

그림 *1.* 스마트 전기 계량

그림 *2.* 스마트 건강 모니터링



- 텔레메트리 애플리케이션은 건강 데이터를 병원과 의사에게 송신합니다.
- 건강 데이터에 대한 분석을 기반으로 MQTT 메시지 경보 또는 피드백을 송신할 수 있습니다.
- 자세한 정보는 텔레메트리 시나리오: 가정 환자 모니터링을 참조하십시오.

MQTT protocol에 대한 사용자 고유의 앱을 작성하여 MQTT 를 소형 디바이스에 빌드할 수 있습니다. 이를 지원하기 위해 IBM에서는 MQTT를 통해 실행되는 앱을 지원하는 클라이언트 라이브러리를 제공합니다. 9 페이지의 『MQTT 클라이언트 시작하기』 의 내용을 참조하십시오. IBM은 iOS 앱과 Android 앱용 클라이언트 라이브러리를 제공V 7.5.0.1하고 플랫폼에 관계 없이 사용 가능한 웹 앱용 JavaScript 브라우저 클라이언트를 제공합니다. V 7.5.0.1 JavaScript 클라이언트 페이지는 WebSockets를 통해 MQTT 프로토콜을 사용하여 IBM MessageSight 및 IBM WebSphere MQ 에 연결합니다. IBM 에서는 Linux® 및 Windows의 C및 Java 에 대한 MQTT 샘플 앱도 제공합니다.

C 및 Java 라이브러리는 iOS, Android, Windows 및 다수의 UNIX and Linux 플랫폼에서 실행됩니다. MQTT 클라이언트 라이브러리에 대한 C 소스 코드를 다른 플랫폼에 이식할 수 있습니다. C와 Java용 MQTT 클라이언트 라이브러리는 Eclipse Paho 프로젝트의 오픈 소스 라이센스와 함께 사용할 수 있습니다. Eclipse Paho을 참조하십시오. MQTT protocol 스펙은 공개되어 있으며 MQTT.org에서 제공합니다.

MQTT protocol

MQTT protocol은 클라이언트가 작다는 의미에서 경량이며 네트워크 대역폭을 효율적으로 사용합니다. MQTT 프로토콜은 보장된 전달과 전송 후 삭제 전송을 지원합니다. 이 프로토콜에서 메시지 전달은 애플리케이션에서 분리됩니다. 애플리케이션에서 분리 범위는 MQTT 클라이언트 및 MQTT 서버가 작성되는 방식에 따라 다릅니다. 분리된 전달은 서버 연결에서 애플리케이션을 분리하고 메시지를 기다리지 않게 합니다. 상호작용 모델은 이메일과 비슷하지만 애플리케이션 프로그래밍에 맞게 최적화되어 있습니다.

MQTT V3.1 프로토콜이 발행됩니다. MQTT V3.1 프로토콜 스펙의 내용을 참조하십시오. 스펙은 프로토콜에 대한 다수의 구별되는 기능을 식별합니다.

- 발행/구독 프로토콜입니다.

  일대다 메시지 분배를 제공하는 것 외에도 발행/구독은 애플리케이션을 분리합니다. 두 기능 모두 다수의 클라이언트가 있는 애플리케이션에서 유용합니다.
- 메시지 컨텐츠에 종속되지 않습니다.
- TCP/IP를 통해 실행되어 기본 네트워크 연결을 제공합니다.
- 메시지 전달에 대한 세 가지 서비스 품질을 가지고 있습니다.

  "많아야 한 번"

  메시지는 기본 인터넷 프로토콜 네트워크의 최고 효율에 따라 전달됩니다. 메시지 유실이 발생할 수 있습니다.

  예를 들어, 통신하는 주위 센서 데이터와 함께 이 서비스 품질을 사용하십시오. 다음 수치가 곧 발행되는 경우에는 개별 수치가 유실되는지 여부는 중요하지 않습니다.

"적어도 한 번"
메시지는 확실히 도달하지만 중복이 발생할 수 있습니다.

"정확히 한 번"
메시지는 확실하게 정확히 한 번 도달합니다.

예를 들어, 청구 시스템과 함께 이 서비스 품질을 사용하십시오. 메시지가 중복되거나 유실되면 불편하거나 잘못된 요금이 부과될 수 있습니다.

- 네트워크에서 메시지 플로우를 관리하는 방식 면에서 경제적입니다. 예를 들어, 고정 길이 헤더의 길이는 2바이트이며 프로토콜 교환은 최소화되어 네트워크 트래픽을 줄입니다.
- 여기에는 구독자에게 MQTT 서버로부터 클라이언트의 비정상적인 연결 끊기를 알리는 "이상 종료 시 메시지" 기능이 있습니다. 124 페이지의 『이상 종료 시 메시지 발행』의 내용을 참조하십시오.

MQTT version 3.1은 IBM IBM WebSphere MQ 및 IBM MessageSight에 의해 지원됩니다. MQTT는 TCP/IP를 통해 구현됩니다. 다른 버전의 프로토콜(MQTT-S)을 비TCP/IP 네트워크에 사용할 수 있습니다. MQTT-S version 1.2 스펙을 참조하십시오.

MQTT 커뮤니티

IBM 는 IBM MessageSight 및 IBM WebSphere MQ용 애플리케이션을 작성하는 MQTT 개발자를 위해 IBM Developer 메시징 커뮤니티 를 실행 중입니다.

MQTT.org는 MQTT 프로토콜에 대한 구현 및 확장에 대해 알아보고 토론하기에 적합한 위치입니다.

MQTT는 Eclipse기술 프로젝트 아래의 오픈 소스 Eclipse 프로젝트입니다. Paho 커뮤니티는 오픈 소스 클라이언트와 서버를 개발 중입니다. Eclipse Paho을 참조하십시오.

MQTT 소개

MQ 텔레메트리 전송을 사용하여 모바일 앱 간에 메시지를 전송하는 방법에 대해 학습하십시오 (MQTT). 이 프로토콜은 무선 및 저대역폭 네트워크에서 사용하기 위한 것입니다. MQTT를 사용하는 모바일 애플리케이션은 MQTT 라이브러리를 호출하여 메시지를 송수신합니다. 메시지는 MQTT 메시징 서버를 통해 교환됩니다. MQTT 클라이언트와 서버는 모바일 앱에 대한 메시지 전달의 복잡도를 신뢰성 있게 처리하고 네트워크 관리 비용을 낮게 유지합니다.

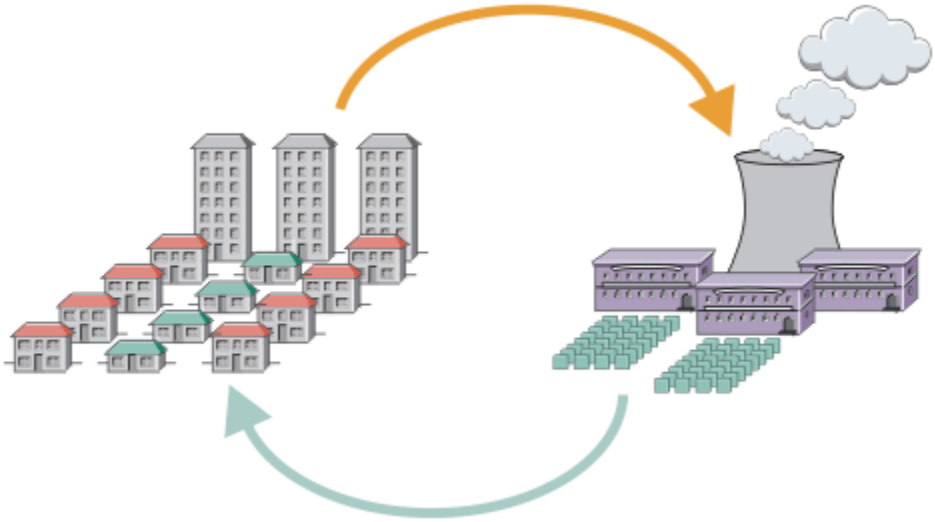
MQTT 애플리케이션은 스마트폰 및 태블릿과 같은 모바일 디바이스에서 실행됩니다. MQTT는 Telemetry가 센서에서 데이터를 수신하여 원격으로 제어하는 데도 사용됩니다. 모바일 디바이스 및 센서에 대해 MQTT는 전달이 보장된 확장성 높은 발행/구독 프로토콜을 제공합니다. MQTT 메시지를 송수신하려면 MQTT 클라이언트 라이브러리를 애플리케이션에 추가하십시오.

MQTT 클라이언트 라이브러리는 작습니다. 이 라이브러리는 편지함 역할을 수행하여 MQTT 서버에 연결된 다른 MQTT 애플리케이션과 메시지를 송수신합니다. 응답을 기다리는 서버에 연결된 상태를 유지하는 대신 메시지를 송신하여 MQTT 애플리케이션은 배터리 수명을 유지합니다. 이 라이브러리는 MQTT version 3.1 프로토콜을 실행 중인 MQTT 서버를 통해 다른 디바이스에 메시지를 송신합니다. 특정 클라이언트에 메시지를 송신하거나 발행/구독 메시징을 사용하여 다수의 디바이스에 연결할 수 있습니다.

MQTT 클라이언트 라이브러리는 MQTT 프로토콜을 사용하여 모바일 디바이스와 센서에 대한 애플리케이션을 MQTT 서버에 연결합니다.

IBM MessageSight 및 IBM WebSphere MQ 는 MQTT 서버입니다. 다량의 MQTT 클라이언트 애플리케이션을 연결할 수 있고 MQTT 및 IBM WebSphere MQ 네트워크를 함께 연결할 수 있습니다. 130 페이지의 『MQTT 서버 시작하기』을(를) 참조하십시오. IBM WebSphere MQ와 IBM MessageSight 모두 모바일 디바이스와 센서로 실행되는 외부 웹 애플리케이션과, 엔터프라이즈 내에서 실행되는 다른 유형의 발행/구독 및 메시징 애플리케이션 사이에서 브릿지를 형성할 수 있습니다. 브릿지를 사용하면 모바일 디바이스와 센서를 통합하는 "스마트 솔루션"을 더 편리하게 빌드할 수 있습니다.

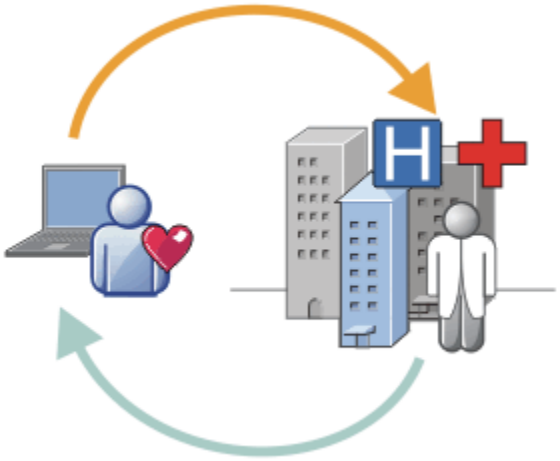
스마트 솔루션은 인터넷에서 사용 가능한 풍부한 정보를 모바일 및 센서 디바이스에서 실행 중인 애플리케이션에 제공합니다. 텔레메트리를 기반으로 하는 스마트 애플리케이션의 두 가지 예는 스마트 전기 및 스마트 건강 서비스입니다.



- 서비스 제공자에게 전송되는 에너지 사용 데이터가 들어 있는 MQTT 메시지입니다.
- 텔레메트리 애플리케이션은 에너지 사용량 데이터에 대한 분석을 기반으로 한 제어 명령을 송신합니다.
- 자세한 정보는 텔레메트리 시나리오: 가정 에너지 모니터링 및 제어를 참조하십시오.

그림 *3.* 스마트 전기 계량

그림 *4.* 스마트 건강 모니터링



- 텔레메트리 애플리케이션은 건강 데이터를 병원과 의사에게 송신합니다.
- 건강 데이터에 대한 분석을 기반으로 MQTT 메시지 경보 또는 피드백을 송신할 수 있습니다.
- 자세한 정보는 텔레메트리 시나리오: 가정 환자 모니터링을 참조하십시오.

MQTT protocol에 대한 사용자 고유의 앱을 작성하여 MQTT 를 소형 디바이스에 빌드할 수 있습니다. 이를 지원하기 위해 IBM에서는 MQTT를 통해 실행되는 앱을 지원하는 클라이언트 라이브러리를 제공합니다. 9 페이지의 『MQTT 클라이언트 시작하기』의 내용을 참조하십시오. IBM은 iOS 앱과 Android 앱용 클라이언트 라이브러리를 제공 V 7.5.0.1 하고 플랫폼에 관계 없이 사용 가능한 웹 앱용 JavaScript 브라우저 클라이언트를 제공합니다. V 7.5.0.1 JavaScript 클라이언트 페이지는 WebSockets를 통해 MQTT 프로토콜을 사용하여 IBM MessageSight 및 IBM WebSphere MQ 에 연결합니다. IBM 에서는 Linux 및 Windows의 C및 Java 에 대한 MQTT 샘플 앱도 제공합니다.

C 및 Java 라이브러리는 iOS, Android, Windows 및 다수의 UNIX and Linux 플랫폼에서 실행됩니다. MQTT 클라이언트 라이브러리에 대한 C 소스 코드를 다른 플랫폼에 이식할 수 있습니다. C와 Java용 MQTT 클라이언트 라이브러리는 Eclipse Paho 프로젝트의 오픈 소스 라이센스와 함께 사용할 수 있습니다. Eclipse Paho을 참조하십시오. MQTT protocol 스펙은 공개되어 있으며 MQTT.org에서 제공합니다.

MQTT protocol

MQTT protocol은 클라이언트가 작다는 의미에서 경량이며 네트워크 대역폭을 효율적으로 사용합니다. MQTT 프로토콜은 보장된 전달과 전송 후 삭제 전송을 지원합니다. 이 프로토콜에서 메시지 전달은 애플리케이션에서 분리됩니다. 애플리케이션에서 분리 범위는 MQTT 클라이언트 및 MQTT 서버가 작성되는 방식에 따라 다릅니다. 분리된 전달은 서버 연결에서 애플리케이션을 분리하고 메시지를 기다리지 않게 합니다. 상호작용 모델은 이메일과 비슷하지만 애플리케이션 프로그래밍에 맞게 최적화되어 있습니다.

MQTT V3.1 프로토콜이 발행됩니다. MQTT V3.1 프로토콜 스펙의 내용을 참조하십시오. 스펙은 프로토콜에 대한 다수의 구별되는 기능을 식별합니다.

- 발행/구독 프로토콜입니다.

  일대다 메시지 분배를 제공하는 것 외에도 발행/구독은 애플리케이션을 분리합니다. 두 기능 모두 다수의 클라이언트가 있는 애플리케이션에서 유용합니다.
- 메시지 컨텐츠에 종속되지 않습니다.
- TCP/IP를 통해 실행되어 기본 네트워크 연결을 제공합니다.
- 메시지 전달에 대한 세 가지 서비스 품질을 가지고 있습니다.

  "많아야 한 번"
  메시지는 기본 인터넷 프로토콜 네트워크의 최고 효율에 따라 전달됩니다. 메시지 유실이 발생할 수 있습니다.

  예를 들어, 통신하는 주위 센서 데이터와 함께 이 서비스 품질을 사용하십시오. 다음 수치가 곧 발행되는 경우에는 개별 수치가 유실되는지 여부는 중요하지 않습니다.

  "적어도 한 번"
  메시지는 확실히 도달하지만 중복이 발생할 수 있습니다.

  "정확히 한 번"
  메시지는 확실하게 정확히 한 번 도달합니다.

  예를 들어, 청구 시스템과 함께 이 서비스 품질을 사용하십시오. 메시지가 중복되거나 유실되면 불편하거나 잘못된 요금이 부과될 수 있습니다.
- 네트워크에서 메시지 플로우를 관리하는 방식 면에서 경제적입니다. 예를 들어, 고정 길이 헤더의 길이는 2바이트이며 프로토콜 교환은 최소화되어 네트워크 트래픽을 줄입니다.
- 여기에는 구독자에게 MQTT 서버로부터 클라이언트의 비정상적인 연결 끊기를 알리는 "이상 종료 시 메시지" 기능이 있습니다. 124 페이지의 『이상 종료 시 메시지 발행』의 내용을 참조하십시오.

MQTT version 3.1은 IBM IBM WebSphere MQ 및 IBM MessageSight에 의해 지원됩니다. MQTT는 TCP/IP를 통해 구현됩니다. 다른 버전의 프로토콜(MQTT-S)을 비TCP/IP 네트워크에 사용할 수 있습니다. MQTT-S version 1.2 스펙을 참조하십시오.

MQTT 커뮤니티

IBM 는 IBM MessageSight 및 IBM WebSphere MQ용 애플리케이션을 작성하는 MQTT 개발자를 위해 IBM Developer 메시징 커뮤니티 를 실행 중입니다.

MQTT.org는 MQTT 프로토콜에 대한 구현 및 확장에 대해 알아보고 토론하기에 적합한 위치입니다.

MQTT는 Eclipse기술 프로젝트 아래의 오픈 소스 Eclipse 프로젝트입니다. Paho 커뮤니티는 오픈 소스 클라이언트와 서버를 개발 중입니다. Eclipse Paho을 참조하십시오.

MQTT 클라이언트 시작하기

MQTT 클라이언트 라이브러리를 사용하는 샘플 MQTT 클라이언트 앱을 빌드하고 실행하여 모바일 또는 시스템 간(M2M) 앱 개발을 시작할 수 있습니다. 샘플 앱 및 연관된 클라이언트 라이브러리는 IBM의 모바일 메시징 및 M2M 클라이언트 팩 에서 사용 가능합니다. Java, JavaScript및 C로 작성된 앱 및 클라이언트 라이브러리의 버전이 있습니다. Apple의 Android 디바이스 및 제품을 포함하여 대부분의 플랫폼 및 디바이스에서 이러한 앱을 실행할 수 있습니다.

시작하기 전에

앱을 빌드하고 실행하려면 대상 디바이스 또는 플랫폼에서 사용되는 앱을 빌드한 경험과 사용 중인 프로그래밍 언어에 대한 경험이 필요합니다. 일반적으로 약간의 경험만 있으면 충분히 선택한 디바이스 또는 플랫폼에서 샘플 앱을 시작하고 실행할 수 있습니다.

IBM WebSphere MQ 또는 IBM MessageSight와 같은 엔터프라이즈급 MQTT 서버를 사용하는 경우 샘플 앱의 정보를 기존 엔터프라이즈 앱과 교환할 수 있습니다.

이 태스크 정보

목표는 다음과 같습니다.

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.
2. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.
3. 클라이언트 팩에서 대상 디바이스 또는 플랫폼에서 사용할 수 있는 샘플 앱을 빌드합니다.
4. 샘플을 MQTT 서버에 연결하여 샘플이 예상대로 작동하는지 확인합니다.

디바이스 또는 플랫폼에서 사용할 샘플 앱을 빌드하고 테스트함으로써 작동하는 개발 환경을 작성하고 이 개발 환경을 사용해서 자체 클라이언트 앱을 빌드할 수 있습니다.

모바일 메시징 및 M2M 클라이언트 팩에는 MQTT SDK가 포함되어 있습니다. 이 SDK에서는 다음과 같은 자원을 제공합니다.

- Java, JavaScript, C로 작성된 샘플 MQTT 클라이언트 앱
- 이 클라이언트 앱을 지원하고 대부분의 플랫폼과 디바이스에서 이들 앱을 사용 가능하게 하는 MQTT 클라이언트 라이브러리

SDK에는 C용 MQTT 클라이언트의 소스 코드도 포함되어 있습니다. 이 소스 코드를 채택하여 다른 플랫폼용 C의 MQTT 클라이언트 라이브러리를 빌드할 수 있습니다. 이를 수행하는 데 필요한 도움말은 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』의 내용을 참조하십시오. C용 MQTT 클라이언트의 소스 코드는 Eclipse Paho의 오픈 소스 라이센스에서도 사용 가능합니다.

프로시저

다음 주제에서는 데스크탑 컴퓨터나, Android용 또는 Apple의 모바일 디바이스에서 샘플 MQTT 앱을 빌드하고 실행하는 플랫폼별 단계를 안내합니다.

- 11 페이지의 『Java용 MQTT 클라이언트 시작하기』
- 17 페이지의 『Android 에서 Java용 MQTT 클라이언트 시작하기』
- V 7.5.0.1
  22 페이지의 『JavaScript용 MQTT 메시징 클라이언트 시작하기』
- 24 페이지의 『C용 MQTT 클라이언트 시작하기』
- 45 페이지의 『iOS에서 C용 MQTT 클라이언트 시작하기』

다음에 수행할 작업

새 MQTT 애플리케이션을 개발하려면 다음과 같은 기술을 가지고 있거나 확보해야 합니다.

- 디바이스 또는 플랫폼에 필요한 언어로 프로그래밍
- 대상 디바이스 또는 플랫폼에 대한 프로그래밍
- 발행/구독 애플리케이션 설계
- MQTT 프로그래밍 모델에 대한 프로그램 설계
- 선택한 모바일 디바이스에서 실행할 프로그램 설계
- SSL 및 JAAS를 사용하여 프로그램 보안

MQTT는 메시징 시스템이고 큐잉 시스템이기도 하므로 MQTT 클라이언트를 다른 디바이스 또는 애플리케이션과 연결하기 위한 네트워크 프로그래밍 기술은 필요하지 않습니다. MQTT 클라이언트 라이브러리는 애플리케이션에 대한 네트워크 연결을 관리합니다.

MQTT 클라이언트를 기존 엔터프라이즈 애플리케이션과 통합하려면 두 가지 선택사항이 있습니다. MQTT 발행/구독 토픽을 IBM WebSphere MQ 또는 JMS 애플리케이션 등과 공유하거나 자체 통합 어댑터를 다른 MQTT 클라이언트로 작성할 수 있습니다.

오늘 문의할 정보의 소스는 다음과 같습니다.

- WebSphere MQ Telemetry용 애플리케이션 개발
- MQTT.org
- Eclipse Paho

관련 개념

130 페이지의 『MQTT 서버 시작하기』

Java용 MQTT 클라이언트 시작하기

MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ를 사용하여 Java 샘플 애플리케이션의 MQTT 클라이언트를 시작하여 실행합니다. The sample applications use a client library from the MQTT software development toolkit (SDK) from IBM. `SampleAsyncCallBack` 샘플 애플리케이션은 Android 및 기타 이벤트 구동 운영 체제에 대해 MQTT 애플리케이션을 작성하기 위한 모델입니다.

시작하기 전에

- JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능". IBM 모바일 메시징 및 M2M 클라이언트 팩 에 대한 시스템 요구사항의 내용을 참조하십시오.
- 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.

이 태스크 정보

The purpose of the task is to check that you can build and run an MQTT client for Java sample application, connect it to IBM WebSphere MQ or IBM MessageSight as the MQTT version 3 server, and exchange messages.

이 태스크를 수행하여 Eclipse 워크벤치 또는 명령행에서 샘플 애플리케이션을 실행하십시오. 예제의 단계는 Windows용입니다. 조금만 수정하면 JSE 1.5 이상을 지원하는 모든 플랫폼에서 샘플 애플리케이션을 실행할 수 있습니다.

IBM WebSphere MQ와 동일한 서버에서 애플리케이션을 실행할 수 있습니다. 여기서 IBM WebSphere MQ에 연결되는 애플리케이션을 실행하는 데 필요한 환경이 구성됩니다. 134 페이지의 『명령행에서 MQTT 서비스 구성』 태스크에 따라 Windows 또는 Linux에 IBM WebSphere MQ Telemetry 옵션을 사용하여 IBM WebSphere MQ 를 설치하고 구성하십시오. 환경이 설치되어 구성된 경우 샘플 애플리케이션 `MQTTV3Sample`을 실행하여 설치를 확인하십시오.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 MQTT version 3.1 프로토콜을 지원해야 합니다. IBM 의 모든 MQTT 서버는 IBM WebSphere MQ 및 IBM MessageSight를 포함합니다. 130 페이지의 『MQTT 서버 시작하기』의 내용을 참조하십시오.
2. 옵션: MQTT 서버를 구성하십시오.
   - IBM WebSphere MQ에서는 다음 태스크 중 하나를 완료하여 큐 관리자를 설정하고 해당 텔레메트리(MQXR) 서비스를 구성해야 합니다.
     - 134 페이지의 『명령행에서 MQTT 서비스 구성』
     - 136 페이지의 『IBM WebSphere MQ Explorer로 MQTT 서비스 구성』

- 기타 서버에서는 서버 문서를 참조하십시오. Really Small Message Broker의 경우에는 구성 단계가 필요하지 않습니다. Really Small Message Broker을 참조하십시오.

3. 모바일 메시징 및 M2M 클라이언트 팩 을 다운로드하고 MQTT SDK를 설치하십시오.

   설치 프로그램은 존재하지 않으며 다운로드한 파일을 펼치기만 하면 됩니다.

   a. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

   b. SDK를 설치할 폴더를 작성하십시오.

      해당 폴더의 이름을 MQTT로 지정할 수도 있습니다. 이 폴더의 경로는 여기에서 *sdkroot*로 참조됩니다.

   c. 압축된 모바일 메시징 및 M2M 클라이언트 팩 파일 컨텐츠를 *sdkroot*로 펼치십시오. 이 펼치기를 통해 *sdkroot*\SDK에서 시작되는 디렉토리 트리가 작성됩니다.

4. Java 개발 킷 (JDK) 버전 6이상을 설치하십시오.

   Java 앱 for Android을 (를) 개발하고 있으므로 JDK 은 Oracle에서 와야 합니다. JDK는 Java SE 다운로드에서 가져올 수 있습니다.

5. Java 샘플 응용프로그램에 대해 하나 이상의 MQTT 클라이언트를 컴파일하고 실행하십시오.

   - 13 페이지의 『명령행에서 Paho 샘플 프로그램 컴파일 및 실행』
   - 14 페이지의 『Eclipse의 모든 MQTT 클라이언트 샘플 Java 앱 컴파일과 실행』
   - 17 페이지의 『Android 에서 Java용 MQTT 클라이언트 시작하기』

   다음과 같은 MQTT 클라이언트 샘플 Java 앱이 SDK에 포함되어 있습니다.

   MQTTV3Sample
   또한 이 샘플은 IBM WebSphere MQ와 함께 포함되어 있으며 `com.ibm.micro.client.mqttv3.jar` 패키지에 링크됩니다.

   Sample
   `Sample` 는 Paho 패키지에 있으며 `org.eclipse.paho.client.mqttv3` 패키지에 대한 링크입니다. 이는 `MQTTV3Sample`과 비슷하여 각각의 MQTT 조치가 완료될 때까지 대기합니다.

   SampleAsyncWait
   `SampleAsyncWait`은 `org.eclipse.paho.client.mqttv3` 패키지에 있습니다. 이는 비동기 MQTT API를 사용합니다. 이는 조치가 완료될 때까지 다른 스레드를 기다립니다. 기본 스레드는 MQTT 조치가 완료되기를 기다리는 스레드에서 동기화될 때까지 다른 작업을 수행할 수 있습니다.

   SampleAsyncCallBack
   `SampleAsyncCallBack`은 `org.eclipse.paho.client.mqttv3` 패키지에 있습니다. 이는 비동기 MQTT API를 호출합니다. 비동기 API는 MQTT가 호출 처리를 완료할 때까지 기다리지 않고 애플리케이션으로 돌아갑니다. 애플리케이션은 다른 태스크를 계속 수행한 후 다음 이벤트가 처리를 위해 도착하기를 기다립니다. MQTT는 처리를 완료할 때 이벤트 알림을 애플리케이션에 게시합니다. 이벤트 구동 MQTT 인터페이스는 Android의 서비스 및 활동 프로그래밍 모델 및 기타 이벤트 구동 운영 체제에 적합합니다.

   예를 들어, `mqttExerciser` 샘플이 서비스 및 활동 프로그래밍 모델을 사용하여 MQTT를 Android에 통합하는 방법을 보십시오.

   mqttExerciser
   `mqttExerciser` 는 Android의 샘플 프로그램입니다. 이 프로그램은 다르게 빌드되고 실행되므로 이에 대해 별도로 설명합니다. 17 페이지의 『Android 에서 Java용 MQTT 클라이언트 시작하기』의 내용을 참조하십시오.

결과

MQTT 서버로서의 IBM WebSphere MQ또는 IBM MessageSight에 연결된 MQTT Java 샘플 애플리케이션을 컴파일하고 실행했습니다.

다음에 수행할 작업

Javadoc 참조 정보를 연구하십시오. 14 페이지의 『Eclipse의 모든 MQTT 클라이언트 샘플 Java 앱 컴파일과 실행』의 15 페이지의 『3』 단계를 참조하십시오. 또는 모바일 메시징 및 M2M 클라이언트 팩의 SDK\clients\java\doc\javadoc 디렉토리에 있는 html 파일 여십시오.

명령행에서 Paho 샘플 프로그램 컴파일 및 실행

명령행에서 Paho 샘플 애플리케이션 Sample.java 을 컴파일하고 실행하십시오. 샘플은 MQTT SDK에 있습니다. 샘플은 org.eclipse.paho.client.mqttv3 패키지의 MQTT Paho 클라이언트 라이브러리를 사용하여 빌드됩니다. 이 샘플은 MQTT 발행자 및 구독자를 보여줍니다. 동일한 디렉토리에 있는 두 개의 다른 Paho 샘플을 동일한 방식으로 빌드하고 실행할 수 있습니다. 이들은 MQTT 라이브러리를 비동기식으로 호출하여 구별됩니다.

시작하기 전에

기본 태스크의 11 페이지의 『1』 - 12 페이지의 『4』 단계를 수행하여 MQTT 서버를 구성하고 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

이 태스크 정보

SDK 클라이언트 샘플 서브디렉토리 SDK\clients\java\samples에서 Sample.java를 컴파일하고 실행하십시오. Java 코드는 SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app 디렉토리에 있습니다.

프로시저

1. 클라이언트 샘플 디렉토리에서 스크립트를 작성하여 컴파일하고 선택한 플랫폼에서 Sample 를 실행하십시오.

   다음 스크립트는 Windows에서 샘플을 컴파일하고 실행합니다.

   ```
   @echo on
   setlocal
   set classpath=
   set JAVADIR=C:\Program Files\IBM\Java70\bin
   "%JAVADIR%\javac"
   -cp ..\org.eclipse.paho.client.mqttv3.jar .\org\eclipse\paho\sample\mqttv3app\Sample.java
   start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
   org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b localhost -p 1883
   @rem Sleep for 2 seconds
   ping -n 2 127.0.0.1 > NUL 2>&1
   "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
   org.eclipse.paho.sample.mqttv3app.Sample -b localhost -p 1883
   pause
   endlocal
   ```

   그림 *5. Sample.java* 컴파일 및 실행

2. 스크립트를 실행하십시오.

   결과:

   ```
   Connected to tcp://localhost:1883 with client ID SampleJavaV3_subscribe
   Subscribing to topic "Sample/#" qos 2
   Press <Enter> to exit
   Time:   2012-11-09 17:23:22.718  Topic: Sample/Java/v3  Message:        Message
   from blocking MQTTv3 Java client sample  QoS:   2
   ```

   그림 *6. MQTTV3Sample* 구독자

```
Connecting to tcp://localhost:1883 with client ID SampleJavaV3_publish
Connected
Publishing at: 2012-11-09 17:22:07.734 to topic "Sample/Java/v3" qos 2
Disconnected
```

그림 *7. MQTTV3Sample* 발행자

구독자 애플리케이션이 계속 실행되도록 하는 경우에는 동일한 구독자를 다시 실행할 수 없습니다. 새 구독자의 클라이언트 ID는 이전 구독자와 동일합니다. 동시에 동일한 클라이언트 ID를 가진 두 MQTT 클라이언트를 실행할 수 없습니다. 동시에 서로 다른 구독자를 실행할 수 있도록 `-i` 옵션을 설정하여 클라이언트 ID를 설정할 수 있습니다.

동일한 클라이언트를 다시 실행하는 경우에는 `-c` 옵션을 활용하여 `cleansession`이 false로 설정된 상태로 클라이언트를 시작할 수 있습니다. 이 옵션을 사용하면 인터럽트된 클라이언트 세션의 작동을 탐색할 수 있습니다.

3. Enter를 누르거나 창을 닫아 구독자를 종료하십시오.

다음에 수행할 작업

스크립트를 작성하여 `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` 서브디렉토리에서 다른 샘플을 컴파일하고 실행하십시오. 13 페이지의 그림 5에 있는 스크립트를 복사한 후 `Sample`을 `SampleAsyncWait` 또는 `SampleAsyncCallBack`으로 바꾸십시오. 기타 샘플은 동기 `Sample` 프로그램의 비동기 버전입니다.

`SampleAsyncWait`
`SampleAsyncWait`은 `org.eclipse.paho.client.mqttv3` 패키지에 있습니다. 이는 비동기 MQTT API를 사용합니다. 이는 조치가 완료될 때까지 다른 스레드를 기다립니다. 기본 스레드는 MQTT 조치가 완료되기를 기다리는 스레드에서 동기화될 때까지 다른 작업을 수행할 수 있습니다.

`SampleAsyncCallBack`
`SampleAsyncCallBack`은 `org.eclipse.paho.client.mqttv3` 패키지에 있습니다. 이는 비동기 MQTT API를 호출합니다. 비동기 API는 MQTT가 호출 처리를 완료할 때까지 기다리지 않고 애플리케이션으로 돌아갑니다. 애플리케이션은 다른 태스크를 계속 수행한 후 다음 이벤트가 처리를 위해 도착하기를 기다립니다. MQTT는 처리를 완료할 때 이벤트 알림을 애플리케이션에 게시합니다. 이벤트 구동 MQTT 인터페이스는 Android의 서비스 및 활동 프로그래밍 모델 및 기타 이벤트 구동 운영 체제에 적합합니다.

예를 들어, `mqttExerciser` 샘플이 서비스 및 활동 프로그래밍 모델을 사용하여 MQTT를 Android에 통합하는 방법을 보십시오.

비동기 샘플은 MQTT 애플리케이션이 MQTT 클라이언트를 기다리는 동안 차단하는 시간을 줄이는 방법을 보여줍니다. 모바일 환경에서 응답성을 및 배터리 수명을 늘리려면 기본 스레드에서 차단하는 호출을 제거하는 것이 중요합니다.

샘플에서는 MQTT 클라이언트에서 비동기 인터페이스를 호출하는 두 가지 패턴을 보여줍니다.

1. `SampleAsyncWait` 은 (는) MQTT 가 네트워크 상호작용을 대기하는 동안 차단되지 않습니다.
2. `SampleAsyncCallBack`은 MQTT 클라이언트가 조치를 완료할 때까지 기다리는 동안 차단하지 않습니다. JavaScript 페이지가 브라우저에서 MQTT 클라이언트를 호출하는 경우 후자가 필요합니다. JavaScript 페이지는 차단되지 않아야 합니다. 조치에 대한 응답은 기본 브라우저 스레드에 다시 게시되어야 하며 이는 알림을 처리하기 위해 사용자가 작성하는 MQTT 이벤트 핸들러를 호출합니다.

Eclipse의 모든 MQTT 클라이언트 샘플 Java 앱 컴파일과 실행

모바일 메시징 및 M2M 클라이언트 팩에 있는 MQTT 클라이언트 샘플 Java 애플리케이션을 컴파일하고 실행하십시오. 샘플에서는 MQTT 발행자 및 구독자를 보여줍니다.

이 태스크 정보

Eclipse에서 MQTT Java 샘플, `Sample`및 `MQTTV3Sample` 을 컴파일하고 실행하십시오. `Sample`은 *`sdkroot`*`\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` SDK 클라이언트 서브디렉토리에 있고 `MQTTV3Sample.java`는 *`sdkroot`*`\SDK\clients\java\samples`에 있습니다.

프로시저

1. Eclipse IDE for Java Developers를 다운로드하십시오.
2. Eclipse에서 `MQTT Samples` 라는 Java 프로젝트를 작성하십시오.
   a) **파일** > **새로 작성** > **Java 프로젝트** 를 선택하고 `MQTT Samples`를 입력하십시오. **다음**을 클릭하십시오.

   JRE가 올바른 버전 또는 최신 버전에 있는지 확인하십시오.JSE 는 버전 1.5 이상이어야 합니다.

   b) **Java 설정** 창에서 **추가 소스 폴더 링크**를 클릭하십시오.
   c) MQTT Java SDK 폴더를 설치한 디렉토리로 이동하십시오. *sdkroot*`\SDK\clients\java\samples` 폴더를 선택하고 **확인** > **다음** > **완료**를 클릭하십시오.
   d) **Java 설정** 창에서 **라이브러리** > **외부 Jar 추가**를 클릭하십시오.
   e) MQTT Java SDK 폴더를 설치한 디렉토리로 이동하십시오. *sdkroot*`\SDK\clients\java` 폴더를 찾고 `org.eclipse.paho.client.mqttv3.jar` 및 `com.ibm.micro.client.mqttv3.jar` 파일을 선택하십시오. **열기** > **완료**를 클릭하십시오.

   `MQTTV3Sample.java` 샘플은 `com.ibm.micro.client.mqttv3.jar`에 링크되고 paho 디렉토리 트리에 있는 샘플은 `org.eclipse.paho.client.mqttv3.jar`에 링크됩니다. 기존 MQTT 애플리케이션이 변경 없이 계속 빌드되고 실행되도록 `com.ibm.micro.client.mqttv3.jar` 가 보유됩니다.

   `MQTT Samples` 프로젝트가 빌드되고 일부 경고가 표시되지만 오류는 표시되지 않습니다.

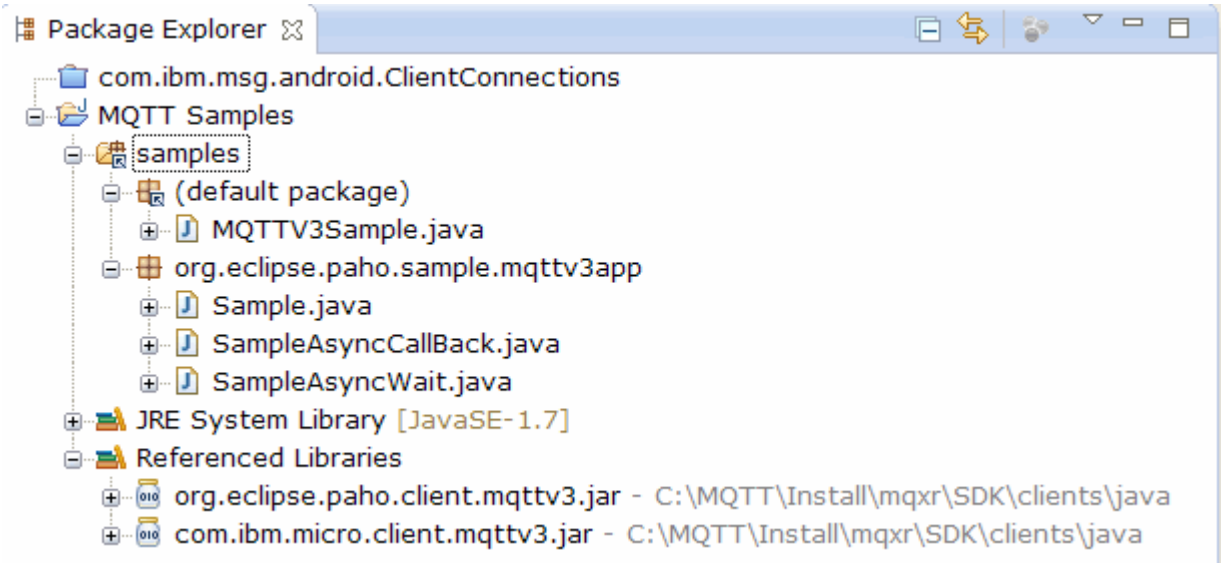


그림 *8. Java*용 *MQTT* 클라이언트 프로젝트

3. 옵션: MQTT 클라이언트 Javadoc을 설치하십시오.

   MQTT 클라이언트 설치된 경우, Java 편집기는 풍선 도움말의 MQTT 클래스를 설명합니다.

   a) Java 프로젝트에서 **패키지 탐색기** > **참조된 라이브러리** 를 여십시오.
   `org.eclipse.paho.client.mqttv3.jar` **> 특성**을 마우스 오른쪽 단추로 클릭하십시오.
   b) 특성 네비게이터에서 **Javadoc 위치**를 클릭하십시오.
   c) **Javadoc 위치** 페이지에서 **Javadoc URL** > **찾아보기** 를 누르고 `SDK\clients\java\doc\javadoc` 폴더 **> 확인**을 찾으십시오.
   d) **유효성 검증** > **확인**을 클릭하십시오.

   문서를 보기 위해 브라우저를 열라는 메시지가 표시됩니다.

   e) `com.ibm.micro.client.mqttv3.jar` 파일에 대해 이 프로시저를 반복하십시오.
4. 발행자 및 구독자 런타임 구성을 작성하여 `mqttv3app.Sample` 애플리케이션을 실행하십시오.
   a) **샘플** 클래스를 마우스 오른쪽 단추로 클릭하고 **실행 도구** > **실행 구성**을 클릭하십시오.

b) **Java Application** > **새로 작성** 을 마우스 오른쪽 단추로 클릭하고 이름 SampleSubscriber을 입력하십시오.

c) 인수 탭을 클릭하여 프로그램 인수를 입력한 후 **적용**을 클릭하십시오.

```
-a subscribe -b localhost -p 1883
```

d) -a subscribe 매개변수를 생략하여 SamplePublisher 구성을 작성하는 마지막 단계를 반복하십시오.

5. 발행자가 뒤따라오는 mqttv3app.Sample 구독자를 실행하십시오.

a) **실행** > **구성 실행**을 클릭하십시오.

b) **SampleSubscriber** > **실행**을 누르십시오.

콘솔 보기를 여십시오. 구독자가 발행물을 기다립니다.

```
Connected to tcp://localhost:1883 with client ID SampleJavaV3_subscribe
Subscribing to topic "Sample/#" qos 2
Press <Enter> to exit
```

c) **SamplePublisher** > **실행**을 누르십시오.

콘솔 보기를 여십시오. 발행자가 작성하는 발행물이 표시됩니다.

```
Connecting to tcp://localhost:1883 with client ID SampleJavaV3_publish
Connected
Publishing at: 2012-11-09 14:09:29.859 to topic "Sample/Java/v3" qos 2
Disconnected
```

d) 콘솔 보기를 구독자 콘솔로 전환하십시오.

콘솔 전환 아이콘은  입니다.

구독자가 발행물을 수신했습니다.

```
...
Time: 2012-11-09 14:09:30.593  Topic: Sample/Java/v3  Message: Message from blocking
MQTTv3 Java client sample  QoS: 2
```

6. 옵션: MQTTV3Sample에 대한 발행자 및 구독자 런타임 구성을 작성하십시오.

a) **MQTTV3Sample** 클래스를 마우스 오른쪽 단추로 클릭하고 **실행 도구** > **구성 실행**을 클릭하십시오.

b) **Java Application** > **새로 작성** 을 마우스 오른쪽 단추로 클릭하고 이름 MQTTV3SampleSubscriber을 입력하십시오.

c) 인수 탭을 클릭하여 프로그램 인수를 입력한 후 **적용**을 클릭하십시오.

```
-a subscribe -b localhost -p 1883
```

d) -a subscribe 매개변수를 생략하여 MQTTV3SamplePublisher 구성을 작성하는 마지막 단계를 반복하십시오.

7. 옵션: 발행자가 뒤따라오는 MQTTV3Sample 구독자를 실행하십시오.

a) **실행** > **구성 실행**을 클릭하십시오.

b) **MQTTV3SampleSubscriber** > **실행**을 누르십시오.

콘솔 보기를 여십시오. 구독자가 발행물을 기다립니다.

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
```

c) **MQTTV3SamplePublisher** > **실행**을 누르십시오.

콘솔 보기를 여십시오. 발행자가 작성하는 발행물을 확인할 수 있습니다.

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
```

d) 콘솔 보기를 구독자 콘솔로 전환하십시오.

콘솔 전환 아이콘은  입니다.

구독자가 발행물을 수신했습니다.

```
MQTTV3Sample/Java/v3
Message: Message from MQTTv3 Java client
QoS:  2
```

Android 에서 Java용 MQTT 클라이언트 시작하기

MQTT 서버와 메시지를 교환하는 Android에 대한 MQTT 클라이언트 샘플 Java 앱 를 설치할 수 있습니다. 앱은 IBM의 MQTT SDK에서 클라이언트 라이브러리를 사용합니다. 직접 앱을 빌드하거나 사전 빌드된 샘플 앱을 다운로드할 수 있습니다.

시작하기 전에

- 지원되는 MQTT 클라이언트 플랫폼과 참조 MQTT 클라이언트 플랫폼은 IBM 모바일 메시징 및 M2M 클라이언트 팩 에 대한 시스템 요구사항의 내용을 참조하십시오.
- 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.
- MQTT 클라이언트 샘플 앱은 Ice Cream Sandwich(Android 4.0) 이상에서 작동합니다. Android의 이 버전도 태블릿에서 선명한 화면 해상도를 제공합니다.

이 태스크 정보

Android에 대한 MQTT 클라이언트 샘플 Java 앱을 "`mqttExerciser`"라고 합니다. 이 앱에서는 MQTT SDK의 클라이언트 라이브러리를 사용하며 MQTT 서버와 메시지를 교환합니다.

직접 샘플 앱을 빌드한 후 Eclipse 에서 `mqttExerciser.apk`로 내보내거나 모바일 메시징 및 M2M 클라이언트 팩의 *`sdkroot`*`\SDK\clients\android\samples\apks` 폴더에서 `mqttExerciser.apk` 파일로 사용 가능한 사전 빌드된 샘플 앱을 사용할 수 있습니다. 직접 앱을 빌드하는 경우 사용자가 빌드하는 개발 환경은 모바일 메시징을 for Android 앱에 포함하도록 조정됩니다. 이는 모바일 메시징을 사용자 자신의 앱에 포함시킬 때 유용합니다.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 MQTT version 3.1 프로토콜을 지원해야 합니다. IBM 의 모든 MQTT 서버는 IBM WebSphere MQ 및 IBM MessageSight를 포함합니다. 130 페이지의 『MQTT 서버 시작하기』의 내용을 참조하십시오.
2. 적절한 도구를 가져오십시오.

   Java 개발 킷 (JDK) 버전 6이상을 설치하십시오. Java 앱 for Android을 (를) 개발하고 있으므로 JDK 은 Oracle에서 와야 합니다. JDK는 Java SE 다운로드에서 가져올 수 있습니다.

   또한 Eclipse 개발 환경도 필요합니다. 이는 Eclipse 3.6.2(Helios) 이상이어야 합니다. Eclipse에는 사용 중인 JDK에 부합하는 Java 컴파일러 레벨 6 이상이 있어야 합니다. Eclipse Foundation에서 이 모든 것을 가져올 수 있습니다.

   마지막으로 Android SDK가 필요합니다. 이는 Android SDK 가져오기를 통해 가져올 수 있습니다.
3. 모바일 메시징 및 M2M 클라이언트 팩 을 다운로드하고 MQTT SDK를 설치하십시오.

   설치 프로그램은 존재하지 않으며 다운로드한 파일을 펼치기만 하면 됩니다.

   a. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

b. SDK를 설치할 폴더를 작성하십시오.

해당 폴더의 이름을 MQTT로 지정할 수도 있습니다. 이 폴더의 경로는 여기에서 *sdkroot*로 참조됩니다.

c. 압축된 모바일 메시징 및 M2M 클라이언트 팩 파일 컨텐츠를 *sdkroot*로 펼치십시오. 이 펼치기를 통해 *sdkroot*\SDK에서 시작되는 디렉토리 트리가 작성됩니다.

4. 옵션: for Android mqttExerciser 샘플 앱을 빌드하십시오.

Eclipse 및 Android 도구를 구성하고 MQTT SDK에서 `mqttExerciser` 프로젝트를 가져와서 빌드하십시오.

참고: 지금 당장 이를 수행하려는 경우가 아니면 MQTT SDK의 *sdkroot*\SDK\clients\android\samples\apks 폴더에 `mqttExerciser.apk` 파일로 제공되는 사전 빌드된 샘플 앱을 사용할 수 있습니다.

a) JDK의 JRE를 사용하여 Eclipse 개발 환경을 시작하십시오.

```
eclipse -vm "JRE path"
```

b) Android SDK에서 패키지와 플랫폼 세트를 선택하여 설치하십시오.

Google 에서 권장하는 플랫폼 및 패키지 목록은 플랫폼 및 패키지 추가 를 참조하십시오.

참고: SDK 플랫폼 레벨은 Android API 레벨 16 이상이어야 합니다. 이전 API 레벨에서는 프로젝트를 컴파일할 수 없습니다.

c) Android 개발 도구(ADT) 플러그인을 Eclipse에 추가하십시오.

d) 샘플 `mqttExerciser` 앱 프로젝트를 Eclipse로 가져와서 오류를 수정하십시오.

i) *sdkroot*\SDK\clients\android\samples\mqttExerciser경로의 MQTT SDK에서 샘플 앱 프로젝트를 가져오십시오.

문제점 보기에 다수의 빌드 오류가 나열됩니다. 다음 몇 단계에서 빌드 오류를 해결합니다.

ii) `org.eclipse.paho.client.mqttv3.jar` 라이브러리를 Android 프로젝트의 **libs** 폴더에 복사하십시오. Windows 예를 들어, Windows에서는 *sdkroot*\SDK\clients\java 폴더 아래에 있습니다. **파일 조작** 창이 열립니다. **파일 복사** 선택사항을 승인한 후 **확인**을 클릭하십시오.

iii) 프로젝트 폴더 `com.ibm.msg.android`를 마우스 오른쪽 단추로 클릭하고 클릭하십시오. **Android 도구 ...** > **지원 라이브러리 추가 ...**. 라이센스 조항을 읽고 동의한 후 **설치**를 클릭하십시오.

iv) 프로젝트 폴더 `com.ibm.msg.android`를 마우스 오른쪽 단추로 클릭하고 클릭하십시오. **Android 도구 ...** > **프로젝트 특성 수정**.

v) 작업공간에 수퍼 클래스 메소드 대체를 언급하는 약 84개의 오류가 있는 경우에는 컴파일러 준수 레벨이 1.5 이하로 설정되어 있습니다. Android SDK 버전 16에서는 컴파일러 준수 레벨이 1.5 이하여야 합니다. 남아 있는 오류를 수정하려면 다음 단계를 완료하십시오.

a) Android SDK 및 해당 Eclipse 플러그인을 확인하여 필요한 경우 Android SDK 버전 17로 업데이트하십시오.

b) **com.ibm.msg.android** 프로젝트 폴더를 마우스 오른쪽 단추로 클릭한 후 **특성** > **Java 컴파일러**를 선택하십시오. 컴파일러 준수 레벨을 확인하고 1.6 이상으로 설정한 후 작업공간을 다시 빌드하십시오.

프로젝트가 빌드되고 일부 경고가 표시되지만 오류는 표시되지 않습니다.

5. Android 디바이스에서 MQTT 클라이언트 샘플 Java 앱 를 설치하고 시작하십시오.

`developer.android.com` 페이지의 앱 실행을 참조하십시오.

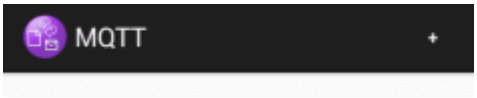
앱을 Eclipse 프로젝트로 직접 빌드한 경우 Eclipse에서 앱을 시작할 수 있습니다.

애플리케이션 패키지(APK) 파일 `mqttExerciser.apk`가 있는 경우, Android Debug Bridge(ADB) 설치 명령을 사용하여 Eclipse 외부에 이를 설치할 수 있습니다. 이 명령은 APK 파일의 위치를 인수로 사용합니다. 사전 빌드된 샘플 앱을 사용 중인 경우 이 위치는 *sdkroot*\SDK\clients\android\samples\apks\`mqttExerciser.apk`입니다.

6. for Android mqttExerciser 샘플 앱을 사용하여 토픽에 연결, 구독 및 발행하십시오.

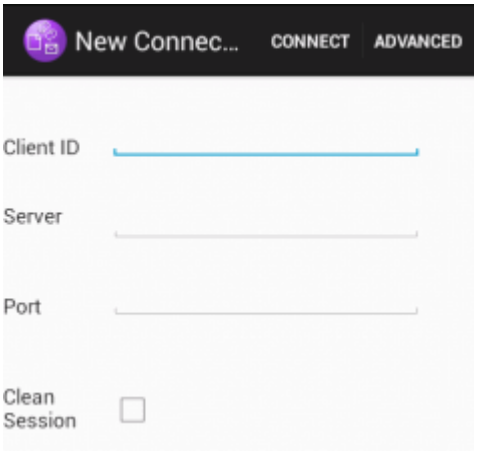
a) Android에 대한 MQTT 클라이언트 샘플 Java 앱를 여십시오.

이 창은 Android 디바이스에서 열립니다.



b) MQTT 서버에 연결하십시오.

i) **+** 부호를 클릭하여 새 MQTT 연결을 여십시오.



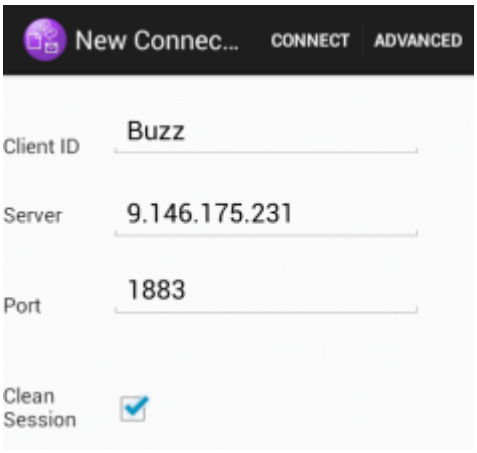
ii) **클라이언트 ID** 필드에 고유한 ID를 입력하십시오. 키 입력은 느릴 수 있으므로 기다려야 할 수도 있습니다.

iii) **서버** 필드에 MQTT 서버의 IP 주소를 입력하십시오.

이는 첫 기본 단계에서 선택한 서버입니다. IP 주소는 `127.0.0.1`일 수 없습니다.

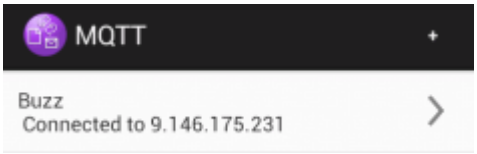
iv) MQTT 연결의 포트 번호를 입력하십시오.

일반 MQTT 연결의 기본 포트 번호는 1883입니다.



v) **연결**을 클릭하십시오.

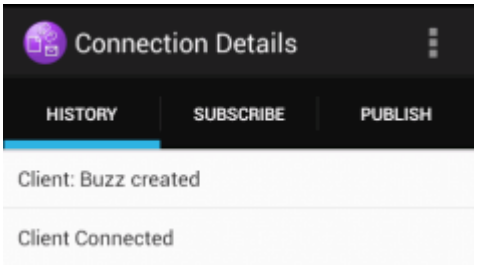
연결에 성공하면 이 창 다음에 "연결 중" 메시지가 나타납니다.



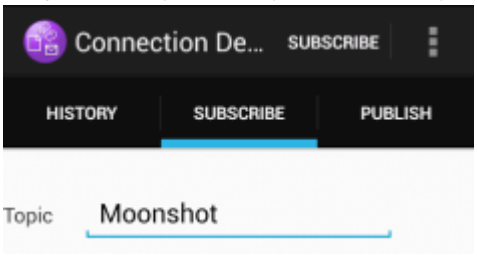
c) 토픽을 구독하십시오.

i) **연결됨** 메시지를 클릭하십시오.

연결 세부사항 창이 열리고 실행 기록이 나열됩니다.



ii) **구독** 탭을 클릭하고 토픽 문자열을 입력하십시오.

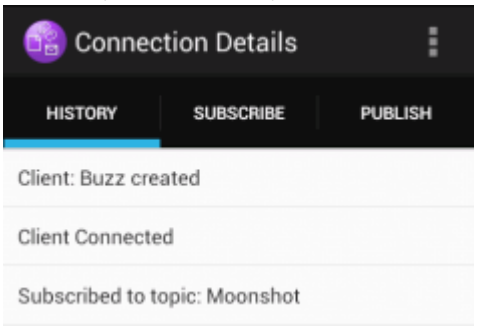


iii) **구독** 조치를 클릭하십시오.

"구독 처리됨" 메시지가 잠깐 동안 표시됩니다.

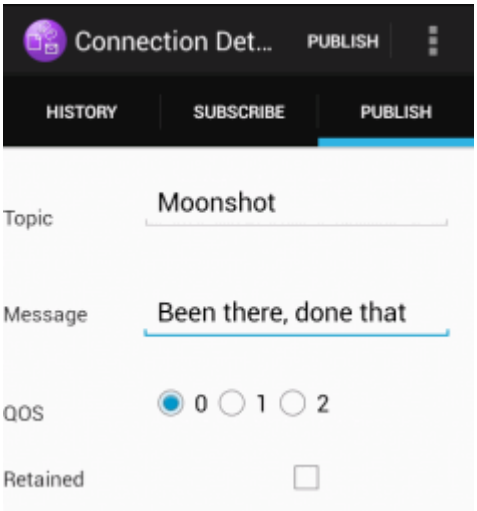
iv) **실행 기록** 탭을 클릭하십시오.

실행 기록에는 이제 구독이 포함됩니다.



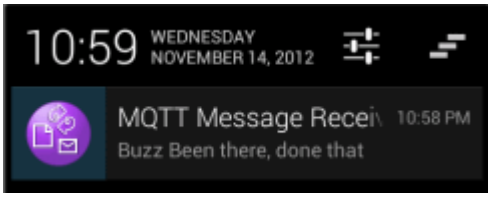
d) 이제 동일한 토픽으로 발행하십시오.

i) **발행** 탭을 클릭하고 구독 시 입력한 것과 동일한 토픽 문자열을 입력하십시오. 메시지를 입력하십시오.

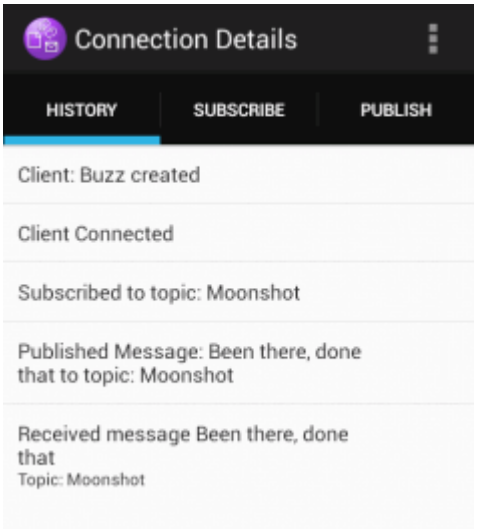


ii) **발행** 조치를 클릭하십시오.

두 개의 메시지 즉, "발행됨"과 "구독 처리됨"이 잠깐 동안 표시됩니다. 발행물이 상태 영역에 표시됩니다. (구분 막대를 아래로 당겨 상태 창을 여십시오.)



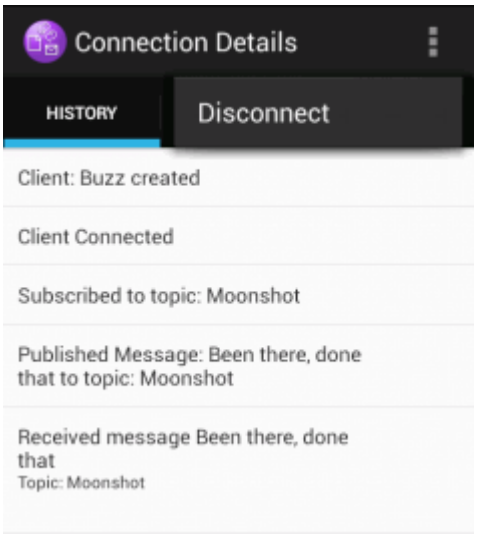
iii) **실행 기록** 탭을 클릭하여 전체 실행 기록을 보십시오.



e) 클라이언트 인스턴스의 연결을 끊으십시오.

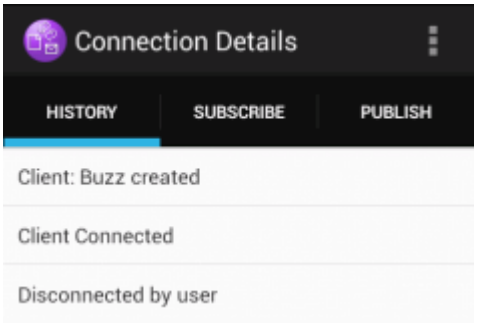
i) 조치 막대의 메뉴 아이콘을 클릭하십시오.

Android에 대한 MQTT 클라이언트 샘플 Java 앱은 **연결 끊기** 단추를 MQTT **연결 세부사항** 창에 추가합니다.

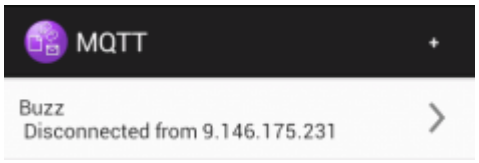


ii) **연결 끊기**를 클릭하십시오.

연결됨 상태가 연결 끊김으로 변경됩니다.



f) **이전** 을 클릭하여 MQTT 클라이언트 샘플 Java 앱 세션의 목록으로 되돌아가십시오.



- 더하기 부호를 클릭하여 새 MQTT 클라이언트 샘플 Java 앱 세션을 시작하십시오.
- 연결이 끊긴 클라이언트를 클릭하여 다시 연결하십시오.
- **이전**을 클릭하여 런치패드로 되돌아가십시오.

g) 태스크 단추를 클릭하여 실행 중인 앱을 나열하십시오. MQTT 클라이언트 샘플 Java 앱을 찾으십시오. 아이콘을 화면 밖으로 밀어서 닫으십시오.

다음에 수행할 작업

샘플 앱을 빌드하였으면, MQTT 라이브러리를 호출하여 메시지를 교환하는 나만의 Android 앱을 개발할 준비가 되었습니다. `mqttExerciser`의 클래스에서 Android 앱을 모델링할 수 있습니다. 샘플을 학습하려면 `mqttExerciser` 프로젝트의 `com.ibm.msg.android` 및 `com.ibm.msg.android.service` 에 있는 클래스에 대해 Javadoc 를 생성하십시오.

관련 정보

ADT로 Eclipse의 프로젝트 관리

JavaScript용 MQTT 메시징 클라이언트 시작하기

메시징 클라이언트 샘플 홈 페이지를 표시하고 이 홈 페이지에서 링크하는 자원을 찾아봄으로써 JavaScript용 MQTT 메시징 클라이언트를 시작할 수 있습니다. To display this home page, you configure an MQTT server to accept connections from the MQTT 메시징 클라이언트 샘플 JavaScript 페이지, then you type the URL that you have configured on the server into a web browser. JavaScript용 MQTT 메시징 클라이언트가 디바이스에서 자동으로 시작되고 메시징 클라이언트 샘플 홈 페이지가 표시됩니다. 이 페이지에는 유틸리티, 프로그래밍 인터페이스 문서, 학습서, 기타 유용한 정보에 대한 링크가 있습니다.

시작하기 전에

고급 사용의 경우 또는 프로덕션에서 사용하는 경우 메시징 클라이언트 샘플 홈 페이지의 모양을 바꾸거나 홈 페이지를 제거할 수 있습니다. 샘플 코드에서 생성된 사용자 인터페이스는 접근성 표준 또는 접근성 요구사항의 준수가 보장되지 않습니다.

JavaScript용 MQTT 메시징 클라이언트를 지원하려면 MQTT 서버가 필요합니다. 이 서버는 MQTT WebSockets V3.1 프로토콜을 지원해야 합니다. IBM MessageSight, IBM WebSphere MQ Version 7.5.0, Fix Pack 1 이상 버전에서는 WebSockets에서 MQTT protocol 를 지원합니다. 130 페이지의 『MQTT 서버 시작하기』을 참조하십시오. IBM WebSphere MQ 90일 무료 평가판을 설치하려면 132 페이지의 『설치 중IBM WebSphere MQ』의 내용을 참조하십시오.

WebSocket protocol은 최근에 설정되었습니다. 클라이언트와 서버 사이에 방화벽이 있는 경우 WebSockets 트래픽을 차단하지 않는지 확인하십시오. 이와 유사하게, 브라우저가 아직 WebSocket protocol 를 지원하지 않는 경우[1]메시징 클라이언트 샘플 홈 페이지에서 제공되는 학습서 또는 클라이언트 유틸리티를 사용할 수 없습니다.

23 페이지의 표 1에 최신 버전이 테스트되어 메시징 클라이언트와 함께 작업할 수 있도록 표시된 브라우저가 나열되어 있습니다.

표 *1. JavaScript*용 *MQTT* 메시징 클라이언트와 함께 사용할 수 있도록 지원되는 브라우저

Android	**iOS**	**Linux**	**Windows**
Firefox for Android 19.0 이상 Chrome for Android 25.0 이상	Safari 6.0 이상 Chrome 14.0 이상	Firefox 6.0 이상 Chrome 14.0 이상	Firefox 6.0 이상 Chrome 14.0 이상

이 태스크 정보

이 태스크의 단계는 대부분 MQTT 서버를 구성하는 단계입니다. JavaScript용 메시징 클라이언트에 액세스하려면 WebSocket protocol을 지원하는 브라우저를 실행하기만 하면 됩니다.

IBM WebSphere MQ의 경우 샘플 채널을 작성하여 IBM WebSphere MQ Telemetry를 사용하는 단계를 수행하십시오. 포트 1883에서 샘플 기본 MQTT WebSockets 채널에 연결하십시오. 메시징 클라이언트 샘플 홈 페이지 URL은 IBM WebSphere MQ에서 `http://`*`hostname`*`:1883` 입니다.

IBM MessageSight에서는 어플라이언스를 설치한 후 설정하고 연결을 승인하도록 메시징 허브를 구성한 후 MQTT WebSockets 엔드 포인트를 작성하십시오.

프로시저

1. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하고 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   9 페이지의 『MQTT 클라이언트 시작하기』의 내용을 참조하십시오.
2. JavaScript용 MQTT 메시징 클라이언트 샘플 HTML 페이지에서 연결을 승인하도록 MQTT 서버를 구성하십시오.
   - IBM WebSphere MQ:
     - MQTT에 대해 구성된 IBM WebSphere MQ 큐 관리자가 이미 있는 경우, 채널 정의의 프로토콜을 변경하여 MQTT와 HTTP를 모두 지원하십시오. **`ALTER CHANNEL`**의 내용을 참조하십시오.
     - IBM WebSphere MQ 큐 관리자를 작성하고 샘플 MQTT WebSockets 엔드포인트를 구성하려면 다음 태스크 중 하나를 완료하십시오.
       - 134 페이지의 『명령행에서 MQTT 서비스 구성』
       - 136 페이지의 『IBM WebSphere MQ Explorer로 MQTT 서비스 구성』
3. 디바이스에서 웹 브라우저를 여십시오.
4. 메시징 클라이언트 샘플 홈 페이지의 URL을 입력하십시오.
   - IBM WebSphere MQ에서는 `http://`*`hostname`*`:1883` 입니다.
   - IBM MessageSight의 경우 URL이 `http://`*`hostname`*`:`*`port`*입니다.

     여기서 *`hostname`*은 IBM MessageSight 어플라이언스에서 클라이언트를 연결할 엔드 포인트로 구성한 이더넷 소켓의 DNS 이름 또는 IP 주소이고 *`port`*는 클라이언트의 엔드 포인트에 지정한 TCP/IP 포트 번호입니다.

   메시징 클라이언트 샘플 홈 페이지가 표시됩니다.

[1] 특히 RFC 6455 (WebSocket) 표준을 지원하지 않는 경우입니다.

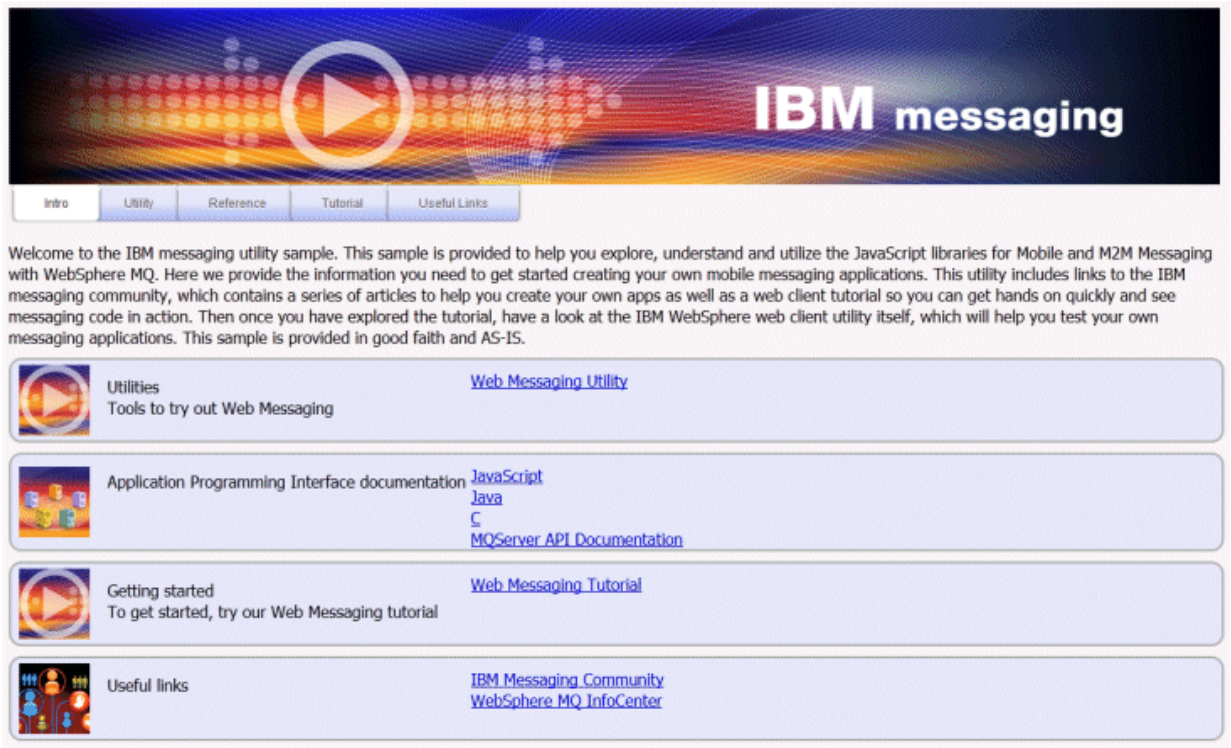


그림 *9. JavaScript*용 *MQTT* 메시징 클라이언트 샘플 홈 페이지

결과

WebSockets의 MQTT 채널을 구성했습니다.

JavaScript용 MQTT 메시징 클라이언트의 샘플 홈 페이지에서 **웹 메시징 유틸리티**를 클릭하여 메시징 클라이언트 API의 여러 기능을 시험해 보십시오. 예를 들면 큐 관리자에 연결하고 메시지를 구독한 후 일부 메시지를 발행할 수 있습니다. **웹 메시징 학습서**를 클릭하여 JavaScript API의 MQTT 메시징 클라이언트를 호출하는 웹 페이지를 작성하는 방법을 학습할 수도 있습니다.

관련 개념

112 페이지의 『JavaScript용 MQTT 메시징 클라이언트와 웹 앱』
116 페이지의 『JavaScript에서 메시징 앱을 프로그래밍하는 방법』

관련 태스크

74 페이지의 『SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets 연결』
SSL및 WebSocket protocol를 사용하는 JavaScript용 MQTT 메시징 클라이언트 샘플 HTML 페이지를 사용하여 웹 애플리케이션을 IBM WebSphere MQ 에 안전하게 연결하십시오.

C용 MQTT 클라이언트 시작하기

C 소스를 컴파일할 수 있는 모든 플랫폼에서 샘플 C용 MQTT 클라이언트를 시작하고 실행할 수 있습니다. MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ를 사용하여 C에 대해 샘플 MQTT 클라이언트를 실행할 수 있는지 확인하십시오.

시작하기 전에

- 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.
- 지원되는 C용 MQTT 클라이언트와 참조 MQTT 클라이언트 플랫폼은 IBM 모바일 메시징 및 M2M 클라이언트 팩 에 대한 시스템 요구사항의 내용을 참조하십시오.

이 태스크 정보

이 태스크를 수행하여 명령행 또는 Microsoft Visual Studio 2010에서 Windows의 샘플 C용 MQTT 클라이언트를 컴파일하고 실행하십시오. Microsoft Visual Studio 2010은 명령행 예제에서 클라이언트를 컴파일하는 데도 사용됩니다. 명령행 스크립트를 수정하여 다른 플랫폼에서 샘플을 컴파일하고 실행하십시오.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 MQTT version 3.1 프로토콜을 지원해야 합니다. IBM 의 모든 MQTT 서버는 IBM WebSphere MQ 및 IBM MessageSight를 포함합니다. 130 페이지의 『MQTT 서버 시작하기』의 내용을 참조하십시오.
2. 빌드 중인 플랫폼에서 C 개발 환경을 설치하십시오.

   이 토픽에서 예제에 나오는 Make 파일은 다음과 같은 도구를 대상으로 합니다.

   - 

     For iOS, on Apple Mac with OS X 10.8.2 with the iOS development tools from Xcode.
   - 

     Linux의 경우 Red Hat® Enterprise Linux 버전 6.2의 gcc 버전 4.4.6.

     `glibc` C 라이브러리의 지원되는 최소 레벨은 2.12이고 Linux 커널의 지원되는 최소 레벨은 2.6.32입니다.
   - 

     Microsoft Windows의 경우 Visual Studio 버전 10.0.
3. 모바일 메시징 및 M2M 클라이언트 팩 을 다운로드하고 MQTT SDK를 설치하십시오.

   설치 프로그램은 존재하지 않으며 다운로드한 파일을 펼치기만 하면 됩니다.

   a. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

   b. SDK를 설치할 폴더를 작성하십시오.

      해당 폴더의 이름을 `MQTT`로 지정할 수도 있습니다. 이 폴더의 경로는 여기에서 *`sdkroot`*로 참조됩니다.

   c. 압축된 모바일 메시징 및 M2M 클라이언트 팩 파일 컨텐츠를 *`sdkroot`*로 펼치십시오. 이 펼치기를 통해 *`sdkroot`*`\SDK`에서 시작되는 디렉토리 트리가 작성됩니다.
4. 옵션: 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』의 단계를 수행하십시오.

   모바일 메시징 및 M2M 클라이언트 팩에 대상 플랫폼의 C 클라이언트 라이브러리가 포함되지 않은 경우에만 이 단계를 수행하십시오.
5. MQTT 클라이언트 샘플 C앱 (`MQTTV3Sample.c`) 을 컴파일하고 실행하십시오.

   - 명령행에서 25 페이지의 『명령행에서 MQTT 클라이언트 샘플 C 앱 컴파일하고 실행』의 단계를 수행하십시오.
   - IDE에서 26 페이지의 『Microsoft Visual Studio에서 MQTT 클라이언트 샘플 C 앱 컴파일하고 실행』의 단계를 수행하십시오.

명령행에서 MQTT 클라이언트 샘플 C 앱 컴파일하고 실행

명령행에서 MQTT 클라이언트 샘플 C 앱을 컴파일하고 실행합니다. 샘플은 MQTT SDK에 있습니다. 이 샘플은 MQTT 발행자 및 구독자를 보여줍니다.

시작하기 전에

C 개발 환경을 설치하십시오 (예: Microsoft Visual Studio 2010).

이 태스크 정보

SDK 클라이언트 서브디렉토리(*`sdkroot`*`\SDK\clients\c\samples`)에 있는 C 샘플(`MQTTV3Sample`)을 컴파일하고 실행하십시오.

프로시저

클라이언트 샘플 디렉토리에서 스크립트를 작성하여 컴파일하고 선택한 플랫폼에서 `Sample` 를 실행하십시오.

다음 스크립트는 Microsoft Visual Studio 2010으로 빌드된 Windows 32비트 플랫폼에서 샘플을 컴파일하고 실행합니다. *sdkroot*\SDK\clients\c\samples 서브디렉토리에서 스크립트를 실행하십시오.

```
@echo off
setlocal
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3Sample.c" /link /
nologo ..\windows_ia32\mqttv3c.lib
set path=%path%;..\windows_ia32;
start "MQTT Subscriber" MQTTV3Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
MQTTV3Sample -b localhost -p 1883
pause
endlocal
```

결과

발행자와 구독자는 출력을 명령 창에 기록합니다.

```
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
MQTTV3Sample.c
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/C/v3" qos 2
Disconnected
Press any key to continue . . .
```

그림 *10.* 발행자의 출력

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Topic:           MQTTV3Sample/C/v3
Message:         Message from MQTTv3 C client
QoS:             2
```

그림 *11.* 구독자의 출력

Microsoft Visual Studio에서 MQTT 클라이언트 샘플 C 앱 컴파일하고 실행

Microsoft Visual Studio에서 MQTT 클라이언트 샘플 C 앱을 컴파일하고 실행합니다. 샘플은 모바일 메시징 및 M2M 클라이언트 팩에 있습니다. 이 샘플은 MQTT 발행자 및 구독자를 보여줍니다.

시작하기 전에

예제에서는 Microsoft Visual Studio 2010을 사용합니다. Windows 및 다른 플랫폼에서 다른 C 개발 환경을 사용할 수 있습니다(예: Eclipse IDE for C/C++ Developers).

이 태스크 정보

Microsoft Visual Studio 2010을 사용하여 C 샘플 `MQTTV3Sample` 를 컴파일하고 실행하십시오. `MQTTV3Sample.c`는 SDK 클라이언트 서브디렉토리(*sdkroot*\SDK\clients\c\samples)에 있습니다.

프로시저

1. Microsoft Visual Studio를 시작하십시오.
2. 기존 코드를 사용하여 새 프로젝트를 작성하십시오.
   a) **파일** > **새로 작성** > **기존 코드에서 프로젝트**를 클릭하십시오.
   b) 작성할 프로젝트의 유형으로 **Visual C++**를 선택하십시오.
   c) **다음**을 클릭하십시오.
3. **프로젝트 위치 및 소스 파일** 창에서 매개변수를 지정하십시오.

a) **찾아보기**를 클릭하여 *sdkroot*\SDK\clients\c\samples 디렉토리를 찾으십시오.

b) 프로젝트 이름을 MQTTV3Sample로 지정하십시오.

c) **다음**을 클릭하십시오.

4. **프로젝트 유형** 목록에서 **콘솔 애플리케이션 프로젝트**를 선택하십시오. **완료**를 클릭하십시오.
5. 디버그 구성만 구성하십시오.

   기본적으로 Microsoft Visual Studio는 릴리스 구성과 디버그 구성을 둘 다 작성합니다. 학습서에서는 디버그 구성을 구성합니다. 빌드 오류를 표시하지 않으려면 릴리스 구성에 대한 **빌드** 옵션을 선택 취소하십시오.

   a) **프로젝트** > **MQTTV3Sample 등록 정보** > **구성 관리자**를 누르십시오. **릴리스**를 **활성 솔루션 구성**으로 선택한 후 **빌드**를 선택 취소하십시오.

   b) **활성 솔루션 구성** > **닫기**로 **디버그** 를 선택하십시오.

   다음과 같은 단계 모두에서 디버그 구성을 수정하는지 확인하십시오.
6. **MQTTV3Sample 특성 페이지**에서 **C/C++** 설정을 수정하십시오.

   a) **MQTTV3Sample 등록 정보 페이지** 창에서 **구성 특성** > **C/C++** > **일반**을 여십시오.

   b) 일반 특성 목록에서 **추가 포함 디렉토리**를 클릭하여 디렉토리 경로를 *sdkroot*\SDK\clients\c\include에 추가한 후 **적용**을 클릭하십시오.
7. **링커** 설정을 수정하십시오.

   a) **구성 특성** > **링커** > **일반**을 여십시오.

   b) 일반 특성 목록에서 **추가 라이브러리 디렉토리**를 클릭하여 디렉토리 경로를 *sdkroot*\SDK\clients\c\windows_ia32에 추가하십시오.

   c) 링커 특성 목록에서 **명령행**을 클릭하십시오. **추가 옵션** 데이터 입력 항목 영역에 mqttv3c.lib를 입력한 후 **적용**을 클릭하십시오.
8. 프로젝트에서 MQTTV3SSample.c 소스 파일을 제거하십시오.

   a) **솔루션 탐색기** 창에서 **MQTTV3sample** > **소스 파일** 폴더를 여십시오.

   b) **MQTTV3SSample.c** > **프로젝트에서 제외** 를 마우스 오른쪽 단추로 누르십시오.
9. MQTTV3Sample 프로젝트를 빌드하십시오.

   a) 솔루션 탐색기에서 **MQTTV3sample** 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **빌드**를 클릭하십시오.

   오류 없이 빌드가 완료됩니다.
10. 2개의 새 프로젝트를 추가하여 **MQTTV3Sample**을 구독자 및 발행자 런타임 인스턴스로 실행하십시오.

    발행자 및 구독자 프로젝트는 **MQTTV3Sample**을 실행하는 데 필요한 명령을 포함해야 합니다. 두 프로젝트는 빌드되지 않으며 코드를 포함하고 있지 않습니다.

    a) **솔루션 탐색기**에서 **솔루션 'MQTTV3Sample'** > **추가** > **새 프로젝트**를 마우스 오른쪽 단추로 클릭하십시오.

    b) **이름** 필드에 Subscriber를 입력하십시오. **Win32 콘솔 애플리케이션**은 선택된 상태로 두십시오. **확인**을 클릭하십시오.

    Win32 애플리케이션 마법사가 시작됩니다.

    c) **Win32 애플리케이션 마법사**에서 **다음**을 클릭하십시오. **비어 있는 프로젝트** > **완료**를 선택하십시오.

    d) 이 단계를 반복하여 발행자 프로젝트를 추가하십시오.

    e) 구독자 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **시작 프로젝트로 설정**을 클릭하십시오.

    솔루션 탐색기 창이 28 페이지의 그림 12에 표시됩니다.

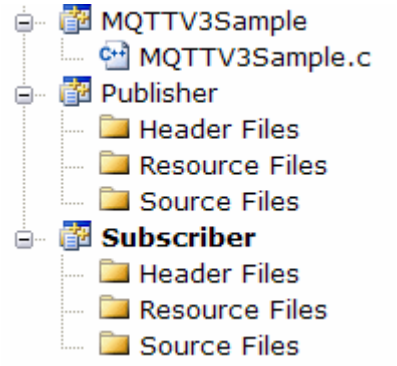


그림 *12. MQTTV3Sample* 솔루션

11. 구독자 특성 페이지를 구성하십시오.
    a) 솔루션 탐색기에서 **Subscriber** 를 마우스 오른쪽 단추로 클릭하십시오. **특성** > **구성 특성** > **특성** > **디버깅**

       창 제목이 **구독자 특성 페이지**인지 확인하십시오.

    b) **환경**을 클릭하십시오. `path=%path%;`*`sdkroot`*`\SDK\clients\c\windows_ia32`를 입력한 후 **적용**을 클릭하십시오.

       사용자 환경에 맞게 *`sdkroot`* 를 변경하십시오.

    c) **명령**을 클릭한 후 `$(TargetPath)`를 `MQTTV3Sample` 모듈의 경로로 바꾸십시오.

       예: *`sdkroot`*`\SDK\clients\c\samples\debug\MQTTV3Sample`

       사용자 환경에 맞게 *`sdkroot`* 를 변경하십시오.

    d) **명령 인수** 를 누르고 `-a subscribe -b localhost -p 1883` 를 입력하고 **적용**을 누르십시오.

12. 발행자 특성 페이지를 구성하십시오.

    팁: 솔루션 탐색기 창에서 프로젝트를 클릭하여 특성 페이지 프로젝트를 전환할 수 있습니다.

    a) 발행자에 대해 구독자 단계를 반복하십시오.

       명령 인수는 `-b localhost -p 1883` 입니다.

13. 발행자 및 구독자 프로젝트를 빌드하는 빌드 프로세스를 중지하십시오.
    a) 프로젝트의 특성 페이지에서 **구성 관리자**를 클릭한 후 릴리스 구성과 디버그 구성에서 발행자 및 구독자에 대해 **빌드**를 선택 취소하십시오. **닫기**를 클릭하십시오.
14. 샘플을 실행하십시오.
    a) **F5**를 클릭하여 구독자를 시작하십시오.
    b) 솔루션 탐색기에서 **발행자**를 마우스 오른쪽 단추로 클릭한 후 **디버그** > **새 인스턴스 시작**을 클릭하십시오.

결과

발행자 및 구독자가 명령 창에 출력됩니다. Visual Studio가 발행자 창을 닫습니다. 다음 그림에 표시된 구독자 창을 본 후, 구독자 창을 닫으십시오.

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Topic:          MQTTV3Sample/C/v3
Message:        Message from MQTTv3 C client
QoS:            2
```

그림 *13.* 구독자의 출력

다음에 수행할 작업

비동기 발행자 및 구독자를 빌드하고 실행하십시오.The examples are `MQTTV3ASample.c` and `MQTTV3ASSample.c` in *sdkroot*`\SDK\clients\c\samples`.

C용 MQTT 클라이언트 라이브러리 빌드

다음 단계를 수행하여 C용 MQTT 클라이언트 라이브러리를 빌드하십시오. 이 토픽에는 다수의 플랫폼에 대한 컴파일 및 링크 스위치와 iOS 및 Windows에서 라이브러리를 빌드하는 예제가 포함되어 있습니다.

시작하기 전에

1. 필요한 경우에만 C 클라이언트 라이브러리를 빌드하십시오. 대상 플랫폼에 일치하는 항목이 있는 경우에만 `SDK\clients\c` 서브디렉토리의 SDK(Software Development Kit)에 있는 사전 빌드된 클라이언트 라이브러리를 링크하십시오.
2. MQTT 서버를 구성하여 MQTT 클라이언트 샘플 C 앱으로 빌드하는 라이브러리를 테스트하십시오. 130 페이지의 『MQTT 서버 시작하기』를 참조하십시오. MQTT 클라이언트 샘플 앱 중 하나를 실행하여 서버 구성을 확인하십시오.
3. SSL(Secure Sockets Layer)을 지원하는 C 라이브러리의 보안 버전을 빌드하는 경우 OpenSSL 라이브러리도 빌드해야 합니다. 43 페이지의 『OpenSSL 패키지 빌드』를 참조하십시오.

   중요사항: OpenSSL 패키지의 다운로드 및 재분배는 엄격한 가져오기 및 내보내기 규제 및 오픈 소스 라이센싱 조건에 적용됩니다. 패키지를 다운로드할지 여부를 결정하기 전에 제한 및 경고에 주의하십시오.

이 태스크 정보

43 페이지의 『OpenSSL 패키지 빌드』의 지시사항에 따라 OpenSSL 라이브러리를 다운로드하고 빌드하십시오. C용 MQTT 클라이언트 라이브러리의 보안 버전을 빌드하려면 OpenSSL을 빌드해야 합니다. MQTT 라이브러리의 보안이 설정되지 않은 버전을 빌드하는 데는 OpenSSL이 필요하지 않습니다. 단계에는 iOS 및 Windows용 라이브러리를 빌드하는 예제가 포함되어 있습니다.

빌드 플랫폼에 C 개발 라이브러리 도구와 MQTT SDK(Software Development Toolkit)를 다운로드하여 C용 MQTT 클라이언트 라이브러리를 빌드하십시오. Make 파일을 작성하여 대상 플랫폼에 대한 라이브러리를 빌드하여 30 페이지의 『MQTT 다른 플랫폼에 대한 빌드 옵션』에 문서화되어 있는 옵션을 통합하십시오. 플랫폼별 Make 파일을 빌드하고 실행하는 단계는 다음 위치에 있습니다.

- iOS 32 페이지의 『iOS 디바이스에서 사용할 Apple Mac의 C용 MQTT 클라이언트 라이브러리 빌드』
- Windows 37 페이지의 『Windows에서 MQTT 라이브러리 빌드』

프로시저

1. 빌드 중인 플랫폼에서 C 개발 환경을 설치하십시오.

   이 토픽에서 예제에 나오는 Make 파일은 다음과 같은 도구를 대상으로 합니다.

   - iOS For iOS, on Apple Mac with OS X 10.8.2 with the iOS development tools from Xcode.
   - Linux Linux의 경우 Red Hat Enterprise Linux 버전 6.2의 gcc 버전 4.4.6.

     `glibc` C 라이브러리의 지원되는 최소 레벨은 2.12이고 Linux 커널의 지원되는 최소 레벨은 2.6.32입니다.
   - Windows Microsoft Windows의 경우 Visual Studio 버전 10.0.
2. 모바일 메시징 및 M2M 클라이언트 팩 을 다운로드하고 MQTT SDK를 설치하십시오.

   설치 프로그램은 존재하지 않으며 다운로드한 파일을 펼치기만 하면 됩니다.

   a. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

   b. SDK를 설치할 폴더를 작성하십시오.

해당 폴더의 이름을 MQTT로 지정할 수도 있습니다. 이 폴더의 경로는 여기에서 *sdkroot*로 참조됩니다.

c. 압축된 모바일 메시징 및 M2M 클라이언트 팩 파일 컨텐츠를 *sdkroot*로 펼치십시오. 이 펼치기를 통해 *sdkroot*\SDK에서 시작되는 디렉토리 트리가 작성됩니다.

3. C용 MQTT 클라이언트 라이브러리의 소스 코드를 펼치십시오.

   소스 코드 압축 파일은 *sdkroot*\SDK\clients\c\source.zip입니다.

4. 옵션: OpenSSL을 빌드하십시오.

   43 페이지의 『OpenSSL 패키지 빌드』를 참조하십시오.

5. C용 MQTT 클라이언트 라이브러리를 빌드하십시오.

   라이브러리를 빌드하는 명령 및 옵션이 30 페이지의 『MQTT 다른 플랫폼에 대한 빌드 옵션』에 나열됩니다.

   다음 예제의 단계를 수행하여 대상 플랫폼에서 사용할 C용 MQTT 클라이언트 라이브러리를 빌드하는 Make 파일을 작성하십시오.

   - 32 페이지의 『iOS 디바이스에서 사용할 Apple Mac의 C용 MQTT 클라이언트 라이브러리 빌드』
   - 37 페이지의 『Windows에서 MQTT 라이브러리 빌드』

MQTT 다른 플랫폼에 대한 빌드 옵션

다음 테이블에는 다양한 플랫폼에서 C용 MQTT 클라이언트 라이브러리를 빌드하는 컴파일러 및 빌드 옵션이 나열됩니다.

표 *2. MQTT* 다른 플랫폼에 대한 빌드 옵션

플랫폼	Compiler	Compiler Options	Linker Options	Extra Options
AIX®	gcc	-fPIC -Os -Wall -DREVERSED -I *MQTTCLIENT_DIR*	-Wl,-G	
Linux s390x			-shared -Wl,-soname, libmqttv3c.so	-m64
Linux x86-64		-fPIC -Os -Wall -I *MQTTCLIENT_DIR*		
Linux x86-32				-m32
Linux ARM (glibc)	arm-linux-gcc			
Linux ARM (uclibc)	arm-unknown-linux -uclibcgnueabi-gcc			

표 2. *MQTT* 다른 플랫폼에 대한 빌드 옵션 (계속)

플랫폼	Compiler	Compiler Options	Linker Options	Extra Options
Windows 32 비트	cl	/D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS" / nologo /c /O2 /W3 /Fd /MD /TC	/nologo /machine:x86 /manifest kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib ws2_32.lib /implib:mqttv3c.lib) /pdb:mqttv3c.pdb) /map:mqttv3c.map)	
iOS ARMv7	gcc -arch armv7	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot /Applications/Xcode.app/\ Contents/Developer/Platforms/\ iPhoneOS.platform/Developer/SDKs/\ iPhoneOS6.0.sdk	-L/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk/usr/lib/system	
iOS ARMv7s	gcc -arch armv7s			
iOS ARMi386	gcc -arch i386	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot /Applications/Xcode.app/\ Contents/Developer/Platforms/\ iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk	-L/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk/usr/lib/system	

iOS *iOS 디바이스에서 사용할 Apple Mac의 C용 MQTT 클라이언트 라이브러리 빌드*

다음 단계를 수행하여 iOS 디바이스에서 사용할 수 있도록 Apple Mac의 C용 MQTT 클라이언트 라이브러리를 빌드하는 Make 파일을 작성하십시오.

시작하기 전에

1. OS X 10.8.2 이상이 설치된 Apple Mac에서 빌드 도구를 설치하고 Make 파일을 개발 및 실행하십시오.
2. **make** 프로그램을 포함하는 Xcode에 대한 명령행 도구를 설치하십시오. Xcode에서 명령행 도구를 다운로드하십시오.

이 태스크 정보

ARMv7 또는 ARMv7s 프로세서를 실행하는 iPhone이나 iPad, i386-64비트 프로세서에서 실행되는 iPhone 시뮬레이터의 C용 MQTT 클라이언트 라이브러리를 빌드하는 Make 파일을 작성합니다. iOS 디바이스 목록을 참조하십시오.

팁: 36 페이지의 『MQTTios.mak Make 파일 나열』 에 전체 Make 파일이 나열됩니다.

1. 목록을 복사하여 파일에 붙여넣으십시오.
2. 대상 뒤에 오는 각 행의 선두 문자를 탭으로 변환하십시오. 34 페이지의 『8』 단계를 참조하십시오.
3. 프로시저의 35 페이지의 『9』 단계에 나열된 명령을 사용하여 실행하십시오.

프로시저

1. iOS 개발 도구를 다운로드하여 설치하십시오.
   a. 관리 권한이 있는 사용자 ID로 로그온하십시오.
   b. AppleMac 버전이 10.8.2 또는 이상인지 확인하십시오.
   c. 웹 사이트 Xcode 로 이동하여 Mac 앱 스토어에서 Xcode 를 다운로드하십시오.
   d. Xcode, 명령행 환경, 시뮬레이터를 설치하십시오.

   Mac 앱 스토어에서 여러 버전의 시뮬레이터를 제공하는 경우에는 사용자 앱의 대상으로 지정하는 iOS 레벨과 호환되는 버전을 선택하십시오.
2. Make 파일 `MQTTios.mak`를 작성하십시오.

   프롤로그 추가:

```
# 빌드 출력은 현재 디렉토리에서 생성됩니다.
# MQTTCLIENT_DIR는 MQTT 클라이언트 소스 코드를 포함하고 있는 기본 디렉토리를 가리켜야 한다.
# 기본 MQTTCLIENT_DIR는 현재 디렉토리이다.
# 기본 TOOL_DIR은 /Applications/Xcode.app/Contents/Developer/Platforms 입니다.
# sdkroot/sdk/clients/c/mqttv3c/src에 상대적인 기본 OPENSSL_DIR은 sdkroot/openssl입니다.
# OPENSSL_DIR은 OpenSSL 빌드를 포함하는 기본 디렉토리를 가리켜야 합니다.
# 예제: -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. MQTT 소스 코드의 위치를 설정하십시오.

   MQTT 소스 파일과 동일한 디렉토리에서 make 파일을 실행하거나 `MQTTCLIENT_DIR` 명령행 매개변수를 설정하십시오.

```
make -f makefile MQTTCLIENT_DIR=sdkroot/SDK/clients/c/mqttv3c/src
```

   다음 행을 Make 파일에 추가하십시오.

```
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
```

   예제는 `VPATH` 를 디렉토리로 설정합니다. 여기서 **make** 는 명시적으로 식별되지 않은 소스 파일을 검색합니다 (예: 빌드에 필요한 모든 헤더 파일).

4. 옵션: OpenSSL 라이브러리의 위치를 설정하십시오.

   이 단계는 C 라이브러리에 대한 MQTT 클라이언트의 SSL 버전을 빌드하는 데 필요합니다.

   MQTT SDK를 확장할 때 OpenSSL 라이브러리의 기본 경로를 동일한 디렉토리로 설정하십시오. 그렇지 않으면 OPENSSL_DIR을 명령행 매개변수로서 설정하십시오.

```
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../openssl-1.0.1c
endif
```

   팁: *OpenSSL* 은 모든 OpenSSL 서브디렉토리를 포함하는 OpenSSL 디렉토리입니다. 디렉토리 트리에 불필요한 빈 상위 디렉토리가 있으므로 이를 펼친 곳에서 이동해야 할 수도 있습니다.

5. 개발 도구 디렉토리를 설정하십시오.

   다른 위치에 Xcode를 설치한 경우에는 명령행에서 TOOL_DIR을 설정하십시오.

```
ifndef TOOL_DIR
   TOOL_DIR = /Applications/Xcode.app/Contents/Developer/Platforms endif
IPHONE_SDK = iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
IPHONESIM_SDK = iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
SDK_ARM = ${TOOL_DIR}/${IPHONE_SDK}
SDK_i386 = ${TOOL_DIR}/${IPHONESIM_SDK}
```

6. 각 MQTT 라이브러리를 빌드하는 데 필요한 모든 소스 파일을 선택하십시오. 또한 빌드할 MQTT 라이브러리의 이름 및 위치를 설정하십시오.

   make 파일에 다음 행을 추가하여 모든 MQTT 소스 파일을 나열하십시오.

```
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
```

   소스 파일은 The source files depend o동기 또는 비동기 라이브러리를 빌드 중인지 여부와 라이브러리에 SSL이 포함되는지 여부에 달려 있습니다.

   다음 행 중 하나 이상을 추가하십시오(빌드할 대상에 따라 다름). 공유 라이브러리가 darwin_x86_64 디렉토리에서 작성됩니다.

   - 동기, 보안 설정되지 않음:

```
MQTTLIB = mqttv3c
SOURCE_FILES = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB} .a
```

   - 동기, 보안 설정됨:

```
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S}.
```

   - 비동기, 보안 설정되지 않음:

```
MQTTLIB_A = mqttv3a
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/
SSLSocket.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A} .a
```

   - 비동기 보안 설정됨:

```
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS}.
```

7. 컴파일러 및 컴파일러 옵션을 정의하십시오.

   다양한 플랫폼에 대한 MQTT 빌드 옵션에 표시된 다른 플랫폼에 대한 옵션을 참조하십시오.

   a) Gnu 프로젝트 C 및 C++(**gcc**)를 컴파일러로 설정하십시오.

세 가지 교차 컴파일러를 선택하여 다양한 디바이스 및 iPhone 시뮬레이터용 라이브러리를 빌드하십시오.

```
CC = iPhoneOS.platform/Developer/usr/bin/gcc
CC_armv7 = ${TOOL_DIR}/${CC} -arch armv7
CC_armv7s = ${TOOL_DIR}/${CC} -arch armv7s
CC_i386 = ${TOOL_DIR}/${CC} -arch i386
```

b) 컴파일러 옵션을 추가하십시오.

```
CCFLAGS = -Os -Wall -fomit-frame-pointer
```

c) 포함 경로를 추가하십시오.

```
CCFLAGS_SO_ARM = ${CCFLAGS} -isysroot ${SDK_ARM} -I${OPENSSL_DIR}/include -L$
{SDK_ARM}/usr/lib/system
CCFLAGS_SO_i386 = ${CCFLAGS} -isysroot ${SDK_i386} -I${OPENSSL_DIR}/include--L$
{SDK_i386}/usr/lib/system
```

8. 빌드 대상을 정의하십시오.

팁: 대상의 구현을 정의하는 각 연속된 행은 탭 문자로 시작해야 합니다.

a) **all** 대상을 정의하십시오.

"all" 대상은 모든 라이브러리를 빌드합니다.

```
all: ${MQTTLIB_DARWIN} ${MQTTLIB_DARWIN_A} ${MQTTLIB_DARWIN_AS} ${MQTTLIB_DARWIN_S}
```

이를 처음에 나열하면 기본 대상이 됩니다.

a) 동기, 보안 설정되지 않은 라이브러리 `libmqttv3c.a`를 빌드하십시오.

```
${MQTTLIB_DARWIN}: ${SOURCE_FILES}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@
```

명령문 `rm *.o` 은 각 라이브러리에 대해 작성된 모든 오브젝트 파일을 삭제합니다. `lipo` 는 세 개의 라이브러리를 모두 하나의 파일로 연결합니다.

b) 비동기, 보안 설정되지 않은 라이브러리 `libmqttv3a.a`를 빌드하십시오.

```
${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@
```

명령문 `rm *.o` 은 각 라이브러리에 대해 작성된 모든 오브젝트 파일을 삭제합니다. `lipo` 는 세 개의 라이브러리를 모두 하나의 파일로 연결합니다.

c) 동기, 보안 설정된 라이브러리 `libmqttv3cs.a`를 빌드하십시오.

```
${MQTTLIB_DARWIN_S}: ${SOURCE_FILES_S}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o
$@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o
$@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o
$@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@
```

명령문 `rm *.o` 은 각 라이브러리에 대해 작성된 모든 오브젝트 파일을 삭제합니다. `lipo` 는 세 개의 라이브러리를 모두 하나의 파일로 연결합니다.

d) 비동기, 보안 설정된 라이브러리 `libmqttv3as.a`를 빌드하십시오.

```
${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o
$@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o
$@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o
$@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@
```

명령문 `rm *.o` 은 각 라이브러리에 대해 작성된 모든 오브젝트 파일을 삭제합니다. `lipo` 는 세 개의 라이브러리를 모두 하나의 파일로 연결합니다.

e) **`clean`** 대상을 정의하십시오.

"clean" 대상은 Make 파일에 의해 생성되는 모든 파일 및 디렉토리를 제거합니다.

```
.PHONY: 깨끗한
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

9. Make 파일을 실행하십시오.

```
make -f  MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
```

결과

다음과 같은 파일이 *sdkroot/sdk/clients/c/mqttv3c/src*/darwin_x86_64 디렉토리에서 작성됩니다.

```
libmqttv3c.a.armv7
libmqttv3c.a.armv7s
libmqttv3c.a.i386
libmqttv3c.a
libmqttv3a.a.armv7
libmqttv3a.a.armv7s
libmqttv3a.a.i386
libmqttv3a.a
libmqttv3cs.a.armv7
```

```
libmqttv3cs.a.armv7s
libmqttv3cs.a.i386
libmqttv3cs.a
libmqttv3as.a.armv7
libmqttv3as.a.armv7s
libmqttv3as.a.i386
libmqttv3as.a
```

MQTTios.mak Make 파일 나열

```
# 빌드 출력은 현재 디렉토리에서 생성됩니다.
# MQTTCLIENT_DIR는 MQTT 클라이언트 소스 코드를 포함하고 있는 기본 디렉토리를 가리켜야 한다.
# 기본 MQTTCLIENT_DIR는 현재 디렉토리이다.
# 기본 TOOL_DIR은 /Applications/Xcode.app/Contents/Developer/Platforms 입니다.
# sdkroot/sdk/clients/c/mqttv3c/src에 상대적인 기본 OPENSSL_DIR은 sdkroot/openssl입니다.
# OPENSSL_DIR은 OpenSSL 빌드를 포함하는 기본 디렉토리를 가리켜야 합니다.
# 예제: -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../openssl-1.0.1c
endif
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
ifndef TOOL_DIR
   TOOL_DIR = /Applications/Xcode.app/Contents/Developer/Platforms endif
IPHONE_SDK = iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
IPHONESIM_SDK = iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
SDK_ARM = ${TOOL_DIR}/${IPHONE_SDK}
SDK_i386 = ${TOOL_DIR}/${IPHONESIM_SDK}

MQTTLIB = mqttv3c
SOURCE_FILES = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB} .a
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S}.
MQTTLIB_A = mqttv3a
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A} .a
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS}.

# compiler
CC = iPhoneOS.platform/Developer/usr/bin/gcc
CC_armv7 = ${TOOL_DIR}/${CC} -arch armv7
CC_armv7s = ${TOOL_DIR}/${CC} -arch armv7s
CC_i386 = ${TOOL_DIR}/${CC} -arch i386
CCFLAGS = -Os -Wall -fomit-frame-pointer
CCFLAGS_SO_ARM = ${CCFLAGS} -isysroot ${SDK_ARM} -I${OPENSSL_DIR}/include -L${SDK_ARM}/usr/lib/
system
CCFLAGS_SO_i386 = ${CCFLAGS} -isysroot ${SDK_i386} -I${OPENSSL_DIR}/include--L$
{SDK_i386}/usr/lib/system

# targets
all: ${MQTTLIB_DARWIN} ${MQTTLIB_DARWIN_A} ${MQTTLIB_DARWIN_AS} ${MQTTLIB_DARWIN_S}

${MQTTLIB_DARWIN}: ${SOURCE_FILES}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@

${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
```

```
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@

${MQTTLIB_DARWIN_S}: ${SOURCE_FILES_S}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o $@.armv7s
*.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@

${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o $@.armv7s
*.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
    rm *.o
    lipo-작성 $@.armv7 $@.armv7s $@.i386 -output $@

.PHONY: 깨끗한
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

Windows *Windows*에서 *MQTT* 라이브러리 빌드

다음 단계를 수행하여 Windows의 C용 MQTT 클라이언트 라이브러리를 빌드하는 Make 파일을 작성하십시오.

시작하기 전에

1. 필요한 경우, Gnu 에 대해 작성된 make 파일과 호환 가능한 **Make** 버전을 빌드 워크스테이션에 설치하십시오. 그렇지 않으면 Gnu 를 다운로드하여 빌드하십시오. Gnu 작성을 참조하십시오. 웹 사이트 Make for Windows는 Windows에 대한 **Make** 의 설치 가능 버전을 제공합니다.
2. Make 파일 예제에서 정리 대상을 사용하려면 Windows용 Linux 명령도 필요합니다. Cygwin과 같은 웹 사이트에서 Windows 에 대한 Linux 명령을 얻을 수 있습니다.

이 태스크 정보

Windows 32비트에서 사용할 C용 MQTT 클라이언트 라이브러리를 빌드하는 Make 파일을 작성합니다.

팁: 41 페이지의 『MQTTwin.mak Make 파일 나열』 에 전체 Make 파일이 나열됩니다.

1. 목록을 복사하여 파일에 붙여넣으십시오.
2. 대상 뒤에 오는 각 행의 선두 문자를 탭으로 변환하십시오. 40 페이지의 『7』 단계를 참조하십시오.

3. 프로시저의 41 페이지의 『9』 단계에 나열된 명령을 사용하여 실행하십시오.

프로시저

1. Make 파일 MQTTwin.mak를 작성하십시오.

   프롤로그 추가:

   ```
   # 빌드 출력은 현재 디렉토리에서 생성됩니다.
   # MQTTCLIENT_DIR는 MQTT 클라이언트 소스 코드를 포함하고 있는 기본 디렉토리를 가리켜야 한다.
   # 기본 MQTTCLIENT_DIR는 현재 디렉토리이다.
   # 기본 OPENSSL_DIR은 sdkroot\openSSL이며 sdkroot\sdk\clients\c\mqttv3c\src에 상대적입니다.
   # OPENSSL_DIR은 OpenSSL 빌드를 포함하는 기본 디렉토리를 가리켜야 합니다.
   # 예제: -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
   # 빌드 환경을 설정하십시오. 예를 들면 다음과 같습니다.
   # %comspec% /k " " C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" " x86
   # set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
   ```

2. MQTT 소스 코드의 위치를 설정하십시오.

   MQTT 소스 파일과 동일한 디렉토리에서 make 파일을 실행하거나 MQTTCLIENT_DIR 명령행 매개변수를 설정하십시오.

   ```
   make -f makefile MQTTCLIENT_DIR=sdkroot/SDK/clients/c/mqttv3c/src
   ```

   다음 행을 Make 파일에 추가하십시오.

   ```
   ifndef MQTTCLIENT_DIR
     MQTTCLIENT_DIR = ${CURDIR}
   endif
   VPATH = ${MQTTCLIENT_DIR}
   ```

   예제는 VPATH 를 디렉토리로 설정합니다. 여기서 **make** 는 명시적으로 식별되지 않은 소스 파일을 검색합니다 (예: 빌드에 필요한 모든 헤더 파일).

3. 옵션: OpenSSL 라이브러리의 위치를 설정하십시오.

   이 단계는 C 라이브러리에 대한 MQTT 클라이언트의 SSL 버전을 빌드하는 데 필요합니다.

   MQTT SDK를 확장할 때 OpenSSL 라이브러리의 기본 경로를 동일한 디렉토리로 설정하십시오. 그렇지 않으면 OPENSSL_DIR을 명령행 매개변수로서 설정하십시오.

   ```
   ifndef OPENSSL_DIR
     OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../openssl-1.0.1c
   endif
   ```

   팁: *OpenSSL* 은 모든 OpenSSL 서브디렉토리를 포함하는 OpenSSL 디렉토리입니다. 디렉토리 트리에 불필요한 빈 상위 디렉토리가 있으므로 이를 펼친 곳에서 이동해야 할 수도 있습니다.

4. 각 MQTT 라이브러리를 빌드하는 데 필요한 모든 소스 파일을 선택하십시오. 또한 빌드할 MQTT 라이브러리의 이름 및 위치를 설정하십시오.

   make 파일에 다음 행을 추가하여 모든 MQTT 소스 파일을 나열하십시오.

   ```
   ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
   ```

   소스 파일은 The source files depend o동기 또는 비동기 라이브러리를 빌드 중인지 여부와 라이브러리에 SSL이 포함되는지 여부에 달려 있습니다.

   다음 행 중 하나 이상을 추가하십시오(빌드할 대상에 따라 다름). 공유 라이브러리 및 속성 정의 파일이 windows_ia32 디렉토리에서 작성됩니다.

   - 동기, 보안 설정되지 않음:

     ```
     MQTTLIB = mqttv3c
     MQTTDLL = windows_ia32/${MQTTLIB} .dll
     SOURCE_FILES = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
     ```

```
{ALL_SOURCE_FILES}}
MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource: ${MQTTDLL}\; 2
```

- 동기, 보안 설정됨:

```
MQTTLIB_S = mqttv3cs
MQTTDLL_S = windows_ia32/${MQTTLIB_S} .dll
SOURCE_FILES_S = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
```

- 비동기, 보안 설정되지 않음:

```
MQTTLIB_A = mqttv3a
MQTTDLL_A = windows_ia32/${MQTTLIB_A} .dll
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/
SSLSocket.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
```

- 비동기 보안 설정됨:

```
MQTTLIB_AS = mqttv3as
MQTTDLL_AS = windows_ia32/${MQTTLIB_AS} .dll
SOURCE_FILES_AS = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
```

5. 컴파일러 및 컴파일러 옵션을 정의하십시오.

   다양한 플랫폼에 대한 MQTT 빌드 옵션에 표시된 다른 플랫폼에 대한 옵션을 참조하십시오.

   a) Microsoft Visual C++를 컴파일러로 설정하십시오.

```
CC = cl
```

   b) 프리프로세서 옵션을 추가하십시오.

```
CPPFLAGS = /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
```

   c) 컴파일러 옵션을 추가하십시오.

```
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
```

   d) 포함 경로를 추가하십시오.

```
INC =  /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
```

   e) 옵션: 보안 라이브러리를 빌드하는 경우에는 프리프로세서 옵션을 추가하십시오.

```
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
```

   f) 옵션: 보안 라이브러리를 빌드하는 경우에는 OpenSSL 헤더 파일을 추가하십시오.

```
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/
```

   팁: 헤더 파일은 `${OPENSSL_DIR}/inc32/openssl`에 있지만 `ssl.h` 파일은 `"openssl/ssl.h"`와 함께 포함되어 있습니다.

6. 링커 및 링커 옵션을 설정하십시오.

   a) Microsoft Visual C++를 링커로 설정하십시오.

```
LD = link
```

   b) 링커 옵션을 추가하십시오.

```
LINKFLAGS = /nologo /machine:x86 /manifest /dll
```

   c) 라이브러리 경로를 추가하십시오.

```
WINLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
          advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib\
          odbc32.lib odbccp32.lib ws2_32.lib
```

d) 중간 출력 파일을 추가하십시오.

```
IMP = /함축: ${@:.dll=.lib}
LIBPDB = /pdb: ${@:.dll=.pdb}
LQAP = /map: ${@:.dll=.map}
```

e) 옵션: 보안 라이브러리를 빌드하는 경우에는 OpenSSL 라이브러리를 추가하십시오.

```
WINLIBS_S = ${WINLIBS} crypt32.lib ssleay32.lib libeay32.lib
```

f) 옵션: 보안 라이브러리를 빌드하는 경우에는 OpenSSL 라이브러리 경로를 추가하십시오.

```
LIBPATH_S = /LIBPATH:${OPENSSL_DIR}/lib
```

7. 4개의 빌드 대상을 정의하십시오.

a) **all** 대상을 정의하십시오.

팁: 대상의 구현을 정의하는 각 연속된 행은 탭 문자로 시작해야 합니다.

"all" 대상은 모든 라이브러리를 빌드합니다.

```
all: ${MQTTDLL} ${MQTTDLL_A} ${MQTTDLL_AS} ${MQTTDLL_S}
```

b) 동기, 보안 설정되지 않은 라이브러리 `mqttv3c.dll`를 빌드하십시오.

```
${MQTTDLL}: ${SOURCE_FILES}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out:${MQTTDLL}
    ${MANIFEST}
```

`-rm ${CURDIR}/MQTTAsync.obj` 명령문은 이전 대상에 대해 작성된 모든 `MQTTAsync.obj`를 삭제합니다. `MQTTAsync.obj`와 `MQTTClient.obj`는 상호 배타적입니다.

c) 비동기, 보안 설정되지 않은 라이브러리 `mqttv3a.dll`를 빌드하십시오.

```
${MQTTDLL_A}: ${SOURCE_FILES_A}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES_A}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out:${MQTTDLL_A}
    ${MANIFEST_A}
```

`-rm ${CURDIR}/MQTTClient.obj` 명령문은 이전 대상에 대해 작성된 모든 `MQTTClient.obj`를 삭제합니다. `MQTTAsync.obj`와 `MQTTClient.obj`는 상호 배타적입니다.

d) 동기, 보안 설정된 라이브러리 `mqttv3cs.dll`를 빌드하십시오.

```
${MQTTDLL_S}: ${SOURCE_FILES_S}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:$
{MQTTDLL_S}
    ${MANIFEST_S}
```

`-rm ${CURDIR}/MQTTAsync.obj` 명령문은 이전 대상에 대해 작성된 모든 `MQTTAsync.obj`를 삭제합니다. `MQTTAsync.obj`와 `MQTTClient.obj`는 상호 배타적입니다.

e) 비동기, 보안 설정된 라이브러리 `mqttv3as.dll`를 빌드하십시오.

```
${MQTTDLL_AS}: ${SOURCE_FILES_AS}
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES_AS}
```

```
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:$
(MQTTDLL_AS}
    $(MANIFEST_AS}
```

`-rm $(CURDIR}/MQTTClient.obj` 명령문은 이전 대상에 대해 작성된 모든 `MQTTClient.obj`를 삭제합니다. `MQTTAsync.obj`와 `MQTTClient.obj`는 상호 배타적입니다.

f) **clean** 대상을 정의하십시오.

"clean" 대상은 Make 파일에 의해 생성되는 모든 파일 및 디렉토리를 제거합니다.

```
.PHONY: 깨끗한
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

8. Windows 경로를 설정하여 Make 파일을 실행하십시오.

설치와 일치하도록 이탤릭체로 된 부분을 설정하십시오.

a) Microsoft Visual Studio 환경을 설정하십시오.

```
%comspec% /k ""C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
```

b) make 프로그램 및 Linux 명령 환경을 포함하도록 `Path` 변수를 설정하십시오.

```
set path=%path%;C:\Program Files\GnuWin32\bin;C:\cygwin\bin
```

9. Make 파일을 실행하십시오.

```
make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/SDK/clients/c/mqttv3c/src
```

팁: 파일 구분 기호 문자는 백슬래시가 아니라 슬래시여야 합니다.

결과

다음과 같은 파일이 *sdkroot**SDK**clients**c**mqttv3c**src*\windows_ia32 디렉토리에서 작성됩니다.

```
mqttv3a.dll
mqttv3a.dll.manifest
mqttv3a.exp
mqttv3a.lib
mqttv3a.map
mqttv3as.dll
mqttv3as.dll.manifest
mqttv3as.exp
mqttv3as.lib
mqttv3as.map
mqttv3c.dll
mqttv3c.dll.manifest
mqttv3c.exp
mqttv3c.lib
mqttv3c.map
mqttv3cs.dll
mqttv3cs.dll.manifest
mqttv3cs.exp
mqttv3cs.lib
mqttv3cs.map
```

MQTTwin.mak Make 파일 나열

```
# 빌드 출력은 현재 디렉토리에서 생성됩니다.
# MQTTCLIENT_DIR는 MQTT 클라이언트 소스 코드를 포함하고 있는 기본 디렉토리를 가리켜야 한다.
# 기본 MQTTCLIENT_DIR는 현재 디렉토리이다.
# 기본 OPENSSL_DIR은 sdkroot\openSSL이며 sdkroot\sdk\clients\c\mqttv3c\src에 상대적입니다.
# OPENSSL_DIR은 OpenSSL 빌드를 포함하는 기본 디렉토리를 가리켜야 합니다.
# 예제: -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# 빌드 환경을 설정하십시오. 예를 들면 다음과 같습니다.
# %comspec% /k " " C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" " x86
# set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
```

```
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../openssl-1.0.1c
endif

ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}

MQTTLIB = mqttv3c
MQTTDLL = windows_ia32/${MQTTLIB} .dll
SOURCE_FILES = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource: ${MQTTDLL}\; 2
MQTTLIB_S = mqttv3cs
MQTTDLL_S = windows_ia32/${MQTTLIB_S} .dll
SOURCE_FILES_S = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
MQTTLIB_A = mqttv3a
MQTTDLL_A = windows_ia32/${MQTTLIB_A} .dll
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
MQTTLIB_AS = mqttv3as
MQTTDLL_AS = windows_ia32/${MQTTLIB_AS} .dll
SOURCE_FILES_AS = ${필터-아웃 ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2

# compiler
CC = cl
CPPFLAGS = /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
INC =  /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/

# linker
LD = link
LINKFLAGS = /nologo /machine:x86 /manifest /dll
WINLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
          advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib\
          odbc32.lib odbccp32.lib ws2_32.lib
IMP = /함축: ${@:.dll=.lib}
LIBPDB = /pdb: ${@:.dll=.pdb}
LQAP = /map: ${@:.dll=.map}
WINLIBS_S = ${WINLIBS} crypt32.lib ssleay32.lib libeay32.lib
LIBPATH_S = /LIBPATH:${OPENSSL_DIR}/lib

# targets
all: ${MQTTDLL} ${MQTTDLL_A} ${MQTTDLL_AS} ${MQTTDLL_S}

${MQTTDLL}: ${SOURCE_FILES}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out:${MQTTDLL}
    ${MANIFEST}

${MQTTDLL_A}: ${SOURCE_FILES_A}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES_A}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out:${MQTTDLL_A}
    ${MANIFEST_A}

${MQTTDLL_S}: ${SOURCE_FILES_S}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:$
{MQTTDLL_S}
    ${MANIFEST_S}

${MQTTDLL_AS}: ${SOURCE_FILES_AS}
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES_AS}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:$
(MQTTDLL_AS}
    $(MANIFEST_AS}
```

```
.PHONY: 깨끗한
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

OpenSSL 패키지 빌드

C, `mqttv3cs` 및 `mqttv3as`에 대한 보안 MQTT 클라이언트 라이브러리를 빌드하기 전에 OpenSSL 패키지를 빌드하십시오. 이 빌드에서는 C용 MQTT 클라이언트 라이브러리의 보안 버전과 OpenSSL 인증서 관리 도구를 빌드하는 데 필요한 라이브러리를 작성합니다.

시작하기 전에

1. iOS iOS Make 파일 사용자 정의는 iOS6를 실행하는 대상 디바이스를 위한 것입니다. iOS의 이전 버전 또는 최신 버전의 경우 이 사용자 정의가 다를 수 있습니다.
2. Windows Windows Make 파일 사용자 정의는 32비트 Windows를 위한 것입니다.

이 태스크 정보

OpenSSL 패키지 및 필수 소프트웨어를 다운로드하여 설치하십시오. OpenSSL Make 파일을 사용자 정의하고 대상 플랫폼에 대한 OpenSSL 라이브러리를 빌드하십시오. Windows 및 Linux의 경우 Make는 OpenSSL 키 작성 및 관리 도구도 빌드합니다.

프로시저

1. OpenSSL 패키지를 설치하십시오.
   a) OpenSSL에서 OpenSSL 패키지를 다운로드하십시오.

   중요사항: OpenSSL 패키지의 다운로드 및 재분배는 엄격한 가져오기 및 내보내기 규제 및 오픈 소스 라이센싱 조건에 적용됩니다. 패키지를 다운로드할지 여부를 결정하기 전에 제한 및 경고에 주의하십시오.

   b) 압축된 파일 컨텐츠를 *sdkroot*에 펼치십시오.

   OpenSSL 사이트의 **뉴스** 탭에서 최신 패키지의 다운로드 위치를 찾으십시오. 이 패키지는 tar 파일로 압축되며 확장자는 `tar.gz`입니다. 패키지가 펼쳐지면 최상위 레벨 폴더 `openssl`*version*이 작성됩니다(예: *openssl-1.0.1c*). The examples refer to the path to the folder as %openssl% on Windows and $openssl on iOS; for example, on Windows, %openssl% is *sdkroot*\openssl-1.0.1c.

   팁: OpenSSL 패키지를 추출하여 작성되는 디렉토리 경로를 확인하십시오. 일부 패키지에는 `openssl`*version* 폴더의 중복 레벨이 있습니다.

2. Windows
   옵션: Windows의 경우 perl을 다운로드하여 설치하십시오. perl.org를 참조하십시오.

   예제에서는 ActivePerl 다운로드에서 perl을 다운로드했습니다.

3. iOS
   옵션: iOS의 경우 세 가지 추가 디렉토리를 작성하십시오.

   ```
   $ssarm7 ≡ $openssl/arm7
   $sslarm7s ≡ $openssl/arm7s
   $ssli386 ≡ $openssl/i386
   ```

   iOS의 경우에는 세 가지 다른 하드웨어 플랫폼에 대한 OpenSSL 패키지를 빌드해야 합니다.
4. OpenSSL Make 파일을 생성하여 사용자의 하드웨어 및 운영 체제에 대한 OpenSSL 패키지를 빌드하십시오.
   a) %openssl% 또는 $openssl 디렉토리에서 명령 창을 여십시오.
   b) 적절한 매개변수로 **Configure** perl 명령을 실행하십시오.
   - Windows

```
perl Configure VC-WIN32 enable-capieng no-asm no-idea no-mdc2 no-rc5 --prefix=%openssl%
ms\do_ms.bat
```

- iOS

```
./Configure BSD-generic32 no-idea no-mdc2 no-rc5 --prefix=$openssl
```

5. iOS

iOS의 경우 생성되는 OpenSSL Make 파일을 다양한 Apple 디바이스에 맞게 사용자 정의하십시오.

a) 생성되는 Make 파일 $openssl/Makefile의 세 가지 사본을 작성하십시오.

$openssl/Makefile_armv7
$openssl/Makefile_armv7s
$openssl/Makefile_i386

b) 각각의 Make 파일에서 "CC=gcc" 명령문을 변경하십시오.

CC=gcc 명령문은 62행 또는 그 주위에 있습니다. 이 명령문을 다음과 같은 명령으로 변경하십시오.

$openssl/Makefile_armv7

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch armv7
```

$openssl/Makefile_armv7s

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch armv7s
```

$openssl/Makefile_i386

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch i386
```

c) 각각의 Make 파일에서 "CFLAG=..." 명령문을 변경하십시오.

이 명령문은 63행 또는 그 주위에 있습니다(가독성을 위해 3개의 행으로 구분됨).

```
CFLAG= -DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

표시된 SDK의 위치는 Xcode 설치 선택사항에 따라 다릅니다. SDK의 버전은 Make 파일을 빌드하는 운영 체제 레벨에 따라 다릅니다.

iPhone 시뮬레이터

iPhone 시뮬레이터 Make 파일은 $openssl/Makefile_i386입니다.

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/
iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
-DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

iOS

iOS Make 파일은 $openssl/Makefile_arm7 및 $openssl/Makefile_arm7s입니다.

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/
iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
-DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

6. 생성되는 Make 파일을 실행하십시오.

- Windows

```
nmake -clean
nmake -f ms\nt.mak
nmake -f ms\nt.mak install
```

- iOS:

```
make clean
make -f $openssl/Makefile_arm7
mv $openssl/libcrypto.a $ssarm7/libcrypto.a
mv $openssl/libssl.a $ssarm7/libssl.a
make clean
make -f $openssl/Makefile_arm7s
mv $openssl/libcrypto.a $sslarm7s/libcrypto.a
mv $openssl/libssl.a $sslarm7s/libssl.a
make clean
make -f $openssl/Makefile_i386
mv $openssl/libcrypto.a $ssli386/libcrypto.a
mv $openssl/libssl.a $ssli386/libssl.a
```

결과

이 빌드에서는 C용 MQTT 클라이언트 라이브러리의 보안 버전을 빌드하는 데 필요한 헤더 파일과 공유 라이브러리(lib)를 생성합니다.

iOS에서 C용 MQTT 클라이언트 시작하기

iOS 애플리케이션을 가져와 MQTT 서버와 메시지를 교환하는 방법에 대해 학습합니다. iOS 디바이스(즉, iPhone과 iPad)에서 사용하려면 MQTT SDK(Software Development Kit)의 일부로 제공되는 소스 코드에서 C용 MQTT 클라이언트 라이브러리를 빌드해야 합니다.

시작하기 전에

1. iOS Dev 센터 에 링크하고 iOS에 대한 애플리케이션을 개발하는 방법을 알고 있습니다.
2. OS X 10.8.2 이상이 설치된 Apple Mac을 확보하여 Xcode 통합 개발 환경(IDE)을 실행하십시오.
3. (선택사항) Windows 또는 Linux에서 C 개발 환경을 구성하십시오. MQTT iOS 앱을 개발하기 전에 Windows 또는 Linux에서 MQTT 클라이언트 샘플 C 앱을 빌드하여 실행하면 유용합니다. 또는 샘플을 빌드하지 않고 샘플 소스 코드를 연구하십시오.
4. C에 대해 지원되는 MQTT 클라이언트 플랫폼과 참조 MQTT 클라이언트 플랫폼은 IBM 모바일 메시징 및 M2M 클라이언트 팩 에 대한 시스템 요구사항의 내용을 참조하십시오.
5. 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.

이 태스크 정보

프로시저에서는 다음과 같은 단계에 대해 안내합니다.

1. C용 MQTT 클라이언트 라이브러리와 MQTT 클라이언트 샘플 앱을 연구하고 빌드하며 실행하여 MQTT의 프로그래밍에 대해 알아보십시오.
2. Apple Mac에 iOS용 Xcode 개발 환경을 설치하십시오.
3. Do the task 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』 to build the MQTT client for C libraries for iOS devices.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 MQTT version 3.1 프로토콜을 지원해야 합니다. IBM 의 모든 MQTT 서버는 IBM WebSphere MQ 및 IBM MessageSight를 포함합니다. 130 페이지의 『MQTT 서버 시작하기』의 내용을 참조하십시오.
2. 모바일 메시징 및 M2M 클라이언트 팩 을 다운로드하고 MQTT SDK를 설치하십시오.

   설치 프로그램은 존재하지 않으며 다운로드한 파일을 펼치기만 하면 됩니다.

a. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

b. SDK를 설치할 폴더를 작성하십시오.

해당 폴더의 이름을 MQTT로 지정할 수도 있습니다. 이 폴더의 경로는 여기에서 *sdkroot*로 참조됩니다.

c. 압축된 모바일 메시징 및 M2M 클라이언트 팩 파일 컨텐츠를 *sdkroot*로 펼치십시오. 이 펼치기를 통해 *sdkroot*\SDK에서 시작되는 디렉토리 트리가 작성됩니다.

3. 옵션: MQTT 클라이언트 샘플 C 앱를 학습하여 MQTT API에 익숙해져야 합니다.

a) Windows 또는 Linux에 대해 동기 MQTT 클라이언트 샘플 C 애플리케이션 MQTTV3sample.c 을 빌드하십시오. 24 페이지의 『C용 MQTT 클라이언트 시작하기』를 참조하십시오.

b) MQTT version 3 서버에 연결한 후 서버에서 토픽을 발행하고 구독하십시오.

c) 소스 코드 및 MQTT API 문서를 연구하십시오. MQTT 클라이언트 라이브러리에 대한 클라이언트 API 문서에 대한 링크는 MQTT 클라이언트 프로그래밍 참조를 참조하십시오.

동기 샘플을 연구하여 MQTT 토픽을 발행 및 구독하고 MQTT 클라이언트를 작성하고 계속하는 방법에 대해 알아보십시오. 동기 샘플은 비동기 샘플보다 단순합니다. 이전에 MQTT를 프로그래밍한 적이 없는 경우에는 동기 MQTT 프로그램을 작성하여 MQTT 프로그래밍 모델 및 API에 익숙해지십시오.

d) Windows 또는 Linux에서 C의 비동기 MQTT 클라이언트 라이브러리를 빌드하십시오. 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』의 내용을 참조하십시오.

e) 비동기 MQTT 클라이언트 샘플 발행과 구독 C 앱을 빌드하고 실행하십시오.

f) MQTT 클라이언트 샘플 비동기 C 앱 소스 코드와 MQTT 참조 문서를 연구하십시오.

비동기 인터페이스를 사용하여 모바일 디바이스용 MQTT 애플리케이션을 작성해야 합니다. 비동기 인터페이스를 호출하는 잘 작성된 앱은 응답성이 높으며 동기 인터페이스에 작성된 앱보다 배터리 수명이 깁니다.

비동기 인터페이스는 두 가지 수준의 비동기성을 가지고 있습니다.

i) 첫 번째 수준은 MQTT 클라이언트 라이브러리가 서버의 발행물을 기다리는 동안 애플리케이션의 차단을 해제하는 것입니다.

ii) 두 번째 수준은 클라이언트 라이브러리가 서버에 연결하고 구독을 작성하고 발행물을 게시하는 동안 애플리케이션의 차단을 해제하는 것입니다.

4. iOS 개발 도구를 다운로드하여 설치하십시오.

a. 관리 권한이 있는 사용자 ID로 로그온하십시오.

b. AppleMac 버전이 10.8.2 또는 이상인지 확인하십시오.

c. 웹 사이트 Xcode 로 이동하여 Mac 앱 스토어에서 Xcode 를 다운로드하십시오.

d. Xcode, 명령행 환경, 시뮬레이터를 설치하십시오.

Mac 앱 스토어에서 여러 버전의 시뮬레이터를 제공하는 경우에는 사용자 앱의 대상으로 지정하는 iOS 레벨과 호환되는 버전을 선택하십시오.

5. iOS의 C용 MQTT 클라이언트 라이브러리를 빌드하십시오. 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』의 내용을 참조하십시오.

다음에 수행할 작업

1. 빌드한 C용 MQTT 클라이언트 라이브러리를 확인하십시오.

a. Xcode 개발 환경을 사용하여 비동기 MQTT 클라이언트 샘플 C 앱을 컴파일하고 C용 비보안 비동기 MQTT 클라이언트 라이브러리에 링크하십시오.

b. Xcode 개발 환경을 사용하여 iOS 디바이스에서 비동기 MQTT 클라이언트 샘플 C 앱을 실행하십시오. 구성한 MQTT version 3 서버에 샘플을 연결하십시오. 134 페이지의 『명령행에서 MQTT 서비스 구성』의 내용을 참조하십시오.

2. iOS용 MQTT 클라이언트 C 앱을 작성하십시오. 비동기 C 애플리케이션에 대한 코딩 예제가 도움이 될 수 있습니다. The examples are MQTTV3ASample.c and MQTTV3ASSample.c in *sdkroot*\SDK\clients\c\samples. 실습으로 MQTT 발행/구독 샘플을 구현하는 것부터 시작하십시오.

팁: 앱의 모양과 앱이 수행하는 작업에 대해 알아보려면 MQTT 클라이언트 샘플 C 앱의 화면 캡처를 확인하십시오. 17 페이지의 『Android 에서 Java용 MQTT 클라이언트 시작하기』의 내용을 참조하십시오.

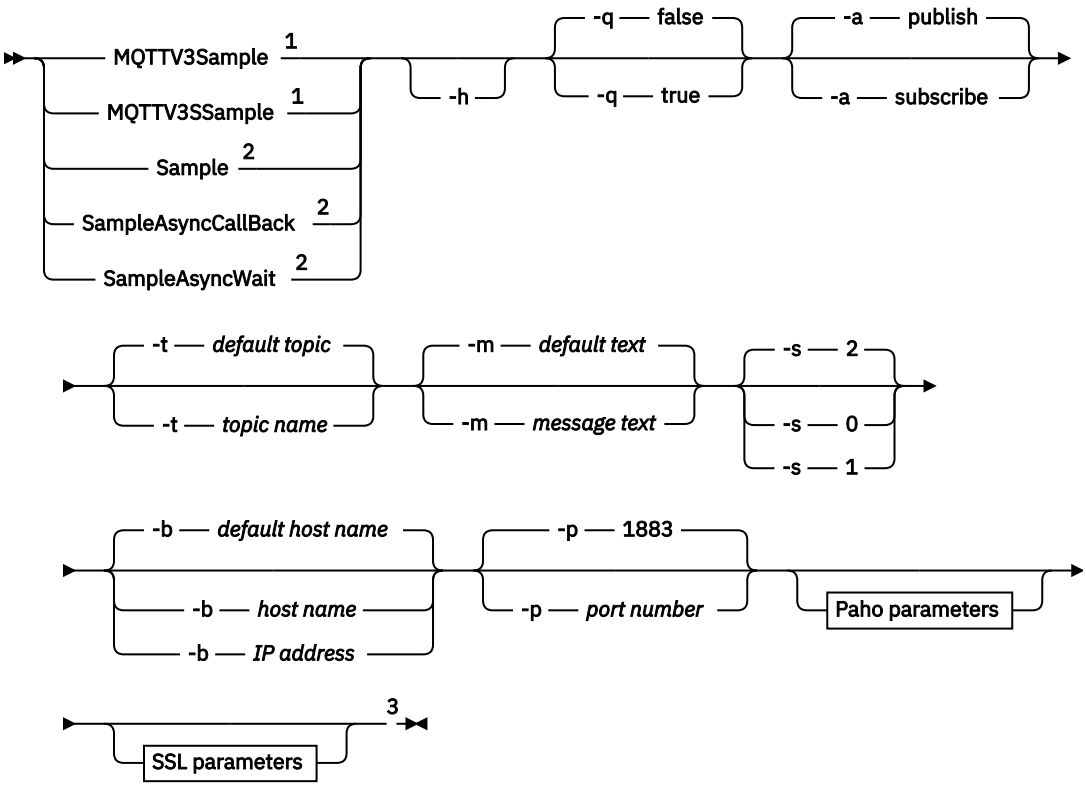
MQTT 명령행 샘플 프로그램

MQTT 명령행 샘플 프로그램의 구문 및 매개변수입니다.

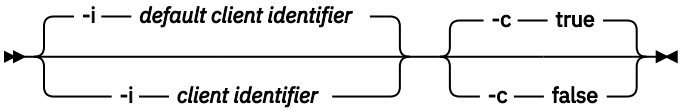
목적

토픽을 발행하고 구독하십시오.

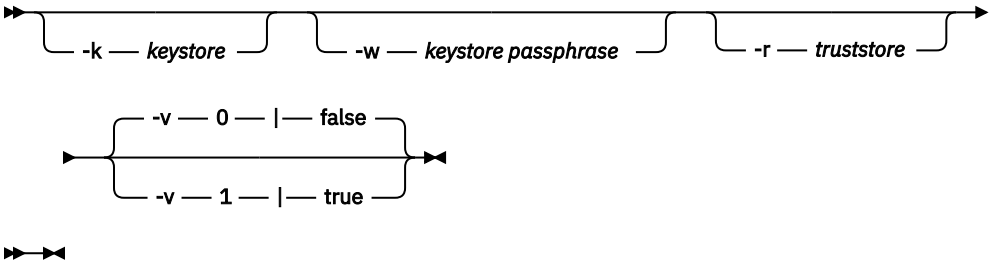
Syntax



Paho parameters



SSL parameters



참고:

[1] IBM WebSphere MQ sample
[2] Paho sample
[3] *Not* `MQTTV3Sample`.

매개변수

-h
이 도움말 텍스트를 인쇄하고 종료합니다.

-q
기본 모드인 false를 사용하는 대신 자동 모드를 설정합니다.

-a publish|subscribe
기본 조치인 publishing 대신 조치를 publish 또는 subscribe로 설정합니다.

-t *topic name*
기본 토픽을 발행하거나 구독하는 대신 *topic name*을 발행하거나 구독합니다. 기본 토픽은 다음과 같습니다.

Paho 샘플

발행
Sample/Java/v3

구독
Sample/#

IBM WebSphere MQ 샘플

발행
MQTTV3Sample/Java/v3 또는 MQTTV3Sample/C/v3

구독
MQTTV3Sample/#

-m *message text*
기본 텍스트를 송신하는 대신 *message text*를 발행합니다. 기본 텍스트는 "Message from MQTTv3 C client" 또는 "Message from MQTTv3 Java client"입니다.

-s 0|1|2
기본 서비스 품질(QoS)인 2를 사용하는 대신 QoS를 설정합니다.

-b *host name*
기본 호스트 이름에 연결하는 대신 *host name* 또는 IP 주소에 연결합니다. Paho 샘플의 기본 호스트 이름은 m2m.eclipse.org입니다. IBM WebSphere MQ 샘플의 경우에는 localhost입니다.

-p *port number*
기본 포트인 1883을 사용하는 대신 *port number* 포트를 사용합니다.

Paho 매개변수

-i *client identifier*
Set the client identifier to *client identifier*. The default client identifier is SampleJavaV3_"+action, where action is publish or subscribe.

-c true|false
정리 세션 플래그를 설정합니다. 기본값은 true입니다(구독이 지속되지 않음).

SSL 매개변수

-k *keystore*
클라이언트를 식별하는 개인 키가 포함된 키 저장소의 경로를 *keystore*로 설정합니다.C 샘플의 경우 저장소는 PEM(Privacy-Enhanced Mail) 파일입니다. Java 샘플의 경우에는 Java 키 저장소(JKS)입니다.

-w *keystore passphrase*
클라이언트에 키 저장소에 대한 액세스 권한을 부여하는 데 필요한 비밀번호 문구를 *keystore passphrase*로 설정합니다.

-r *truststore*
클라이언트가 신뢰하는 MQTT 서버의 공용 키를 포함하는 키 저장소에 대한 경로를 *truststore*로 설정하십시오. 키 저장소는 PEM (Privacy-Enhanced Mail) 파일입니다. C 샘플의 경우 저장소는 PEM(Privacy-Enhanced Mail) 파일입니다. Java 샘플의 경우에는 Java 키 저장소(JKS)입니다.

-v 0|false|1|true
확인 옵션을 1|true로 설정하여 서버 인증서를 요구합니다. 기본값은 0|false입니다. 서버 인증서가 확인되지 않습니다. SSL 채널은 항상 암호화됩니다.

C 프로그램의 경우 0|1 , Java 프로그램의 경우 true|false 옵션을 설정하십시오.

관련 태스크

11 페이지의 『Java용 MQTT 클라이언트 시작하기』
MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ를 사용하여 Java 샘플 애플리케이션의 MQTT 클라이언트를 시작하여 실행합니다. The sample applications use a client library from the MQTT software development toolkit (SDK) from IBM. SampleAsyncCallBack 샘플 애플리케이션은 Android 및 기타 이벤트 구동 운영 체제에 대해 MQTT 애플리케이션을 작성하기 위한 모델입니다.

24 페이지의 『C용 MQTT 클라이언트 시작하기』
C 소스를 컴파일할 수 있는 모든 플랫폼에서 샘플 C용 MQTT 클라이언트를 시작하고 실행할 수 있습니다. MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ를 사용하여 C에 대해 샘플 MQTT 클라이언트를 실행할 수 있는지 확인하십시오.

29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』
다음 단계를 수행하여 C용 MQTT 클라이언트 라이브러리를 빌드하십시오. 이 토픽에는 다수의 플랫폼에 대한 컴파일 및 링크 스위치와 iOS 및 Windows에서 라이브러리를 빌드하는 예제가 포함되어 있습니다.

MQTT 보안

MQTT 보안에는 ID, 인증, 권한 부여라는 세 가지 기본 개념이 있습니다. ID는 권한 부여 중이고 권한 부여된 클라이언트의 이름을 지정하는 것과 관련되어 있습니다. 인증은 클라이언트의 ID를 증명하는 것과 관련되어 있고 권한 부여는 클라이언트에 제공되는 권한을 관리하는 것과 관련되어 있습니다.

보안 샘플 시도

- 52 페이지의 『보안 MQTT 클라이언트 샘플 Java 앱 빌드와 실행』
- 60 페이지의 『SSL을 통해 Android 에 MQTT 클라이언트 샘플 Java 어플리케이션 연결』
- 69 페이지의 『JAAS 을 사용하여 MQTT 클라이언트 Java 애플리케이션 인증』
- V7.5.0.1 74 페이지의 『SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets 연결』
- 82 페이지의 『보안 MQTT 클라이언트 샘플 C 앱 빌드와 실행』

ID

클라이언트 ID, 사용자 ID 또는 공용 디지털 인증서로 MQTT 클라이언트를 식별하십시오. 이 속성 중 하나 또는 다른 속성이 클라이언트 ID를 정의합니다. MQTT 서버는 SSL 프로토콜을 사용하여 클라이언트가 송신한 인증서를 인증하거나 클라이언트가 설정한 비밀번호를 사용하여 클라이언트 ID를 인증합니다. 서버는 클라이언트 ID를 기반으로 클라이언트가 액세스할 수 있는 자원을 제어합니다.

MQTT 서버는 IP 주소 또는 디지털 인증서로 클라이언트에 대해 자신을 식별합니다. MQTT 클라이언트는 SSL 프로토콜을 사용하여 서버가 송신한 인증서를 인증합니다. 일부 경우 이 클라이언트는 서버의 DNS 이름을 사용하여 이 클라이언트에 인증서를 송신한 서버가 인증서 보유자로 등록되어 있는지 확인합니다.

다음 방법 중 하나로 클라이언트의 ID를 설정하십시오.

클라이언트 ID
MqttClient 클래스(C에서는 MQTTClient_create 또는 MQTTAsync_create)는 클라이언트 ID를 설정합니다. 클래스 구성자를 호출하여 클라이언트 ID를 매개변수로 설정하거나 임의로 생성된 클라이언트 ID를 리턴하십시오.클라이언트 ID는 서버에 연결되는 모든 클라이언트에서 고유해야 하며 서버의 큐 관리자

이름과 동일해서는 안됩니다. 모든 클라이언트에는 ID 검사에 사용되지 않더라도 클라이언트 ID가 있어야 한다. 122 페이지의 『클라이언트 ID』를 참조하십시오.

사용자 ID
MqttClient 클래스(C에서는 MQTTClient_create 또는 MQTTAsync_create)는 클라이언트 사용자 ID를 MqttConnectOptions(C에서는 MqttClient_ConnectOptions)의 속성으로 설정합니다. 사용자 ID는 클라이언트에 대해 고유하지 않아도 됩니다.

클라이언트 디지털 인증서
클라이언트 디지털 인증서는 클라이언트 키 저장소에 저장됩니다. 키 저장소 위치는 클라이언트에 따라 다릅니다.

- **Java**

  MqttConnectOptions의 setSSLProperties 메소드를 호출한 후 키 저장소 특성을 전달하여 클라이언트 키 저장소의 위치와 특성을 설정하십시오. Example.java의 SSL 수정을 참조하십시오. **keytool** 도구는 Java키 및 키 저장소를 관리합니다.

- **C**

  MQTTClient_create 또는 MQTTAsync_create 는 키 저장소 특성을 MQTTClient_SSLOptions ssl_opts의 속성으로 설정합니다. **openSSL** 도구는 C의 MQTT 클라이언트에서 액세스하는 키 및 키 스토어를 작성하고 관리합니다.

- **Android**

  설정 > **보안** 메뉴에서 Android 디바이스 키 저장소를 관리하십시오. SD 카드에서 새 인증서를 로드하십시오.

개인 키를 서버 키 저장소에 저장하여 서버의 ID를 설정하십시오.

IBM WebSphere MQ
MQTT 서버 키 저장소는 클라이언트가 연결되는 텔레메트리 채널의 속성입니다.

키 저장소 위치 및 속성을 IBM WebSphere MQ Explorer로 설정하거나 **DEFINE CHANNEL** 명령을 사용하여 설정하십시오. DEFINE CHANNEL (MQTT)을 참조하십시오. 여러 채널이 하나의 키 저장소를 공유할 수 있습니다.

인증

MQTT 클라이언트는 연결되는 MQTT 서버를 인증할 수 있고 해당 서버는 연결되는 클라이언트를 인증할 수 있습니다.

클라이언트는 SSL 프로토콜을 사용하여 서버를 인증합니다. MQTT 서버는 SSL 프로토콜, 비밀번호 또는 둘 다를 사용하여 클라이언트를 인증합니다.

클라이언트가 서버를 인증하지만 서버는 클라이언트를 인증하지 않는 경우 해당 클라이언트는 익명 클라이언트로 알려질 수 있습니다. SSL을 통해 익명 클라이언트 연결을 설정한 후 SSL 세션에 의해 암호화된 비밀번호를 사용하여 클라이언트를 인증하는 것이 일반적입니다. 인증서 배포 및 관리 문제로 인해 클라이언트 인증서보다는 비밀번호를 사용하여 클라이언트를 인증하는 경우가 일반적입니다. 고가치 디바이스(예: ATM 및 칩 앤 핀 시스템)와 사용자 정의 디바이스(예: 스마트 전력량계)에서 사용되는 클라이언트 인증서를 찾으려는 경우가 많습니다.

클라이언트에 의한 서버 인증
MQTT 클라이언트는 SSL 프로토콜을 사용하여 서버 인증서를 인증하여 올바른 서버에 연결되어 있는지 확인합니다. 이 양식의 확인은 HTTPS 프로토콜을 통해 웹 사이트를 이동할 때 익숙한 양식입니다.

서버는 인증 기관이 서명한 공용 인증서를 클라이언트에 송신합니다. 클라이언트는 인증 기관의 공개 키를 사용하여 서버 인증서에서 인증 기관의 서명을 확인합니다. 클라이언트는 인증서가 최신 인증서인지도 확인합니다. 이 확인을 통해 인증서의 유효성이 보장됩니다.

인증 기관 인증서(루트 인증서라고도 함)는 클라이언트 신뢰 저장소에 저장됩니다.

- **Java**

MqttConnectOptions의 setSSLProperties 메소드를 호출하고 신뢰 저장소 특성을 전달하여 클라이언트 신뢰 저장소의 위치와 특성을 설정하십시오. Example.java의 SSL 수정을 참조하십시오. **keytool** 도구를 사용하여 인증서 및 신뢰 저장소를 관리하십시오.

- **C**

  MQTTClient_create 또는 MQTTAsync_create 는 신뢰 저장소 특성을 MQTTClient_SSLOptions ssl_opts의 속성으로 설정합니다. **openSSL** 도구를 사용하여 인증서 및 신뢰 저장소를 관리하십시오.

- **Android**

  설정 > **보안** 메뉴에서 Android 디바이스 신뢰 저장소를 관리하십시오. SD 카드에서 새 루트 인증서를 로드하십시오.

서버에 의한 클라이언트 인증

MQTT 서버는 SSL 프로토콜을 사용하여 클라이언트 인증서를 인증하거나 비밀번호를 사용하여 클라이언트 ID를 인증하여 올바른 클라이언트에 연결되어 있는지 확인합니다.

이 서버는 클라이언트가 서버의 MQTT protocol 헤더에 송신하는 비밀번호를 사용하여 클라이언트를 인증합니다. 이 서버는 클라이언트 ID 또는 사용자 ID를 인증하거나 비밀번호를 사용하여 인증하도록 선택할 수 있습니다. 서버에 따라 다르게 선택할 수 있습니다. 일반적으로 서버는 사용자 ID를 인증합니다. 보안 설정되지 않은 상태로 비밀번호를 송신하지 않도록 서버를 확인하여 보안 설정된 SSL 연결을 통해 비밀번호를 확인하십시오.

- **IBM WebSphere MQ**

  IBM WebSphere MQ는 SSL 프로토콜을 사용하여 클라이언트 인증서를 인증합니다. 루트 인증서를 IBM WebSphere MQ Telemetry 키 저장소에 저장하십시오. 상호 SSL 인증의 일부로만 클라이언트 인증서를 인증할 수 있습니다. 즉, 클라이언트에 서버 공용 인증서를 제공하고 서버에 클라이언트 공용 인증서를 제공해야 합니다.

  IBM WebSphere MQ Telemetry는 자체 개인용 및 공용 인증서와 기타 공용 인증서(예: 인증 기관이 제공한 루트 인증서)에 대해 동일한 저장소를 사용합니다.

  키 저장소 위치 및 속성을 IBM WebSphere MQ Explorer로 설정하거나 **DEFINE CHANNEL** 명령을 사용하여 설정하십시오. DEFINE CHANNEL (MQTT)을 참조하십시오. 여러 채널이 하나의 키 저장소를 공유할 수 있습니다.

  IBM WebSphere MQ는 JAAS(Java Authentication and Authorization Service)를 호출하여 클라이언트 사용자 ID(또는 클라이언트 ID)를 인증합니다.

  jaas.config 파일에 저장되는 MQXRConfig 구성 스탠자에서 JAAS를 구성하십시오. 이 파일은 qmgrs*QmgrName*\mqxr 디렉토리의 IBM WebSphere MQ 데이터 경로에 저장됩니다.

  JAASLoginModule의 login 메소드를 작성하여 클라이언트의 인증서를 검사하십시오. 110 페이지의 『텔레메트리 채널 JAAS 구성』의 내용을 참조하십시오.

  IBM WebSphere MQ Telemetry 는 JAASLoginModule.login 메소드를 다음 매개변수로 전달합니다.

  - 사용자 ID
  - 암호
  - 클라이언트 ID
  - 네트워크 ID
  - 채널 이름
  - ValidPrompts

권한 부여

권한 부여는 MQTT protocol의 일부가 아닙니다. 권한 부여는 MQTT 서버에 의해 제공됩니다. 권한 부여되는 항목은 서버가 수행하는 작업에 따라 다릅니다. MQTT 서버는 발행/구독 브로커이며 유용한 MQTT 권한 부여 규칙은 서버에 연결될 수 있는 클라이언트와 클라이언트가 발행하거나 구독할 수 있는 토픽을 제어합니다. MQTT 클

라이언트가 서버를 관리할 수 있는 경우에는 더 많은 권한 부여 규칙이 서버의 다양한 측면을 관리할 수 있는 클라이언트를 제어합니다.

가능한 클라이언트 수는 아주 많으므로 각각의 클라이언트에 개별적으로 권한을 부여하는 것은 가능하지 않습니다. MQTT 서버는 그룹 또는 프로파일별로 클라이언트를 그룹화하는 방법을 가지고 있습니다.

액세스 및 권한 부여의 관점에서 클라이언트의 ID는 MQTT 클라이언트에 대해 고유하지 않습니다. 클라이언트의 ID를 클라이언트 ID와 동일시하지 마십시오. 이들은 동일할 수 있지만 일반적으로는 다릅니다. 예를 들어, 다수의 서비스에서 공용인 사용자 이름을 가지고 있으며 이 서비스 중 일부가 "싱글 사인온"으로 협력합니다. 엔터프라이즈 스케일 MQTT 서버는 서로 다른 애플리케이션에 대해 공용 ID 및 권한을 제공하는 권한 서비스를 호출하려고 합니다.

IBM WebSphere MQ

IBM WebSphere MQ에는 플러그 가능한 권한 서비스가 포함되어 있습니다. Windows와 Linux에서 제공되는 기본 권한 서비스는 OAM(Object Authority Manager)입니다. UNIX, Linux 및 Windows 시스템에서 OAM을 사용하여 오브젝트에 대한 액세스 제어를 참조하십시오. 이 서비스는 운영 체제 사용자 ID 및 그룹을 토픽 및 큐와 같은 IBM WebSphere MQ 오브젝트에 대한 조작과 연관시킵니다.

고정 사용자 ID를 사용하여 IBM WebSphere MQ에 액세스하도록 텔레메트리 채널을 구성할 수 있습니다. 이는 샘플 채널을 설정하는 방법입니다. 또는 MQTT 클라이언트가 설정한 사용자 ID로 IBM WebSphere MQ에 액세스할 수 있습니다. WebSphere MQ 오브젝트에 액세스할 수 있도록 MQTT 클라이언트에 권한 부여에서는 IBM WebSphere MQ Telemetry를 설정하여 대단위, 중간 단위, 세분화된 클라이언트 액세스 제어를 달성하는 방법에 대해 설명합니다.

관련 태스크

82 페이지의 『보안 MQTT 클라이언트 샘플 C 앱 빌드와 실행』
Windows 예제를 기반으로 C 소스를 컴파일할 수 있는 모든 운영 체제에서 보안 샘플 C 앱을 시작하고 실행할 수 있습니다. MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 샘플 C 앱을 실행할 수 있는지 확인하십시오.

52 페이지의 『보안 MQTT 클라이언트 샘플 Java 앱 빌드와 실행』
Windows 예제를 기반으로 하여 MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 보안 샘플 Java 애플리케이션을 시작하여 실행할 수 있습니다. JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능"

74 페이지의 『SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets 연결』
SSL및 WebSocket protocol를 사용하는 JavaScript용 MQTT 메시징 클라이언트 샘플 HTML 페이지를 사용하여 웹 애플리케이션을 IBM WebSphere MQ 에 안전하게 연결하십시오.

60 페이지의 『SSL을 통해 Android 에 MQTT 클라이언트 샘플 Java 어플리케이션 연결』
SSL을 통해 IBM WebSphere MQ 에 연결된 샘플 Android MQTT 클라이언트를 시작하여 실행하십시오.

69 페이지의 『JAAS 을 사용하여 MQTT 클라이언트 Java 애플리케이션 인증』
JAAS를 사용하여 클라이언트를 인증하는 방법에 대해 학습합니다. 이 태스크의 단계를 완료하여 샘플 프로그램 `JAASLoginModule.java` 를 수정하고 IBM WebSphere MQ 를 JAAS로 MQTT 클라이언트 Java 애플리케이션을 인증하도록 구성하십시오.

보안 MQTT 클라이언트 샘플 Java 앱 빌드와 실행

Windows 예제를 기반으로 하여 MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 보안 샘플 Java 애플리케이션을 시작하여 실행할 수 있습니다. JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능"

시작하기 전에

1. SSL에서 MQTT protocol 를 지원하는 MQTT version 3.1 서버에 대한 액세스 권한이 있어야 합니다.
2. 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.
3. JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능". IBM 모바일 메시징 및 M2M 클라이언트 팩 에 대한 시스템 요구사항의 내용을 참조하십시오.
4. SSL 채널이 시작되어야 합니다.

이 태스크 정보

예를 들어, 이 기사에서는 명령행에서 Windows 의 보안 MQTT 클라이언트 샘플 Java 앱 를 컴파일 및 실행하는 방법을 보여줍니다.

인증 기관 서명된 키 또는 자체 서명된 키를 사용하여 SSL 채널을 보안 설정하십시오.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 SSL을 통해 MQTT version 3.1 프로토콜을 지원해야 합니다. IBM 의 모든 MQTT 서버는 IBM WebSphere MQ 및 IBM MessageSight를 포함합니다. 130 페이지의 『MQTT 서버 시작하기』의 내용을 참조하십시오.
2. 옵션: 버전 7이상에 Java 개발 킷 (JDK) 을 설치하십시오.

   버전 7은 인증서를 인증하기 위해 **keytool** 명령을 실행해야 합니다. 인증서를 인증하지 않으려면 버전 7 JDK가 필요하지 않습니다.
3. 모바일 메시징 및 M2M 클라이언트 팩 을 다운로드하고 MQTT SDK를 설치하십시오.

   설치 프로그램은 존재하지 않으며 다운로드한 파일을 펼치기만 하면 됩니다.

   a. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

   b. SDK를 설치할 폴더를 작성하십시오.

      해당 폴더의 이름을 MQTT로 지정할 수도 있습니다. 이 폴더의 경로는 여기에서 *sdkroot*로 참조됩니다.

   c. 압축된 모바일 메시징 및 M2M 클라이언트 팩 파일 컨텐츠를 *sdkroot*로 펼치십시오. 이 펼치기를 통해 *sdkroot*\SDK에서 시작되는 디렉토리 트리가 작성됩니다.
4. 키 쌍 및 인증서를 생성하기 위해 스크립트를 작성하고 실행하고 IBM WebSphere MQ 를 MQTT 서버로 구성하십시오.

   92 페이지의 『키 및 인증서 생성』 의 단계를 수행하여 스크립트를 작성하고 실행하십시오.스크립트는 55 페이지의 『Windows에 대한 SSL 인증서를 구성하는 스크립트 예』에도 나열되어 있습니다.
5. SSL 채널이 실행 중이고 예상대로 설정되어 있는지 확인하십시오.

   IBM WebSphere MQ에서 다음 명령을 명령 창에 입력하십시오.

   - Linux

     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```

   - Windows

     ```
     echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     ```
6. 스크립트를 작성하여 보안 MQTT 클라이언트 샘플 Java 앱을 빌드하고 실행하십시오.

   a) `ssjavaclient.bat`을 작성한 후 실행하여 자체 서명 인증서로 보안 설정된 SSL 채널을 테스트하십시오.

   b) `cajavaclient.bat`을 작성한 후 실행하여 인증 기관이 서명한 인증서로 보안 설정된 SSL 채널을 테스트하십시오.

MQTT 보안 Java 클라이언트를 실행하는 데 필요한 스크립트

이 스크립트를 실행하기 전에 55 페이지의 『Windows에 대한 SSL 인증서를 구성하는 스크립트 예』의 스크립트를 실행하십시오.

자체 서명된 인증서를 사용하여 MQTT 보안 Java 클라이언트를 실행하십시오.

`sscerts.bat` 스크립트를 실행하여 작성한 자체 서명된 인증서를 사용하여 이 스크립트를 실행하십시오.

```
@echo off
setlocal
cd %jsamppath%
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
cd %
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar .\org\eclipse\paho\sample\mqttv3app\Sample.java
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportopt% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportopt% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
pause
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportreq% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
pause
endlocal
```

그림 *14. ssjavaclient.bat*

인증 기관이 서명한 인증서를 사용하여 MQTT 보안 Java 클라이언트를 실행하십시오.

cacerts.bat 스크립트를 실행하여 작성한 인증 기관이 서명한 인증서를 사용하여 이 스크립트를 실행하십시오.

```
@echo off
setlocal
cd %jsamppath%
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
cd %
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar .\org\eclipse\paho\sample\mqttv3app\Sample.java
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportopt% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltcajkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportopt% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltcajkstruststore% -v true
pause
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportreq% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltcajkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltcajkstruststore% -v true
pause
endlocal
```

그림 *15. cajavaclient.bat*

관련 개념

49 페이지의 『MQTT 보안』

MQTT 보안에는 ID, 인증, 권한 부여라는 세 가지 기본 개념이 있습니다. ID는 권한 부여 중이고 권한 부여된 클라이언트의 이름을 지정하는 것과 관련되어 있습니다. 인증은 클라이언트의 ID를 증명하는 것과 관련되어 있고 권한 부여는 클라이언트에 제공되는 권한을 관리하는 것과 관련되어 있습니다.

관련 태스크

92 페이지의 『키 및 인증서 생성』
Java 클라이언트와 C 클라이언트(Android 앱과 iOS 앱, IBM WebSphere MQ 서버와 IBM MessageSight 서버 포함)의 키와 인증서를 생성하려면 다음 프로시저를 수행하십시오.

60 페이지의 『SSL을 통해 Android 에 MQTT 클라이언트 샘플 Java 어플리케이션 연결』
SSL을 통해 IBM WebSphere MQ 에 연결된 샘플 Android MQTT 클라이언트를 시작하여 실행하십시오.

69 페이지의 『JAAS 을 사용하여 MQTT 클라이언트 Java 애플리케이션 인증』
JAAS를 사용하여 클라이언트를 인증하는 방법에 대해 학습합니다. 이 태스크의 단계를 완료하여 샘플 프로그램 `JAASLoginModule.java` 를 수정하고 IBM WebSphere MQ 를 JAAS로 MQTT 클라이언트 Java 애플리케이션을 인증하도록 구성하십시오.

Windows에 대한 SSL 인증서를 구성하는 스크립트 예

** **

예제 명령 파일은 태스크의 단계에 설명된 대로 인증서 및 인증서 저장소를 작성합니다. 또한 예제에서는 서버 인증서 저장소를 사용하도록 MQTT 클라이언트 큐 관리자를 설정합니다. 예제는 IBM WebSphere MQ와 함께 제공되는 `SampleMQM.bat` 스크립트를 호출하여 큐 관리자를 삭제하고 다시 작성합니다.

`initcert.bat`

`initcert.bat`는 **keytool** 및 **openSSL** 명령에 필요한 인증서의 이름 및 경로와 기타 매개변수를 설정합니다. 이 설정은 스크립트의 주석에서 설명합니다.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
```

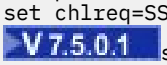
```
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
```

```
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
```


```
set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 스크립트의 명령은 서버 인증서 저장소가 잠기지 않도록 하기 위해 MQTT 클라이언트 큐 관리자를 삭제한 다음 샘플 보안 스크립트가 작성한 모든 키 저장소 및 인증서를 삭제합니다.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

genkeys.bat 스크립트의 명령은 개인 인증 기관, 서버 및 클라이언트에 대한 키 쌍을 작성합니다.

```
@rem
@echo ____________________________________________________________________________
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

sscerts.bat 스크립트의 명령은 해당 키 저장소에서 클라이언트 및 서버 자체 서명 인증서를 내보낸 후 서버 인증서를 클라이언트 신뢰 저장소로 가져오고 클라이언트 인증서를 서버 키 저장소로 가져옵니다. 서버에는 신뢰 저장소가 없습니다. 명령은 클라이언트 JKS 신뢰 저장소에서 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

이 스크립트는 인증 기관 루트 인증서를 개인용 키 저장소로 가져옵니다. CA 루트 인증서는 루트 인증서와 서명된 인증서 간의 키 체인을 작성하기 위해 필요합니다. cacerts.bat 스크립트는 해당 키 저장소에서 클라이언트 및 서버 인증서 요청을 내보냅니다. 스크립트는 cajkskeystore.jks 키 저장소에서 개인용 인증 기관의 키를 사용하여 인증서 요청에 서명한 다음 서명한 인증서를 요청이 나온 곳과 같은 키 저장소로 다시 가져옵니다. 이 가져오기를 통해 CA 루트 인증서가 포함된 인증서 체인이 작성됩니다. 이 스크립트는 클라이언트 JKS 신뢰 저장소로부터 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
```

```
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
```

```
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

이 스크립트는 인증 디렉토리에 있는 키 저장소 및 인증서를 나열합니다. 그런 다음 MQTT 샘플 큐 관리자를 작성하고 보안 텔레메트리 채널을 구성합니다.

```
@echo ____________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

SSL을 통해 Android 에 MQTT 클라이언트 샘플 Java 어플리케이션 연결

SSL을 통해 IBM WebSphere MQ 에 연결된 샘플 Android MQTT 클라이언트를 시작하여 실행하십시오.

시작하기 전에

이 글에서는 최소한 Android API 레벨 14(ICS 4.0)를 실행 중이라고 가정합니다. 이전 레벨에는 키 저장소가 있지만 시스템 앱만 키 저장소에 액세스할 수 있습니다.

1. SSL에서 MQTT protocol 를 지원하는 MQTT version 3.1 서버에 대한 액세스 권한이 있어야 합니다.
2. 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.
3. 이전 Android 디바이스에서 연결을 테스트할 경우 디바이스에 인증서를 전송할 SD 카드가 필요합니다.
4. 가상 Android 디바이스에서 연결을 테스트할 경우 가상 디바이스에서 사용할 수 있게 SD 카드를 구성하십시오.
5. SSL 채널이 시작되어야 합니다.

이 태스크 정보

이 태스크를 완료하면 SSL을 통해 Android에 대한 MQTT 클라이언트 샘플 Java 앱이 실행됩니다. SSL이 정상적으로 연결되면 Android 디바이스와 MQTT 서버 사이에 보안 암호화된 채널이 설정됩니다. 서버의 ID는 인증됩니다.

Android에서는 SSL을 사용하여 서버를 인증할 수 있습니다. 샘플 앱에서는 지원하지 않지만 디바이스를 인증할 수도 있습니다. 디바이스를 인증하려면 KeyChain API를 사용하거나, JAAS를 사용하여 클라이언트 ID, 클라이언트 IP 주소를 인증하거나, MQTT Android 앱에서 제공하는 사용자 이름과 비밀번호를 사용하십시오.

Android 신뢰 저장소에 설치한 X.509 인증서는 인증 기관에서 서명한 것이어야 합니다. 이 예제에서는 Android 디바이스에 설치한 인증서에 서명하는 인증 기관을 작성합니다. 여러 루트 인증서가 Android 디바이스에 사전설치됩니다.

신뢰할 수 있는 인증서를 설치하기 전에 Android 디바이스에 잠금을 작성해야 합니다. 잠금은 내가 모르게 다른 사람이 디바이스에 인증서를 설치할 수 없도록 합니다.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 SSL을 통해 MQTT version 3.1 프로토콜을 지원해야 합니다. IBM 의 모든 MQTT 서버는 IBM WebSphere MQ 및 IBM MessageSight를 포함합니다. 130 페이지의 『MQTT 서버 시작하기』의 내용을 참조하십시오.
2. 보안이 설정되지 않은 MQTT 채널에서 for Android MQTT 클라이언트 샘플 앱 "MQTTExerciser"를 실행하십시오. 17 페이지의 『Android 에서 Java용 MQTT 클라이언트 시작하기』의 내용을 참조하십시오.

   다시 앱을 사용하여 보안 채널을 테스트하십시오.

   Android 가상 디바이스를 시작한 경우 이를 실행 중인 상태로 두십시오.
3. 옵션: 버전 7이상에 Java 개발 킷 (JDK) 을 설치하십시오.

   버전 7은 인증서를 인증하기 위해 **keytool** 명령을 실행해야 합니다. 인증서를 인증하지 않으려면 버전 7 JDK가 필요하지 않습니다.
4. 키 쌍 및 인증서를 생성하기 위해 스크립트를 작성하고 실행하고 IBM WebSphere MQ 를 MQTT 서버로 구성하십시오.

   92 페이지의 『키 및 인증서 생성』 의 단계를 수행하여 스크립트를 작성하고 실행하십시오.스크립트는 64 페이지의 『 Windows에 대한 SSL 인증서를 구성하는 스크립트 예』에도 나열되어 있습니다.

   인증 기관 공용 인증서와 서버 키 저장소가 필요합니다. 클라이언트 인증서나 `.pem` 또는 `.p12` 양식의 인증서는 필요하지 않습니다.
5. SSL 채널이 실행 중이고 예상대로 설정되어 있는지 확인하십시오.

   IBM WebSphere MQ에서 다음 명령을 명령 창에 입력하십시오.

   - Linux

     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```

   - Windows

     ```
     echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     ```
6. Android 신뢰 저장소에 인증 기관 인증서를 설치하십시오.

   예제의 인증 기관 파일은`cacert.cer`입니다.

   a) 인증서 이름을 `cacert.crt`로 바꾸십시오.

   b) 루트 내부 스토리지 또는 SD 카드에 인증서를 복사하십시오.

   실행 중인 가상 디바이스의 경우 Eclipse를 열거나 ADB(Android Debug Bridge)를 실행하여 가상 디바이스에 인증서를 복사하십시오.

   Eclipse

   i) Eclipse를 실행하고 DDMS 퍼스펙티브를 여십시오.

   ii) 기본 보기에서 **파일 탐색기** 창을 여십시오.

iii) `mnt/sdcard` 디렉토리를 여십시오.

iv) `cacert.crt` 파일을 `mnt/sdcard` 디렉토리로 끌어오십시오.

ADB

i) 명령 창을 열고 현재 디렉토리를 Android 설치 디렉토리의 `android-sdk\platform-tools` (예: `C:\Program Files\Android\android-sdk\platform-tools`) 로 설정하십시오.

ii) 인증서를 `mnt/sdcard` 디렉토리에 복사하십시오.

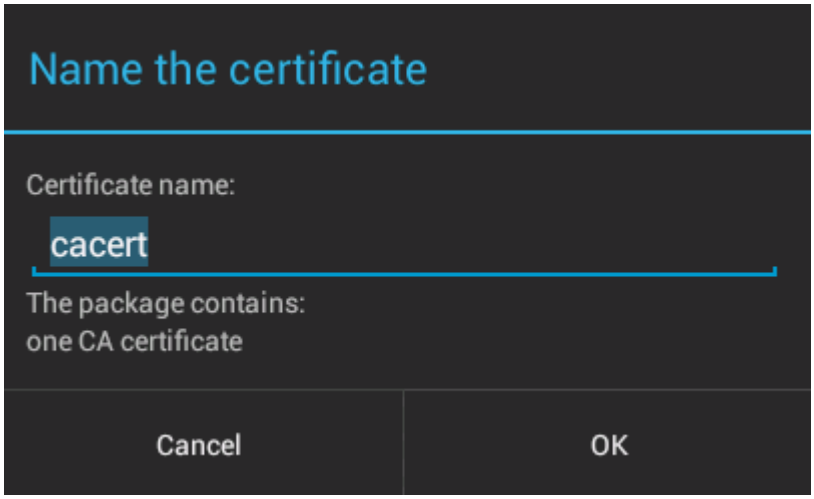
```
adb push %cacert% /mnt/sdcard/cacert.crt
```

7. Android 디바이스의 인증서 신뢰 저장소에 인증서를 설치하십시오.

   인증서에는 값이 `Subject Type=CA`인 기본 제한조건 절이 있어야 합니다.

   a) 디바이스를 잠금을 해제하고 **위젯** 단추를 클릭하십시오.

   b) **설정** > **보안** > **신임 스토리지** > **SD 카드에서 설치**를 클릭하십시오.

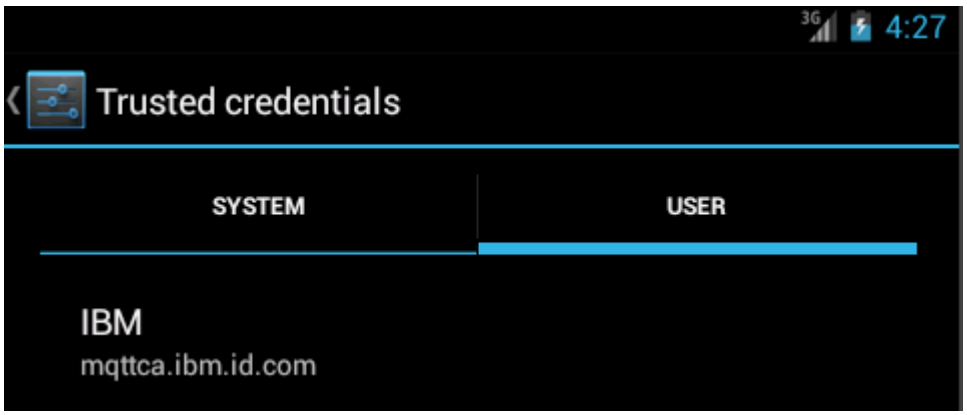


   c) 인증서 파일 이름이 올바른지 확인한 후 **확인**을 클릭하십시오.

   참고: 디바이스의 잠금을 정의하지 않은 경우 Android가 잠금을 설정하라는 프롬프트를 표시합니다.

8. 디바이스에 인증서가 설치되었는지 확인하십시오.

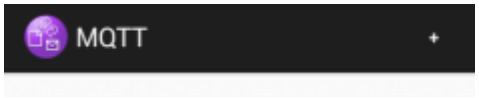
   a) **신뢰할 수 있는 신임 정보** > **사용자** 를 클릭하고 인증서가 사용자 인증서 목록에 표시되도록 몇 분 정도 기다리십시오.



9. `MQTTExerciser` 앱을 다시 실행하고 익명의 SSL 클라이언트가 사용할 수 있도록 구성한 MQTT 채널에 연결하십시오.

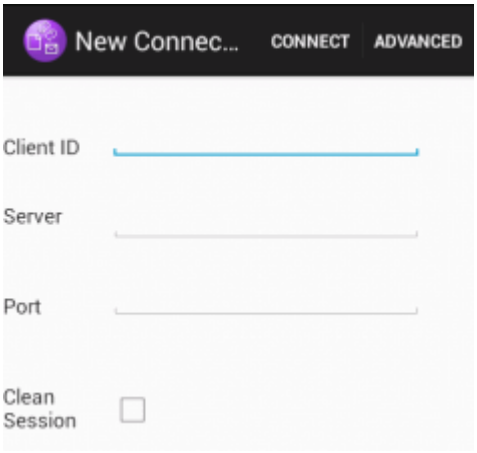
   a) Android에 대한 MQTT 클라이언트 샘플 Java 앱를 여십시오.

   이 창은 Android 디바이스에서 열립니다.



b) MQTT 서버에 연결하십시오.

i) **+** 부호를 클릭하여 새 MQTT 연결을 여십시오.



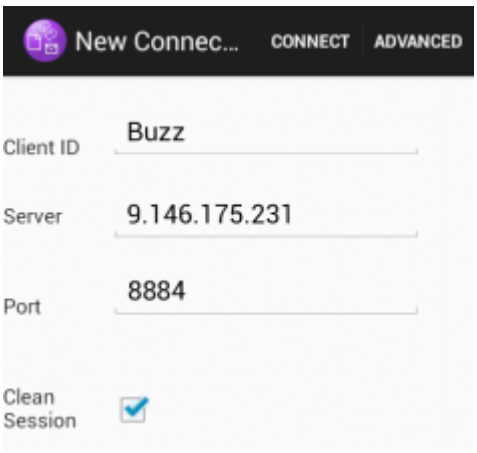
ii) **클라이언트 ID** 필드에 고유한 ID를 입력하십시오. 키 입력은 느릴 수 있으므로 기다려야 할 수도 있습니다.

iii) **서버** 필드에 MQTT 서버의 IP 주소를 입력하십시오.

이는 첫 기본 단계에서 선택한 서버입니다. IP 주소는 `127.0.0.1`일 수 없습니다.

iv) MQTT 연결의 포트 번호를 입력하십시오.

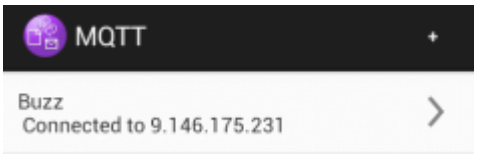
포트 번호를 8884로 설정하십시오. 포트 번호는 예제 스크립트의 `%sslportopt%` 변수로 설정됩니다. 이 포트 번호는 61 페이지의 『4』 단계에서 샘플 스크립트를 실행하여 익명의 SSL 클라이언트가 사용할 수 있도록 구성한 MQTT 채널의 포트 번호입니다.



v) **고급** 탭을 클릭하고 **SSL** 옵션을 선택하십시오. **저장**을 클릭하십시오.

vi) **연결**을 클릭하십시오.

연결에 성공하면 이 창 다음에 "연결 중" 메시지가 나타납니다.



결과

MQTTExerciser 앱은 연결하고 메시지를 교환하는 데 시간이 좀 더 걸리지만 그 외에는 비보안 연결에서 연결되는 것과 다르게 작동하지 않습니다.

관련 개념

49 페이지의 『MQTT 보안』
MQTT 보안에는 ID, 인증, 권한 부여라는 세 가지 기본 개념이 있습니다. ID는 권한 부여 중이고 권한 부여된 클라이언트의 이름을 지정하는 것과 관련되어 있습니다. 인증은 클라이언트의 ID를 증명하는 것과 관련되어 있고 권한 부여는 클라이언트에 제공되는 권한을 관리하는 것과 관련되어 있습니다.

관련 태스크

92 페이지의 『키 및 인증서 생성』
Java 클라이언트와 C 클라이언트(Android 앱과 iOS 앱, IBM WebSphere MQ 서버와 IBM MessageSight 서버 포함)의 키와 인증서를 생성하려면 다음 프로시저를 수행하십시오.

52 페이지의 『보안 MQTT 클라이언트 샘플 Java 앱 빌드와 실행』
Windows 예제를 기반으로 하여 MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 보안 샘플 Java 애플리케이션을 시작하여 실행할 수 있습니다. JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능"

69 페이지의 『JAAS 을 사용하여 MQTT 클라이언트 Java 애플리케이션 인증』
JAAS를 사용하여 클라이언트를 인증하는 방법에 대해 학습합니다. 이 태스크의 단계를 완료하여 샘플 프로그램 JAASLoginModule.java 를 수정하고 IBM WebSphere MQ 를 JAAS로 MQTT 클라이언트 Java 애플리케이션을 인증하도록 구성하십시오.

Windows에 대한 SSL 인증서를 구성하는 스크립트 예

** **

예제 명령 파일은 태스크의 단계에 설명된 대로 인증서 및 인증서 저장소를 작성합니다. 또한 예제에서는 서버 인증서 저장소를 사용하도록 MQTT 클라이언트 큐 관리자를 설정합니다. 예제는 IBM WebSphere MQ와 함께 제공되는 SampleMQM.bat 스크립트를 호출하여 큐 관리자를 삭제하고 다시 작성합니다.

initcert.bat

initcert.bat는 **keytool** 및 **openSSL** 명령에 필요한 인증서의 이름 및 경로와 기타 매개변수를 설정합니다. 이 설정은 스크립트의 주석에서 설명합니다.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
```

```
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
```

```
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 스크립트의 명령은 서버 인증서 저장소가 잠기지 않도록 하기 위해 MQTT 클라이언트 큐 관리자를 삭제한 다음 샘플 보안 스크립트가 작성한 모든 키 저장소 및 인증서를 삭제합니다.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

genkeys.bat 스크립트의 명령은 개인 인증 기관, 서버 및 클라이언트에 대한 키 쌍을 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

sscerts.bat 스크립트의 명령은 해당 키 저장소에서 클라이언트 및 서버 자체 서명 인증서를 내보낸 후 서버 인증서를 클라이언트 신뢰 저장소로 가져오고 클라이언트 인증서를 서버 키 저장소로 가져옵니다. 서버에는 신뢰 저장소가 없습니다. 명령은 클라이언트 JKS 신뢰 저장소에서 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

이 스크립트는 인증 기관 루트 인증서를 개인용 키 저장소로 가져옵니다. CA 루트 인증서는 루트 인증서와 서명된 인증서 간의 키 체인을 작성하기 위해 필요합니다. cacerts.bat 스크립트는 해당 키 저장소에서 클라이언트 및 서버 인증서 요청을 내보냅니다. 스크립트는 cajkskeystore.jks 키 저장소에서 개인용 인증 기관의 키를 사용하여 인증서 요청에 서명한 다음 서명한 인증서를 요청이 나온 곳과 같은 키 저장소로 다시 가져옵니다. 이 가져오기를 통해 CA 루트 인증서가 포함된 인증서 체인이 작성됩니다. 이 스크립트는 클라이언트 JKS 신뢰 저장소로부터 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
```

```
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

이 스크립트는 인증 디렉토리에 있는 키 저장소 및 인증서를 나열합니다. 그런 다음 MQTT 샘플 큐 관리자를 작성하고 보안 텔레메트리 채널을 구성합니다.

```
@echo ------------------------------------------------------------------------------
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V 7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

JAAS 을 사용하여 MQTT 클라이언트 Java 애플리케이션 인증

JAAS를 사용하여 클라이언트를 인증하는 방법에 대해 학습합니다. 이 태스크의 단계를 완료하여 샘플 프로그램 `JAASLoginModule.java` 를 수정하고 IBM WebSphere MQ 를 JAAS로 MQTT 클라이언트 Java 애플리케이션을 인증하도록 구성하십시오.

시작하기 전에

1. JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능". IBM 모바일 메시징 및 M2M 클라이언트 팩 에 대한 시스템 요구사항의 내용을 참조하십시오.
2. 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.

3. You must have access to the MQXR JAASLoginModule and JAASPrincipal Java samples in an IBM WebSphere MQ installation. 샘플은 %MQ_FILE_PATH%\mqxr\samples. 경로에 있습니다.
4. Windows 또는 Linux에서 단계를 완료하십시오. 이 예제는 Windows에서 가져온 것입니다.
5. 70 페이지의 『1』 단계를 완료하려면 IBM WebSphere MQ에서 MQXR_SAMPLE_QM 큐 관리자를 작성할 수 있는 권한이 있어야 합니다.

이 태스크 정보

In the task, you output the MQTT Sample client identification parameters from your version of JAASLoginModule. 클라이언트 매개변수 작성은 샘플 JAASLoginModule 프로그램 수정 및 JAASLoginModule버전을 로드하도록 IBM WebSphere MQ 를 구성하는 것을 수반합니다.

프로시저

1. 14 페이지의 『Eclipse의 모든 MQTT 클라이언트 샘플 Java 앱 컴파일과 실행』의 단계를 완료하여 Paho MQTT Sample 클라이언트를 실행하십시오.

   목표는 JAAS 인증을 개발하고 테스트하기 위해 개발 환경을 준비하는 것입니다. JAAS 인증 모듈을 조정하려면 Java 개발 환경이 필요합니다. 이 예제에서는 Java용 샘플 Paho 클라이언트를 실행하여 JAAS 구성을 테스트합니다. 간략화하기 위해 샘플 클라이언트와 샘플 JAAS 로그인 모듈을 수정하는 데 동일한 개발 환경을 사용합니다. 또는 C용 MQTT 클라이언트나 기타 MQTT 클라이언트를 사용하여 JAAS 로그인 모듈을 테스트합니다.

2. 옵션: MQTT Paho 샘플에 사용자 이름과 비밀번호 매개변수를 추가하십시오.

   참고: Java용 Paho 클라이언트에 사용자 이름과 비밀번호 매개변수가 포함되어 있는 경우에는 이 단계가 필요 없습니다. 업데이트가 있는지 다운로드 사이트를 확인하십시오. IBM 메시징 커뮤니티 다운로드를 참조하거나 그렇지 않으면 Sample.java의 사본을 변경하십시오.

   a) Paho 샘플 프로젝트의 org.eclipse.paho.sample.mqttv3app 패키지에서 패키지 탐색기를 여십시오.

   b) Sample.java **복사** > **붙여넣기**를 마우스의 오른쪽 단추로 클릭하십시오. **이름 충돌** 창에 이름 SampleForJAAS를 입력하십시오.

   c) 다음 코드 행을 main 메소드에 추가하십시오.

      i) "boolean ssl = false;" 행 뒤에 userName 변수와 password 변수를 선언하십시오.

      ```
      String password = null;
      String userName = null;
      ```

      이전 MQTT 서버와의 호환성을 위해 기본적으로 비밀번호와 사용자 이름 매개변수를 설정하지 마십시오.

      ii) "case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;" 행 뒤에 두 개의 새 입력 매개변수를 구문 분석하십시오.

      ```
      case 'u': userName = args[++i]; break;
      case 'z': password = args[++i]; break;
      ```

      iii) "if (action.equals("publish")) {" 행 앞에서 userName과 password를 Sample 생성자 인수에 추가하십시오.

      ```
      Sample sampleClient = new Sample(url, clientId, cleanSession, quietMode, userName,
      password);
      ```

   d) userName과 password를 Sample의 생성자에 추가하십시오.

      다음을

      ```
      public Sample(String brokerUrl, String clientId, boolean cleanSession,
             boolean quietMode) throws MqttException {
      ```

대상:

```
public Sample(String brokerUrl, String clientId, boolean cleanSession,
       boolean quietMode, String userName, char[] password) throws MqttException {
```

e) 다음 코드 행을 Sample 생성자에 추가하십시오.

"conOpt.setCleanSession(clean);" 행 뒤에 Sample 메소드의 conOpt 오브젝트에서 userName과 password 변수를 설정하십시오.

```
if(password != null ) {
   conOpt.setPassword(this.password.toCharArray());
}
if(userName != null) {
   conOpt.setUserName(this.userName);
}
```

f) publish와 subscribe 메소드에서 다음 코드 행을 수정하십시오.

"client.connect();" 행을 다음과 같이 변경하십시오.

```
client.connect(conOpt);
```

3. JAAS 예제에서 사용할 수 있게 Java 프로젝트인 JAASSample을 작성하십시오.
   a) Eclipse 작업공간에서 **새 Java 프로젝트** 마법사를 여십시오. **파일** > **새로 작성** > **Java 프로젝트**를 클릭하십시오.
   b) **프로젝트 이름** 필드에 JAASSample을 입력하십시오.
   c) JRE 옵션에서 실행 환경 JRE로 J2SE-1.5를 선택하고 **다음**을 클릭하십시오.

      JRE는 IBM WebSphere MQ 서버가 실행하는 JRE와 일치해야 합니다. IBM WebSphere MQ Version 7.5 는 J2SE-1.5를 실행합니다.
   d) **Java 설정** 창에서 **라이브러리** 탭을 클릭한 후 **외부 JARS 추가...**를 클릭하십시오. 찾아보기%MQ_FILE_PATH%\mqxr\lib 디렉토리를 선택하고 **MQXR.jar**을 선택하십시오. **완료**를 클릭하십시오.
4. Import the JAAS samples JAASLoginModule and JAASPrincipal classes.
   a) 패키지 탐색기에서 JAASSample 프로젝트를 마우스 오른쪽 단추로 클릭하십시오. **가져오기 ...** > **일반** > **파일 시스템** 을 클릭하고 **다음**을 클릭하십시오.
   b) %MQ_FILE_PATH%\mqxr\samples를 찾아보고 JAASLoginModule.java와 JAASPrincipal.java를 확인한 후 **완료**를 클릭하십시오.
   c) 패키지 탐색기에서 Java 파일을 모두 선택하고 마우스 오른쪽 단추로 클릭하십시오. **리팩터 ...** > **이동**.
   d) **이동** 창에서 두 요소의 목적지로 JAASSample이 선택되었는지 확인하고 **패키지 작성...**을 클릭하십시오.
   e) **새 Java 패키지** 마법사의 **이름** 필드에 samples 을 입력하십시오. **완료** > **확인** 을 클릭하십시오.

   Eclipse가 사용되지 않은 값에 대한 다수의 경고가 있는 가져온 Java 클래스를 빌드합니다.
5. JAASLoginModule 클래스의 이름을 바꾸십시오.

   IBM WebSphere MQ와 함께 제공되는 샘플 JAASLoginModule 클래스와 구별하기 쉽도록 클래스의 이름을 바꾸십시오.
   a) 패키지 탐색기에서 JAASloginModule.java 를 마우스 오른쪽 단추로 누르십시오. **리팩터 ...** > **이름 바꾸기**.
   b) **컴파일 장치 이름 바꾸기** 창에서 **새 이름** 필드를 JAASloginModule에서 MyJAASloginModule로 변경한 후 **완료**를 클릭하십시오.
6. 콜백 필드의 컨텐츠를 출력하도록 MyJAASloginModule 클래스를 수정하십시오.
   a) 다음 코드 행을 MyJAASloginModule.java에 추가하십시오.

```
System.out.println("Username=" + username
                 + "\nPassword=" + new String(password)
                 + "\nClientId=" + clientId
                 + "\nNetwork address=" + networkAddress);
```

다음 명령문 바로 앞에 행을 배치하십시오."`if (true) loggedIn = true;`"

b) CTRL+Shift+O를 눌러 가져오기를 재구성한 후 파일을 저장하십시오.

7. JAASPrincipal을 MyJAASPrincipal로 바꾸십시오.

샘플 JAASPrincipal 클래스와 혼동되지 않도록 클래스의 이름을 바꾸십시오. 예제에서는 MyJAASPrincipal 클래스의 컨텐츠를 변경하지 말고 그대로 두십시오.

8. 큐 관리자 프로세스를 실행 중인 사용자 ID에 JAAS 클래스에 대한 read 권한과 execute 권한을 부여하십시오.

a) Windows 탐색기에서 Eclipse 작업공간 디렉토리를 여십시오. 예제에서 Eclipse 작업공간 위치는 Eclipse 변수 *workspace_loc*로 표시됩니다.

b) MyJAASLoginModule 클래스와 MyJAASPrincipal 클래스가 있는 디렉토리를 찾아보십시오.

디렉토리 경로는 *workspace_loc*\JAASSample\bin\samples입니다.

c) 두 클래스를 모두 선택한 후 마우스의 오른쪽 단추로 클릭하고 **특성**을 클릭한 다음 **특성** 창에서 **보안** 탭을 클릭하십시오.

d) **추가 ...**를 클릭하십시오. 오브젝트 이름 mqm을 입력하고 **이름 확인** 을 클릭하여 확인하십시오. **확인**을 클릭하십시오.

e) **그룹 또는 사용자 이름**의 목록에서 **mqm**을 선택하고 mqm의 권한 목록에서 **읽기와 실행**, **읽기**를 선택한 후 확인을 클릭하십시오.

9. MyJAASLoginModule 클래스를 실행하도록 IBM WebSphere MQ 를 구성하십시오.

a) service.env 파일을 IBM WebSphere MQ 구성에 추가하여 MyJAASLoginModule 클래스를 로드하도록 클래스 경로를 정의하십시오.

WMQ_DATA_PATH 디렉토리에서 다음 클래스 경로 명령문을 사용하여 service.env 파일을 작성하십시오.

```
CLASSPATH=user.dir\JAASSample\bin
```

여기서, *user.dir* 는Eclipse 작업공간에서 컴파일된 클래스 파일의 디렉토리 루트입니다. *WMQ_DATA_PATH* 디렉토리에는 qmgrs 디렉토리가 포함되어 있습니다. 추가 환경 변수를 참조하십시오.

팁: service.env 파일에 CLASSPATH=*user.dir*\JAASSample\bin 명령문만 있을 수 있습니다.

b) 스탠자 MyJAASStanza를 jaas.config 파일에 추가하여 service.env 파일의 클래스 경로와 관련된 MyJAASLoginModule 클래스를 식별하십시오.

jaas.config는 큐 관리자 mqxr 디렉토리인 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr에 있습니다.

스탠자는 다음과 같습니다.

```
MyJAASStanza {
  samples.MyJAASLoginModule required debug=true;
};
```

c) JAAS 구성 스탠자의 이름을 사용하여 IBM WebSphere MQ Telemetry 채널을 구성하십시오.

명령 창에서 다음 명령을 실행하십시오.

```
echo DEFINE CHANNEL('MyJAAS') CHLTYPE(MQTT) TRPTYPE(TCP) PORT(1890)
JAASCFG('MyJAASStanza') | runmqsc MQXR_SAMPLE_QM
```

이 명령은 MyJAAS 채널을 jaas.config 파일의 MyJAASStanza에 연결합니다. MCAUSER 옵션을 지정하지 않거나, 채널 정의에서 USECLTID 옵션을 지정하면 채널이 MQTT 클라이언트 프로그램에서 제

공하는 사용자 이름을 사용하여 큐 관리자 자원에 대한 액세스 권한을 부여합니다. 예제에서는 클라이언트가 제공하는 사용자 이름을 "Guest"로 설정합니다. SampleMQM 명령 파일에서 설정한 Guest의 기존 권한도 이 예제에서 사용됩니다.

10. 새 구성 데이터를 읽으려면 IBM WebSphere MQ Telemetry 서비스를 재시작하십시오.
    IBM WebSphere MQ Telemetry 서비스를 재시작하려면 큐 관리자 또는 IBM WebSphere MQ Explorer의 서비스를 시작하거나 샘플 구성에 대해 다음 명령을 실행하십시오.

    ```
    echo stop  service(SYSTEM.MQXR.SERVICE) | runmqsc  MQXR_SAMPLE_QM
    echo start service(SYSTEM.MQXR.SERVICE) | runmqsc  MQXR_SAMPLE_QM
    ```

11. Sample 프로그램을 실행하십시오.

    SampleForJAAS의 실행 구성을 설정하려면 다음과 같이 수정하여 70 페이지의 『1』 단계와 동일한 프로시저를 수행하십시오.

    a) MQTT 채널 구성과 일치하도록 포트 번호를 1890으로 설정하십시오.

    b) Sample 프로그램에서 사용하기 위해 작성한 구독자 구성과 발행자 구성의 **(x)= Arguments** 탭에서 비밀번호에 `-u Guest -z password` 매개변수를 추가하십시오.

    샘플 프로그램이 실행되고, 포트 번호가 1883이 아니라 1890인 것 외에는 출력에 변경사항이 없습니다.

    WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM 디렉토리에서 mqxr.stdout 파일을 여십시오. MyJAASLoginModule의 출력이 mqxr.stdout에 기록됩니다.

    ```
    Username=Guest
    Password=passord
    ClientId=SampleJavaV3_subscribe
    Network address=/127.0.0.1
    ```

다음에 수행할 작업

예제가 작동하지 않는 경우 JAAS 관련 문제점 해결 주제인 173 페이지의 『문제점 해결: 텔레메트리 서비스가 JAAS 로그인 모듈을 호출하지 않음』을 읽고 다음 디버깅 팁을 시도하십시오.

1. -verbose를 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties의 매개변수에 추가하십시오. 이 로그에서 클래스가 정상적으로 로드되었는지 확인할 수 있습니다.

   출력은 *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr에 기록됩니다.

2. *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log를 조사하여 MyJAASLoginModule에서 처리된 예외가 있는지 확인하십시오. 예를 들어 문자열이 아니라 문자 배열인 널 password를 출력하려는 경우 예외가 처리됩니다.
3. *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log를 조사하십시오. userName의 사용자 이름에 큐 관리자 자원에 액세스할 권한이 부여되지 않고 채널이 MCAUSER 또는 USECLTID 옵션으로 구성된 경우 여기에 오류가 보고됩니다.
4. *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config 에서 스탠자의 이름이 Sample 클라이언트가 연결을 시도 중인 포트에 대해 구성된 MQTT 채널의 이름과 동일한지 확인하십시오.
5. 스탠자의 경로가 Eclipse의 MyJAASLoginModule 클래스에 대한 경로와 일치하는지 확인하십시오. 예:

   ```
   MyJAASStanza {
     samples.MyJAASLoginModule required debug=true;
   };
   ```

6. *WMQ_DATA_PATH*의 service.env 파일에서 클래스 경로가 올바로 선택되지 않는 결함이 발생할 수 있는 여부를 제거하려면 사용자의 클래스 경로를 포함하도록 %MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT의 "set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%" 행을 변경하십시오. 클래스 경로를 에코할 수도 있습니다. 그러나 클래스 경로에 service.env에서 설정된 클래스 경로는 포함되지 않으므로 controlMQXR.BAT 파일을 수정하는 경우에만 에코가 작동합니다.

관련 개념

49 페이지의 『MQTT 보안』
MQTT 보안에는 ID, 인증, 권한 부여라는 세 가지 기본 개념이 있습니다. ID는 권한 부여 중이고 권한 부여된 클라이언트의 이름을 지정하는 것과 관련되어 있습니다. 인증은 클라이언트의 ID를 증명하는 것과 관련되어 있고 권한 부여는 클라이언트에 제공되는 권한을 관리하는 것과 관련되어 있습니다.

110 페이지의 『텔레메트리 채널 JAAS 구성』
클라이언트에서 보내는 `Username`을 인증하기 위해 JAAS를 구성하십시오.

관련 태스크

173 페이지의 『문제점 해결: 텔레메트리 서비스가 JAAS 로그인 모듈을 호출하지 않음』
텔레메트리(MQXR) 서비스가 JAAS 로그인 모듈을 호출 중이지 않은지 알아내고 문제점을 수정하도록 JAAS를 구성합니다.

52 페이지의 『보안 MQTT 클라이언트 샘플 Java 앱 빌드와 실행』
Windows 예제를 기반으로 하여 MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 보안 샘플 Java 애플리케이션을 시작하여 실행할 수 있습니다. JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능"

60 페이지의 『SSL을 통해 Android 에 MQTT 클라이언트 샘플 Java 어플리케이션 연결』
SSL을 통해 IBM WebSphere MQ 에 연결된 샘플 Android MQTT 클라이언트를 시작하여 실행하십시오.

관련 정보

추가 환경 변수

SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets 연결

SSL및 WebSocket protocol를 사용하는 JavaScript용 MQTT 메시징 클라이언트 샘플 HTML 페이지를 사용하여 웹 애플리케이션을 IBM WebSphere MQ 에 안전하게 연결하십시오.

시작하기 전에

1. WebSockets을 통해 MQTT protocol을 지원하는 MQTT version 3 서버에 대한 액세스 권한이 있어야 합니다.
2. 브라우저에서 SSL 및 WebSocket protocol을 지원해야 합니다. 164 페이지의 『SSL을 통한 모바일 메시징 웹 앱에 대한 브라우저 지원의 제한사항』의 내용을 참조하십시오.
3. SSL 채널이 시작되어야 합니다.

이 태스크 정보

이 태스크를 완료하면 SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 샘플 페이지가 실행됩니다. 이 태스크는 92 페이지의 『키 및 인증서 생성』을 수행하여 인증서를 작성하고 IBM WebSphere MQ를 구성합니다.

인증 기관 서명된 키 또는 자체 서명된 키를 사용하여 SSL 채널을 보안 설정하십시오.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 보안 WebSockets을 통해 MQTT protocol을 지원해야 합니다.

   - IBM MessageSight와 IBM WebSphere MQ 버전 7.5.0.1 이상의 릴리스에서 이를 수행합니다.
2. 옵션: 버전 7 이상의 Java 개발 킷(JDK)을 설치하십시오.

   테스트 시스템을 설정 중이며 자체 서명된 인증서를 사용하려는 경우에는 JDK 버전 7 **keytool** 명령을 사용하여 인증서를 인증해야 합니다. 프로덕션 시스템을 설정 중이며 인증서 서명 요청(CSR)을 외부 인증 기관에 보낼 경우에는 버전 7 JDK가 필요하지 않습니다.
3. 키 쌍 및 인증서를 생성하기 위해 스크립트를 작성하고 실행하고 IBM WebSphere MQ 를 MQTT 서버로 구성하십시오.

   92 페이지의 『키 및 인증서 생성』의 단계를 수행하여 스크립트를 작성하고 실행하십시오.스크립트는 76 페이지의 『 Windows에 대한 SSL 인증서를 구성하는 스크립트 예』에도 나열되어 있습니다.

서버 인증서의 공용 이름은 서버 채널의 DNS 이름과 일치해야 합니다. 일부 브라우저는 공용 이름의 목록이 포함된 인증서를 승인합니다. 예:

```
"CN=localhost, CN=*.example.com"
```

다른 브라우저는 하나의 공용 이름만 승인합니다. 예를 들어 Firefox는 최대 버전 18까지 하나의 공용 이름만 승인합니다. 이후 버전은 다를 수 있습니다.

4. SSL 채널이 실행 중이고 예상대로 설정되어 있는지 확인하십시오.

   IBM WebSphere MQ에서 다음 명령을 명령 창에 입력하십시오.

   - 

     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```

   - 

     ```
     echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     ```

5. 브라우저 인증서 저장소에서 인증서를 설치하십시오.

   예를 들어 다음 서버 인증서 중 하나를 선택하십시오.

   a. 자체 서명된 서버 인증서의 경우 인증서는 `srvcertselfsigned.cer`입니다.

   b. 사설 인증 기관이 서명한 서버 인증서의 경우 인증서는 `cacert.cer`입니다.

   c. 외부 인증 기관이 서명한 서버 인증서의 경우 인증 기관의 루트 인증서가 이미 인증서 저장소에 설치되어 있는지 확인하십시오.

   브라우저에서 사용할 수 있는 지원에 따라 신뢰할 수 있는 루트 인증 기관 목록에 `cacert.cer`을 설치하십시오. 164 페이지의 『SSL을 통한 모바일 메시징 웹 앱에 대한 브라우저 지원의 제한사항』의 내용을 참조하십시오.

6. 옵션: 클라이언트를 인증하십시오.

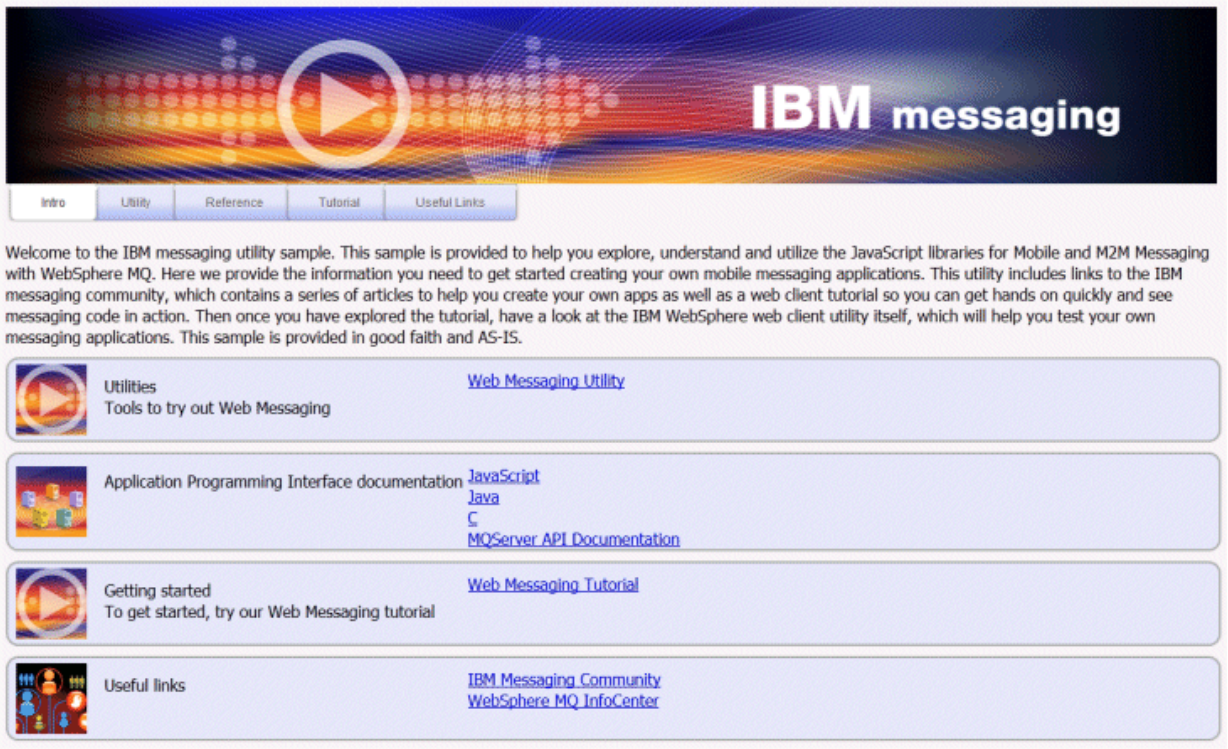
   예제에서는 `cacert.cer`을 설치하여 서버를 인증하고 채널을 암호화하지만 클라이언트를 인증하지는 않습니다. 클라이언트를 인증하려면 브라우저에서 클라이언트 키 저장소(`cltkeystore.p12`)를 설치해야 합니다. 일부 브라우저는 클라이언트 인증을 지원하지 않습니다. 164 페이지의 『SSL을 통한 모바일 메시징 웹 앱에 대한 브라우저 지원의 제한사항』의 내용을 참조하십시오.

7. 보안 WebSockets 채널에 연결하십시오.

   브라우저를 열고 주소 표시줄에 WebSockets 채널의 URL을 입력하십시오. 예:

   ```
   https://localhost:8886
   ```

   IBM WebSphere MQ가 MQTT 메시징 클라이언트 샘플 JavaScript 페이지의 첫 페이지로 응답합니다.



연결에 실패하는 경우 예제 스크립트를 실행하여 샘플 MQTT 큐 관리자를 설정했으면 포트 1886에서 정상 WebSockets 채널에 연결하십시오. 포트 1886에서 성공하면 해당 실패가 SSL 연결로 격리됩니다.

```
https://localhost:1886
```

관련 개념

112 페이지의 『JavaScript용 MQTT 메시징 클라이언트와 웹 앱』

116 페이지의 『JavaScript에서 메시징 앱을 프로그래밍하는 방법』

관련 태스크

92 페이지의 『키 및 인증서 생성』

Java 클라이언트와 C 클라이언트(Android 앱과 iOS 앱, IBM WebSphere MQ 서버와 IBM MessageSight 서버 포함)의 키와 인증서를 생성하려면 다음 프로시저를 수행하십시오.

22 페이지의 『JavaScript용 MQTT 메시징 클라이언트 시작하기』

메시징 클라이언트 샘플 홈 페이지를 표시하고 이 홈 페이지에서 링크하는 자원을 찾아봄으로써 JavaScript용 MQTT 메시징 클라이언트를 시작할 수 있습니다. To display this home page, you configure an MQTT server to accept connections from the MQTT 메시징 클라이언트 샘플 JavaScript 페이지, then you type the URL that you have configured on the server into a web browser. JavaScript용 MQTT 메시징 클라이언트가 디바이스에서 자동으로 시작되고 메시징 클라이언트 샘플 홈 페이지가 표시됩니다. 이 페이지에는 유틸리티, 프로그래밍 인터페이스 문서, 학습서, 기타 유용한 정보에 대한 링크가 있습니다.

관련 참조

164 페이지의 『SSL을 통한 모바일 메시징 웹 앱에 대한 브라우저 지원의 제한사항』

다른 플랫폼에서는 다른 브라우저 간에 기능의 차이가 있습니다. 이러한 차이를 이해하면 SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets를 사용하여 연결하도록 애플리케이션, 인증서 권한 (CAs) 및 클라이언트 인증서를 구성하는 데 도움이 됩니다.

Windows에 대한 SSL 인증서를 구성하는 스크립트 예

** **

예제 명령 파일은 태스크의 단계에 설명된 대로 인증서 및 인증서 저장소를 작성합니다. 또한 예제에서는 서버 인증서 저장소를 사용하도록 MQTT 클라이언트 큐 관리자를 설정합니다. 예제는 IBM WebSphere MQ와 함께 제공되는 `SampleMQM.bat` 스크립트를 호출하여 큐 관리자를 삭제하고 다시 작성합니다.

initcert.bat

initcert.bat는 **keytool** 및 **openSSL** 명령에 필요한 인증서의 이름 및 경로와 기타 매개변수를 설정합니다. 이 설정은 스크립트의 주석에서 설명합니다.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
```

```
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 스크립트의 명령은 서버 인증서 저장소가 잠기지 않도록 하기 위해 MQTT 클라이언트 큐 관리자를 삭제한 다음 샘플 보안 스크립트가 작성한 모든 키 저장소 및 인증서를 삭제합니다.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
```

```
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

genkeys.bat 스크립트의 명령은 개인 인증 기관, 서버 및 클라이언트에 대한 키 쌍을 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

sscerts.bat 스크립트의 명령은 해당 키 저장소에서 클라이언트 및 서버 자체 서명 인증서를 내보낸 후 서버 인증서를 클라이언트 신뢰 저장소로 가져오고 클라이언트 인증서를 서버 키 저장소로 가져옵니다. 서버에는 신뢰 저장소가 없습니다. 명령은 클라이언트 JKS 신뢰 저장소에서 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

이 스크립트는 인증 기관 루트 인증서를 개인용 키 저장소로 가져옵니다. CA 루트 인증서는 루트 인증서와 서명된 인증서 간의 키 체인을 작성하기 위해 필요합니다. cacerts.bat 스크립트는 해당 키 저장소에서 클라이언트 및 서버 인증서 요청을 내보냅니다. 스크립트는 cajkskeystore.jks 키 저장소에서 개인용 인증기관의 키를 사용하여 인증서 요청에 서명한 다음 서명한 인증서를 요청이 나온 곳과 같은 키 저장소로 다시 가져옵니다. 이 가져오기를 통해 CA 루트 인증서가 포함된 인증서 체인이 작성됩니다. 이 스크립트는 클라이언트 JKS 신뢰 저장소로부터 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
```

```
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

이 스크립트는 인증 디렉토리에 있는 키 저장소 및 인증서를 나열합니다. 그런 다음 MQTT 샘플 큐 관리자를 작성하고 보안 텔레메트리 채널을 구성합니다.

```
@echo ------------------------------------------------------------------------------
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
```

```
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
```
V 7.5.0.1
```
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopttws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

보안 MQTT 클라이언트 샘플 C 앱 빌드와 실행

Windows 예제를 기반으로 C 소스를 컴파일할 수 있는 모든 운영 체제에서 보안 샘플 C 앱을 시작하고 실행할 수 있습니다. MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 샘플 C 앱을 실행할 수 있는지 확인하십시오.

시작하기 전에

1. SSL에서 MQTT protocol 를 지원하는 MQTT version 3.1 서버에 대한 액세스 권한이 있어야 합니다.
2. 클라이언트와 서버 사이에 방화벽이 있는 경우 MQTT 트래픽을 차단하지 않는지 확인하십시오.
3. 여러 운영 체제에 C용 클라이언트 라이브러리에 필요한 2진 버전이 제공됩니다. 이 운영 체제 중 일부의 경우에는 클라이언트의 보안 버전이 2진 파일로 제공되지 않습니다. 해당 운영 체제의 경우에는 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』의 지시사항을 따라야 합니다.
4. 문제점을 해결하기 위해 IBM 지원 센터에서 사용자에게 참조 플랫폼에서 C용 MQTT 클라이언트를 실행하도록 요청할 수 있습니다.
5. SSL 채널이 시작되어야 합니다.

지원되는 플랫폼과 참조 플랫폼의 개요는 IBM 모바일 메시징 및 M2M 클라이언트 팩 에 대한 시스템 요구사항의 내용을 참조하십시오. C용 클라이언트에 대해 지원되는 항목의 자세한 내용은 WebSphere MQ V7.5 Telemetry 의 시스템 요구사항에서 관련 절을 참조하십시오.

이 태스크 정보

예를 들어, 이 기사에서는 명령행에서 Windows 의 보안 MQTT 클라이언트 샘플 C 앱 를 컴파일 및 실행하는 방법을 보여줍니다. 예시에서 Microsoft Visual Studio 2010을 사용하여 클라이언트를 컴파일합니다. 명령행 스크립트를 수정하여 Linux와 iOS 같은 다른 운영 체제에서 샘플 앱을 컴파일하고 실행할 수 있습니다.

참고:

이 기사에서 제공되는 Windows 스크립트에서는 소스에서 전체 OpenSSL 패키지를 빌드하는 것으로 가정합니다. IBM에서 제공하는 사전컴파일된 라이브러리를 사용할 경우 OpenSSL의 사전컴파일된 2진 릴리스를 가져오는 것을 선호할 수도 있습니다. iOS에서는 사전컴파일된 라이브러리를 사용할 수 없습니다.

인증 기관 서명된 키 또는 자체 서명된 키를 사용하여 SSL 채널을 보안 설정하십시오.

프로시저

1. 클라이언트 앱을 연결할 수 있는 MQTT 서버를 선택하십시오.

   서버는 SSL을 통해 MQTT version 3.1 프로토콜을 지원해야 합니다. IBM 의 모든 MQTT 서버는 IBM WebSphere MQ 및 IBM MessageSight를 포함합니다. 130 페이지의 『MQTT 서버 시작하기』의 내용을 참조하십시오.
2. 옵션: 버전 7이상에 Java 개발 킷 (JDK) 을 설치하십시오.

   버전 7은 인증서를 인증하기 위해 **keytool** 명령을 실행해야 합니다. 인증서를 인증하지 않으려면 버전 7 JDK가 필요하지 않습니다.
3. 빌드 중인 플랫폼에서 C 개발 환경을 설치하십시오.

이 토픽에서 예제에 나오는 Make 파일은 다음과 같은 도구를 대상으로 합니다.

- iOS For iOS, on Apple Mac with OS X 10.8.2 with the iOS development tools from Xcode.
- Linux Linux의 경우 Red Hat Enterprise Linux 버전 6.2의 gcc 버전 4.4.6.

  glibc C 라이브러리의 지원되는 최소 레벨은 2.12이고 Linux 커널의 지원되는 최소 레벨은 2.6.32입니다.
- Windows Microsoft Windows의 경우 Visual Studio 버전 10.0.

4. 모바일 메시징 및 M2M 클라이언트 팩 을 다운로드하고 MQTT SDK를 설치하십시오.

   설치 프로그램은 존재하지 않으며 다운로드한 파일을 펼치기만 하면 됩니다.

   a. 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오.

   b. SDK를 설치할 폴더를 작성하십시오.

      해당 폴더의 이름을 MQTT로 지정할 수도 있습니다. 이 폴더의 경로는 여기에서 *sdkroot*로 참조됩니다.

   c. 압축된 모바일 메시징 및 M2M 클라이언트 팩 파일 컨텐츠를 *sdkroot*로 펼치십시오. 이 펼치기를 통해 *sdkroot*\SDK에서 시작되는 디렉토리 트리가 작성됩니다.
5. 옵션: 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』의 단계를 수행하십시오.

   MQTT SDK에 대상 운영 체제의 보안 C 클라이언트 라이브러리가 포함되지 않는 경우에만 이 단계를 수행하십시오.

   - Windows 컴파일에 사용할 라이브러리는 `mqttv3cs.lib`이고 실행에 사용할 라이브러리는 `mqttv3cs.dll`입니다.
   - Linux 라이브러리는 `libmqttv3cs.so`입니다.
   - iOS 라이브러리는 `libmqttv3cs.a`입니다.
6. 키 쌍 및 인증서를 생성하기 위해 스크립트를 작성하고 실행하고 IBM WebSphere MQ 를 MQTT 서버로 구성하십시오.

   92 페이지의 『키 및 인증서 생성』의 단계를 수행하여 스크립트를 작성하고 실행하십시오.스크립트는 86 페이지의 『Windows에 대한 SSL 인증서를 구성하는 스크립트 예』에도 나열되어 있습니다.
7. SSL 채널이 실행 중이고 예상대로 설정되어 있는지 확인하십시오.

   IBM WebSphere MQ에서 다음 명령을 명령 창에 입력하십시오.

   - Linux

     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```
   - Windows

     ```
     echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     ```
8. 스크립트를 작성하여 보안 MQTT 클라이언트 샘플 C 앱을 빌드하고 실행하십시오.

   a) `sscclient.bat`을 작성한 후 실행하여 자체 서명 인증서로 보안 설정된 SSL 채널을 테스트하십시오.

   b) `cacclient.bat`을 작성한 후 실행하여 인증 기관이 서명한 인증서로 보안 설정된 SSL 채널을 테스트하십시오.

결과

결과는 보안 설정되지 않은 클라이언트 실행 결과와 비슷합니다.

```
Connected to ssl://localhost:8884
Subscribing to topic "MQTTV3SSample/#" qos 2
Topic:          MQTTV3SSample/C/v3
Message:        Message from MQTTv3 SSL C client
QoS:            2
Connected to ssl://localhost:8885
Subscribing to topic "MQTTV3SSample/#" qos 2
Topic:          MQTTV3SSample/C/v3
Message:        Message from MQTTv3 SSL C client
QoS:            2
```

그림 *16.* 보안 구독자

```
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
MQTTV3SSample.c
Connected to ssl://localhost:8884
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .
Connected to ssl://localhost:8885
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .
```

그림 *17.* 보안 발행자

보안 MQTT 클라이언트 샘플 C 앱을 실행할 스크립트

이 스크립트를 실행하기 전에 86 페이지의 『 Windows에 대한 SSL 인증서를 구성하는 스크립트 예』 의 스크립트를 실행하십시오.

자체 서명된 인증서를 사용하여 MQTT 클라이언트 샘플 C 앱에 보안을 설정하십시오.
sscerts.bat 스크립트를 실행하여 작성한 자체 서명된 인증서를 사용하여 이 스크립트를 실행하십시오.

```
@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal
```

그림 *18. sscclient.bat*

인증 기관이 사인한 인증서를 사용하여 MQTT 보안 클라이언트 샘플 C 앱을 실행하십시오.

cacerts.bat 스크립트를 실행하여 작성한 인증 기관이 서명한 인증서를 사용하여 이 스크립트를 실행하십시오.

```
@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal
```

그림 *19. cacclient.bat*

관련 개념

49 페이지의 『MQTT 보안』

MQTT 보안에는 ID, 인증, 권한 부여라는 세 가지 기본 개념이 있습니다. ID는 권한 부여 중이고 권한 부여된 클라이언트의 이름을 지정하는 것과 관련되어 있습니다. 인증은 클라이언트의 ID를 증명하는 것과 관련되어 있고 권한 부여는 클라이언트에 제공되는 권한을 관리하는 것과 관련되어 있습니다.

관련 태스크

92 페이지의 『키 및 인증서 생성』

Java 클라이언트와 C 클라이언트(Android 앱과 iOS 앱, IBM WebSphere MQ 서버와 IBM MessageSight 서버 포함)의 키와 인증서를 생성하려면 다음 프로시저를 수행하십시오.

Windows에 대한 SSL 인증서를 구성하는 스크립트 예

예

예제 명령 파일은 태스크의 단계에 설명된 대로 인증서 및 인증서 저장소를 작성합니다. 또한 예제에서는 서버 인증서 저장소를 사용하도록 MQTT 클라이언트 큐 관리자를 설정합니다. 예제는 IBM WebSphere MQ와 함께 제공되는 `SampleMQM.bat` 스크립트를 호출하여 큐 관리자를 삭제하고 다시 작성합니다.

`initcert.bat`

`initcert.bat`는 **keytool** 및 **openSSL** 명령에 필요한 인증서의 이름 및 경로와 기타 매개변수를 설정합니다. 이 설정은 스크립트의 주석에서 설명합니다.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
```

```
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
```

```
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 스크립트의 명령은 서버 인증서 저장소가 잠기지 않도록 하기 위해 MQTT 클라이언트 큐 관리자를 삭제한 다음 샘플 보안 스크립트가 작성한 모든 키 저장소 및 인증서를 삭제합니다.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
```

```
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

genkeys.bat 스크립트의 명령은 개인 인증 기관, 서버 및 클라이언트에 대한 키 쌍을 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

sscerts.bat 스크립트의 명령은 해당 키 저장소에서 클라이언트 및 서버 자체 서명 인증서를 내보낸 후 서버 인증서를 클라이언트 신뢰 저장소로 가져오고 클라이언트 인증서를 서버 키 저장소로 가져옵니다. 서버에는 신뢰 저장소가 없습니다. 명령은 클라이언트 JKS 신뢰 저장소에서 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-server trust store from the jks client-server trust store:
```

```
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

이 스크립트는 인증 기관 루트 인증서를 개인용 키 저장소로 가져옵니다. CA 루트 인증서는 루트 인증서와 서명된 인증서 간의 키 체인을 작성하기 위해 필요합니다. cacerts.bat 스크립트는 해당 키 저장소에서 클라이언트 및 서버 인증서 요청을 내보냅니다. 스크립트는 cajkskeystore.jks 키 저장소에서 개인용 인증기관의 키를 사용하여 인증서 요청에 서명한 다음 서명한 인증서를 요청이 나온 곳과 같은 키 저장소로 다시 가져옵니다. 이 가져오기를 통해 CA 루트 인증서가 포함된 인증서 체인이 작성됩니다. 이 스크립트는 클라이언트 JKS 신뢰 저장소로부터 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
```

```
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

이 스크립트는 인증 디렉토리에 있는 키 저장소 및 인증서를 나열합니다. 그런 다음 MQTT 샘플 큐 관리자를 작성하고 보안 텔레메트리 채널을 구성합니다.

```
@echo ________________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

키 및 인증서 생성

Java 클라이언트와 C 클라이언트(Android 앱과 iOS 앱, IBM WebSphere MQ 서버와 IBM MessageSight 서버 포함)의 키와 인증서를 생성하려면 다음 프로시저를 수행하십시오.

시작하기 전에

1. **keytool** 명령의 사본을 가지고 있어야 합니다. 모든 버전의 **keytool** 가 Java 키스토어 (JKS) 에서 공개 키 암호 시스템 (PKCS) 으로의 키 스토어 변환 또는 인증 요청의 서명을 지원하지는 않습니다. 예제에서는 이 기능을 모두 지원하는 JDK 버전 7.0에서 **keytool** 명령을 사용합니다.
2. PEM(Privacy-Enhanced Mail) 양식의 C용 클라이언트에 대한 키와 인증서를 생성하려면 **openSSL** 명령의 사본이 있어야 합니다. 29 페이지의 『C용 MQTT 클라이언트 라이브러리 빌드』의 단계를 수행하여 openSSL 패키지를 빌드하십시오.
3. 요구에 맞게 `initcert.bat` 스크립트에서 매개변수 값을 변경하십시오. 특히 비밀번호를 기록해 두지 않게 비밀번호 매개변수를 생략하도록 선택할 수 있습니다. **keytool** 명령은 비밀번호가 누락되었다는 프롬프트를 표시합니다.

이 태스크 정보

MQTT 클라이언트와 서버 사이에 보안 SSL 연결을 작성하려면 키와 인증서가 필요합니다. 이 태스크에서는 필요한 키 및 인증서를 작성하는 두 가지 방법(자체 서명됨, 자체 인증 기관에 의해 서명됨)을 보여줍니다. 수행하는 방법은 키 저장소 및 인증서 관리 계획에 따라 다릅니다.

외부 인증 기관이 서명한 인증서를 사용하려면 `cacerts.bat`의 서명 단계를 외부 인증 기관에 인증서 요청을 보내는 것으로 바꾸십시오. 인증 기관은 서명된 인증서 외에도 중간 인증서와 루트 인증서를 리턴할 수 있습니다. 리턴된 인증서를 설치할 외부 CA에서 제공하는 지침에 따르십시오.

IBM WebSphere MQ 서버는 사용자가 텔레메트리 채널 구성 매개변수에서 지정하는 인증서 저장소에서만 인증서를 검색합니다. 이 서버는 JSE `cacerts` 저장소에서 추가로 검색하지 않습니다. Java 클라이언트는 사용자가 지정하는 신뢰 저장소에서 인증서를 검색합니다. 사용자가 신뢰 저장소를 지정하지 않는 경우 이 서버는 `cacerts` 키 저장소의 JSE `jre\lib\security` 디렉토리에서 검색합니다. Android 클라이언트는 Android 디바이스의 사전정의된 인증서 저장소에서 인증서를 검색합니다. C 클라이언트 앱과 iOS 앱은 애플리케이션이 지정하는 인증서 저장소에서만 검색합니다.

Android 및 Java 클라이언트는 사전 구성된 신뢰 저장소에서 신뢰할 수 있는 인증서를 검색합니다. CA 루트 인증서는 Android 신뢰 인증서 저장소 및 JSE `jre\lib\security\cacerts` 저장소에 저장됩니다. 서버 인증서를 인증한 CA의 루트 인증서가 사전 구성된 신뢰 저장소에 이미 설치되어 있는 경우에는 클라이언트 신뢰 저장소를 정의하지 마십시오. 필요한 유일한 구성은 보안 설정된 MQTT 서버 채널의 TCP/IP 포트를 설정하는 것입니다.

키 및 인증서를 작성하고 다양한 양식을 모두 관리하는 도구는 사용이 간단하지 않습니다. 이 도구에는 관리할 매개변수가 많이 있으며 **openSSL**에는 구성 파일(`openssl.cnf`)과 명령행 매개변수가 필요합니다. C와 Java 둘 다에서 실행되는 애플리케이션에 대한 키 및 인증서를 관리하기 위해 필요한 모든 기능을 제공하는 도구는 없습니다. IBM WebSphere MQ의 텔레메트리 채널에는 JKS 키 저장소가 필요하므로 예제에서는 주로 Java 인증서 도구인 **ikeyman**과 **keytool**을 사용합니다. 그러나 Java 도구는 C 클라이언트 앱에 필수인 PEM 양식을 지원하지 않습니다. PEM 양식의 키 저장소를 작성하려면 **openSSL** 도구를 실행하십시오. **openSSL** 도구는 키 저장소를 PKCS12 양식에서 PEM 양식으로 변환하고 **keytool**은 키 저장소를 JKS 양식과 PKCS12 양식 사이에서 변환합니다. 키 저장소 변환을 위해서는 `openssl.cnf` 파일이 필요하지 않습니다. C 클라이언트 앱 또는 iOS 앱을 빌드하려는 경우에만 **openSSL**이 필수입니다. **openSSL**로 작업하는 것을 선호하는 경우에는 **keytool**로 인증서를 서명하는 대신 이를 사용하여 인증서를 서명할 수 있습니다.

프로시저

1. 명령 창을 열고 다음과 같은 스크립트를 실행하십시오.
2. `initcert.bat` 스크립트를 작성하고 실행하여 MQTT 보안 샘플 클라이언트를 실행하는 데 필요한 매개변수를 설정하십시오.
3. `cleancert.bat` 스크립트를 작성한 후 실행하여 새 키 저장소와 인증서를 작성할 수 있도록 환경을 지우십시오.

4. genkeys.bat 스크립트를 작성한 후 실행하여 필요한 키 쌍을 생성하십시오.
5. 다음 옵션 중 하나를 수행하십시오.
   - sscerts.bat 스크립트를 작성한 후 실행하여 자체 서명된 인증서를 작성하십시오.
   - cacerts.bat 스크립트를 작성한 후 실행하여 인증 기관이 서명한 인증서 체인을 작성하십시오.
6. mqcerts.bat 스크립트를 작성한 후 실행하여 MQXR_SAMPLE_QM 큐 관리자를 작성하고 해당 텔레메트리 채널을 구성하십시오.

관련 태스크

82 페이지의 『보안 MQTT 클라이언트 샘플 C 앱 빌드와 실행』
Windows 예제를 기반으로 C 소스를 컴파일할 수 있는 모든 운영 체제에서 보안 샘플 C 앱을 시작하고 실행할 수 있습니다. MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 샘플 C 앱을 실행할 수 있는지 확인하십시오.

52 페이지의 『보안 MQTT 클라이언트 샘플 Java 앱 빌드와 실행』
Windows 예제를 기반으로 하여 MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ에서 보안 샘플 Java 애플리케이션을 시작하여 실행할 수 있습니다. JSE 1.5 이상의 플랫폼에서 Java용 MQTT 클라이언트 앱을 실행할 수 있습니다. "Java 호환 가능"

Windows에 대한 SSL 인증서를 구성하는 스크립트 예

예

예제 명령 파일은 태스크의 단계에 설명된 대로 인증서 및 인증서 저장소를 작성합니다. 또한 예제에서는 서버 인증서 저장소를 사용하도록 MQTT 클라이언트 큐 관리자를 설정합니다. 예제는 IBM WebSphere MQ와 함께 제공되는 SampleMQM.bat 스크립트를 호출하여 큐 관리자를 삭제하고 다시 작성합니다.

initcert.bat

initcert.bat는 **keytool** 및 **openSSL** 명령에 필요한 인증서의 이름 및 경로와 기타 매개변수를 설정합니다. 이 설정은 스크립트의 주석에서 설명합니다.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
```

```
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
```

```
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

cleancert.bat 스크립트의 명령은 서버 인증서 저장소가 잠기지 않도록 하기 위해 MQTT 클라이언트 큐 관리자를 삭제한 다음 샘플 보안 스크립트가 작성한 모든 키 저장소 및 인증서를 삭제합니다.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

genkeys.bat 스크립트의 명령은 개인 인증 기관, 서버 및 클라이언트에 대한 키 쌍을 작성합니다.

```
@rem
@echo ____________________________________________________________________________
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

sscerts.bat 스크립트의 명령은 해당 키 저장소에서 클라이언트 및 서버 자체 서명 인증서를 내보낸 후 서버 인증서를 클라이언트 신뢰 저장소로 가져오고 클라이언트 인증서를 서버 키 저장소로 가져옵니다. 서버에는 신뢰 저장소가 없습니다. 명령은 클라이언트 JKS 신뢰 저장소에서 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

이 스크립트는 인증 기관 루트 인증서를 개인용 키 저장소로 가져옵니다. CA 루트 인증서는 루트 인증서와 서명된 인증서 간의 키 체인을 작성하기 위해 필요합니다. cacerts.bat 스크립트는 해당 키 저장소에서 클라이언트 및 서버 인증서 요청을 내보냅니다. 스크립트는 cajkskeystore.jks 키 저장소에서 개인용 인증 기관의 키를 사용하여 인증서 요청에 서명한 다음 서명한 인증서를 요청이 나온 곳과 같은 키 저장소로 다시

가져옵니다. 이 가져오기를 통해 CA 루트 인증서가 포함된 인증서 체인이 작성됩니다. 이 스크립트는 클라이언트 JKS 신뢰 저장소로부터 PEM 형식으로 클라이언트 신뢰 저장소를 작성합니다.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

이 스크립트는 인증 디렉토리에 있는 키 저장소 및 인증서를 나열합니다. 그런 다음 MQTT 샘플 큐 관리자를 작성하고 보안 텔레메트리 채널을 구성합니다.

```
@echo ____________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V 7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

MQTT 클라이언트 ID, 권한 및 인증

텔레메트리(MQXR) 서비스는 MQTT 채널을 사용하여 MQTT 클라이언트 대신 WebSphere MQ 토픽을 발행하거나 구독합니다. WebSphere MQ 관리자는 WebSphere MQ 권한에 사용되는 MQTT 채널 ID를 구성합니다. 관리자는 채널에 대한 공통 ID를 정의하거나 채널에 연결된 클라이언트의 Username 또는 ClientIdentifier를 사용할 수 있습니다.

텔레메트리(MQXR) 서비스는 클라이언트 인증서를 사용하거나 클라이언트가 제공하는 Username을 사용하여 클라이언트를 인증할 수 있습니다. Username은 클라이언트에서 제공하는 비밀번호를 사용하여 인증됩니다.

요약: 클라이언트 ID는 클라이언트 ID의 선택입니다. 컨텍스트에 따라, 클라이언트는 ClientIdentifier, Username, 관리자에서 작성하는 공통 클라이언트 ID 또는 클라이언트 인증서에 의해 식별됩니다. 인증 검사에 사용되는 클라이언트 ID는 권한 부여에 사용되는 ID와 동일할 필요는 없습니다.

MQTT 클라이언트 프로그램에서는 MQTT 채널을 통해 서버로 전송되는 Username과 Password를 설정합니다. 이는 연결을 암호화하고 인증하기 위해 필요한 SSL 특성을 설정할 수도 있습니다. 관리자는 MQTT 채널을 인증할지 선택하고 채널을 인증하는 방법을 결정합니다.

MQTT 클라이언트에 IBM WebSphere MQ에 액세스할 수 있는 권한을 부여하려면 클라이언트의 ClientIdentifier 또는 Username에 권한을 부여하거나 공통 클라이언트 ID에 권한을 부여해야 합니다. 클라이언트가 IBM WebSphere MQ에 연결되도록 허용하려면 Username에 권한을 부여하거나 클라이언트 인증서를 사용하십시오. JAAS를 구성하여 Username을 인증하고 SSL을 구성하여 클라이언트 인증서를 인증하십시오.

클라이언트에서 Password를 설정할 경우 VPN을 통해 연결을 암호화하거나 SSL을 사용하도록 MQTT 채널을 구성하여 비밀번호를 개인용으로 유지합니다.

클라이언트 인증서는 관리하기 어렵습니다. 이러한 이유로, 비밀번호 인증과 연관되는 위험을 감수할 수 있는 경우, 비밀번호 인증이 종종 클라이언트를 인증하기 위해 사용됩니다.

클라이언트 인증서를 관리하고 저장하기 위한 안전한 방법이 있는 경우, 인증서 인증에 의존할 수 있습니다. 그러나, 텔레메트리가 사용되는 환경의 유형에서 인증서가 안전하게 관리될 수 있는 경우는 드뭅니다. 대신, 클라이언트 인증서를 사용하는 디바이스의 인증이 서버에서 클라이언트 비밀번호를 인증하여 보완됩니다. 클라이언트 인증서 사용은 더 복잡하므로 매우 민감한 애플리케이션에서만 사용됩니다. 두 가지 형식의 인증 사용을 두 요소 인증이라고 합니다. 한 가지 요소(예: 비밀번호)를 알고 있고 다른 요소(예: 인증서)가 있어야 합니다.

매우 민감한 애플리케이션(예: 칩 앤 핀 디바이스)에서 디바이스는 제조할 때 내부 하드웨어와 소프트웨어가 도용되지 않도록 잠겨집니다. 신뢰할 수 있는 시간 제한된 클라이언트 인증서가 디바이스로 복사됩니다. 디바이스는 사용할 수 있는 위치에 배치됩니다. 디바이스를 사용할 때마다 비밀번호 또는 스마트 카드의 다른 인증서가 사용되어 추가적인 인증이 수행됩니다.

MQTT 클라이언트 ID 및 권한

ClientIdentifier, Username 또는 권한 부여에 필요한 공통 클라이언트 ID를 사용하여 WebSphere MQ 오브젝트에 액세스합니다.

IBM WebSphere MQ 관리자가 MQTT 채널의 ID를 선택할 수 있는 방법은 세 가지가 있습니다. 관리자는 클라이언트가 사용하는 MQTT 채널을 정의하거나 수정할 때 선택할 수 있습니다. ID는 IBM WebSphere MQ 토픽에 액세스 권한을 부여하기 위해 사용됩니다. 선택 항목은 다음과 같습니다.

1. 클라이언트 ID
2. 관리자가 채널에 제공하는 ID.
3. MQTT 클라이언트에서 전달된 사용자 이름.

사용자 이름은 MqttConnectOptions 클래스의 속성입니다. 클라이언트가 서비스에 연결되기 전에 설정되어야 합니다. 기본값은 널입니다.

IBM WebSphere MQ **setmqaut** 명령을 사용하여 사용 권한을 부여할 오브젝트와 조치를 MQTT 채널과 연관된 ID를 통해 선택합니다. 예를 들어, 큐 관리자인 QM1의 관리자가 제공한 채널 ID MQTTClient에 권한을 부여하려면 다음을 수행하십시오.

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

관련 정보

WebSphere MQ 오브젝트에 액세스할 수 있도록 MQTT 클라이언트에 권한 부여

비밀번호를 사용한 MQTT 클라이언트 인증

클라이언트 비밀번호를 사용하여 Username을 인증하십시오. 클라이언트에 권한을 부여하기 위해 사용되는 다른 ID를 사용하여 토픽에 발행하고 구독하는 클라이언트를 인증할 수 있습니다.

텔레메트리(MQXR) 서비스는 JAAS를 사용하여 클라이언트 Username을 인증합니다. JAAS는 MQTT 클라이언트가 제공하는 비밀번호를 사용합니다.

IBM WebSphere MQ 관리자는 클라이언트가 연결된 MQTT 채널을 구성하여 사용자 이름을 인증할지 또는 인증하지 않을지에 대해 결정합니다. 클라이언트는 다른 채널에 지정될 수 있고 각 채널은 다른 방법으로 해당 클라이언트를 인증하기 위해 구성될 수 있습니다. JAAS를 사용하여 클라이언트를 인증해야 하는 메소드 및 클라이언트를 선택적으로 인증할 수 있는 메소드를 구성할 수 있습니다.

인증을 위한 ID 선택은 권한 부여 ID 선택에 영향을 미치지 않습니다. 관리상의 편의를 위해 권한 부여를 위한 공통 ID를 설정할 수 있지만 해당 ID를 사용하도록 사용자별로 인증하십시오. 다음 프로시저는 개별 사용자가 공용 ID를 사용하도록 인증하는 단계를 파악합니다.

1. IBM WebSphere MQ 관리자는 IBM WebSphere MQ Explorer를 사용하여 `MQTTClientUser`와 같은 이름으로 MQTT 채널 ID를 설정합니다.
2. IBM WebSphere MQ 관리자는 토픽에 발행하고 구독하도록 `MQTTClient`에 권한을 부여합니다.

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

3. MQTT 클라이언트 앱 개발자는 서버에 연결하기 전에 `MqttConnectOptions` 오브젝트를 작성하고 `Username`과 `Password`를 설정합니다.
4. 보안 개발자는 비밀번호를 가지는 사용자 이름에 권한을 부여하기 위한 JAAS `LoginModule`을 작성하고 JAAS 구성 파일에 이를 포함시킵니다.
5. IBM WebSphere MQ 관리자는 JAAS를 사용하는 클라이언트의 `UserName`을 인증하도록 MQTT 채널을 구성합니다.

SSL을 사용한 MQTT 클라이언트 인증

MQTT 클라이언트와 큐 관리자 사이의 연결은 항상 MQTT 클라이언트가 시작합니다. MQTT 클라이언트는 항상 SSL 클라이언트입니다. 서버의 클라이언트 인증 및 MQTT 클라이언트의 서버 인증은 모두 선택사항입니다.

개인용 서명 디지털 인증서를 클라이언트에 제공하여 IBM WebSphere MQ에 대해 MQTT 클라이언트를 인증할 수 있습니다. IBM WebSphere MQ 관리자는 MQTT 클라이언트가 SSL을 사용하여 큐 관리자에 대해 인증하도록 강제 실행할 수 있습니다. 클라이언트를 상호 인증의 부분으로만 요청할 수 있습니다.

SSL 사용의 대안으로서 IPsec 등과 같은 일종의 가성 사설망(VPN)은 TCP/IP 연결의 엔드 포인트를 인증합니다. VPN은 네트워크를 통해 플로우하는 각 IP 패킷을 암호화합니다. 이러한 VPN 연결이 설정되면 신뢰할 수 있는 네트워크가 설정된 것입니다. VPN 네트워크를 통해 TCP/IP를 사용하여 MQTT 클라이언트를 임시 채널에 연결할 수 있습니다.

SSL을 사용하는 클라이언트 인증 방식은 비밀 키를 가진 클라이언트에 따라 달라집니다. 본인확인정보는 자체 서명 인증서의 경우에 클라이언트의 개인 키이거나 인증 기관이 제공하는 키입니다. 키는 클라이언트의 디지털 인증서에 서명하는 데 사용됩니다. 해당하는 공개 키를 가진 사용자는 누구나 디지털 인증서를 확인할 수 있습니다. 인증서는 신뢰할 수 있거나, 체인 연결된 경우 인증서 체인을 통해 신뢰되는 루트 인증서로 추적될 수 있습니다. 클라이언트 확인은 클라이언트가 제공한 인증서 체인에 있는 모든 인증서를 서버로 보냅니다. 서버는 신뢰하는 인증서를 찾을 때까지 인증서 체인을 검사합니다. 신뢰되는 인증서는 자체 서명 인증서에서 생성된 공용 인증서이거나 일반적으로 인증 기관이 발행한 루트 인증서입니다. 마지막으로, 신뢰되는 인증서를 "라이브" 인증서 폐기 목록과 비교할 수 있는 선택적 단계가 있습니다.

신뢰되는 인증서는 인증 기관이 발행하거나 JRE 인증서 저장소에 이미 포함되었을 수 있습니다. 이는 자체 서명 인증서이거나 신뢰되는 인증서로서 텔레메트리 채널 키 저장소에 추가된 인증서일 수도 있습니다.

참고: 텔레메트리 채널에는 하나 이상의 텔레메트리 채널 및 클라이언트를 인증하는 데 필요한 모든 공용 인증서 둘 모두를 보유하는 결합된 키 저장소/신뢰 저장소가 있습니다. SSL 채널에는 키 저장소가 있어야 하고 채널 신뢰 저장소와 동일한 파일이므로 JRE 인증서 저장소는 절대 참조되지 않습니다. 클라이언트의 인증에 CA 루트 인증서가 필요한 경우 CA 루트 인증서가 이미 JRE 인증서 저장소에 있더라도 채널의 키 저장소에 루트 인증서를 배치해야 한다는 의미입니다. JRE 인증서 저장소는 절대 참조되지 않습니다.

클라이언트 인증이 카운터하려는 위협과 클라이언트 및 서버가 위협 카운터링 시에 수행하는 역할에 대해 생각하십시오. 클라이언트 인증서만을 인증하는 것은 시스템에 비인가 액세스를 방지하기에는 충분하지 않습니다. 다른 누군가가 클라이언트 디바이스를 보유한 경우 클라이언트 디바이스는 인증서 홀더의 권한을 사용하여 수행할 필요가 없습니다. 원치 않는 공격에 대응하는 방법으로 한 가지 방어책에만 의존하지 마십시오. 적어도 두 요소 인증 방법을 사용하고 개인용 정보에 대한 지식이 있는 인증서를 보조로 소유하십시오. 예를 들어, JAAS를 사용하여 서버가 발행한 비밀번호를 사용하여 클라이언트를 인증하십시오.

클라이언트 인증서에 대한 1차 위협은 이것이 올바르지 않은 사용자의 수중에 들어간다는 점입니다. 인증서는 클라이언트에서 비밀번호로 보호된 키 저장소에 보관됩니다. 이를 키 저장소에 배치하는 방법은 무엇입니까? MQTT 클라이언트가 키 저장소에 비밀번호를 가져오는 방법은 무엇입니까? 비밀번호 보호는 얼마나 안전합니까? 텔레메트리 디바이스는 종종 제거하기 쉬우므로 개인적으로 해킹될 수 있습니다. 디바이스 하드웨어가 변경

할 수 없도록 설정되어야 합니까? 클라이언트측 인증서를 분배하고 보호하는 일은 인식하기가 어렵습니다. 이는 키 관리 문제점이라고 불립니다.

2차 위협은 디바이스가 의도하지 않은 방법으로 서버에 액세스하기 위해 잘못 사용되는 것입니다. 예를 들어, MQTT 애플리케이션이 변조되는 경우, 인증된 클라이언트 ID를 사용하여 서버 구성에서 약점을 이용할 수 있습니다.

SSL을 사용하여 MQTT 클라이언트를 인증하려면 텔레메트리 채널을 구성한 다음 클라이언트를 구성하십시오.

-
-

*SSL*을 사용하여 클라이언트 인증에 필요한 *MQTT* 클라이언트 구성

SSL을 사용하여 MQTT 클라이언트를 인증하려면 클라이언트가 SSL을 통해 텔레메트리 채널에 연결해야 합니다. SSL 클라이언트를 인증하도록 구성된 텔레메트리 채널에 해당하는 TCP 포트를 지정해야 합니다.

예를 들어 클라이언트의 경우는 다음과 같습니다.

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

클라이언트 JVM은 JSSE의 표준 소켓 팩토리를 사용해야 합니다. 자바 ME를 사용하는 경우에는, 해당 파일이 로드되었는지 확인해야 한다. Java SE를 사용하는 경우, Java 버전 1.4.1이후로 JRE가 JRE에 포함되었습니다.

SSL 연결을 사용하려면 연결하기 전에 다수의 SSL 특성을 설정해야 합니다. `-D` 스위치를 사용하여 특성을 JVM에 전달하여 특성을 설정하거나 `MqttConnectionOptions.setSSLProperties` 메소드를 사용하여 특성을 설정할 수 있습니다.

`MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`메소드를 호출하여 비표준 소켓 팩토리를 로드하는 경우, SSL 설정이 네트워크 소켓에 전달되는 방법은 응용프로그램이 정의됩니다.

클라이언트의 개인 키를 사용하거나 CA를 통해 서명된 클라이언트의 디지털 인증서를 클라이언트에서 비밀번호로 보호된 키 저장소에 추가합니다. 인증서에 키 체인이 있을 경우 키 체인에서 저장소로 인증서를 추가할 수 있습니다. 서버에서 클라이언트 인증서를 확인할 경우 클라이언트가 보낸 인증서를 사용하여 키 저장소의 인증서와 일치시킵니다. 인증서가 속한 키 체인에서 첫 번째로 일치하는 인증서를 찾습니다. 나머지 키 체인은 무시됩니다.

MQTT 클라이언트는 키 저장소에 있는 모든 인증서를 서버로 보냅니다. 서버에서 클라이언트가 보낸 모든 키 체인을 인증하면 클라이언트가 인증된 것입니다.

클라이언트 인증에 SSL 암호 스위트를 사용할 수도 있습니다.다음은 현재 지원되는 SSL 암호 스위트 목록이며 알파벳순으로 나열되어 있습니다.

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`

- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 SHA-2 암호 스위트를 사용하려는 경우 164 페이지의 『MQTT 클라이언트와 함께 SHA-2 암호 스위트를 사용하기 위한 시스템 요구사항』의 내용을 참조하십시오.

관련 개념

103 페이지의 『SSL을 사용하여 채널 인증에 필요한 MQTT 클라이언트 구성』
SSL을 통해 텔레메트리 채널을 인증하려면 클라이언트가 SSL을 통해 텔레메트리 채널에 연결해야 합니다. 이 경우 SSL에 대해 구성된 텔레메트리 채널에 해당하는 포트를 지정해야 합니다. 구성에는 서버의 개인적으로 서명된 디지털 인증서가 있는 비밀번호 문구로 보호된 키 저장소가 포함되어야 합니다.

SSL을 사용하여 텔레메트리 채널 인증

MQTT 클라이언트와 큐 관리자 사이의 연결은 항상 MQTT 클라이언트가 시작합니다. MQTT 클라이언트는 항상 SSL 클라이언트입니다. 서버의 클라이언트 인증 및 MQTT 클라이언트의 서버 인증은 모두 선택사항입니다.

클라이언트가 익명 연결을 지원하는 CipherSpec을 사용하도록 구성되지 않는 한 클라이언트는 항상 서버를 인증하려고 시도합니다. 인증에 실패하면 연결이 설정되지 않습니다.

SSL 사용의 대안으로서 IPsec 등과 같은 일종의 가성 사설망(VPN)은 TCP/IP 연결의 엔드 포인트를 인증합니다. VPN은 네트워크를 통해 플로우하는 각 IP 패킷을 암호화합니다. 이러한 VPN 연결이 설정되면 신뢰할 수 있는 네트워크가 설정된 것입니다. VPN 네트워크를 통해 TCP/IP를 사용하여 MQTT 클라이언트를 임시 채널에 연결할 수 있습니다.

SSL을 사용한 서버 인증을 통해 기밀 정보를 보낼 서버를 인증합니다. 클라이언트는 서버에서 보낸 인증서와 일치하는 검사, 신뢰 저장소에 있는 인증서 또는 JRE `cacerts` 저장소에 있는 인증서를 수행합니다.

JRE 인증 저장소는 JKS 파일 `cacerts`입니다. 이 파일은 `JRE InstallPath\lib\security\`에 있습니다. 이 파일은 `changeit`라는 기본 비밀번호로 설치됩니다. JRE 인증서 저장소에서 신뢰할 수 있는 인증서나 클라이언트 신뢰 저장소에서 신뢰할 수 있는 인증서 중 하나를 저장할 수 있습니다. 두 저장소를 모두 사용할 수는 없습니다. 클라이언트가 신뢰하는 공개 인증서를 다른 Java 애플리케이션에서 사용하는 인증서와 별도로 보관하려면 클라이언트 신뢰 저장소를 사용하십시오. 클라이언트에서 실행 중인 모든 Java 애플리케이션에 대한 공통 인증서 저장소를 사용하려면 JRE 인증서 저장소를 사용하십시오. JRE 인증서 저장소를 사용하기로 결정하면 해당 저장소에 속한 인증서를 검토하여 이러한 인증서를 신뢰할 수 있는지 확인합니다.

다른 신뢰 제공자를 통해 JSSE 구성을 수정할 수 있습니다. 신뢰 제공자를 사용자 정의하여 인증서에 대해 다른 검사를 수행할 수도 있습니다. MQTT 클라이언트를 사용한 일부 OGSi 환경에서 환경은 다른 신뢰 제공자를 제공합니다.

SSL을 통해 텔레메트리 채널을 인증하려면 서버와 클라이언트를 구성하십시오.

•

관련 개념

103 페이지의 『SSL을 사용하여 채널 인증에 필요한 MQTT 클라이언트 구성』
SSL을 통해 텔레메트리 채널을 인증하려면 클라이언트가 SSL을 통해 텔레메트리 채널에 연결해야 합니다. 이 경우 SSL에 대해 구성된 텔레메트리 채널에 해당하는 포트를 지정해야 합니다. 구성에는 서버의 개인적으로 서명된 디지털 인증서가 있는 비밀번호 문구로 보호된 키 저장소가 포함되어야 합니다.

SSL을 사용하여 채널 인증에 필요한 MQTT 클라이언트 구성

SSL을 통해 텔레메트리 채널을 인증하려면 클라이언트가 SSL을 통해 텔레메트리 채널에 연결해야 합니다. 이 경우 SSL에 대해 구성된 텔레메트리 채널에 해당하는 포트를 지정해야 합니다. 구성에는 서버의 개인적으로 서명된 디지털 인증서가 있는 비밀번호 문구로 보호된 키 저장소가 포함되어야 합니다.

예를 들어 클라이언트의 경우는 다음과 같습니다.

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

클라이언트 JVM은 JSSE의 표준 소켓 팩토리를 사용해야 합니다. 자바 ME를 사용하는 경우에는, 해당 파일이 로드되었는지 확인해야 한다. Java SE를 사용하는 경우, Java 버전 1.4.1이후로 JRE가 JRE에 포함되었습니다.

SSL 연결을 사용하려면 연결하기 전에 다수의 SSL 특성을 설정해야 합니다. `-D` 스위치를 사용하여 특성을 JVM에 전달하여 특성을 설정하거나 `MqttConnectionOptions.setSSLProperties` 메소드를 사용하여 특성을 설정할 수 있습니다.

`MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`메소드를 호출하여 비표준 소켓 팩토리를 로드하는 경우, SSL 설정이 네트워크 소켓에 전달되는 방법은 응용프로그램이 정의됩니다.

SSL을 통해 텔레메트리 채널에 연결하도록 클라이언트를 코딩하고 다음 세 가지 방법 중 하나를 사용하여 서버 인증서를 신뢰하도록 클라이언트를 구성합니다.

`cacerts` 저장소에서 잘 알려진 인증 기관에 의해 서명된 서버 인증서 사용

서버가 인증서 체인에 있는 모든 중간 키를 송신하는 경우 추가 구성이 필요하지 않습니다. 이 경우 클라이언트 JRE의 `cacerts` 저장소에 있는 인증서를 검토하고 비밀번호를 `cacerts` 저장소로 변경하는 것이 좋습니다.

기타 인증서

클라이언트의 신뢰 저장소에 신뢰할 수 있는 인증서를 저장합니다. 신뢰 저장소의 인증서 체인에 하나 이상의 인증서를 저장하고 `MqttConnectionOptions.SSLProperty`에서 신뢰 저장소 매개변수를 설정해야 합니다.

- com.ibm.ssl.trustStore
- com.ibm.ssl.trustStorePassword

사용자 정의 신뢰 관리자 사용

신뢰 제공자를 구현하고 신뢰 제공자에게 사용된 알고리즘 이름을 전달합니다. MqttConnectionOptions.SSLProperty를 사용하도록 제공자 클래스와 알고리즘의 이름을 설정합니다.

- com.ibm.ssl.trustStoreProvider
- com.ibm.ssl.trustStoreManager

채널 인증에 SSL 암호 스위트를 사용할 수도 있습니다.다음은 현재 지원되는 SSL 암호 스위트 목록이며 알파벳 순으로 나열되어 있습니다.

- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_AES_128_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_RC4_128_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA

- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V 7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V 7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V 7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V 7.5.0.2 SHA-2 암호 스위트를 사용하려는 경우 164 페이지의 『MQTT 클라이언트와 함께 SHA-2 암호 스위트를 사용하기 위한 시스템 요구사항』의 내용을 참조하십시오.

관련 개념

101 페이지의 『SSL을 사용하여 클라이언트 인증에 필요한 MQTT 클라이언트 구성』
SSL을 사용하여 MQTT 클라이언트를 인증하려면 클라이언트가 SSL을 통해 텔레메트리 채널에 연결해야 합니다. SSL 클라이언트를 인증하도록 구성된 텔레메트리 채널에 해당하는 TCP 포트를 지정해야 합니다.

텔레메트리 채널의 발행물 개인정보 보호

텔레메트리 채널에서 방향에 관계 없이 송신된 MQTT 발행물의 개인 정보는 SSL로 연결에 대해 전송을 암호화하여 보호됩니다.

텔레메트리 채널에 연결하는 MQTT 클라이언트는 채널에서 대칭 키 암호화를 사용하여 전송된 발행물의 개인 정보를 보호하기 위해 SSL을 사용합니다. 엔드 포인트가 인증되지 않으므로 채널 암호화만 신뢰할 수는 없습니다. 개인정보 보호와 서버 또는 상호 인증을 결합하십시오.

SSL 사용의 대안으로서 IPsec 등과 같은 일종의 가성 사설망(VPN)은 TCP/IP 연결의 엔드 포인트를 인증합니다. VPN은 네트워크를 통해 플로우하는 각 IP 패킷을 암호화합니다. 이러한 VPN 연결이 설정되면 신뢰할 수 있는 네트워크가 설정된 것입니다. VPN 네트워크를 통해 TCP/IP를 사용하여 MQTT 클라이언트를 임시 채널에 연결할 수 있습니다.

채널을 암호화하고 서버를 인증하는 일반 구성의 경우, 102 페이지의 『SSL을 사용하여 텔레메트리 채널 인증』의 내용을 참조하십시오.

서버를 인증하지 않고 SSL 연결을 암호화하면 연결이 중간자 공격(man-in-the-middle attack)에 노출됩니다. 교환하는 정보가 도청에 대해 보호되는 경우에도 누구와 교환하는지 알지 못합니다. 네트워크를 제어하지 않는 경우, IP 전송을 인터셉트하고 엔드 포인트로 위장하는 사람에게 노출됩니다.

서버를 인증하지 않고 익명 SSL을 지원하는 Diffie-Hellman 키 교환 CipherSpec을 사용하여 암호화된 SSL로 연결할 수 있습니다. 클라이언트와 서버 사이에 공유되고 SSL 전송을 암호화하는 데 사용되는 마스터 보안은 개인적으로 사인된 서버 인증서를 교환하지 않고 설정됩니다.

익명 연결은 안전하지 않으므로 대부분의 SSL 구현은 기본적으로 익명 CipherSpec을 사용하지 않습니다. 텔레메트리 채널에서 SSL 연결에 대한 클라이언트 요청이 승인되는 경우, 채널에는 비밀번호 문구로 보호되는 키 저장소가 있어야 합니다. 기본적으로 SSL 구현은 익명 CipherSpec을 사용하지 않으므로 키 저장소에는 클라이언트가 인증할 수 있는 개인적으로 사인된 인증서가 포함되어 있어야 합니다.

익명 CipherSpec을 사용하는 경우 서버 키 저장소가 있어야 하지만 개인적으로 서명된 인증서를 포함할 필요는 없습니다.

암호화된 연결을 설정하는 다른 방법은 클라이언트에서 신뢰 제공자를 사용자 고유의 구현으로 바꾸는 것입니다. 사용자의 신뢰 제공자는 서버 인증서를 인증하지 않지만 연결이 암호화됩니다.

MQTT 클라이언트 및 텔레메트리 채널의 SSL 구성

MQTT 클라이언트와 WebSphere MQ 텔레메트리 (MQXR) 서비스는 SSL을 사용하여 텔레메트리 채널을 연결하기 위해 JSSE (Java Secure Socket Extension) 를 사용합니다. MQTT C 클라이언트 및 디바이스용 WebSphere MQ Telemetry 디먼에서는 SSL을 지원하지 않습니다.

Telemetry 채널 및 MQTT 클라이언트를 인증하고 클라이언트 및 Telemetry 채널 사이의 메시지 전송을 암호화하도록 SSL을 구성하십시오.

SSL 사용의 대안으로서 IPsec 등과 같은 일종의 가성 사설망(VPN)은 TCP/IP 연결의 엔드 포인트를 인증합니다. VPN은 네트워크를 통해 플로우하는 각 IP 패킷을 암호화합니다. 이러한 VPN 연결이 설정되면 신뢰할 수 있는 네트워크가 설정된 것입니다. VPN 네트워크를 통해 TCP/IP를 사용하여 MQTT 클라이언트를 임시 채널에 연결할 수 있습니다.

TCP/IP를 통해 SSL 프로토콜을 사용하기 위해 Java MQTT 클라이언트와 텔레메트리 채널 간의 연결을 구성할 수 있다. 보안 개념은 사용자가 JSSE를 사용하도록 SSL을 구성하는 방법에 따라 다릅니다. 가장 안전한 구성에서 시작하여 세 가지 다른 보안 레벨을 구성할 수 있습니다.

1. 신뢰할 수 있는 MQTT 클라이언트만 연결하도록 허용합니다. MQTT 클라이언트는 신뢰할 수 있는 텔레메트리 채널에만 연결합니다. 클라이언트와 큐 관리자 사이에 메시지를 암호화하십시오. 100 페이지의 『SSL을 사용한 MQTT 클라이언트 인증』의 내용을 참조하십시오.
2. MQTT 클라이언트는 신뢰할 수 있는 텔레메트리 채널에만 연결합니다. 클라이언트와 큐 관리자 사이에 메시지를 암호화하십시오. 102 페이지의 『SSL을 사용하여 텔레메트리 채널 인증』의 내용을 참조하십시오.
3. 클라이언트와 큐 관리자 사이에 메시지를 암호화하십시오. 105 페이지의 『텔레메트리 채널의 발행물 개인정보 보호』의 내용을 참조하십시오.

JSSE 구성 매개변수

SSL 연결 구성 방법을 대체하려면 JSSE 매개변수를 수정하십시오. JSSE 구성 매개변수는 세 가지 세트로 배열됩니다.

1. IBM WebSphere MQ 텔레메트리 채널
2. MQTT Java 클라이언트
3. JRE

IBM WebSphere MQ 탐색기를 사용하여 텔레메트리 채널 매개변수를 구성하십시오. `MqttConnectionOptions.SSLProperties` 속성에서 MQTT Java 클라이언트 매개변수를 설정하십시오. 클라이언트 및 서버 모두의 JRE 보안 디렉토리에서 파일을 편집하여 JRE 보안 매개변수를 수정하십시오.

IBM WebSphere MQ Telemetry 채널

WebSphere MQ Explorer를 사용하여 모든 Telemetry 채널 SSL 매개변수를 설정합니다.

ChannelName

`ChannelName`은 모든 채널의 필수 매개변수입니다.

채널 이름은 특정 포트 번호와 연관된 채널을 식별합니다. MQTT 클라이언트 세트를 관리하는 데 도움이 되도록 채널 이름을 지정하십시오.

PortNumber

`PortNumber`는 모든 채널에서 선택적 매개변수입니다. 기본값은 TCP 채널의 경우 1883으로 설정되고 SSL 채널의 경우 8883으로 설정됩니다.

이 채널과 연관된 TCP/IP 포트 번호입니다. 채널에 정의된 포트를 지정하면 MQTT 클라이언트가 채널에 연결됩니다. 채널에 SSL 특성이 있는 경우, 클라이언트는 SSL 프로토콜을 사용하여 연결해야 합니다. 예:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

KeyFileName

`KeyFileName`은 SSL 채널의 필수 매개변수입니다. TCP 채널의 경우에는 생략되어야 합니다.

KeyFile이름 은 사용자가 제공하는 디지털 인증서가 포함된 Java키 저장소에 대한 경로입니다. 서버에서 키 저장소의 유형으로 JKS, JCEKS 또는 PKCS12를 사용하십시오.

다음 파일 확장자 중 하나를 사용하여 키 저장소 유형을 식별하십시오.

```
.jks
.jceks
.p12
.pkcs12
```

다른 파일 확장자를 사용하는 키 저장소를 JKS 키 저장소라고 가정합니다.

서버의 키 저장소 유형과 클라이언트의 다른 키 저장소 유형을 결합할 수 있습니다.

키 저장소에 서버의 개인용 인증서를 저장하십시오. 이 인증서를 서버 인증서라고 합니다. 인증서는 자체 서명되거나 서명 기관에서 서명한 인증서 체인의 일부가 될 수 있습니다.

인증서 체인을 사용하는 경우에는 연관된 인증서를 서버 키 저장소에 저장하십시오.

서버 인증서 및 해당 인증서 체인의 모든 인증서는 클라이언트로 보내져 서버 ID를 인증합니다.

ClientAuth 를 Required로 설정한 경우 키 저장소는 클라이언트를 인증하는 데 필요한 인증서를 포함해야 합니다. 클라이언트가 자체 서명 인증서 또는 인증서 체인을 송신하면 이를 키 저장소의 인증서와 대조 확인한 후 클라이언트가 인증됩니다. 인증서 체인을 사용하면 클라이언트가 다양한 클라이언트 인증서를 사용하여 실행되더라도 하나의 인증서로 많은 클라이언트를 확인할 수 있습니다.

PassPhrase

PassPhrase는 SSL 채널에 대한 필수 매개변수입니다. TCP 채널의 경우에는 생략되어야 합니다.

비밀번호 문구는 키 저장소를 보호하기 위해 사용됩니다.

ClientAuth

ClientAuth는 선택적 SSL 매개변수입니다. 클라이언트 인증으로 기본값이 설정됩니다. TCP 채널의 경우에는 생략되어야 합니다.

텔레메트리(MQXR) 서비스에서 클라이언트를 인증하도록 하려면 ClientAuth를 설정한 후 클라이언트가 텔레메트리 채널에 연결하도록 허용하십시오.

ClientAuth를 설정하면 클라이언트는 SSL을 사용하여 서버에 연결하고 서버를 인증해야 합니다. ClientAuth 설정에 대한 응답으로 클라이언트는 디지털 인증서 및 키 저장소에 있는 다른 모든 인증서를 서버로 송신합니다. 이 디지털 인증서는 클라이언트 인증서라고 합니다. 이러한 인증서는 채널 키 저장소 및 JRE cacerts 저장소에 있는 인증서에 대응하여 인증됩니다.

CipherSuite

CipherSuite는 선택적 SSL 매개변수입니다. 기본값은 사용 가능한 모든 CipherSpec을 시도하는 것입니다. TCP 채널의 경우에는 생략되어야 합니다.

특정 CipherSpec을 사용하려는 경우에는 CipherSuite를 SSL 연결을 설정하는 데 사용해야 하는 CipherSpec 이름으로 설정하십시오.

텔레메트리 서비스 및 MQTT 클라이언트는 각 끝에서 사용 가능한 모든 CipherSpec의 공통 CipherSpec을 협상합니다. 연결의 한 끝 또는 양 끝에 특정 CipherSpec 지정된 경우에는 반대쪽 끝에 있는 CipherSpec과 일치시켜야 합니다.

JSSE에 추가 제공자를 추가하여 추가 암호를 설치하십시오.

FIPS(Federal Information Processing Standard)

FIPS는 선택적 설정입니다. 기본적으로 설정되지 않습니다.

큐 관리자의 특성 패널 또는 **runmqsc**를 사용하여 SSLFIPS를 설정하십시오. SSLFIPS는 FIPS 인증 알고리즘만 사용되는지 여부를 지정합니다.

폐기 이름 목록

폐기 이름 목록은 선택적 설정입니다. 기본적으로 설정되지 않습니다.

큐 관리자의 특성 패널 또는 **runmqsc**를 사용하여 SSLCRLNL을 설정하십시오. SSLCRLNL은 인증서 폐기 위치를 제공하는 데 사용되는 인증 정보 오브젝트의 이름 목록을 지정합니다.

SSL 특성을 설정하는 다른 큐 관리자 매개변수는 사용되지 않습니다.

MQTT 자바 클라이언트

MqttConnectionOptions.SSLProperties에서 Java 클라이언트의 SSL 특성을 설정하십시오. 예를 들어, 다음과 같습니다.

```
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
com.ibm.micro.client.mqttv3.MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setSSLProperties(sslClientProperties);
```

특정 특성의 이름과 값에 대해서는 MqttConnectOptions에 대한 API 문서에 설명되어 있습니다.MQTT 클라이언트 라이브러리에 대한 클라이언트 API 문서에 대한 링크는 MQTT 클라이언트 프로그래밍 참조를 참조하십시오.

프로토콜

Protocol은 선택사항입니다.

프로토콜은 텔레메트리 서버와 협상할 때 선택됩니다. 특정 프로토콜이 필요한 경우에는 선택할 수 있습니다. 텔레메트리 서버가 해당 프로토콜을 지원하지 않는 경우, 연결이 실패합니다.

ContextProvider

ContextProvider는 선택사항입니다.

KeyStore

KeyStore는 선택사항입니다. 서버에 ClientAuth가 설정되어 클라이언트 인증을 강제 실행하는 경우 이를 구성하십시오.

개인 키를 사용하여 서명된 클라이언트의 디지털 인증서를 키 저장소에 저장하십시오. 키 저장소 경로와 비밀번호를 지정하십시오. 유형 및 제공자는 선택사항입니다. JKS는 기본 유형이며 IBMJCE는 기본 제공자입니다.

새 키 저장소 제공자를 추가하는 클래스를 참조하려면 다른 키 저장소 제공자를 지정하십시오. 키 관리자 이름을 설정하여 KeyManagerFactory를 인스턴스화하려면 키 저장소 제공자가 사용하는 알고리즘 이름을 전달하십시오.

TrustStore

TrustStore는 선택사항입니다. JRE cacerts 저장소에서 신뢰하는 모든 인증서를 배치할 수 있습니다.

클라이언트에 다른 신뢰 저장소를 사용하려면 신뢰 저장소를 구성하십시오. cacerts에 루트 인증서가 이미 저장되어 있는 잘 알려진 CA에서 실행하는 인증서를 서버에서 사용하고 있으면 신뢰 저장소를 구성할 수 없습니다.

공용으로 서명된 서버 인증서 또는 루트 인증서를 신뢰 저장소에 추가하고 신뢰 저장소의 경로와 비밀번호를 지정하십시오. JKS는 기본 유형이며 IBMJCE는 기본 제공자입니다.

새 신뢰 저장소 제공자를 추가하는 클래스를 참조하려면 다른 신뢰 저장소 제공자를 지정하십시오. 신뢰 관리자 이름을 설정하여 TrustManagerFactory를 인스턴스화하려면 키 저장소 제공자가 사용하는 알고리즘 이름을 전달하십시오.

JRE

클라이언트 및 서버의 SSL 동작에 영향을 주는 다른 측면의 Java 보안이 JRE에 구성되어 있습니다. Windows 의 구성 파일은 *Java Installation Directory*\jre\lib\security에 있습니다. 다음 표에서는 IBM WebSphere MQ와 함께 제공되는 JRE를 사용하는 경우의 경로를 보여줍니다.

표 *3. JRE SSL* 구성 파일의 플랫폼별 파일 경로	
플랫폼	**파일 경로**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux for System x(32비트)	*WMQ Installation Directory*/java/jre/lib/security
기타 UNIX and Linux 플랫폼	*WMQ Installation Directory*/java/jre64/jre/lib/security

잘 알려진 인증 기관

cacerts 파일에는 잘 알려진 인증 기관의 루트 인증서가 포함되어 있습니다. 신뢰 저장소를 지정하지 않은 경우 기본적으로 cacerts가 사용됩니다. cacerts 저장소를 사용하거나 신뢰 저장소를 제공하지 않은 경우 cacerts의 서명자 목록을 검토하고 편집하여 보안 요구사항을 충족해야 합니다.

IBM 키 관리 유틸리티를 실행하는 WebSphere MQ 명령 스트레이트 （strmqikm）을 사용하여 cacerts 를 열 수 있습니다. 비밀번호 changeit를 사용하여 cacerts를 JKS 파일로 여십시오. 비밀번호를 수정하여 파일을 보호하십시오.

보안 클래스 구성

java.security 파일을 사용하여 추가 보안 제공자 및 기타 기본 보안 특성을 등록하십시오.

권한

java.policy 파일을 사용하여 자원에 부여된 권한을 수정하십시오. javaws.policy는 javaws.jar에 권한을 부여합니다.

암호화 강도

일부 JRE는 암호화 강도가 약해진 상태로 배송됩니다. 키를 키 저장소로 가져올 수 없는 경우, 약해진 암호화 강도가 원인일 수 있습니다. **strmqikm** 명령을 사용하여 **ikeyman**을 시작하거나, 강력하지만 제한적인 관할 파일을 IBM developer kits, 보안 정보에서 다운로드하십시오.

중요사항: 비밀번호화 소프트웨어의 원산 국가는 다른 나라로 가져오기, 소유권, 사용 또는 다시 내보내기를 수행하는 데 제한사항이 있습니다. 제한 없는 정책 파일을 다운로드하거나 사용하려면 사용자 국가의 법을 확인하십시오. 암호화 소프트웨어의 수입, 소유, 사용 및 재수출과 관련된 규정 및 정책을 조사하여 허용되는지 판별하십시오.

클라이언트가 서버에 연결할 수 있도록 신뢰 제공자 수정

다음 예제에서는 MQTT 클라이언트 코드에서 신뢰 제공자를 추가하고 참조하는 방법에 대해 설명합니다. 예는 클라이언트 또는 서버의 인증을 수행하지 않습니다. 따라서 SSL 연결은 인증되지 않고 암호화됩니다.

109 페이지의 그림 20의 코드 스니펫에서는 MQTT 클라이언트에 대한 AcceptAllProviders 신뢰 제공자와 신뢰 관리자를 설정합니다.

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager","TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider","AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

그림 *20. MQTT* 클라이언트 코드 스니펫

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
    private static final long serialVersionUID = 1L;
    public AcceptAllProvider() {
        super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
        put("TrustManagerFactory.TrustAllCertificates",
                AcceptAllTrustManagerFactory.class.getName());
    }
```

그림 *21. AcceptAllProvider.java*

```
    protected static class AcceptAllTrustManagerFactory extends
            javax.net.ssl.TrustManagerFactorySpi {
        public AcceptAllTrustManagerFactory() {}
        protected void engineInit(java.security.KeyStore keystore) {}
        protected void engineInit(
                javax.net.ssl.ManagerFactoryParameters parameters) {}
        protected javax.net.ssl.TrustManager[] engineGetTrustManagers() {
            return new javax.net.ssl.TrustManager[] { new AcceptAllX509TrustManager() };
        }
```

그림 *22. AcceptAllTrustManagerFactory.java*

```
    protected static class AcceptAllX509TrustManager implements
            javax.net.ssl.X509TrustManager {
        public void checkClientTrusted(
                java.security.cert.X509Certificate[] certificateChain,
                String authType) throws java.security.cert.CertificateException {
            report("Client authtype=" + authType);
            for (java.security.cert.X509Certificate certificate : certificateChain) {
                report("Accepting:" + certificate);
            }
        }
        public void checkServerTrusted(
                java.security.cert.X509Certificate[] certificateChain,
                String authType) throws java.security.cert.CertificateException {
            report("Server authtype=" + authType);
            for (java.security.cert.X509Certificate certificate : certificateChain) {
                report("Accepting:" + certificate);
            }
        }
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return new java.security.cert.X509Certificate[0];
        }
    private static void report(String string) {
           System.out.println(string);
      }
    }
```

그림 *23. AcceptAllX509TrustManager.java*

텔레메트리 채널 JAAS 구성

클라이언트에서 보내는 Username을 인증하기 위해 JAAS를 구성하십시오.

WebSphere MQ 관리자는 JAAS를 사용한 클라이언트 인증이 필요한 MQTT 채널을 구성합니다. JAAS 인증을 수행할 각 채널에 대한 JAAS 구성 이름을 지정하십시오. 채널은 동일한 JAAS 구성을 모두 사용할 수 있거나 다른 JAAS 구성을 사용할 수 있습니다. 구성은 *WMQData directory*\qmgrs*qMgrName*\mqxr\jaas.config에 정의되어 있습니다.

jaas.config 파일은 JAAS 구성 이름에 의해 조직됩니다. 로그인 구성의 목록은 각 구성 이름 아래에 있습니다. 111 페이지의 그림 24의 내용을 참조하십시오.

JAAS에서는 네 개의 표준 로그인 모듈을 제공합니다. 표준 NT 및 UNIX 로그인 모듈의 값은 제한적입니다.

JndiLoginModule
JNDI(Java Naming and Directory Interface) 아래에 구성된 디렉토리 서비스에 대해 인증합니다.

Krb5LoginModule
Kerberos 프로토콜을 사용하여 인증합니다.

NTLoginModule
현재 사용자에 대한 NT 보안 정보를 사용하여 인증합니다.

UnixLoginModule
현재 사용자의 UNIX 보안 정보를 사용하여 인증합니다.

NTLoginModule 또는 UnixLoginModule을 사용할 때 발생하는 문제점은 텔레메트리(MQXR) 서비스가 MQTT 채널의 ID가 아니라 mqm ID로 실행된다는 점입니다. mqm은 클라이언트의 ID가 아닌 인증을 위해 NTLoginModule 또는 UnixLoginModule로 전달되는 ID입니다.

이 문제점을 해결하려면 자체 로그인 모듈을 작성하거나 기타 표준 로그인 모듈을 사용하십시오. 샘플 JAASLoginModule.java가 WebSphere MQ Telemetry와 함께 제공됩니다. 이는 javax.security.auth.spi.LoginModule 인터페이스의 구현입니다. 이를 사용하여 자체 인증 메소드를 개발하십시오.

사용자가 제공하는 새 LoginModule 클래스는 텔레메트리(MQXR) 서비스의 클래스 경로에 있어야 합니다. 클래스 경로에 있는 WebSphere MQ 디렉토리에는 클래스를 배치할 수 없습니다. 자체 디렉토리를 작성하고 텔레메트리(MQXR) 서비스에 대한 전체 클래스 경로를 정의하십시오.

service.env 파일에서 클래스 경로를 설정하여 텔레메트리(MQXR) 서비스가 사용하는 클래스를 보강할 수 있습니다. CLASSPATH는 대문자여야 하고 클래스 경로 명령문에는 리터럴만 포함될 수 있습니다. 변수를 CLASSPATH에서 사용할 수 없습니다. 예를 들어, CLASSPATH=%CLASSPATH%는 올바르지 않습니다. 텔레메트리(MQXR) 서비스는 자체 CLASSPATH를 설정합니다. service.env에 정의된 CLASSPATH는 해당 파일에 추가됩니다.

텔레메트리(MQXR) 서비스는 MQTT 채널에 연결된 클라이언트에 Username과 Password를 리턴하는 두 개의 콜백을 제공합니다. 사용자 이름 및 비밀번호 는 MqttConnectOptions 오브젝트에 설정됩니다. Username 및 Password에 액세스하는 방법에 대한 예제를 보려면 112 페이지의 그림 25의 내용을 참조하십시오.

예:

이름 지정된 구성 MQXRConfig가 있는 JAAS 구성 파일의 예입니다.

```
MQXRConfig {
  samples.JAASLoginModule required debug=true;
  //com.ibm.security.auth.module.NTLoginModule required;
  //com.ibm.security.auth.module.Krb5LoginModule required
  //                  principal=principal@your_realm
  //                  useDefaultCcache=TRUE
  //                  renewTGT=true;
  //com.sun.security.auth.module.NTLoginModule required;
  //com.sun.security.auth.module.UnixLoginModule required;
  //com.sun.security.auth.module.Krb5LoginModule required
  //                  useTicketCache="true"
  //                  ticketCache="${user.home}${/}tickets";
};
```

그림 *24.* 샘플 *jaas.config* 파일

MQTT 클라이언트가 제공하는 Username 및 Password를 수신하도록 코딩된 JAAS 로그인 모듈의 예제입니다.

```
public boolean login()
    throws javax.security.auth.login.LoginException {
  javax.security.auth.callback.Callback[] callbacks =
    new javax.security.auth.callback.Callback[2];
  callbacks[0] = new javax.security.auth.callback.NameCallback("NameCallback");
  callbacks[1] = new javax.security.auth.callback.PasswordCallback(
      "PasswordCallback", false);
  try {
    callbackHandler.handle(callbacks);
    String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
        .getName();
    char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
        .getPassword();
    // Accept everything.
    if (true) {
      loggedIn = true;
    } else
      throw new javax.security.auth.login.FailedLoginException("Login failed");

    principal= new JAASPrincipal(username);

  } catch (java.io.IOException exception) {
    throw new javax.security.auth.login.LoginException(exception.toString());
  } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
    throw new javax.security.auth.login.LoginException(exception.toString());
  }

  return loggedIn;
}
```

그림 *25.* 샘플 *JAASLoginModule.Login()* 메소드

클라이언트 프로그래밍 개념

이 절에서 설명하는 개념은 MQTT protocol의 버전 3.1용 Java 클라이언트에 대한 이해를 돕기 위해 제공됩니다. 이 개념은 com.ibm.micro.client.mqttv3 패키지를 수반하는 API 문서를 보완합니다.

com.ibm.micro.client.mqttv3 에는 MQTT 버전 3.1 프로토콜의 Java 구현에 대한 공용 메소드를 제공하는 클래스가 포함되어 있습니다. com.ibm.micro.client.mqttv3 패키지 및 Java SE및 ME에 대한 프로토콜을 구현하는 동반되는 패키지는 IBM WebSphere MQ Telemetry설치와 함께 제공됩니다.

MQTT 클라이언트를 개발 및 실행하려면 클라이언트 디바이스에 이러한 패키지를 복사하거나 설치해야 합니다. 별도의 클라이언트 런타임을 설치할 필요는 없습니다.

클라이언트용 라이센스 부여 조건은 클라이언트를 연결하고 있는 서버와 연관됩니다.

Java 클라이언트는 MQTT protocol 버전 3.1의 참조 구현입니다. 디바이스 플랫폼에 따라 알맞은 언어로 사용자 고유의 클라이언트를 구현할 수 있습니다. 자세한 내용은 MQ Telemetry Transport 양식과 프로토콜을 참조하십시오.

com.ibm.micro.client.mqttv3 패키지의 클라이언트 API 문서는 클라이언트가 연결되는 서버에 대해 어떠한 가정도 하지 않습니다. MQTT 클라이언트 라이브러리에 대한 클라이언트 API 문서에 대한 링크는 MQTT 클라이언트 프로그래밍 참조를 참조하십시오.다른 서버에 연결될 경우 클라이언트의 작동이 약간 달라질 수 있습니다. 뒤따른 설명에서는 IBM WebSphere MQ 텔레메트리(MQXR) 서비스에 연결되는 경우 클라이언트의 동작에 대해 설명합니다.

JavaScript용 MQTT 메시징 클라이언트와 웹 앱

최근까지 웹 앱 프로그래밍과 메시징 앱 작성은 서로 별개의 분야였습니다. 이전 경험과 관계 없이 JavaScript와 메시징을 함께 사용하는 데는 상당한 이점이 있습니다. 메시징 앱을 웹 앱으로 코드화하면 모든 최신 브라우저에 이를 가져와 실행할 수 있습니다. 앱을 변경하는 경우 브라우저를 새로 고칠 때마다 최신 버전을 가져옵니다. 또한 브라우저가 메시지의 보안과 신뢰할 수 있는 전송을 담당합니다.

웹 앱을 사용하면 애플리케이션 배치가 간단해지는 이유

IBM WebSphere MQ 등에서 기존의 메시징 앱을 개발하고 배치한 경험이 있는 경우 다음과 같은 배치 프로세스에 익숙할 것입니다.

1. 시스템 관리자가 클라이언트 라이브러리를 설치하거나 임베드합니다.
2. 시스템 관리자가 메시징 앱을 일반 사용자에게 분배하고 일반 사용자의 로컬 시스템에 설치하는 작업을 처리합니다.
3. 코드가 변경되면 시스템 관리자가 앞의 단계를 반복합니다(따라서 변경 관리가 복잡함).

메시징 앱을 웹 앱으로 코드화하는 경우 배치 단계는 다음과 같습니다.

1. 시스템 관리자가 웹 앱과 클라이언트 라이브러리를 URL에서 제공합니다.
2. 일반 사용자의 브라우저가 웹 앱과 클라이언트 라이브러리를 함께 가져옵니다.
3. 코드가 변경되면 브라우저를 새로 고칠 때 업데이트된 버전이 선택됩니다(따라서 변경 관리가 간편함).

웹 앱의 브라우저에서 직접 메시징을 사용하는 경우

JavaScript에서 앱을 프로그래밍한 경험이 있는 경우 IBM WebSphere MQ와 같은 메시징 시스템에서 제공하는 이점은 다음과 같습니다.

- 메시징 시스템을 통해 메시지를 송수신하는 경우 해당 시스템이 메시지가 확실히 전달되도록 책임을 져야 합니다.
- 메시징 시스템이 전달을 책임지므로 웹 앱은 "시작 후 삭제"될 수 있습니다. 이는 프로그래밍 논리를 상당히 단순화합니다. 메시지가 전달되는 경우 코드에서 메시지가 도착했는지 확인할 필요가 없습니다. 사용자의 앱이 더 이상 수신 확인을 처리하거나 미배달 메시지를 저장한 후 나중에 재시도하지 않아도 됩니다.
- 메시징 시스템에서 이벤트 중심 메시징을 제공합니다. 클라이언트 앱은 더 이상 요청을 송신한 후 계속해서 응답을 폴링할 필요가 없습니다. 대신 흥미로운 이벤트가 발생하는 경우 메시징 서버가 클라이언트 앱에 메시지를 보냅니다. 이는 이벤트가 발생하는 경우 다음에 앱이 서버를 폴링할 때까지 대기하지 않고 즉시 클라이언트 앱에 경보를 보내는 것을 의미합니다.
- 또한 이벤트 중심 메시징은 클라이언트 앱을 호스팅하는 디바이스의 로드, 브라우저와 메시징 서버 간 네트워크 트래픽, 메시징 서버의 로드를 상당히 줄입니다. 점점 더 많은 시스템이 모바일 디바이스에서 실행되고 무선 네트워크를 통해 연결되므로 로드 감소는 더욱 중요합니다.

함께 작동하는 원리

JavaScript용 MQTT 메시징 클라이언트에는 클라이언트 라이브러리와 이 라이브러리를 사용하는 예제 웹 앱이 포함되어 있습니다. 사용자는 이 라이브러리를 사용하는 자체 웹 앱을 코드화합니다. 그러면 MQ 큐 관리자(다음 다이어그램에 표시) 또는 애플리케이션 서버 등을 통해 사용자가 선택한 URL에서 웹 앱과 클라이언트 라이브러리를 사용할 수 있습니다. 브라우저가 웹 앱과 클라이언트 라이브러리를 가져오면 웹 앱이 브라우저를 사용하여 IBM WebSphere MQ Telemetry 또는 IBM MessageSight와 같은 MQTT 서버에 연결하고 메시지를 교환합니다.

플로우는 다음과 같습니다.

1. 브라우저의 각 인스턴스가 웹 앱을 사용할 수 있는 URL에 대한 연결을 새로 고치고 웹 앱과 클라이언트 라이브러리의 최신 버전이 브라우저에 로드됩니다.
2. WebSocket protocol을 통한 MQTT를 사용하여 웹 앱이 큐 관리자에 연결하고 관심 있는 토픽을 구독합니다.
3. 큐 관리자는 동일한 연결을 사용하여 구독과 일치하는 메시지를 다시 웹 앱에 보냅니다.

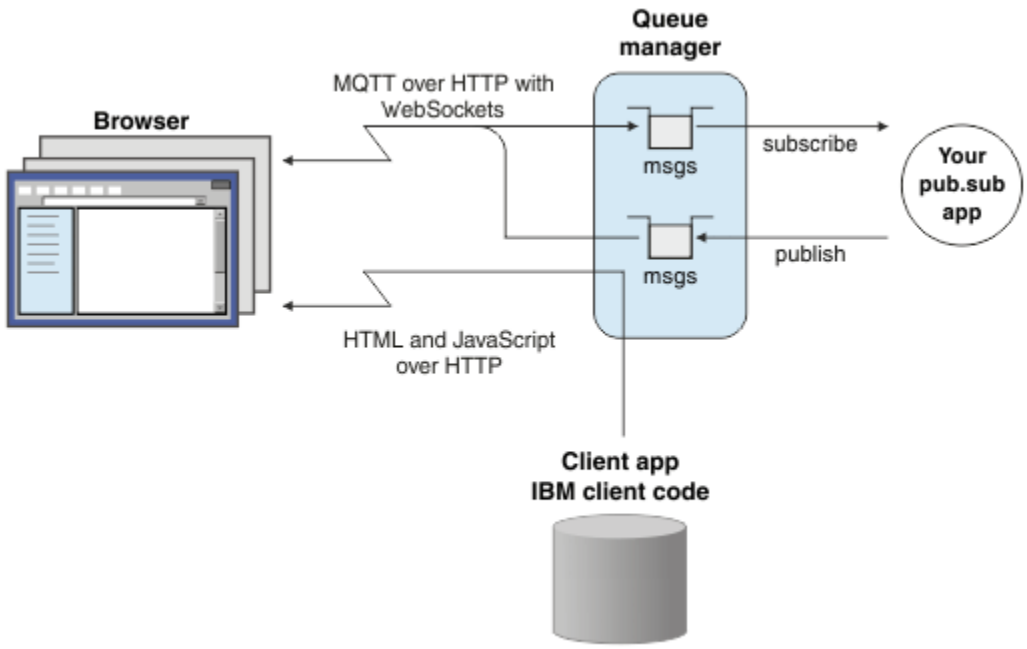


그림 *26.* 발행/구독 메시징과 함께 *JavaScript*용 *MQTT* 메시징 클라이언트 사용

웹 앱에는 애플리케이션 논리와 MQTT 서버의 URL이 포함되어 있습니다. 브라우저에서 앱이 열리면 앱이 MQTT 서버에 연결되고 필요한 구독을 작성한 후 이벤트 중심 경보를 수신하도록 대기하고 경보에 대한 조치를 수행합니다.

웹 앱은 WebSockets을 통해 실행되는 MQTT를 전송 프로토콜로 사용하여 연결됩니다. 최신 브라우저에서는 WebSockets 연결이 가능합니다. WebSockets을 사용하면 웹 앱이 HTTP와 WebSocket protocol을 허용하는 방화벽을 통과해 메시지를 전달할 수 있으며 IP를 통한 TCP를 사용하는 것과 같이 데이터 패킷("프레임"이라고 함)을 송신할 수 있습니다.

웹 앱에서 송신한 메시지가 MQTT 서버에 도착하면 서버 측 앱에서는 이를 메시지로만 봅니다. 메시지가 브라우저에서 송신된 것은 알지 못합니다.

MQTT 서버 관리와 제어

MQTT 서버는 메시징의 서버 측 복잡도를 처리합니다. 웹 앱에서 수신하는 메시지의 전달을 확인하고 웹 앱에 응답하는 발행/구독 애플리케이션을 호스팅합니다. MQTT 서버와 관련하여 다음 단계를 완료해야 합니다.

- 서버를 작성하십시오.
- 포트를 선택하십시오.
- 새 MQTT 채널을 정의하십시오.
- 새 MQTT 채널을 통해 선택한 포트에 연결하도록 클라이언트 웹 앱을 구성하십시오.

브라우저에 웹 앱 실행 가능 JavaScript도 제공해야 합니다. IBM WebSphere MQ Telemetry를 사용 중인 경우에는 기본적으로 MQTT 서버가 웹 앱에서 MQTT 서버에 연결하는 데 사용하는 것과 동일한 MQTT 채널을 사용하여 이를 수행합니다. 이는 MQTT를 시험 중인 경우 빨리 시작하여 실행하는 데 유용합니다. 프로덕션 특히 처리량이 많은 환경에서 사용할 경우 WebSphere Application Server와 같은 전용 애플리케이션 서버를 사용하여 분리된 채널에서 웹 앱 실행 가능 JavaScript를 제공하는 것이 좋을 수 있습니다.

참고: 처리량이 많은 환경에서 사용되도록 설계되었으므로 IBM MessageSight에서는 사용자가 이를 수행할 것으로 예상합니다.

예를 들어 IBM WebSphere MQ Telemetry를 사용 중인 경우 IBM WebSphere MQ Explorer **새 텔레메트리 채널** 마법사를 사용하여 다음 단계를 완료하십시오.

1. 서버를 작성하십시오.

2. 포트를 선택하십시오(기본적으로 1883).
3. 새 MQTT 채널 정의를 수행하십시오.
4. 새 MQTT 채널을 통해 선택한 포트에 연결하도록 클라이언트 웹 앱을 구성하십시오.

웹 앱 실행 가능 JavaScript는 (선택적으로) 동일한 채널에 있는 큐 관리자를 통해서도 제공됩니다. 이를 수행하려면 큐 관리자가 MQTT와 HTTP를 모두 지원해야 합니다. MQTT에서 사용할 수 있게 구성된 큐 관리자가 이미 있는 경우에는 MQSC 명령행 도구를 사용하여 MQTT와 HTTP를 모두 지원하도록 채널 정의에서 프로토콜을 대체할 수 있습니다. ALTER CHANNEL의 내용을 참조하십시오.

웹 앱과 JavaScript용 MQTT 메시징 클라이언트 클라이언트 라이브러리는 애플리케이션 서버 또는 큐 관리자가 정의한 구조의 디스크에 저장됩니다. IBM WebSphere MQ Telemetry를 사용 중인 경우 웹 앱과 클라이언트 라이브러리는 다음과 같은 디렉토리 구조에 저장됩니다.

```
MQINSTALL
  |
  mqxr
  |  |
  |  SDK
  |    |
  |    WebContent
  |       sample web app (the "Web Messaging Utility" sample HTML pages)
  |       |
  |       WebSocket
  |          IBM client library (the messaging client JavaScript classes)
  |
  qmgrs
    |
    qmgr_name
       |
       mqxr
          |
          WebContent
             |
             your_client_app (your own JavaScript pages)
```

샘플 웹 앱과 클라이언트 라이브러리는 *MQINSTALL*/mqxr/SDK/WebContent 디렉토리에 저장됩니다. 모든 큐 관리자가 이 디렉토리의 자료를 제공합니다. 사용자가 이 자료를 모두 보고 사용하지 못하도록 하려면 고유한 앱 버전을 작성해야 합니다. 이 앱 또는 특정 큐 관리자에서 사용할 수 있는 고유 대체 앱을 작성하려면 앱을 *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent 디렉토리에 두십시오. URL에서 제공할 앱 및 연관된 JavaScript 클래스를 선택하기 위해 큐 관리자는 먼저 고유 WebContent 디렉토리에서 글로벌 WebContent 디렉토리를 검색합니다. 이전 예제 디렉토리 트리에서는 큐 관리자가 *your_client_app*과 JavaScript 클래스의 글로벌 사본을 제공합니다.

큐 관리자가 웹 앱 실행 가능 파일을 제공하는 것을 중지하거나 큐 관리자가 실행 파일을 검색하는 위치를 수정하려면 **webcontentpath** 특성을 구성한 후 이를 mqxr.properties 파일에 추가하십시오. MQXR 특성을 참조하십시오.

관련 개념

116 페이지의 『JavaScript에서 메시징 앱을 프로그래밍하는 방법』

관련 태스크

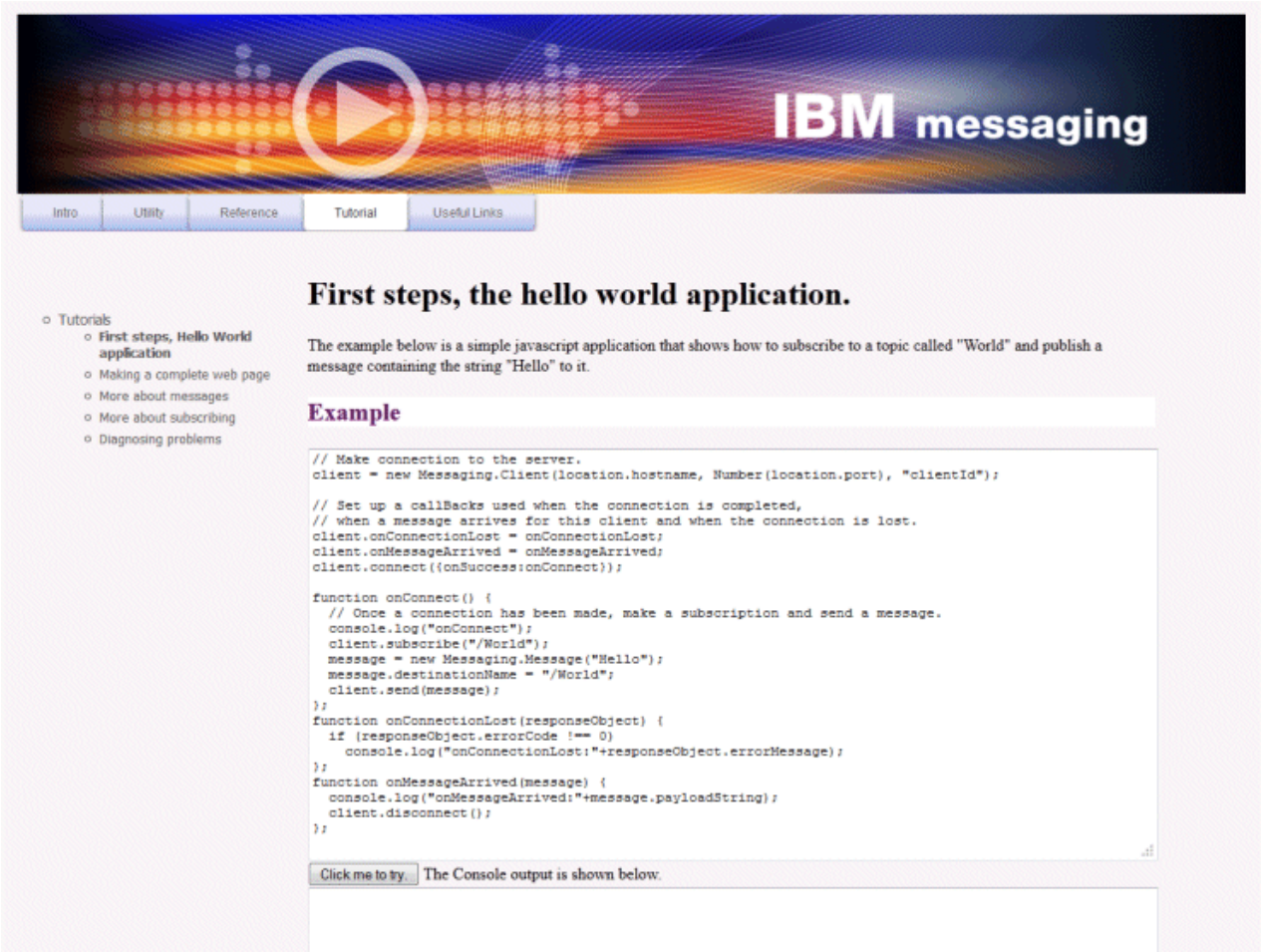
74 페이지의 『SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets 연결』
SSL및 WebSocket protocol를 사용하는 JavaScript용 MQTT 메시징 클라이언트 샘플 HTML 페이지를 사용하여 웹 애플리케이션을 IBM WebSphere MQ 에 안전하게 연결하십시오.

22 페이지의 『JavaScript용 MQTT 메시징 클라이언트 시작하기』
메시징 클라이언트 샘플 홈 페이지를 표시하고 이 홈 페이지에서 링크하는 자원을 찾아봄으로써 JavaScript용 MQTT 메시징 클라이언트를 시작할 수 있습니다. To display this home page, you configure an MQTT server to accept connections from the MQTT 메시징 클라이언트 샘플 JavaScript 페이지, then you type the URL that you have configured on the server into a web browser. JavaScript용 MQTT 메시징 클라이언트가 디바이스에서 자동으로 시작되고 메시징 클라이언트 샘플 홈 페이지가 표시됩니다. 이 페이지에는 유틸리티, 프로그래밍 인터페이스 문서, 학습서, 기타 유용한 정보에 대한 링크가 있습니다.

JavaScript에서 메시징 앱을 프로그래밍하는 방법

JavaScript용 MQTT 메시징 클라이언트에는 단순한 발행/구독 웹 앱을 작성하는 방법을 설명하는 학습서가 포함되어 있습니다. "첫 단계, Hello world" 애플리케이션 코드를 탐색하여 메시징에 사용할 웹 앱을 프로그래밍하는 기술의 기초를 이해할 수 있습니다.

지금까지의 경험이 주로 일반적인 메시징 애플리케이션을 개발하고 배치하는 작업에 관한 것인 경우 116 페이지의 『JavaScript 코드화 팁』 절의 내용도 유용할 것입니다. 메시징을 처음 접하는 경험이 풍부한 JavaScript 개발자의 경우 118 페이지의 『메시징 기본』 절에 주요 메시징 개념이 간략히 소개되어 있습니다.



JavaScript 코드화 팁

메시징 애플리케이션을 개발하는 데 익숙하지만 웹 앱을 처음 접하는 경우 다음과 같은 팁이 유용할 수 있습니다.

onSuccess 콜백에서 각 이벤트의 코드 랩핑

메시징 애플리케이션을 코드화하는 경우 다음 이벤트를 다음과 같은 순서로 코드화합니다.

1. connect
2. 구독(subscribe)
3. 발행(Publish)
4. 메시지 수신

JavaScript용 MQTT 메시징 클라이언트 API는 완전히 비동기적이며 이는 연결 또는 구독과 같은 호출이 적용되도록 대기하는 동안 애플리케이션 스레드가 차단되지 않음을 의미합니다. 대신 해당 호출은 `onSuccess` 또는 `onFailure` 콜백을 호출하여 완료를 알립니다. 다음 이벤트가 트리거되기 전에 각 이벤트가 완료되었는지 확인하려면 `onSuccess` 콜백에서 각 이벤트의 코드를 랩핑해야 합니다. 예를 들면 JavaScript 애플리케이션이 연결이 작성되기 전에 연결 호출 중에 되돌아올 수 있습니다. 구독하기 전에 연결이 발생했는지 확인하려면 연결과 관련된 `onSuccess` 호출에 구독 코드를 넣어야 합니다.

"첫 단계, Hello world" 애플리케이션 코드에서 이 접근 방식을 사용합니다.

HTML 마크업에 애플리케이션 코드 임베드

다음은 JavaScript 페이지 예입니다.

다음은 애플리케이션 코드를 HTML 마크업에 임베드하는 방법을 보여주는 이전 페이지의 소스입니다.

```
<!DOCTYPE html>


<head>
  <script type="text/javascript" src="../WebSocket/mqttws31.js"></script>

  <script type="text/javascript">

  var client;
  var form = document.getElementById("tutorial");

  function doConnect() {
      client = new Messaging.Client("whistler1.hursley.ibm.com", 1883, "ClientId");
      client.onConnect = onConnect;
      client.onMessageArrived = onMessageArrived;
      client.onConnectionLost = onConnectionLost;
      client.connect({onSuccess:onConnect});
  }

  function doSubscribe() {
      client.subscribe("/World");
  }

  function doSend() {
      message = new Messaging.Message("Hello");
      message.destinationName = "/World";
      client.send(message);
  }

  function doDisconnect() {
      client.disconnect();
  }

  // Web Messaging API callbacks

  function onConnect() {
      var form = document.getElementById("example");
      form.connected.checked= true;
  }

  function onConnectionLost(responseObject) {
      var form = document.getElementById("example");
      form.connected.checked= false;
      if (responseObject.errorCode !== 0)
          alert(client.clientId+"\n"+responseObject.errorCode);
  }
```

```
  function onMessageArrived(message) {
      var form = document.getElementById("example");
      form.receiveMsg.value = message.payloadString;
  }

  </script>
</head>

<body>
  <h1>Example Web Messaging web page.</h1>
  <form id="example">
  <fieldset>
  <legend id="Connect" > Connect </legend>
    Make a connection to the server, and set up a call back used if a
    message arrives for this client.
    <br>
    <input type="button" value="Connect" onClick="doConnect(this.form)" name="Connect"/>
    <input type="checkbox" name="connected" disabled="disabled"/>
  </fieldset>

  <fieldset>
  <legend id="Subscribe" > Subscribe </legend>
  Make a subscription to topic "/World".
  <br> <input type="button" value="Subscribe" onClick="doSubscribe(this.form)"/>
  </fieldset>

  <fieldset>
  <legend id="Send" > Send </legend>
    Create a Message object containing the word "Hello" and then publish it at
    the server.
    <br>
    <input type="button" value="Send" onClick="doSend(this.form)"/>
  </fieldset>

  <fieldset>
  <legend id="Receive" > Receive </legend>
    A copy of the published Message is received in the callback we created earlier.
  <textarea name="receiveMsg" rows="1" cols="40" disabled="disabled"></textarea>
  </fieldset>

  <fieldset>
  <legend id="Disconnect" > Disconnect </legend>
    Now disconnect this client from the server.
  <br> <input type="button" value="Disconnect" onClick="doDisconnect()"/>
  </fieldset>
  </form>
</body>
</html>
```

메시징 기본

다음은 메시징을 처음 접하는 웹 앱 개발자를 위한 몇 가지 메시징 배경 정보입니다.

비동기 메시징과 시작 후 삭제 메시징

MQTT 프로토콜은 보장된 전달과 전송 후 삭제 전송을 지원합니다. 프로토콜에서는 메시지 전달이 비동기입니다. 앱은 메시지를 클라이언트 API에 전달한 후 메시지가 전달되었는지 확인하기 위해 추가적인 조치를 수행하지 않습니다. 이러한 접근 방식을 시작 후 삭제라고 합니다. 응답이 제공되는 경우 응답이 자동으로 앱에 송신됩니다.

비동기 전달을 사용하면 앱이 서버 연결에서 분리되고 메시지를 기다리지 않습니다. 상호작용 모델은 이메일과 비슷하지만 애플리케이션 프로그래밍에 맞게 최적화되어 있습니다.

5 페이지의 『MQTT 소개』의 "MQTT 프로토콜" 절도 참조하십시오.

발행/구독 메시징의 개요

정보 제공자는 발행자라고 합니다. 발행자는 주제에 대한 정보를 제공하며 해당 정보에 관심이 있는 애플리케이션에 대해 아무 것도 알 필요가 없습니다. 발행자는 특정 주제에 대한 메시지의 컨테이너인 토픽을 선택합니다. 그런 다음 발행자는 해당 주제에 대한 각 정보 조각을 메시지(발행물이라고 함)로 생성하고 이를 연관된 토픽에 게시합니다.

정보 이용자를 구독자라고 합니다. 구독자는 관심 있는 토픽에 대한 구독을 작성합니다. 새 메시지가 토픽에 게시되면 해당 토픽의 모든 구독자에게 메시지가 전달됩니다. 구독자는 여러 구독을 작성하고 많은 다양한 발행자로부터 정보를 수신할 수 있습니다.

IBM WebSphere MQ 발행/구독 메시징 소개 도 참조하십시오.

구독과 토픽의 일치 방법

IBM WebSphere MQ를 MQTT 서버로 사용 중인 경우 IBM WebSphere MQ가 토픽을 지정하는 방법을 이해해야 합니다. IBM WebSphere MQ에서 발행자는 메시지를 작성하고 발행물의 주제에 가장 적합한 토픽 문자열을 사용하여 메시지를 발행합니다. 구독자는 발행을 수신하기 위해 발행 토픽을 선택하도록 토픽 문자열과 일치하는 패턴의 구독을 작성합니다. 큐 관리자는 발행 토픽과 일치하는 구독을 가진 구독자에게 발행을 전달하며 이 큐 관리자에게 발행을 수신할 수 있는 권한이 부여됩니다.

일반적으로 주제는 계층 구조로 된 토픽 트리로 구성되며 '/' 문자를 사용해서 토픽 문자열에 하위 토픽을 작성합니다. 토픽은 토픽 트리의 노드입니다. 토픽은 추가 하위 토픽이 없는 리프 노드 또는 하위 토픽이 있는 중간 노드입니다. 구독자는 와일드카드를 사용하여 한 번에 둘 이상의 토픽을 구독할 수 있습니다. 예를 들어 `/sport/tennis`를 구독하면 테니스 하위 주제에 게시되는 메시지만 가져오지만 `/sport/#`을 구독하면 `/sport`의 모든 하위 토픽에 게시되는 메시지를 가져옵니다.

토픽, 토픽 트리, 와일드카드 설계의 내용도 참조하십시오.

관련 개념

112 페이지의 『JavaScript용 MQTT 메시징 클라이언트와 웹 앱』

관련 태스크

74 페이지의 『SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets 연결』
SSL및 WebSocket protocol를 사용하는 JavaScript용 MQTT 메시징 클라이언트 샘플 HTML 페이지를 사용하여 웹 애플리케이션을 IBM WebSphere MQ 에 안전하게 연결하십시오.

22 페이지의 『JavaScript용 MQTT 메시징 클라이언트 시작하기』
메시징 클라이언트 샘플 홈 페이지를 표시하고 이 홈 페이지에서 링크하는 자원을 찾아봄으로써 JavaScript용 MQTT 메시징 클라이언트를 시작할 수 있습니다. To display this home page, you configure an MQTT server to accept connections from the MQTT 메시징 클라이언트 샘플 JavaScript 페이지, then you type the URL that you have configured on the server into a web browser. JavaScript용 MQTT 메시징 클라이언트가 디바이스에서 자동으로 시작되고 메시징 클라이언트 샘플 홈 페이지가 표시됩니다. 이 페이지에는 유틸리티, 프로그래밍 인터페이스 문서, 학습서, 기타 유용한 정보에 대한 링크가 있습니다.

MQTT 클라이언트 애플리케이션의 콜백 및 동기화

MQTT 클라이언트 프로그래밍 모델에서는 스레드를 광범위하게 사용합니다. 스레드는 MQTT 클라이언트 애플리케이션을 서버 간에 메시지 전송이 지연되는 것을 방지할 수 있는 만큼 분리합니다. 발행물, 전달 토큰 및 연결 손실 이벤트는 `MqttCallback`을 구현하는 콜백 클래스의 메소드로 전달됩니다.

콜백

`MqttCallback` 인터페이스에는 세 가지 콜백 메소드가 있습니다. Callback.java의 예 구현을 참조하십시오.

`connectionLost(java.lang.Throwable cause)`

`connectionLost`는 통신 오류로 연결이 중단된 경우에 호출됩니다. 또한 연결이 설정된 후에 서버에서 오류가 발생하여 서버가 연결을 중단한 경우에도 호출됩니다. 서버 오류는 큐 관리자 오류 로그에 로그됩니다. 서버는 클라이언트에 대한 연결을 중단하고 클라이언트는 `MqttCallback.connectionLost`를 호출합니다.

클라이언트 앱과 동일한 스레드에서 예외로 처리되는 원격 오류는 `MqttClient.connect`의 예외뿐입니다. 연결이 설정된 후 서버가 감지한 오류는 다시 `MqttCallback.connectionLost` 콜백 메소드에 `throwables`로 보고됩니다.

`connectionLost`가 발생하는 일반적인 서버 오류는 권한 부여 오류입니다. 예를 들어, 텔레메트리 서버가 토픽에 발행할 권한이 없는 클라이언트의 동작에서 토픽에 발행하려고 시도하는 경우입니다. 텔레메트리 서버로 `MQCC_FAIL` 조건 코드가 리턴되면 연결이 중단될 수 있습니다.

`deliveryComplete(MqttDeliveryToken token)`

MQTT 클라이언트가 `deliveryComplete` 를 호출하여 전달 토큰을 클라이언트 앱으로 다시 전달합니다. 123 페이지의 『전달 토큰』을 참조하십시오. 콜백은 전달 토큰을 사용하여 `token.getMessage` 메소드로 발행한 메시지에 액세스할 수 있습니다.

애플리케이션 콜백이 `deliveryComplete` 메소드에 의해 호출된 후 MQTT 클라이언트에 제어를 리턴하면 전달이 완료됩니다. 전달이 완료될 때까지 QoS 1 또는 2의 메시지는 지속 클래스에 보유됩니다.

deliveryComplete 호출은 애플리케이션과 지속 클래스 사이의 동기화 지점입니다. deliveryComplete 메소드는 동일한 메시지에 대해 절대 두 번 호출되지 않습니다.

애플리케이션 콜백이 deliveryComplete 에서 MQTT 클라이언트로 리턴하는 경우, 클라이언트는 QoS 1 또는 2의 메시지에 대해 MqttClientPersistence.remove 를 호출합니다. MqttClientPersistence.remove는 발행된 메시지의 로컬에 저장된 사본을 삭제합니다.

트랜잭션 처리 퍼스펙티브에서 deliveryComplete 호출은 전달을 커미트하는 단일 단계 트랜잭션입니다. 콜백 중에 처리가 실패하는 경우, 클라이언트를 다시 시작할 때 MqttClientPersistence.remove가 다시 호출되어 발행된 메시지의 로컬 저장 사본을 삭제합니다. 콜백은 다시 호출되지 않습니다. 전달된 메시지의 로그를 저장하기 위해 콜백을 사용하는 경우 MQTT 클라이언트와 로그를 동기화할 수 없습니다. 로그를 신뢰할 수 있도록 저장하려면 MqttClientPersistence 클래스의 로그를 업데이트하십시오.

전달 토큰 및 메시지는 기본 애플리케이션 스레드 및 MQTT 클라이언트에서 참조됩니다. MQTT 클라이언트는 전달이 완료되면 MqttMessage 오브젝트의 참조를 해제하며 클라이언트가 연결을 끊으면 전달 토큰 오브젝트의 참조를 해제합니다. 클라이언트 앱이 참조를 해제하는 경우 전달이 완료된 후 MqttMessage 오브젝트의 가비지 콜렉션이 수행될 수 있습니다. 전달 토큰은 세션 연결이 끊어진 후 가비지 콜렉션이 수행될 수 있습니다.

메시지가 발행된 후 MqttDeliveryToken 및 MqttMessage 속성을 가져올 수 있습니다. 메시지가 발행된 후 MqttMessage 속성을 설정하려고 시도하면 결과는 정의되지 않습니다.

MQTT 클라이언트가 동일한 ClientIdentifier를 가진 이전 세션에 다시 연결되면 이 클라이언트는 계속해서 전달 수신확인을 처리합니다(121 페이지의 『정리 세션』 참조). MQTT 클라이언트 애플리케이션은 이전 세션에 대해 MqttClient.CleanSession 를 false 로 설정하고 새 세션에서 false 로 설정해야 합니다. MQTT 클라이언트는 보류 중인 전달에 대한 새 세션에 새 전달 토큰 및 메시지 오브젝트를 작성합니다. 그러면 MqttClientPersistence 클래스를 사용하여 오브젝트가 복구됩니다. 애플리케이션 클라이언트에 여전히 이전 전달 토큰 및 메시지에 대한 참조가 있는 경우에는 해당 참조를 해제하십시오. 애플리케이션 콜백은 이전 세션에서 시작되고 이 세션에서 완료된 모든 전달에 대해 새 세션에서 호출됩니다.

애플리케이션 콜백은 애플리케이션 클라이언트가 연결된 후 보류 중인 전달이 완료될 때 호출됩니다. 애플리케이션 클라이언트는 연결하기 전에 MqttClient.getPendingDeliveryTokens 메소드를 사용하여 보류 중인 전달을 검색할 수 있습니다.

클라이언트 앱이 원래 발행된 메시지 오브젝트와 페이로드(payload) 바이트 배열을 작성했습니다. MQTT 클라이언트는 이러한 오브젝트를 참조합니다. token.getMessage 메소드에서 전달 토큰이 리턴한 메시지 오브젝트는 클라이언트가 작성한 동일한 메시지 오브젝트에서 필수가 아닙니다. 새 MQTT 클라이언트 인스턴스가 전달 토큰을 작성할 경우 MqttClientPersistence 클래스는 MqttMessage 오브젝트를 다시 작성합니다. token.isCompleted가 true인 경우, 일관성 token.getMessage는 메시지 오브젝트가 애플리케이션 클라이언트에서 작성되었는지 또는 MqttClientPersistence 클래스에서 작성되었는지에 관계없이 null을 리턴합니다.

messageArrived(MqttTopic topic, MqttMessage message)

messageArrived는 구독 토픽에 일치하는 발행물이 클라이언트에 도달하는 경우 호출됩니다. topic 은 구독 필터가 아니라 발행물 토픽입니다. 필터에 와일드카드가 포함된 경우 이 둘은 다를 수 있습니다. 토픽이 클라이언트가 작성한 여러 구독에 일치하는 경우, 클라이언트는 발행물의 여러 사본을 수신합니다. 클라이언트가 구독하기도 하는 토픽에 발행하는 경우에는 자신의 발행물 사본을 수신합니다.

메시지가 QoS 1 또는 2로 송신되는 경우 MQTT 클라이언트가 messageArrived를 호출하기 전에 MqttClientPersistence 클래스가 메시지를 저장합니다. messageArrived는 deliveryComplete처럼 작동합니다. 이는 발행을 위해 한 번만 호출되며 발행의 로컬 사본은 messageArrived가 MQTT 클라이언트로 리턴되면 MqttClientPersistence.remove에 의해 제거됩니다. MQTT 클라이언트는 messageArrived가 MQTT 클라이언트로 리턴되면 토픽 및 메시지에 대한 해당 참조를 삭제합니다. 애플리케이션 클라이언트가 오브젝트에 대한 참조를 보유하지 않는 경우, 토픽 및 메시지 오브젝트는 가비지 콜렉션됩니다.

콜백, 스레딩, 클라이언트 앱 동기화

MQTT 클라이언트는 별도의 스레드에서 콜백 메소드를 기본 애플리케이션 스레드로 호출합니다. 클라이언트 앱은 콜백에 대한 스레드를 작성하지 않습니다. 이는 MQTT 클라이언트가 작성합니다.

MQTT 클라이언트는 콜백 메소드를 동기화합니다. 한 번에 하나의 콜백 메소드만 실행됩니다. 동기화를 사용하면 전달된 발행물을 합산하는 오브젝트를 쉽게 업데이트할 수 있습니다. 한 번에 하나의 `MqttCallback.deliveryComplete` 인스턴스가 실행되므로 추가적인 동기화 없이 합산을 안전하게 업데이트할 수 있습니다. 한 번에 하나의 발행물만 도착하는 경우에도 해당합니다. `messageArrived` 메소드의 사용자 코드는 동기화하지 않고 오브젝트를 업데이트할 수 있습니다. 다른 스레드에서 합산 또는 업데이트 중인 오브젝트를 참조하는 경우에는 합산 또는 오브젝트를 동기화하십시오.

전달 토큰은 기본 애플리케이션 스레드와 발행물 전달 사이의 동기화 메커니즘을 제공합니다. `token.waitForCompletion` 메소드는 특정 발행물이 전달될 때까지 또는 선택적 제한시간이 만기될 때까지 대기합니다. `token.waitForCompletion`을 사용하여 몇 가지 간단한 방법으로 한 번에 하나의 발행물을 처리할 수 있습니다.

1. 발행물의 전달이 완료될 때까지 애플리케이션 클라이언트를 일시정지하려면 PubSync.java의 내용을 참조하십시오.
2. `MqttCallback.deliveryComplete` 메소드와 동기화하려면 다음을 수행하십시오. `MqttCallback.deliveryComplete`가 MQTT에 리턴될 경우에만 클라이언트가 `token.waitForCompletion`을 계속합니다. 기본 애플리케이션 스레드에서 코드를 실행하기 전에 이 메커니즘을 사용하여 `MqttCallback.deliveryComplete`의 실행 코드를 동기화할 수 있습니다.

각 발행물이 전달되기를 기다리지 않고 발행하기를 원하지만 모든 발행물이 전달되면 확인을 원하십니까? 단일 스레드에서 발행하는 경우 송신되는 마지막 발행물은 전달되는 마지막 발행물입니다.

서버로 송신된 요청 동기화

표 *4.* 서버에 요청하는 메소드 작동 동기화.

이 테이블은 서버에 요청을 보내는 MQTT Java 클라이언트의 메소드를 나열합니다. 각 메소드마다 메소드가 대기 또는 리턴하는 조건과 메소드의 대기 시간이 표에 설명되어 있습니다.

방법	동기화	제한시간 간격
`MqttClient.Connect`	서버에 대해 연결이 설정되기를 기다립니다.	기본값은 30초이며 매개변수를 통해 설정됩니다.
`MqttClient.Disconnect`	MQTT 클라이언트가 수행해야 하는 모든 작업을 마치고 TCP/IP 세션의 연결을 끊을 때까지 기다리십시오.	기본값은 30초이며 매개변수를 통해 설정됩니다.
`MqttClient.Subscribe`	구독 요청이 완료될 때까지 대기합니다.	기본값은 30초이며 매개변수를 통해 설정됩니다.
`MqttClient.UnSubscribe`	구독 해제 요청이 완료될 때까지 대기합니다.	기본값은 30초이며 매개변수를 통해 설정됩니다.
`MqttClient.Publish`	MQTT 클라이언트에 요청을 전달한 후에 애플리케이션 스레드에 즉시 리턴됩니다.	없음
`MqttDeliveryToken.waitForCompletion`	전달 토큰이 리턴될 때까지 기다립니다.	기본값은 무한이며 매개변수를 통해 설정됩니다.

정리 세션

MQTT 클라이언트와 텔레메트리(MQXR) 서비스는 세션 상태 정보를 유지합니다. 상태 정보는 "적어도 한 번"과 "정확히 한 번"의 전달, 발행의 "정확히 한 번" 수신을 보장하는 데 사용됩니다. 세션 상태는 MQTT 클라이언트에 의해 작성된 구독도 포함합니다. MQTT 클라이언트를 실행하면서 세션 간에 상태 정보를 유지하거나 유지하지

않도록 선택할 수 있습니다. 연결하기 전에 MqttConnectOptions.cleanSession 를 설정하여 정리 세션 모드를 변경하십시오.

MqttClient.connect 메소드를 사용하여 MQTT 클라이언트 앱을 연결하는 경우 클라이언트는 클라이언트 ID와 서버의 주소를 사용하여 연결을 식별합니다. 서버는 이전에 서버에 연결된 세션의 정보가 저장되었는지 확인합니다. 이전 세션이 여전히 있으며 cleanSession=true인 경우, 클라이언트와 서버에서 이전 세션 정보가 삭제됩니다. cleanSession=false인 경우에는 이전 세션이 재개됩니다. 이전 세션이 없는 경우에는 새 세션이 시작됩니다.

참고: WebSphere MQ Administrator는 열린 세션을 강제로 닫고 모든 세션 정보를 삭제할 수 있습니다. 클라이언트가 세션을 cleanSession=false로 다시 여는 경우에는 새 세션이 시작됩니다.

발행물

기본 MqttConnectOptions를 사용하거나 클라이언트에 연결하기 전에 MqttConnectOptions.cleanSession을 true로 설정한 경우에는 클라이언트가 연결할 때 해당 클라이언트에 대해 보류 중인 모든 발행물 전달이 제거됩니다.

정리 세션 설정은 QoS=0으로 송신된 발행물에는 아무런 영향을 미치지 않습니다. QoS=1 및 QoS=2의 경우 cleanSession=true를 사용하면 발행물이 손실될 수 있습니다.

구독

클라이언트를 연결하기 전에 기본 MqttConnectOptions를 사용하거나 MqttConnectOptions.cleanSession 를 true 로 설정하면 클라이언트가 연결될 때 클라이언트에 대한 이전 등록이 제거됩니다. 세션 중에 클라이언트가 만드는 모든 새 구독은 연결을 해제할 때 삭제됩니다.

연결하기 전에 MqttConnectOptions.cleanSession 를 false 로 설정하면 클라이언트가 작성하는 모든 구독이 연결되기 전에 클라이언트에 대해 존재하는 모든 구독에 추가됩니다. 클라이언트가 연결이 끊길 때 모든 구독은 활성인 상태로 남아 있습니다.

cleanSession 속성이 구독에 영향을 미치는 방식을 이해하는 다른 방법은 모달 속성으로 간주하는 것입니다. 기본 모드 cleanSession=true에서 클라이언트를 구독을 작성하고 세션의 범위 내에서만 발행물을 수신합니다. 대체 모드 cleanSession=false에서 구독은 지속됩니다. 클라이언트는 연결하거나 연결을 끊을 수 있고 해당 구독은 활성인 상태로 남아 있습니다. 클라이언트가 다시 연결하면 전달되지 않은 모든 발행물을 수신합니다. 연결된 동안 대신 활성인 구독 세트를 수정할 수 있습니다.

연결하기 전에 cleanSession 모드를 설정해야 합니다. 모드는 전체 세션 동안 지속됩니다. 설정을 변경하려면 클라이언트의 연결을 끊은 후 다시 연결해야 합니다. 모드를 cleanSession=false 사용에서 cleanSession=true로 변경하는 경우, 클라이언트에 대한 모든 이전 등록 및 수신되지 않은 발행물을 버립니다.

클라이언트 ID

클라이언트 ID는 MQTT 클라이언트를 식별하는 23바이트문자열입니다. 클라이언트 ID는 서버에 연결되는 모든 클라이언트에서 고유해야 하며 서버의 큐 관리자 이름과 동일해서는 안됩니다. 이러한 제한조건 내에서 모든 ID 문자열을 사용할 수 있습니다. 클라이언트 ID를 할당하는 프로시저와 선택한 ID로 클라이언트를 구성할 수단이 있는 것이 중요합니다.

클라이언트 ID는 MQTT 시스템 관리에 사용됩니다. 관리해야 하는 클라이언트가 수십만에 이를 수 있는 만큼 특정 클라이언트를 빠르게 식별할 수 있어야 합니다. 예를 들어, 디바이스에 기능 결함이 있고 문의 데스크에 전화한 고객이 이를 알려왔다고 가정합니다. 고객이 디바이스를 식별하는 방법과 일반적으로 클라이언트에 연결된 서버와 ID를 상관하는 방법은 무엇입니까? 각 디바이스를 클라이언트 ID 및 서버에 맵핑하는 데이터베이스를 참조해야 합니까? 디바이스 이름을 통해 연결된 서버를 식별할 수 있습니까? MQTT 클라이언트 연결을 통해 찾아볼 때 각 연결은 클라이언트 ID로 레이블이 지정됩니다. 클라이언트 ID를 물리적 디바이스에 맵핑하기 위해 테이블을 검색해야 합니까?

클라이언트 ID로 특정 디바이스, 사용자 또는 클라이언트에서 실행 중인 애플리케이션을 식별할 수 있습니까? 고객이 결함이 있는 디바이스를 새 디바이스로 바꾸는 경우 새 디바이스의 ID는 이전 디바이스의 ID와 동일합니

까? 새 ID를 할당하십니까? 물리적 디바이스를 변경하지만 동일한 ID를 유지하는 경우 미해결 발행물 및 활성 구독은 자동으로 새 디바이스로 전송됩니다.

클라이언트 ID가 고유한지 확인하는 방법은 무엇입니까? 고유한 ID를 생성하는 시스템 외에도 클라이언트에 ID를 설정하는 안정적인 프로세스가 있어야 합니다. 클라이언트 디바이스는 사용자 인터페이스가 없는 "블랙박스"입니다. 클라이언트 ID가 있는 디바이스(예: MAC 주소 사용 등)를 제조합니까? 또는 활성화하기 전에 디바이스를 구성하는 소프트웨어 설치 및 구성 프로세스가 있습니까?

48비트 디바이스 MAC 프로세스에서 클라이언트 ID를 작성하여 ID를 간단하고 고유하게 유지할 수 있습니다. 전송 크기가 크게 문제가 되지 않는 경우에는 나머지 17바이트를 사용하여 주소를 관리하기 쉽게 만들 수 있습니다.

전달 토큰

클라이언트가 토픽에 대해 발행하면 새 전달 토큰이 작성됩니다. 발행물의 전달을 모니터하거나 전달이 완료될 때까지 클라이언트 앱을 차단하려면 전달 토큰을 사용하십시오.

토큰은 `MqttDeliveryToken` 오브젝트입니다. MqttTopic.publish() 메소드를 호출함으로써 작성되며 클라이언트 세션 연결이 끊기고 전달이 완료될때까지 MQTT 클라이언트가 보유합니다.

토큰은 일반적으로 전달이 완료되었는지 확인하는 데 사용합니다. 리턴된 토큰을 사용해서 `token.waitForCompletion`을 호출하여 전달이 완료될 때까지 클라이언트 앱을 차단하십시오. 또는 `MqttCallBack` 핸들러를 제공하십시오. 발행 전달의 한 부분으로서 MQTT 클라이언트가 기대하고 있던 모든 수신확인을 수신하면 전달 토큰을 매개변수로 전달하면서 `MqttCallBack.deliveryComplete`를 호출합니다.

전달이 완료될 때까지 `token.getMessage`를 호출하여 리턴된 전달 토큰을 통해 발행물을 조사할 수 있습니다.

완료된 전달

전달 완료는 비동기식이며 발행과 연관된 서비스의 질에 따라 다릅니다.

최대 한 번

`QoS=0`
전달은 `MqttTopic.publish`에서 리턴되는 즉시 완료됩니다.
`MqttCallback.deliveryComplete`가 즉시 호출됩니다.

최소 한 번

`QoS=1`
큐 관리자로부터 발행에 대한 수신확인이 수신되면 전달은 완료됩니다. 수신확인이 수신되면 `MqttCallback.deliveryComplete`가 호출됩니다. 통신이 느리거나 불안한 경우 `MqttCallback.deliveryComplete`가 호출되기 전에 이 메시지가 두 번 이상 전달될 수 있습니다.

정확히 한 번

`QoS=2`
발행물이 구독자에게 발행되었다는 완료 메시지를 클라이언트가 수신하면 전달이 완료됩니다. `MqttCallback.deliveryComplete`는 발행 메시지가 수신되는 즉시 호출됩니다. 완료 메시지를 기다리지 않습니다.

드물게 클라이언트 앱이 `MqttCallback.deliveryComplete` 에서 MQTT 클라이언트로 정상적으로 리턴되지 않을 수 있습니다. `MqttCallback.deliveryComplete`가 호출되었으므로 전달이 완료된 것을 알 수 있습니다. 클라이언트가 동일한 세션을 다시 시작하는 경우 `MqttCallback.deliveryComplete`는 다시 호출되지 않습니다.

완료되지 않은 전달

클라이언트 세션 연결이 끊긴 후에 전달이 완료되지 않으면 클라이언트를 다시 연결해 전달을 완료할 수 있습니다. `MqttConnectionOptions` 속성이 `false`로 설정된 세션에서 발행된 메시지만 메시지 전달을 완료할 수 있습니다.

동일한 클라이언트 ID와 서버 주소를 사용하여 클라이언트를 작성한 다음 `cleanSession` `MqttConnectionOptions` 속성을 `false`로 설정하여 다시 연결하십시오. `cleanSession`을 `true`로 설정하면 보류 중인 전달 토큰은 삭제됩니다.

`MqttClient.getPendingDeliveryTokens`를 호출하여 보류 중인 전달이 있는지 검사할 수 있습니다. 클라이언트를 연결하기 전에 `MqttClient.getPendingDeliveryTokens`를 호출할 수 있습니다.

이상 종료 시 메시지 발행

MQTT 클라이언트 연결이 예기치 않게 종료되는 경우 WebSphere MQ 텔레메트리를 구성하여 "마지막 유언장" 발행을 전송할 수 있습니다. 발행물의 컨텐츠 및 송신할 토픽을 사전정의하십시오. "이상 종료 시 메시지"는 연결 특성입니다. 클라이언트를 연결하기 전에 작성하십시오.

이상 종료 시 메시지의 토픽을 작성하십시오. `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`와 같은 주제를 작성할 수 있습니다.

`MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` 메소드를 사용하여 "마지막 유언장" 을 설정하십시오.

`lastWillPayload` 메시지에 시간소인을 작성하는 것을 고려하십시오. 클라이언트 및 연결 상황을 식별하는 데 도움이 되는 기타 클라이언트 정보를 포함시키십시오. `MqttConnectionOptions` 오브젝트를 `MqttClient` 구성자로 전달하십시오.

Set `lastWillQos` to 1 or 2, to make the message persistent in IBM WebSphere MQ, and to ensure delivery. 마지막으로 손실된 연결 정보를 보유하려면 `lastWillRetained`를 `true`로 설정하십시오.

예상치 못하게 연결이 종료될 경우 "이상 종료 시 메시지" 발행이 구독자에게 전송됩니다. 이 발행물은 클라이언트가 `MqttClient.disconnect` 메소드를 호출하지 않은 채로 연결이 종료되는 경우에 송신됩니다.

연결을 모니터하려면 연결과 계획된 연결 종료를 기록하기 위한 기타 발행으로 "이상 종료 시 메시지" 발행을 보완하십시오.

MQTT 클라이언트의 메시지 지속성

발행 메시지는 "최소 한 번" 또는 "정확히 한 번"의 QoS(Quality of Service)로 송신되는 경우 지속됩니다. 클라이언트에 사용자 고유의 지속 메커니즘을 구현할 수도 있고 클라이언트가 제공하는 기본 지속 메커니즘을 사용할 수도 있습니다. 지속성은 클라이언트로 송신되거나 클라이언트에서 수신된 발행에 대해 양방향으로 작용합니다.

MQTT에서 메시지 지속성에는 메시지가 전송되는 방법과 메시지가 지속 메시지로 MQTT 서버에서 큐에 대기되는지 여부라는 두 가지 측면이 있습니다.

1. MQTT 클라이언트는 메시지 지속성과 서비스 품질을 결합시킵니다. 메시지에 대해 선택하는 서비스 품질(QoS)에 따라 메시지가 지속됩니다. 메시지 지속성은 요구되는 서비스 품질을 구현하기 위해서 필요합니다.

   "최대 한 번", `QoS=0`을 지정한 경우 클라이언트는 메시지가 발행되는 즉시 이를 제거합니다. 메시지의 업스트림 처리에 장애가 있는 경우 메시지는 다시 송신되지 않습니다. 클라이언트가 활성 상태로 남아 있는 경우에도 메시지는 다시 송신되지 않습니다. `QoS=0` 메시지의 동작은 IBM WebSphere MQ 빠른 비지속 메시지와 동일합니다.

   QoS 1 또는 2로 클라이언트가 메시지를 발행하는 경우에는 지속됩니다. 메시지는 로컬로 저장되며 더 이상 "적어도 한 번"(QoS=1) 또는 "정확히 한 번"(QoS=2) 전달되도록 보장할 필요가 없는 경우에만 클라이언트에서 제거됩니다.
2. 메시지가 QoS 1 또는 2로 표시된 경우 메시지는 지속 메시지로 큐에 대기됩니다. 메시지가 `QoS=0`으로 표시된 경우에는 비지속 메시지로 큐에 대기됩니다. 메시지 채널이 `FAST`로 설정된 `NPMSPEED` 속성을 갖지 않는 한, IBM WebSphere MQ 비지속 메시지는 "정확히 한 번" 큐 관리자 사이에 전송됩니다.

지속적 발행물은 클라이언트 앱이 이를 수신할 때까지 클라이언트에 저장됩니다. `QoS=2`의 경우 애플리케이션 콜백이 제어를 리턴하면 발행물이 클라이언트로부터 제거됩니다. `QoS=1`의 경우 애플리케이션은 장애 발생 시 발행물을 다시 수신할 수 있습니다. `QoS=0`의 경우 콜백은 발행을 한 번 이상 수신하지 않습니다. 장애가 발생한 경우나 발행될 때 클라이언트의 연결이 끊기는 경우 발행을 수신하지 않을 수 있습니다.

토픽을 구독하는 경우 구독자가 지속 용량에 일치하는 메시지를 수신하도록 QoS를 줄일 수 있습니다. 높은 QoS로 작성된 발행은 구독자가 요구하는 가장 높은 QoS로 송신됩니다.

메시지 저장

작은 디바이스에서 데이터 스토리지 구현은 매우 다양합니다. MQTT 클라이언트가 관리하는 스토리지에 지속 메시지를 임시로 저장하는 모델은 지나치게 느리거나 지나치게 많은 스토리지를 요구할 수 있습니다. 모바일 디바이스에서 모바일 운영 체제는 MQTT 메시지에 이상적인 스토리지 서비스를 제공할 수 있습니다.

소형 디바이스의 제한조건을 만족시키는 유연성을 제공하기 위해 MQTT 클라이언트에는 두 가지 지속 인터페이스가 있습니다. 이러한 인터페이스는 지속 메시지 저장을 포함하는 조작을 정의합니다. 이 인터페이스는 Java용 MQTT 클라이언트에 대한 API 문서에 설명되어 있습니다. MQTT 클라이언트 라이브러리에 대한 클라이언트 API 문서에 대한 링크는 MQTT 클라이언트 프로그래밍 참조를 참조하십시오. 사용자는 디바이스에 맞춰 인터페이스를 구현할 수 있습니다. Java SE에서 실행되는 MQTT 클라이언트에는 파일 시스템에 지속 메시지를 저장하는 인터페이스의 기본 구현이 있습니다. 이 구현에서는 `java.io` 패키지를 사용합니다. 또한 클라이언트에는 Java ME에 대한 기본 구현인 `MqttDefaultMIDPPersistence`가 있습니다.

지속 클래스

`MqttClientPersistence`
: `MqttClientPersistence` 사용자 구현의 인스턴스를 `MqttClient` 생성자의 매개변수로서 MQTT 클라이언트에 전달합니다. `MqttClient` 생성자에서 `MqttClientPersistence` 매개변수를 생략할 경우 MQTT 클라이언트는 클래스 `MqttDefaultFilePersistence` 또는 `MqttDefaultMIDPPersistence`를 사용하여 지속 메시지를 저장합니다.

`MqttPersistable`
: `MqttClientPersistence`는 스토리지 키를 사용하여 `MqttPersistable` 오브젝트를 가져오고 넣습니다. `MqttDefaultFilePersistence` 또는 `MqttDefaultMIDPPersistence`를 사용하지 않는 경우에는 `MqttClientPersistence` 구현과 함께 `MqttPersistable` 구현도 제공해야 합니다.

`MqttDefaultFilePersistence`

: MQTT 클라이언트는 `MqttDefaultFilePersistence` 클래스를 제공합니다. 클라이언트 앱에서 `MqttDefaultFilePersistence`를 인스턴스화하는 경우 지속 메시지를 `MqttDefaultFilePersistence` 생성자의 매개변수로 저장할 디렉토리를 제공할 수 있습니다.

: 또는 MQTT 클라이언트는 `MqttDefaultFilePersistence`를 인스턴스화하고 기본 디렉토리에 파일을 저장할 수 있습니다. 디렉토리 이름은 *`client identifier`*`-tcp `*`hostname portnumber`*입니다. `"\", "\\", "/", ":"` 및 `" "`가 디렉토리 이름 문자열에서 제거됩니다.

: 디렉토리의 경로는 시스템 특성 `rcp.data`의 값입니다. `rcp.data`가 설정되지 않은 경우, 경로는 시스템 특성 `usr.data`의 값입니다.

: `rcp.data`는 OSGi 또는 Eclipse 서식있는 클라이언트 플랫폼(RCP)의 설치와 연관된 특성입니다.

: `usr.data` 는 애플리케이션을 시작한 Java 명령이 실행된 디렉토리입니다.

`MqttDefaultMIDPPersistence`
: `MqttDefaultMIDPPersistence`에는 기본 구성자가 있으며 매개변수가 없습니다. `javax.microedition.rms.RecordStore` 패키지를 사용하여 메시지를 저장합니다.

발행물

발행물은 토픽 문자열과 연관된 `MqttMessage` 의 인스턴스입니다. MQTT 클라이언트는 IBM WebSphere MQ에 송신할 발행물을 작성하고 발행물을 수신할 IBM WebSphere MQ MQ의 토픽을 구독할 수 있습니다.

`MqttMessage`에는 페이로드(payload)로 바이트 배열이 있습니다. 메시지를 가능한 한 작게 유지하십시오. MQTT 프로토콜에서 허용하는 최대 메시지 길이는 250MB입니다.

일반적으로 MQTT 클라이언트 프로그램은 `java.lang.String` 또는 `java.lang.StringBuffer`를 사용하여 메시지 컨텐츠를 조정합니다. 편의를 위해 `MqttMessage` 클래스에는 페이로드를 문자열로 변환하는

toString 메소드가 있습니다. java.lang.String 또는 java.lang.StringBuffer에서 바이트 배열 페이로드를 작성하려면 getBytes 메소드를 사용하십시오.

getBytes 메소드는 문자열을 해당 플랫폼의 기본 문자 세트로 변환합니다. 기본 문자 세트는 일반적으로 UTF-8입니다. 텍스트만 포함된 MQTT 발행물은 보통 UTF-8로 인코딩됩니다. getBytes("UTF8") 메소드를 사용하여 기본 문자 세트를 대체하십시오.

IBM WebSphere MQ에서는 MQTT 발행물이 jms-bytes 메시지로 수신됩니다. 메시지는 <mqtt> 및 <mqps> 폴더가 포함된 MQRFH2 폴더를 포함합니다. <mqtt> 폴더는 clientId 및 qos를 포함하지만 이 컨텐츠는 나중에 변경할 수 있습니다.

MqttMessage에는 QoS(quality of service), 보유 여부 및 중복 여부의 세 가지 추가 속성이 있습니다. 중복 플래그는 서비스 품질(QoS)이 "최소 한 번" 또는 "정확히 한 번"인 경우에만 설정됩니다. 메시지가 이전에 송신되었으나 MQTT 클라이언트에 의해 신속하게 수신확인되지 않은 경우 중복 속성이 true로 설정된 채 메시지가 다시 송신됩니다.

발행

MQTT 클라이언트 앱에서 공개를 작성하려면 MqttMessage를 작성하십시오. 해당 페이로드, 서비스 품질 및 보유 여부를 설정하고 MqttTopic.publish(MqttMessage message) 메소드를 호출하십시오. MqttDeliveryToken이 리턴되며 발행 완료는 비동기입니다.

또는 MQTT 클라이언트는 발행물을 작성할 때 MqttTopic.publish(byte [] payload, int qos, boolean retained) 메소드의 매개변수에서 사용자의 임시 메시지 오브젝트를 작성할 수 있습니다.

발행의 QoS(Quality of Service)가 "최소 한 번" 또는 "정확히 한 번", QoS=1 또는 QoS=2인 경우, MQTT 클라이언트는 MqttClientPersistence 인터페이스를 호출합니다. MqttClientPersistence를 호출하면 전달 토큰을 애플리케이션에 전달하기 전에 메시지가 저장됩니다.

애플리케이션은 메시지가 서버에 전달될 때까지 MqttDeliveryToken.waitForCompletion 메소드를 사용하여 블록을 선택할 수 있습니다. 또는 애플리케이션이 블로킹하지 않고 계속 진행할 수도 있습니다. 발행물이 블로킹 없이 전달되는지 검사하려는 경우 MQTT 클라이언트로 MqttCallback을 구현하는 콜백 클래스의 인스턴스를 등록하십시오. MQTT 클라이언트는 발행이 전달되면 즉시 MqttCallback.deliveryComplete 메소드를 호출합니다. 서비스 품질(QoS)에 따라 QoS=0인 경우에는 거의 즉시 전달되며 QoS=2인 경우에는 시간이 걸릴 수 있습니다.

전달이 완료되면 MqttDeliveryToken.isComplete 메소드를 사용하여 폴링하십시오. MqttDeliveryToken.isComplete의 값이 false이면 MqttDeliveryToken.getMessage를 호출하여 메시지 컨텐츠를 가져올 수 있습니다. MqttDeliveryToken.isComplete를 호출한 결과가 true이면 메시지가 제거되었으며 MqttDeliveryToken.getMessage를 호출하면 널 포인터 예외가 전달됩니다. MqttDeliveryToken.getMessage와 MqttDeliveryToken.isComplete 사이에는 내장 동기화가 없습니다.

보류 중인 전달 토큰을 모두 수신하기 전에 클라이언트가 연결을 끊는 경우, 연결하기 전에 클라이언트의 새 인스턴스가 보류 중인 전달 토큰을 조회할 수 있습니다. 클라이언트가 연결할 때까지 새 전달은 완료되지 않으므로 MqttDeliveryToken.getMessage를 호출하는 것이 안전합니다. MqttDeliveryToken.getMessage 메소드를 사용하여 전달되지 않은 발행물을 알아내십시오. MqttConnectOptions.cleanSession을 기본값인 true로 설정하여 연결하는 경우에는 보류 중인 전달 토큰이 제거됩니다.

구독

큐 관리자 또는 IBM MessageSight가 MQTT 구독자에게 송신할 발행물을 작성하는 작업을 담당합니다. 큐 관리자는 MQTT 클라이언트에 의해 작성된 구독의 토픽 필터가 발행의 토픽 문자열과 일치하는지 검사합니다. 일치는 정확한 일치일 수도 있고 일치에 와일드카드가 포함될 수도 있습니다. 큐 관리자는 발행물을 구독자로 전달하기 전에 발행물과 연관된 토픽 속성을 확인합니다. 또한 관리 토픽 오브젝트가 사용자에게 구독 권한을 부여하는지 식별하기 위해 와일드카드 문자가 포함된 토픽 문자열을 사용한 구독에 설명된 검색 프로시저를 수행합니다.

MQTT 클라이언트가 "최소 한 번" 서비스 품질인 발행을 수신하면 MqttCallback.messageArrived 메소드를 호출하여 발행을 처리합니다. 발행의 서비스 품질이 "정확히 한 번" QoS=2인 경우 MQTT 클라이언트는 메시

지 수신 시 메시지를 저장하기 위해 MqttClientPersistence 인터페이스를 호출합니다. 그런 다음 MqttCallback.messageArrived를 호출합니다.

MQTT 클라이언트에서 제공하는 서비스 품질(QoS)

MQTT 클라이언트는 WebSphere MQ 및 MQTT 클라이언트 ("at most once", "at least once" and "정확히 한 번") 에 서적을 전달하기 위한 세 가지 서비스 품질을 제공합니다. MQTT 클라이언트가 IBM WebSphere MQ에 요청을 송신하여 구독을 작성하는 경우 "최소 한 번" 서비스 품질(QoS)을 사용하여 요청이 송신됩니다.

발행의 서비스 품질은 MqttMessage의 속성입니다. 이 속성은 MqttMessage.setQos 메소드로 설정됩니다.

메소드 MqttClient.subscribe는 토픽에서 클라이언트에 송신된 발행에 적용된 서비스 품질을 낮출 수 있습니다. 구독자에게 전달된 발행의 서비스 품질은 발행의 서비스 품질에 따라 달라질 수 있습니다. 두 값 중 낮은 값이 발행물을 전달하는 데 사용됩니다.

최대 한 번
QoS=0

메시지는 최대 한 번 전달되거나 전혀 전달되지 않습니다. 이 네트워크 간 전달은 수신확인되지 않습니다. 메시지는 저장되지 않습니다. 클라이언트 연결이 끊어지거나 서버가 실패하는 경우 메시지는 손실될 수 있습니다.
QoS=0은 가장 빠른 전송 모드입니다. 이 모드는 "실행 후 삭제"라고도 합니다.
MQTT 프로토콜은 서버가 QoS=0인 발행을 클라이언트로 전달하도록 요구하지 않습니다. 서버가 발행물을 수신할 때 클라이언트 연결이 끊어지는 경우 서버에 따라 발행물을 제거할 수 있습니다. 텔레메트리 (MQXR) 서비스에서는 QoS=0으로 송신된 메시지를 제거하지 않습니다. 해당 메시지는 비지속 메시지로 저장되고 큐 관리자가 중지되는 경우에만 제거됩니다.

최소 한 번
QoS=1

QoS=1은 기본 전송 모드입니다.
메시지는 항상 최소 한 번 전달됩니다. 송신자가 수신확인을 수신하지 않는 경우, 메시지는 수신확인이 수신될 때까지 DUP 플래그가 설정되어 다시 송신됩니다. 따라서 수신자는 동일한 메시지를 여러 번 수신하여 여러 번 처리할 수도 있습니다.
메시지가 처리될 때까지 송신자와 수신자는 메시지를 로컬에 저장해야 합니다.
메시지는 처리된 후에 수신자로부터 삭제됩니다. 수신자가 브로커인 경우, 메시지는 구독자에게 발행됩니다. 수신자가 클라이언트인 경우 메시지는 구독자 애플리케이션에게로 전달됩니다. 메시지가 삭제된 후 수신자는 송신자에게 수신확인을 송신합니다.
수신자로부터 수신확인을 받고 나면 송신자로부터 메시지가 삭제됩니다.

정확히 한 번
QoS=2

메시지는 항상 정확히 한 번만 전송됩니다.
메시지가 처리될 때까지 송신자와 수신자는 메시지를 로컬에 저장해야 합니다.
QoS=2는 가장 안전하지만 가장 느린 전송 모드입니다. 메시지가 송신자에서 삭제되기 전에 송신자와 수신자 사이에 최소 두 쌍의 전송이 발생합니다. 메시지는 첫 번째 전송 후에 수신자 측에서 처리될 수 있습니다.
첫 번째 전송 쌍에서 송신자는 메시지를 전송하고 수신자에게서 메시지를 저장했다는 수신확인을 받습니다. 송신자가 수신확인을 수신하지 않는 경우, 메시지는 수신확인이 수신될 때까지 DUP 플래그가 설정되어 다시 송신됩니다.
두 번째 전송 쌍에서 송신자는 수신자에게 메시지 "PUBREL"의 처리를 완료할 수 있다고 전달합니다. 송신자가 "PUBREL" 메시지에 대한 수신확인을 수신하지 않은 경우 수신확인이 수신될 때까지 "PUBREL" 메시지가 다시 송신됩니다. 송신자는 "PUBREL" 메시지에 대한 수신확인을 수신하면 저장했던 메시지를 삭제합니다.
메시지를 다시 처리하지 않는다는 가정 하에 수신자가 첫 번째 또는 두 번째 단계에서 메시지를 처리할 수 있습니다. 수신자가 브로커인 경우 이는 메시지를 구독자에게 발행합니다. 수신자가 클라이언트인 경우 메시지는 구독자 애플리케이션으로 전달됩니다. 수신자는 송신자에게 메시지 처리가 완료되었다는 완료 메시지를 송신합니다.

보유된 발행 및 MQTT 클라이언트

보유된 발행이 있는 토픽에 대한 구독을 작성하는 경우 토픽의 최신 보유된 발행이 즉시 사용자에게 전달됩니다.

MqttMessage.setRetained 메소드를 사용하여 토픽에 대한 발행물의 보유 여부를 지정하십시오.

IBM WebSphere MQ에서 보유된 발행물을 삭제하려면 CLEAR TOPICSTR**CLEAR TOPICSTR** MQSC 명령을 실행하십시오.

널 페이로드(payload)로 발행을 작성하는 경우 비어 있는 발행이 구독자에게 전달됩니다. 다른 MQTT 브로커는 비어 있는 발행을 구독자에게 전달하지 않을 수도 있습니다.

보유되지 않은 발행물을 토픽에 발행하는 경우 보유된 발행믈은 영향을 받지 않습니다. 현재 구독자는 새로운 발행물을 수신합니다. 새 구독자는 보유된 발행물을 먼저 수신하고 새 발행물을 수신합니다.

보유된 발행을 작성하거나 갱신할 때 QoS 또는 1또는 2로 발행물을 송신하십시오. QoS 0으로 전송하는 경우 IBM WebSphere MQ은(는) 비지속적 보유된 발행물을 작성합니다. 큐 관리자가 중지되는 경우 발행물이 보유되지 않습니다.

보유된 발행물을 사용하여 마지막 측정치를 기록하십시오. 보유된 토픽에 대한 새 구독자는 즉시 최신 측정치를 수신합니다. 구독자가 마지막으로 발행 토픽을 구독한 이후로 새 측정치가 측정되지 않은 경우 및 구독자가 다시 구독하는 경우, 구독자가 토픽에 대해 최신으로 보유된 발행물을 다시 수신합니다.

구독

토픽 필터를 사용하여 관심있는 발행 토픽을 등록하려면 구독을 작성하십시오. 클라이언트는 다중 구독 또는 와일드카드를 사용하는 토픽 필터를 포함하는 구독을 작성하여 관심있는 다중 토픽을 등록할 수 있습니다. 필터와 일치하는 토픽의 발행물이 클라이언트로 송신됩니다. 구독은 클라이언트 연결이 끊긴 중에도 활성 상태로 남아 있을 수 있습니다. 발행물은 클라이언트가 다시 연결할 때 여기에 송신됩니다.

하나 이상의 토픽 필터와 서비스 품질(QoS) 매개변수를 전달하는 MqttClient.subscribe 메소드를 사용하여 구독을 작성하십시오. 서비스 품질 매개변수는 메시지를 수신하는 데 구독자가 사용할 준비가 된 최대 서비스 품질을 설정합니다. 이 클라이언트에 송신된 메시지는 이보다 더 높은 서비스 품질로 전달될 수 없습니다. 메시지가 발행되고 구독의 레벨이 지정될 때 서비스 품질은 원래 값이나 이보다 낮은 값으로 설정됩니다. 수신되는 메시지에 대한 기본 서비스 품질은 QoS=1(최소 한 번)입니다.

구독 요청 자체는 QoS=1로 송신됩니다.

구독자는 MQTT 클라이언트가 MqttCallback.messageArrived 메소드를 호출할 때 발행물을 수신합니다. messageArrived 메소드는 발행된 메시지와 함께 토픽 문자열도 구독자에게 전달합니다.

MqttClient.unsubscribe 메소드를 사용하여 구독을 제거하거나 구독을 설정할 수 있습니다.

WebSphere MQ 명령은 구독을 삭제할 수 있습니다. WebSphere MQ 탐색기를 사용하거나 **runmqsc** 또는 PCF 명령을 사용하여 등록을 나열하십시오. 모든 MQTT 클라이언트 구독은 이름이 지정되어 있습니다. 이름에는 다음 양식의 이름이 지정됩니다. *ClientIdentifier*:*Topic name*

클라이언트를 연결하기 전에 기본 MqttConnectOptions를 사용하거나 MqttConnectOptions.cleanSession 를 true 로 설정하면 클라이언트가 연결될 때 클라이언트에 대한 이전 등록이 제거됩니다. 세션 중에 클라이언트가 만드는 모든 새 구독은 연결을 해제할 때 삭제됩니다.

연결하기 전에 MqttConnectOptions.cleanSession 를 false 로 설정하면 클라이언트가 작성하는 모든 구독이 연결되기 전에 클라이언트에 대해 존재하는 모든 구독에 추가됩니다. 클라이언트가 연결이 끊길 때 모든 구독은 활성인 상태로 남아 있습니다.

cleanSession 속성이 구독에 영향을 미치는 방식을 이해하는 다른 방법은 모달 속성으로 간주하는 것입니다. 기본 모드 cleanSession=true에서 클라이언트를 구독을 작성하고 세션의 범위 내에서만 발행물을 수신합니다. 대체 모드 cleanSession=false에서 구독은 지속됩니다. 클라이언트는 연결하거나 연결을 끊을 수 있고 해당 구독은 활성인 상태로 남아 있습니다. 클라이언트가 다시 연결하면 전달되지 않은 모든 발행물을 수신합니다. 연결된 동안 대신 활성인 구독 세트를 수정할 수 있습니다.

연결하기 전에 cleanSession 모드를 설정해야 합니다. 모드는 전체 세션 동안 지속됩니다. 설정을 변경하려면 클라이언트의 연결을 끊은 후 다시 연결해야 합니다. 모드를 cleanSession=false 사용에서

`cleanSession=true`로 변경하는 경우, 클라이언트에 대한 모든 이전 등록 및 수신되지 않은 발행물을 버립니다.

활성 구독과 일치하는 발행물은 발행되는 즉시 클라이언트로 송신됩니다. 클라이언트의 연결이 끊어지면 `MqttConnectOptions.cleanSession`가 `false`로 설정된 경우 동일한 클라이언트 ID로 동일한 서버에 다시 연결될 때 클라이언트로 송신됩니다.

특정 클라이언트에 대한 구독은 클라이언트 ID로 식별됩니다. 다른 클라이언트 디바이스에서 같은 서버에 클라이언트를 다시 연결해 같은 구독을 계속하여 전달되지 않은 발행을 수신할 수 있습니다.

MQTT 클라이언트의 토픽 문자열 및 토픽 필터

토픽 문자열 및 토픽 필터는 발행과 구독에 사용됩니다. MQTT 클라이언트의 토픽 문자열 및 필터 구문은 IBM WebSphere MQ의 토픽 문자열과 거의 동일합니다.

토픽 문자열은 발행물을 구독자에게 송신하는 데 사용됩니다. `MqttClient.getTopic(java.lang.String topicString)` 메소드를 사용하여 토픽 문자열을 작성하십시오.

토픽 필터는 토픽을 구독하고 발행물을 수신하는 데 사용됩니다. 토픽 필터에는 와일드카드가 포함될 수 있습니다. 와일드카드를 사용하여 다중 토픽을 구독할 수 있습니다. 등록 방법 (예: `MqttClient.subscribe(java.lang.String topicFilter)`) 을 사용하여 주제 필터를 작성하십시오.

토픽 문자열

IBM WebSphere MQ 토픽 문자열의 구문은 토픽 문자열에 설명되어 있습니다. MQTT 주제 문자열의 구문은 Java용 MQTT 클라이언트. MQTT 클라이언트 라이브러리에 대한 클라이언트 API 문서에 대한 링크는 MQTT 클라이언트 프로그래밍 참조를 참조하십시오.에 대한 API 문서의 `MqttClient` 클래스에 설명되어 있습니다.

각 유형의 토픽 문자열 구문은 거의 동일합니다. 네 가지 작은 차이점이 있습니다.

1. MQTT 에서 IBM WebSphere MQ 로 전송되는 토픽 문자열은 큐 관리자 이름에 대한 규칙을 따라야 합니다. 특히 토픽 문자열에는 하이픈이 포함될 수 없습니다.
2. 최대 길이는 다양합니다. IBM WebSphere MQ 토픽 문자열은 10,240자로 제한됩니다. MQTT 클라이언트는 최대 65535바이트의 토픽 문자열을 작성할 수 있습니다.
3. MQTT 클라이언트에 의해 작성된 토픽 문자열은 널 문자를 포함할 수 없습니다.
4. WebSphere Message Broker에서 널 토픽 레벨 `'...//...'`이 올바르지 않습니다. 널 토픽 레벨은 IBM WebSphere MQ에서 지원됩니다.

IBM WebSphere MQ 발행/구독과 달리 `mqttv3` 프로토콜에는 관리 토픽 오브젝트의 개념이 없습니다. 토픽 오브젝트와 토픽 문자열에서 토픽 문자열을 구성할 수 없습니다. 그러나 토픽 문자열은 WebSphere MQ의 관리 토픽에 맵핑됩니다. 관리 토픽과 연관된 액세스 제어는 발행물이 토픽에 발행되는지 또는 제거되는지를 결정합니다. 구독자에게 전달될 때 발행물에 적용되는 속성은 관리 토픽의 속성으로부터 영향을 받습니다.

토픽 필터

IBM WebSphere MQ 토픽 필터의 구문은 토픽 기반 와일드카드 스킴에 설명되어 있습니다. MQTT 클라이언트를 사용하여 구성할 수 있는 주제 필터의 구문은 Java용 MQTT 클라이언트에 대한 API 문서의 `MqttClient` 클래스에 설명되어 있습니다. MQTT 클라이언트 라이브러리에 대한 클라이언트 API 문서에 대한 링크는 MQTT 클라이언트 프로그래밍 참조를 참조하십시오.

각 유형의 토픽 필터 구문은 거의 동일합니다. 유일한 차이점은 다른 MQTT 브로커가 토픽 필터를 해석하는 방법입니다. WebSphere Message Broker V6에서 다중 레벨 와일드카드는 토픽 필터의 끝에만 사용될 수 있습니다. WebSphere MQ에서 다중 레벨 와일드카드는 토픽 트리의 모든 레벨에서 사용할 수 있습니다 (예: `USA/#/Dutchess County`).

MQTT 클라이언트 프로그래밍 참조서

모바일 메시징 및 M2M 클라이언트 팩 및 연관된 클라이언트 API 문서에 대한 링크가 있습니다.

모바일 메시징 및 M2M 클라이언트 팩에서는 MQTT 클라이언트 라이브러리가 생성된 API 문서와 함께 번들로 이루어져 있습니다. IBM 메시징 커뮤니티 다운로드에서 클라이언트 팩을 다운로드할 수 있습니다.

Eclipse Paho 프로젝트에 대한 다음 링크를 따라가면 가장 최신 API 문서의 온라인 사본을 볼 수 있습니다.

- Java 클래스의 MQTT 클라이언트
- C용 MQTT 클라이언트 라이브러리
- C용 비동기 MQTT 클라이언트 라이브러리

참고:

1. Link MQTT Java applications to the `org.eclipse.paho.client.mqttv3` package rather than the `com.ibm.micro.client.mqttv3`. 작성합니다. `com.ibm.micro.client.mqttv3` 패키지는 기존 MQTT Java 애플리케이션을 지원하기 위해 제공됩니다.
2. V 7.5.0.1 C의 MQTT 클라이언트 앱을 `MQTTClient` 라이브러리가 아닌 `MQTTAsync` 라이브러리에 링크하십시오. `MQTTClient` 는 C의 기존 MQTT 앱을 지원하기 위해 제공됩니다.
3. JavaScript용 MQTT 메시징 클라이언트 에는 WebSockets를 지원하는 MQTT 서버가 필요합니다. 예를 들어 IBM WebSphere MQ Version 7.5 이상의 버전이 이를 수행합니다.

MQTT 서버 시작하기

MQTT 전송 프로토콜을 지원하는 메시징 서버를 IBM과 기타 회사에서 제공받을 수 있습니다. 가장 기본적인 MQTT 서버에서는 MQTT 클라이언트 라이브러리가 지원하는 모바일 앱과 디바이스를 사용하여 메시지를 교환할 수 있습니다. IBM WebSphere MQ and IBM MessageSight are MQTT servers from IBM. 이들 서버는 기본 MQTT 서버의 기능을 수행하고 MQTT 클라이언트 앱과 엔터프라이즈 앱 사이의 메시지 교환도 수행합니다. IBM의 모든 MQTT 서버는 MQTT version 3.1 프로토콜과 WebSocket protocol을 통한 MQTT를 지원합니다.

IBM의 현재 MQTT 서버

IBM WebSphere MQ

- IBM WebSphere MQ에서는 엔터프라이즈급 메시징을 제공합니다. 텔레메트리 컴포넌트를 사용하면 IBM WebSphere MQ가 MQTT 서버의 기능도 수행할 수 있습니다.
- 이 기능은 모바일 앱, 시스템 간(M2M) 앱, 디바이스 기반 앱을 지원하며 또한 이들 앱이 IBM WebSphere MQ 앱, JMS 앱과 같은 엔터프라이즈 메시징 앱과 메시지를 교환할 수 있도록 합니다.
- IBM WebSphere MQ 설치에는 IBM의 MQTT SDK 사본이 포함됩니다. 이 SDK는 샘플 MQTT 클라이언트 앱과 이 앱을 지원하는 MQTT 클라이언트 라이브러리를 제공합니다.

  참고: 이 SDK의 최신 버전을 가져오려면 모바일 메시징 및 M2M 클라이언트 팩을 다운로드하십시오. 자세한 정보는 9 페이지의 『MQTT 클라이언트 시작하기』의 내용을 참조하십시오.
- MQTT 지원은 IBM WebSphere MQ Version 7.0.1에 처음 포함되었습니다. IBM WebSphere MQ의 각 릴리스에 대한 전체 정보는 다음 제품 문서를 참조하십시오.
  - WebSphere MQ Telemetry 버전 7.5
  - WebSphere MQ Telemetry 버전 7.1

  IBM WebSphere MQ에 대한 간략한 소개와 IBM WebSphere MQ Telemetry 컴포넌트를 시작하는 단계는 132 페이지의 『MQTT 서버로 IBM WebSphere MQ』의 내용을 참조하십시오.

IBM MessageSight

- IBM MessageSight는 대량의 MQTT 클라이언트를 동시에 연결할 수 있고 계속해서 증가하는 많은 모바일 디바이스와 센서를 수용하는 데 필요한 성능과 확장성을 제공하는 어플라이언스 기반 MQTT 서버입니다. MQTT version 3.1 프로토콜과 WebSocket protocol을 통한 MQTT를 지원합니다.

- MQTT 서버로서 IBM MessageSight가 갖고 있는 주요 기능과 이점은 다음과 같습니다.
  - 고성능, 신뢰성, 확장 가능한 메시징
  - 동시 연결된 엔드 포인트에 필요한 대량의 커뮤니티를 지원함으로써 특히 기기 간 통신(M2M)과 사물 인터넷(Internet of Things) 시나리오에 적합하게 설계됨
  - 설치와 사용의 용이성. 30분 이내에 작동되어 실행될 수 있습니다.
  - Android와 iOS를 포함하는 고유 모바일 앱 지원
  - 발행/구독 브로커로 IBM WebSphere MQ와 통합
- IBM MessageSight에 대한 간략한 소개는 유튜브의 MessageSight 소개와 MessageSight 공지사항을 참조하십시오. 자세한 기술 정보는 MessageSight 제품 문서를 참조하십시오.

IBM WebSphere MQ Telemetry daemon for devices

- 이를 IBM WebSphere MQ Telemetry advanced client for C라고도 합니다. 이 디먼은 일반적으로 위성 위치에서 또는 네트워크의 에지 근처에 있는 디바이스(예: 셋톱 박스, 원격 텔레메트리 장치 또는 POS 터미널)에서 실행되는 작은 풋프린트 MQTT 서버입니다.
- 일반적으로 MQTT 클라이언트 연결을 많이 집중하는 것이며, 이 연결은 단일 MQTT 연결로 인터넷을 통해 IBM WebSphere MQ 에 연결됩니다. 예를 들어, 많은 수의 센서를 빌딩에 설치하고 센서를 IBM WebSphere MQ Telemetry daemon for devices에 연결한 다음 디먼을 IBM WebSphere MQ에 연결할 수 있습니다.
- IBM WebSphere MQ Telemetry daemon for devices은 IBM WebSphere MQ와 함께 포함됩니다. 이를 IBM WebSphere MQ에 연결하려면 별도의 라이센스가 필요합니다. IBM 미국 소프트웨어 공지사항 212-091을 참조하십시오.

Really Small Message Broker

- Really Small Message Broker는 IBM WebSphere MQ Telemetry daemon for devices의 한 버전입니다. 주요한 차이점은 사용법에 있습니다. RSMB는 작은 테스트 서버로서 IBM alphaWorks®에서 사용 가능하며 MQTT 기반 솔루션을 평가하거나 실험할 때 사용하기 위해 만들어졌습니다. RSMB는 여러 Linux 플랫폼, Windows XP, Apple Mac OS X Leopard, Unslung (Linksys NSLU12)에서 MQTT를 지원합니다.

IBM의 이전 MQTT 서버

WebSphere Message Broker *(이제 IBM Integration Bus 로 알려짐)*

- WebSphere Message Broker 버전 6에서는 자체 MQTT 서버를 제공했습니다. 지원이 IBM WebSphere MQ의 텔레메트리 구성요소에 의해 WebSphere Message Broker 버전 7에서 대체되었습니다.

기타 MQTT 서버

MQTT.org는 소프트웨어 페이지에서 MQTT 서버 및 브로커 목록을 보존하고 있습니다(오픈 소스 서버 포함).

관련 태스크

9 페이지의 『MQTT 클라이언트 시작하기』

MQTT 클라이언트 라이브러리를 사용하는 샘플 MQTT 클라이언트 앱을 빌드하고 실행하여 모바일 또는 시스템 간(M2M) 앱 개발을 시작할 수 있습니다. 샘플 앱 및 연관된 클라이언트 라이브러리는 IBM의 모바일 메시징 및 M2M 클라이언트 팩 에서 사용 가능합니다. Java, JavaScript및 C로 작성된 앱 및 클라이언트 라이브러리의 버전이 있습니다. Apple의 Android 디바이스 및 제품을 포함하여 대부분의 플랫폼 및 디바이스에서 이러한 앱을 실행할 수 있습니다.

MQTT 서버로 IBM WebSphere MQ

IBM WebSphere MQ에 포함된 MQTT 서버 사용에 대한 소개입니다.

시작하려면 다음 글에 있는 단계를 수행하십시오.

- 132 페이지의 『설치 중IBM WebSphere MQ』
- 134 페이지의 『명령행에서 MQTT 서비스 구성』
- 136 페이지의 『IBM WebSphere MQ Explorer로 MQTT 서비스 구성』

참고: 명령행 인터페이스 예를 사용하여 빨리 시작할 수 있습니다. 그러나 사용자의 구성이 예와 아주 다른 경우에는 명령행 인터페이스를 효율적으로 사용할 수 있는 지식과 기술이 필요합니다. IBM WebSphere MQ Explorer 인터페이스를 사용하여 간편하게 시작하고 표준 구성 태스크를 수행할 수 있습니다.

IBM WebSphere MQ Telemetry 컴포넌트에 대한 핵심 개념 정보는 IBM WebSphere MQ 제품 문서의 다음 글을 참조하십시오.

- 텔레메트리 디바이스를 큐 관리자에 연결
- 텔레메트리(MQXR) 서비스
- 텔레메트리 채널

관련 정보

Linux 및 AIX 텔레메트리에 대한 큐 관리자 구성
Windows에서 Telemetry용 큐 관리자 구성
MQTT 클라이언트에 메시지를 보내도록 분산 큐잉 구성
WebSphere MQ Telemetry 관리

설치 중IBM WebSphere MQ

Follow these instructions to obtain and install IBM WebSphere MQ and configure IBM WebSphere MQ Telemetry on Windows or Linux.

시작하기 전에

IBM WebSphere MQ에서 실행 중인 MQTT 서비스가 지원하는 운영 체제의 경우, IBM WebSphere MQ Telemetry 시스템 요구사항을 참조하십시오.

다음 방법 중 하나를 사용하여 IBM WebSphere MQ 설치 자료의 사본 및 라이센스를 구하십시오.

1. IBM WebSphere MQ 관리자에게 설치 자료를 요청하고 라이센스 계약을 승인할 수 있는지 확인하십시오.
2. IBM WebSphere MQ의 90일 평가판을 가져오십시오. 평가: IBM WebSphere MQ을 참조하십시오.
3. IBM WebSphere MQ를 구매하십시오. IBM WebSphere MQ 제품 페이지을 참조하십시오.

이 태스크 정보

Install IBM WebSphere MQ as `root` on Linux, and as an administrator on Windows. 설치 시, 추가 옵션인 텔레메트리 서비스 및 텔레메트리 클라이언트를 선택하여 IBM WebSphere MQ Telemetry 컴포넌트를 설치하십시오. 사용자 ID를 작성하여 IBM WebSphere MQ를 관리하고 게스트 사용자 ID가 정의되어 있는지 확인하십시오. 게스트 사용자 ID는 IBM WebSphere MQ에 대한 MQTT 액세스 권한을 부여하기 위해 샘플 MQTT 서비스 구성에서 사용됩니다.

IBM WebSphere MQ를 설치한 후 134 페이지의 『명령행에서 MQTT 서비스 구성』 또는 136 페이지의 『IBM WebSphere MQ Explorer로 MQTT 서비스 구성』의 단계를 수행하여 MQTT 서비스를 시작하십시오.

프로시저

1. Linux에서 `root` 로 로그온하거나 Windows의 관리자로 로그온하십시오.
2. IBM WebSphere MQ를 설치하십시오.

   Linux에 WebSphere MQ 서버 설치 또는 Windows에 WebSphere MQ 서버 설치의 지시사항을 따르십시오. 텔레메트리 서비스 및 텔레메트리 클라이언트를 선택하여 IBM WebSphere MQ Telemetry 컴포넌트를 설치하십시오.

   Linux에서는 "다음에 수행할 작업" 섹션의 지시사항을 기록해서 사용자의 설치를 기본 설치로 만드십시오. 이 설치가 워크스테이션의 유일한 IBM WebSphere MQ 설치인 경우에도 이 설치를 기본 설치로 설정하십시오. 기본 설치로 구성된 WebSphere MQ 버전 7.1 이상의 단일 설치를 참조하십시오.

   샘플 구성 지시사항을 정확하게 따르려면 설치를 기본 설치로 만들어야 합니다.

   다중 설치: 기본이 아닌 설치에 대해 작업하려면 setmqenv 명령을 실행하십시오. 이 명령은 워크스테이션의 명령 창에서 IBM WebSphere MQ 환경을 설정합니다. 다중 설치를 참조하십시오.

   설치 프로그램이 제안한 기본 설치 위치를 승인했다고 가정하면 IBM WebSphere MQ가 다음과 같은 디렉토리에 설치됩니다.

   Linux 64비트

   ```
   /opt/mqm
   ```

   Windows 32비트

   ```
   C:\Program Files\IBM\WebSphere MQ
   ```

   Windows 64비트

   ```
   C:\Program Files (x86)\IBM\WebSphere MQ
   ```

   설치 디렉토리는 *MQ_INSTALLATION_PATH*로 표시됩니다.
3. 옵션: IBM WebSphere MQ를 관리하는 데 사용할 사용자를 이 워크스테이션의 `mqm` 그룹에 추가하십시오.

   Windows 관리자로 IBM WebSphere MQ 를 관리할 수 있으므로 Windows 에서 이 단계는 선택사항입니다. UNIX 및 Windows 시스템에서의 WebSphere MQ 관리 권한을 참조하십시오.

   Windows 워크스테이션이 도메인의 멤버인 경우 기본값이 아닌 Windows 2000 도메인 또는 기본값인 Windows 2003 및 Windows Server 2008 도메인, 보안 권한을 참조하십시오.

   On Linux, the installation program creates a user `mqm`, as a member of the group `mqm`. 이 사용자에게 비밀번호를 제공하거나 `mqm`이 기본 그룹인 다른 사용자를 작성하십시오.
4. 옵션: `mqm` 그룹의 구성원으로 만들 사용자로 사인온하십시오.

   Windows 관리자로 IBM WebSphere MQ 를 관리할 수 있으므로 Windows 에서 이 단계는 선택사항입니다.
5. 게스트 사용자 ID가 워크스테이션에 정의되어 있는지 확인하십시오.

   The guest user ID is "`guest`" on Windows and "`nobody`" on Linux. 게스트 사용자 ID에는 운영 체제 권한이 필요하지 않습니다.

결과

IBM WebSphere MQ를 기본 IBM WebSphere MQ 설치로 워크스테이션에 설치하고 `mqm` 그룹을 작성했습니다. 설치 시 `mqm` 그룹의 구성원에게 IBM WebSphere MQ 를 관리할 수 있는 권한이 부여됩니다. Windows에서 관리자 그룹의 구성원은 IBM WebSphere MQ에 대한 관리 권한도 가지고 있습니다.

다음에 수행할 작업

1. 명령행 또는 IBM WebSphere MQ Explorer에서 MQTT 서비스를 구성하십시오. 136 페이지의 『IBM WebSphere MQ Explorer로 MQTT 서비스 구성』 또는 134 페이지의 『명령행에서 MQTT 서비스 구성』를 참조하십시오.

2. Android, iOS, WebSockets, Java 및 "C" MQTT 클라이언트를 테스트하십시오.
3. When you finish testing, remove the queue manager and MQTT service by running the command *MQ_INSTALLATION_PATH*\mqxr\samples\CleanupMQM.bat on Windows and *MQ_INSTALLATION_PATH*/mqxr/samples/CleanupMQM.sh on Linux.

관련 정보

WebSphere MQ Telemetry 설치
Linux 에 WebSphere MQ 서버 설치
Windows에서 WebSphere MQ 서버 설치

명령행에서 MQTT 서비스 구성

샘플 IBM WebSphere MQ Telemetry 애플리케이션을 실행하려면 명령행을 사용하여 IBM WebSphere MQ 를 구성하십시오. 이 단계에서는 새 큐 관리자 MQXR_SAMPLE_QM에서 MQTT 서비스를 작성하기 위해 스크립트를 실행하는 방법을 보여줍니다.

시작하기 전에

MQTT 서비스를 설정하려면 IBM WebSphere MQ 큐 관리자에 대한 관리 액세스 권한이 있어야 합니다. 큐 관리자에는 다음과 같은 여러 가지 방법으로 액세스할 수 있습니다.

1. IBM WebSphere MQ의 사본을 가져오고 고유한 Linux 또는 Windows 워크스테이션에서 큐 관리자를 작성하십시오. 132 페이지의 『설치 중IBM WebSphere MQ』의 지시사항을 수행하여 IBM WebSphere MQ를 확보하고 설치하십시오. 설치 시 텔레메트리 서비스 및 텔레메트리 클라이언트도 선택해야 합니다. 또한 기존 설치를 수정하여 이 옵션을 추가할 수도 있습니다.
2. IBM WebSphere MQ 관리자에게 문의하여 IBM WebSphere MQ Telemetry가 옵션으로 설치된 서버의 큐 관리자에 대한 관리 액세스 권한을 요청하십시오. V7.5.0.1 큐 관리자 이름 외에도 MQTT 및 WebSockets에 대한 MQTT에 대해 두 개 이상의 TCP/IP 포트가 필요합니다. 보안 클라이언트에 연결하려는 경우에는 두 개 이상의 포트가 추가로 필요합니다.

설명된 대로 정확히 태스크의 단계를 수행하려면 MQXR_SAMPLE_QM이라고 하는 큐 관리자를 작성할 수 있어야 하며, TCP/IP 포트 1883이 사용되지 않아야 합니다.

이 태스크 정보

이 태스크에서는 큐 관리자를 작성한 후 MQTT 서비스가 포트 1883에서 MQTT V3.1 클라이언트 연결을 대기하도록 구성하는 스크립트를 실행합니다.구성은 모든 사용자에게 주제를 공개하고 구독할 수 있는 권한을 제공합니다. 보안 및 액세스 제어는 최소한으로 구성하며 제한된 액세스가 있는 보안 네트워크에 있는 큐 관리자만을 대상으로 합니다. 비보안 환경에서 IBM WebSphere MQ 및 MQTT를 실행하려면 보안을 구성해야 합니다. IBM WebSphere MQ 및 MQTT의 보안을 구성하려면 이 태스크의 끝에서 관련 링크를 참조하십시오.

프로시저

1. IBM WebSphere MQ에 대한 관리 권한이 있는 사용자 ID로 로그온하십시오.

   IBM WebSphere MQ에 대한 관리 권한이 있는 사용자 ID를 정의하려면 132 페이지의 『설치 중IBM WebSphere MQ』의 3단계를 참조하십시오.
2. 명령 창을 열고 샘플 명령 스크립트를 실행하여 MQXR_SAMPLE_QM이라는 샘플 큐 관리자와 MQTT 서비스를 작성한 후 시작하십시오.

   The path to the sample script is %MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat on Windows and *MQ_INSTALLATION_PATH*/mqxr/samples/SampleMQM.sh on Linux.

   다음 명령을 입력하여 큐 관리자를 작성하고 구성하십시오.

   - Windows

     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```

- Linux

```
MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
```

결과

샘플은 Windows에서 다음 특성을 사용하여 PlainText 라는 MQTT 채널을 작성합니다.

```
com.ibm.mq.MQXR.channel/PlainText: \
com.ibm.mq.MQXR.Protocol=MQTT;\
com.ibm.mq.MQXR.Port=1883;\
com.ibm.mq.MQXR.Backlog=4096;\
com.ibm.mq.MQXR.UserName=Guest;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

Linux의 채널 특성은 Windows와 동일합니다(com.ibm.mq.MQXR.UserName=nobody 제외).

포트 1883에 연결하는 MQTT V3.1 클라이언트는 *com.ibm.mq.MQXR.UserName*변수에 설정된 사용자 ID로 IBM WebSphere MQ 에 액세스합니다. 샘플 스크립트에서는 다음과 같은 IBM WebSphere MQ 명령으로 사용자 ID에 권한을 부여합니다.

```
setmqaut -m MQXR_SAMPLE_QM -t topic -n SYSTEM.BASE.TOPIC -p com.ibm.mq.MQXR.UserName -all +pub
+sub
setmqaut -m MQXR_SAMPLE_QM -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p com.ibm.mq.MQXR.UserName -all
+put
```

첫 번째 명령은 사용자에게 기본 토픽에서 권한을 상속하는 토픽에 대한 발행 및 구독 권한을 부여합니다. 두 번째 명령은 사용자에게 SYSTEM.MQTT.TRANSMIT.QUEUE 전송 큐에 메시지를 추가할 수 있는 권한을 부여합니다. MQTT 서비스는 SYSTEM.MQTT.TRANSMIT.QUEUE에 있는 메시지를 발행물로 MQTT 구독자에게 송신합니다.

스크립트는 큐 관리자에서 MQTT 서비스를 시작하여 포트 1883에서 연결을 대기합니다.

다음에 수행할 작업

이 단계를 수행하여 샘플 MQTT V3.1 Java 애플리케이션을 실행하여 연결을 테스트하십시오.

샘플 Java 애플리케이션의 소스는 MQTTV3Sample.java 파일에 있습니다.

샘플을 실행하려면 두 개의 명령 창이 필요합니다. 샘플을 한 창에서는 구독자로 실행하고 다른 창에서는 발행자로 실행하십시오.

- Windows 구독자를 시작하려면 다음 명령을 실행하십시오.

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
  ```

  발행하려면 다음 명령을 실행하십시오.

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
  ```

- Linux 구독자를 시작하려면 다음 명령을 실행하십시오.

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
  ```

  발행하려면 다음 명령을 실행하십시오.

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
  ```

발행자와 구독자는 출력을 명령 창에 기록합니다.

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

그림 *27.* 발행자의 출력

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
Topic:          MQTTV3Sample/Java/v3
Message:        Message from MQTTv3 Java client
QoS:            2
```

그림 *28.* 구독자의 출력

서버가 이제 MQTT V3.1 앱을 테스트할 준비가 되었습니다.

관련 태스크

WebSphere MQ Explorer를 통한 MQTT 서비스 구성
샘플 IBM WebSphere MQ Telemetry 클라이언트를 실행하기 위해 IBM WebSphere MQ Explorer 를 사용하여 IBM WebSphere MQ 를 구성하려면 다음 지시사항을 따르십시오. 단계에서는 `Define sample` 구성 마법사를 실행하여 MQTT 서비스를 작성하는 방법을 보여줍니다.

관련 정보

WebSphere MQ Telemetry
WebSphere MQ Telemetry용 애플리케이션 개발
WebSphere MQ Telemetry 관리
WebSphere MQ Telemetry 보안

IBM WebSphere MQ Explorer로 MQTT 서비스 구성

샘플 IBM WebSphere MQ Telemetry 클라이언트를 실행하기 위해 IBM WebSphere MQ Explorer 를 사용하여 IBM WebSphere MQ 를 구성하려면 다음 지시사항을 따르십시오. 단계에서는 `Define sample` 구성 마법사를 실행하여 MQTT 서비스를 작성하는 방법을 보여줍니다.

시작하기 전에

MQTT 서비스를 설정하려면 IBM WebSphere MQ 큐 관리자에 대한 관리 액세스 권한이 있어야 합니다. 큐 관리자에는 다음과 같은 여러 가지 방법으로 액세스할 수 있습니다.

1. IBM WebSphere MQ의 사본을 가져오고 고유한 Linux 또는 Windows 워크스테이션에서 큐 관리자를 작성하십시오. 132 페이지의 『설치 중IBM WebSphere MQ』 의 지시사항을 수행하여 IBM WebSphere MQ를 확보하고 설치하십시오. 설치 시 텔레메트리 서비스 및 텔레메트리 클라이언트도 선택해야 합니다. 또한 기존 설치를 수정하여 이 옵션을 추가할 수도 있습니다.
2. IBM WebSphere MQ 관리자에게 문의하여 IBM WebSphere MQ Telemetry가 옵션으로 설치된 서버의 큐 관리자에 대한 관리 액세스 권한을 요청하십시오. V 7.5.0.1 큐 관리자 이름 외에도 MQTT 및 WebSockets에 대한 MQTT에 대해 두 개 이상의 TCP/IP 포트가 필요합니다. 보안 클라이언트에 연결하려는 경우에는 두 개 이상의 포트가 추가로 필요합니다.

설명된 대로 정확히 태스크의 단계를 수행하려면 `MQXR_SAMPLE_QM`이라고 하는 큐 관리자를 작성할 수 있어야 하며, TCP/IP 포트 1883이 사용되지 않아야 합니다.

이 태스크 정보

이 태스크에서는 IBM WebSphere MQ Explorer `Define sample` 구성 마법사를 실행하여 MQTT 서비스를 작성하여 포트 1883에서 MQTT V3.1 클라이언트 연결을 대기합니다.구성은 모든 사용자에게 주제를 공개하고 구독할 수 있는 권한을 제공합니다. 보안 및 액세스 제어는 최소한으로 구성하며 제한된 액세스가 있는 보안 네트워크에 있는 큐 관리자만을 대상으로 합니다. 비보안 환경에서 IBM WebSphere MQ 및 MQTT를 실행하려면 보안

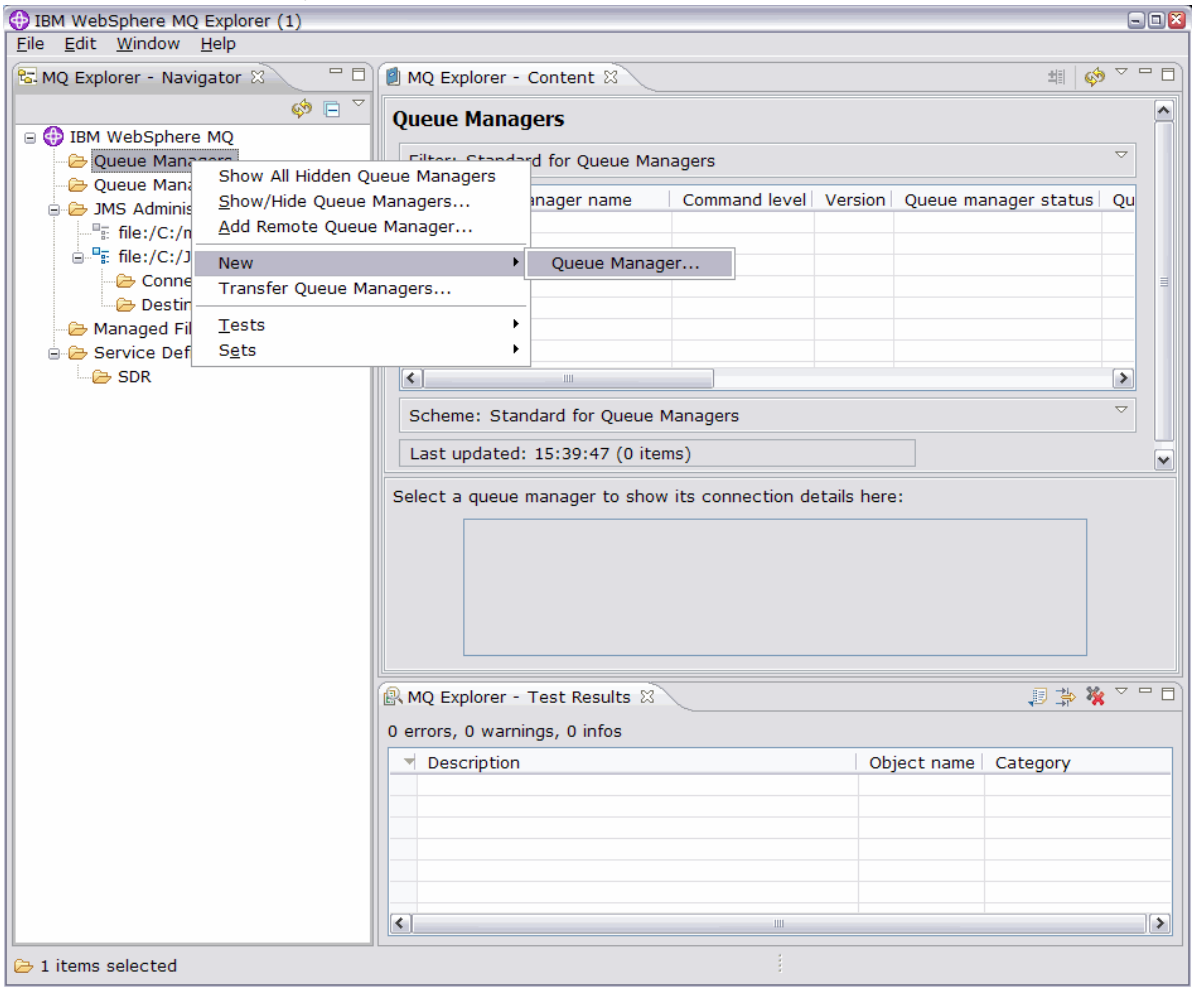
을 구성해야 합니다. IBM WebSphere MQ 및 MQTT의 보안을 구성하려면 이 태스크의 끝에서 관련 링크를 참조하십시오.

프로시저

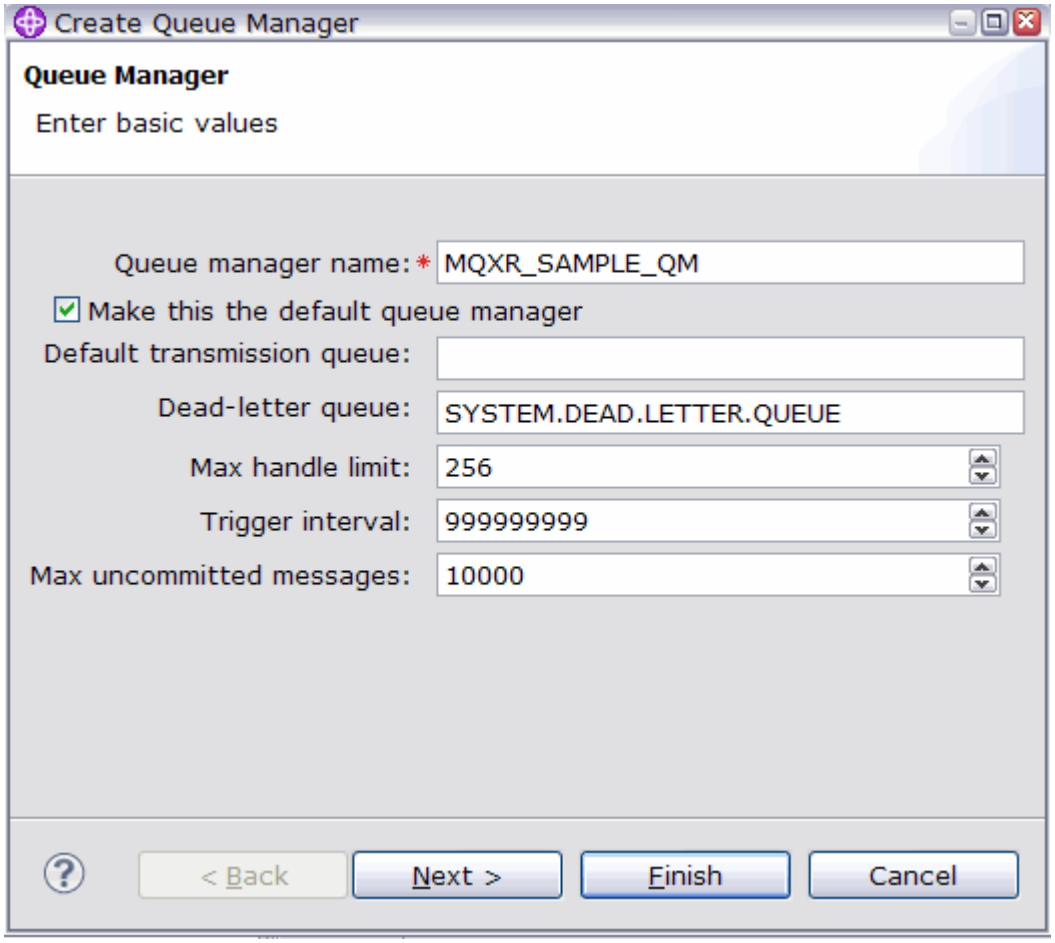
1. IBM WebSphere MQ에 대한 관리 권한이 있는 사용자 ID로 로그온하십시오.

   IBM WebSphere MQ에 대한 관리 권한이 있는 사용자 ID를 정의하려면 132 페이지의 『설치 중IBM WebSphere MQ』의 3단계를 참조하십시오.
2. 명령 창을 열고 IBM WebSphere MQ Explorer 명령 **strmqcfg** 을 실행하여 IBM WebSphere MQ Explorer를 시작하십시오.
3. 큐 관리자 작성

   a) **새 큐 관리자** 마법사를 시작하십시오.

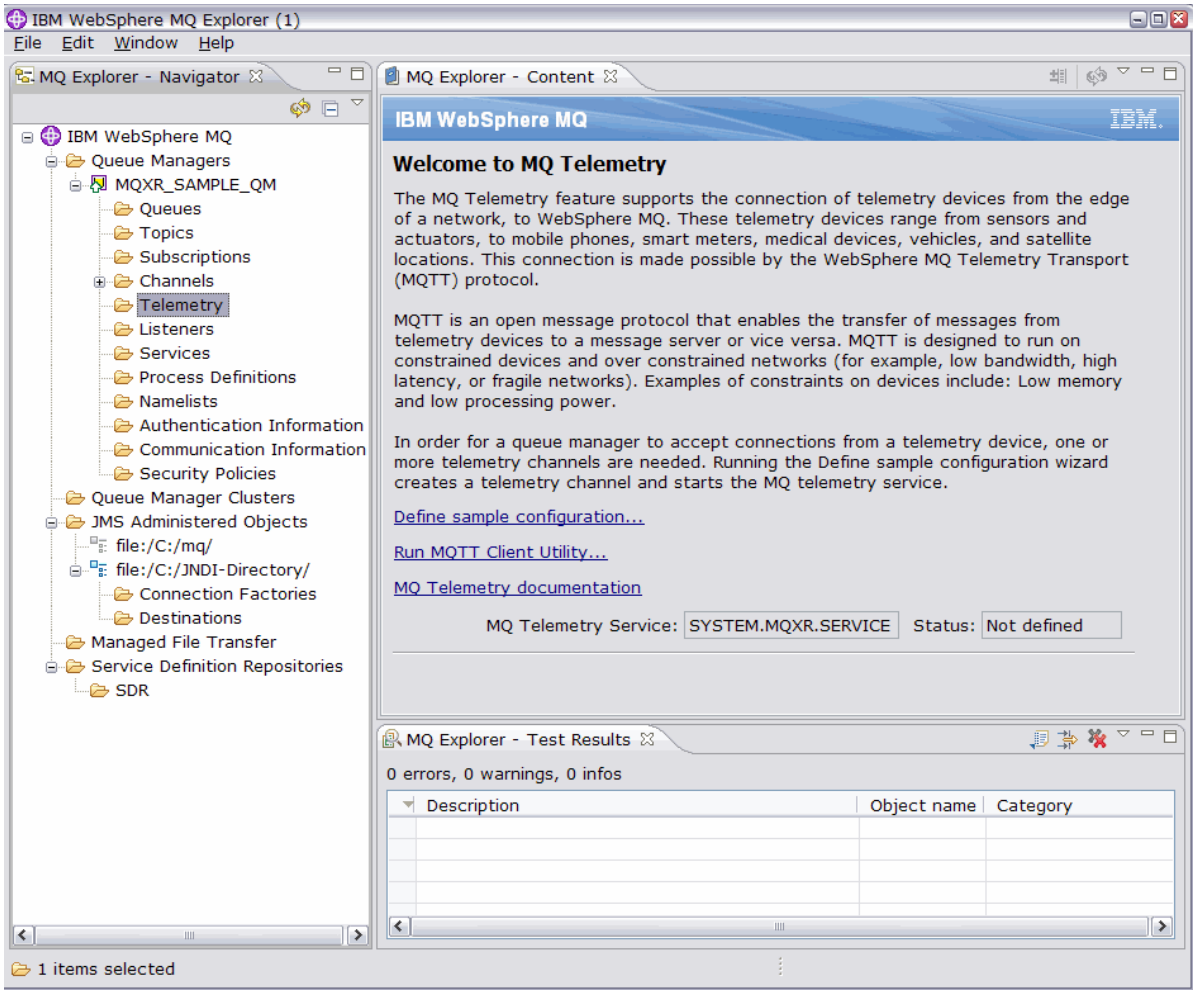


   b) **큐 관리자 이름**을 입력한 후 **데드-레터 큐**의 이름을 입력하십시오. 편의상 해당 이름을 기본 큐 관리자로 지정하십시오. **완료**를 클릭하십시오.



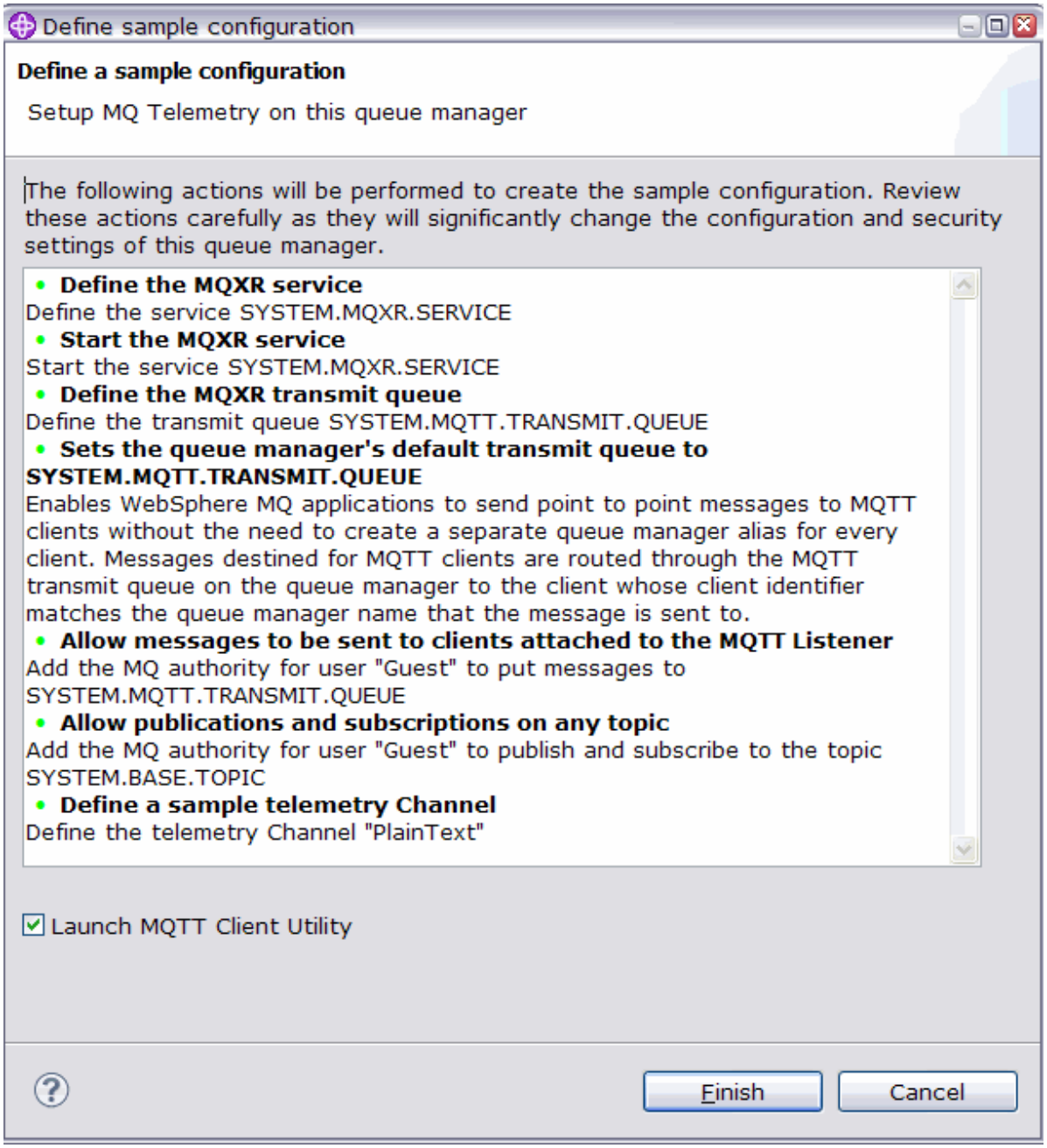
IBM WebSphere MQ Explorer가 큐 관리자를 작성하여 시작합니다.

4. Telemetry **샘플 구성 정의** 마법사를 실행하십시오.
   a) 큐 관리자의 Telemetry 폴더를 여십시오.



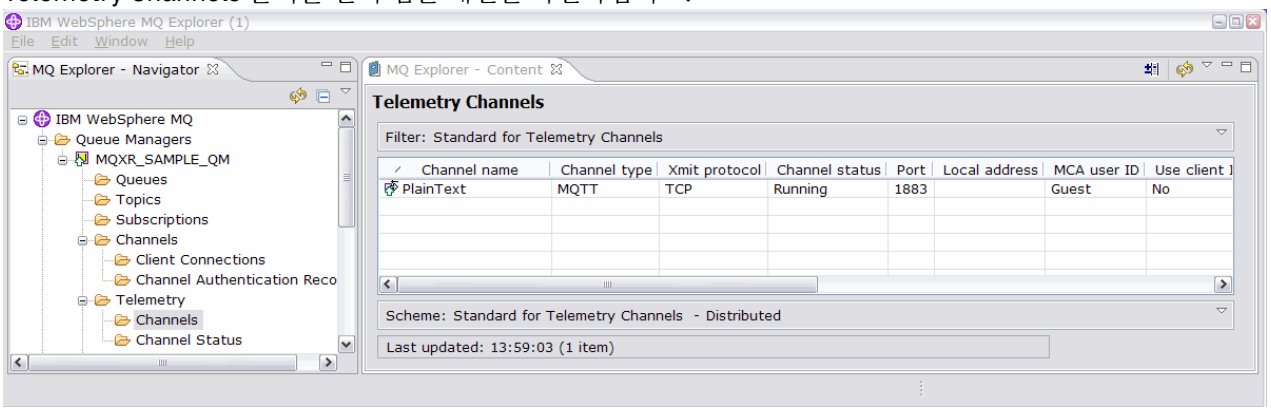
b) **샘플 구성 정의**를 클릭하여 마법사를 시작하십시오.

c) **완료**를 클릭하여 텔레메트리 서비스를 작성하고 MQTT 클라이언트 유틸리티를 실행하십시오.



결과

Telemetry Channels 폴더를 열어 샘플 채널을 나열하십시오.



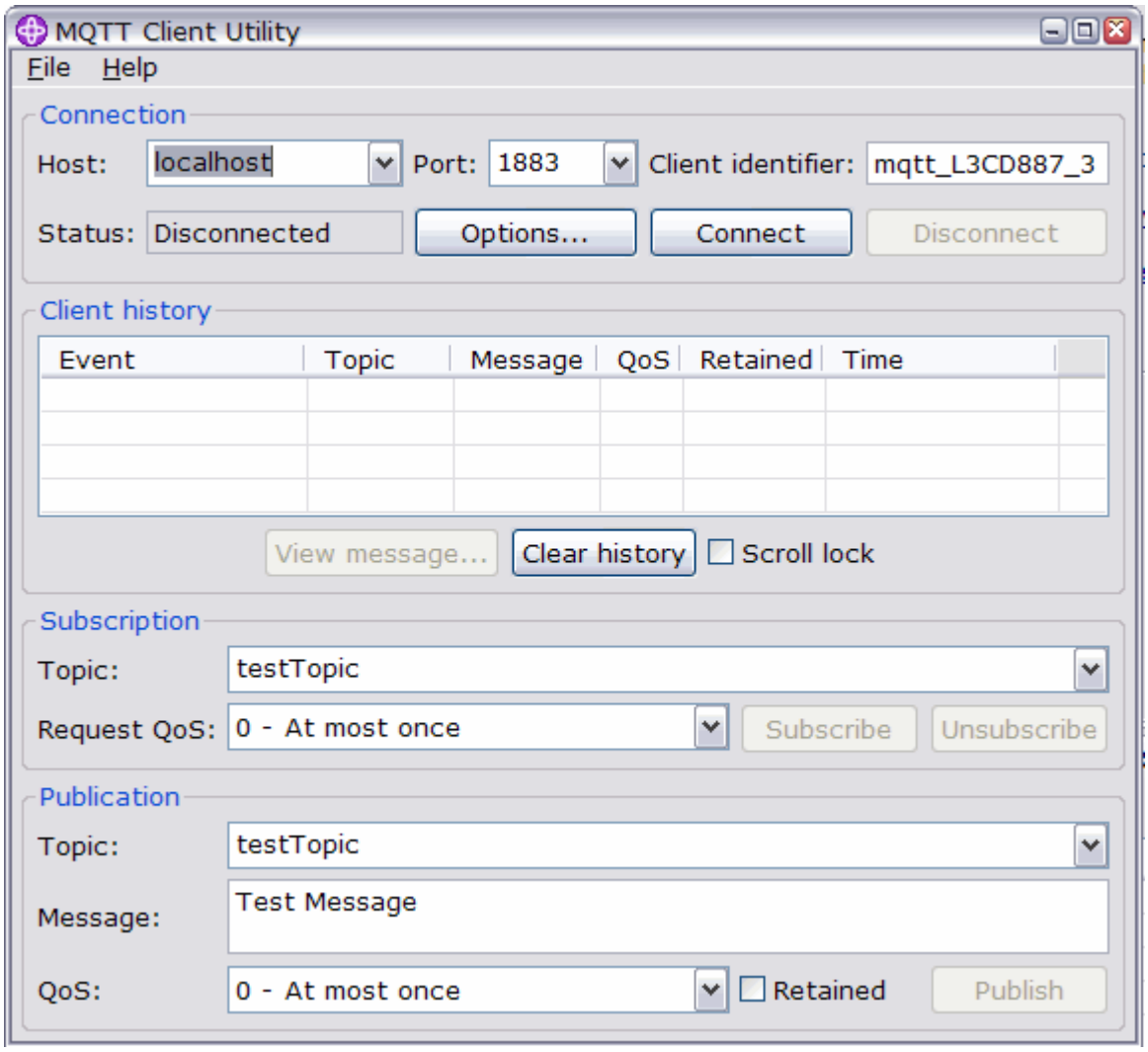
이 채널의 특성을 수정할 수 있으며 이 창에서 채널을 추가하고 삭제할 수 있습니다.

다음에 수행할 작업

MQTT 클라이언트 유틸리티를 실행하여 연결을 테스트하십시오.

1. 클라이언트 유틸리티를 시작하려면 **Telemetry** 폴더를 열고 **MQTT 클라이언트 유틸리티**를 두 번 클릭하십시오.

   서로 다른 클라이언트 ID를 위한 동일한 두 개의 **MQTT 클라이언트 유틸리티** 창이 열립니다.



2. 두 창에서 **연결**을 클릭하십시오.
3. 두 창에서 **구독**을 클릭하십시오.
4. 두 창 중 하나에서 **발행**을 클릭하십시오. 결과가 142 페이지의 그림 29에 표시됩니다.

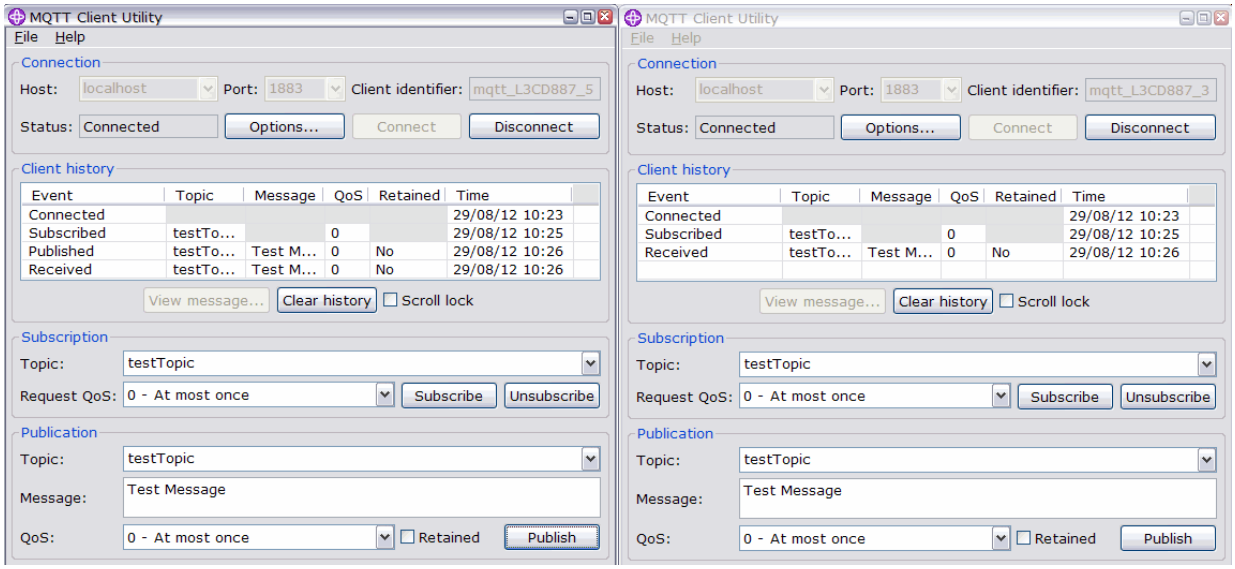


그림 *29.* 결과

5. 두 창에서 **연결 끊기**를 클릭하십시오.

서버가 이제 MQTT V3.1 앱을 테스트할 준비가 되었습니다.

관련 태스크

명령행에서 MQTT 서비스 구성

샘플 IBM WebSphere MQ Telemetry 애플리케이션을 실행하려면 명령행을 사용하여 IBM WebSphere MQ 를 구성하십시오. 이 단계에서는 새 큐 관리자 MQXR_SAMPLE_QM에서 MQTT 서비스를 작성하기 위해 스크립트를 실행하는 방법을 보여줍니다.

WebSphere MQ Telemetry 관리

관련 정보

WebSphere MQ Telemetry

WebSphere MQ Explorer로 WebSphere MQ Telemetry 관리

WebSphere MQ Telemetry용 애플리케이션 개발

보안

WebSphere MQ Telemetry 보안

디바이스용 IBM WebSphere MQ Telemetry 디먼 개념

디바이스용 IBM WebSphere MQ Telemetry 디먼은 고급 MQTT V3 클라이언트 앱입니다. 다른 MQTT 클라이언트에서 온 메시지를 저장 후 전달하려면 이를 사용하십시오. MQTT 클라이언트처럼 IBM WebSphere MQ에 연결하지만 다른 MQTT 클라이언트도 여기 연결할 수 있습니다.

디먼은 발행/구독 브로커입니다. MQTT V3 클라이언트는 발행할 토픽 문자열 및 구독할 토픽 필터를 사용하여 토픽에 발행 및 구독하기 위해 이에 연결합니다. 토픽 문자열은 /로 구분된 토픽 레벨이 있는 계층 구조입니다. 토픽 필터는 단일 레벨 + 와일드카드 및 다중 레벨 # 와일드카드를 토픽 문자열의 마지막 부분으로 포함할 수 있는 토픽 문자열입니다.

참고: 디먼의 와일드카드는 WebSphere Message Broker, v6의 보다 제한적인 규칙을 따릅니다. IBM WebSphere MQ는 다릅니다. 이는 여러 개의 다중 레벨 와일드카드를 지원합니다. 와일드카드는 토픽 문자열에 있는 임의의 계층 구조 레벨 수를 나타냅니다.

다중 MQTT v3 클라이언트는 리스너 포트를 사용하여 디먼에 연결됩니다. 기본 리스너 포트는 수정 가능합니다. 다중 리스너 포트를 정의하고 다른 네임스페이스를 할당할 수 있습니다(149 페이지의 『디바이스용 WebSphere MQ Telemetry 디먼 리스너 포트』 참조). 디먼은 그 자체가 MQTT v3 클라이언트입니다. 다른 디먼의 리스너 포트 또는 WebSphere MQ Telemetry(MQXR) 서비스에 디먼을 연결하도록 디먼 브릿지 연결을 구성하십시오.

디바이스용 WebSphere MQ Telemetry 디먼에 대해 다중 브릿지를 구성할 수 있습니다. 브릿지를 사용하여 발행물을 교환할 수 있는 디먼의 네트워크를 함께 연결하십시오.

각 브릿지는 해당 로컬 디먼에서 토픽을 발행 및 구독할 수 있습니다. 연결되어 있는 다른 디먼, WebSphere MQ 발행/구독 브로커 또는 다른 MQTT v3 브로커에서도 토픽을 발행하고 구독할 수 있습니다. 토픽 필터를 사용하면 브로커 간에 전파할 발행물을 선택할 수 있습니다. 다른 방향으로 발행물을 전파할 수 있습니다. 로컬 디먼에서 연결된 각 리모트 브로커로, 또는 연결된 브로커에서 로컬 디먼으로 발행물을 전파할 수 있습니다(143 페이지의 『디바이스용 IBM WebSphere MQ Telemetry 디먼 브릿지』 참조).

디바이스용 IBM WebSphere MQ Telemetry 디먼 브릿지

디바이스용 IBM WebSphere MQ Telemetry 디먼 브릿지는 MQTT v3 프로토콜을 사용하여 두 개의 발행/구독 브로커를 연결합니다. 브릿지는 브로커 간에 양방향으로 발행물을 전파합니다. 한쪽 끝은 디바이스용 WebSphere MQ Telemetry 디먼 브릿지 연결이고 다른 한쪽 끝은 큐 관리자 또는 다른 디먼이 될 수 있습니다. 큐 관리자는 텔레메트리 채널을 사용하여 브릿지 연결에 연결됩니다. 디먼은 디먼 리스너를 사용하여 브릿지 연결에 연결됩니다.

디바이스용 IBM WebSphere MQ Telemetry 디먼은 다른 브로커에 대한 하나 이상의 동시 연결을 지원합니다. 디먼으로부터의 연결을 브릿지라 하며 디먼 구성 파일에서 연결 항목별로 정의됩니다. IBM WebSphere MQ에 대한 연결은 다음 그림에 표시된 대로 IBM WebSphere MQ 텔레메트리 채널을 사용하여 설정됩니다.

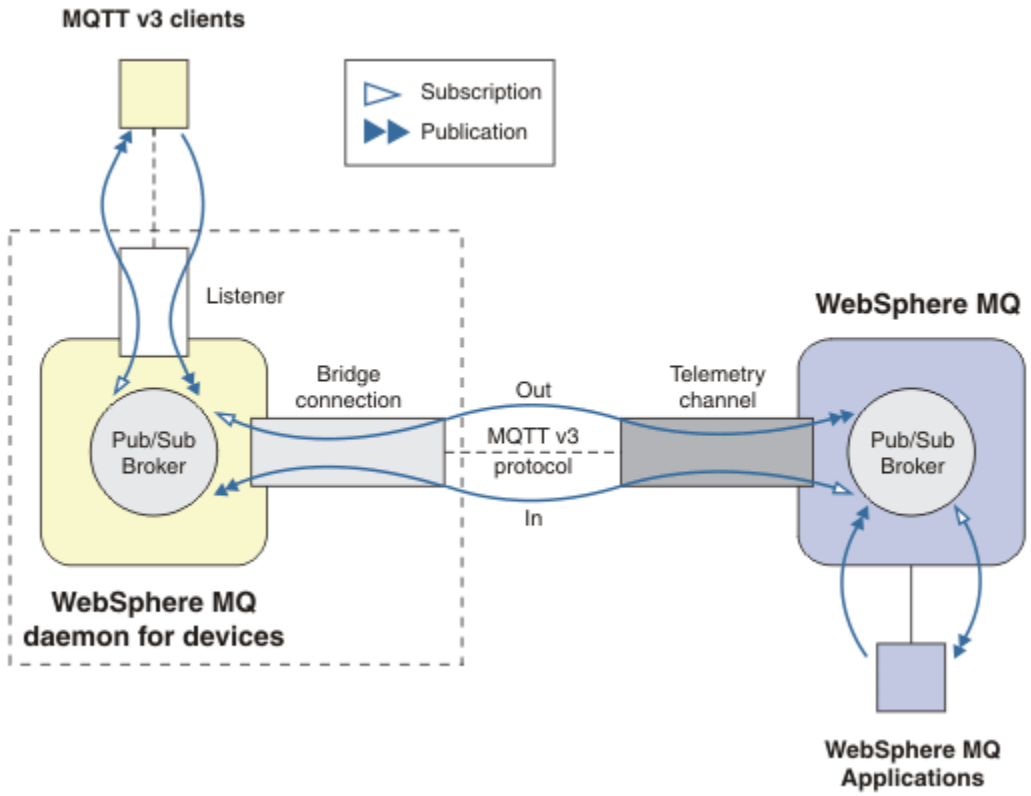


그림 *30. Connecting IBM WebSphere MQ Telemetry daemon for devices to IBM WebSphere MQ*

브릿지는 MQTT v3 클라이언트로 다른 브로커에 디먼을 연결합니다. 브릿지 매개변수는 MQTT v3 클라이언트의 속성을 미러링합니다.

브릿지는 둘 이상의 연결입니다. 이는 두 발행/구독 브로커 사이에서 발행 및 구독 에이전트의 역할을 합니다. 로컬 브로커는 디바이스용 IBM WebSphere MQ Telemetry 디먼이고 원격 브로커는 MQTT v3 프로토콜을 지원하는 발행/구독 브로커입니다. 일반적으로 원격 브로커는 다른 디먼 또는 IBM WebSphere MQ입니다.

브릿지의 역할은 두 브로커 사이에 발행물을 전파하는 것입니다. 브릿지는 양방향입니다. 다른 방향으로 발행물을 전파합니다. 143 페이지의 그림 30에서는 브릿지가 디바이스용 IBM WebSphere MQ Telemetry 디먼을 IBM WebSphere MQ에 연결하는 방식을 보여줍니다. 144 페이지의 『브릿지에 대한 토픽 설정의 예』에서는 토픽 매개변수를 사용하여 브릿지를 구성하는 방법을 보여주기 위한 예를 사용합니다.

143 페이지의 그림 30의 In 및 Out 화살표는 브릿지의 양방향성을 나타냅니다. 화살표의 한쪽 끝에서 구독이 작성됩니다. 구독과 일치하는 발행물은 화살표 반대쪽 끝에 있는 브로커로 구독됩니다. 이 화살표는 발행물의 플로우를 따라 레이블 지정됩니다. 발행물은 디먼으로 플로우인(In)하고 디먼에서 플로우아웃(Out)합니다. 레이블의 중요성은 명령 구문에서 사용된다는 점입니다. In 및 Out은 발행물이 플로우하는 위치를 참조하고 구독이 송신되는 위치는 참조하지 않습니다.

다른 클라이언트, 애플리케이션 또는 브로커는 IBM WebSphere MQ 또는 디바이스용 WebSphere MQ Telemetry 디먼에 연결될 수 있습니다. 이는 연결되는 브로커에서 토픽을 발행 및 구독합니다. 브로커가 IBM WebSphere MQ인 경우 토픽이 클러스터되거나 분배될 수 있으며 로컬 큐 관리자에서 명시적으로 정의되지 않을 수 있습니다.

브릿지 사용

브릿지 연결 및 리스너를 사용하여 디먼을 함께 연결하십시오. 브릿지 연결 및 텔레메트리 채널을 사용하여 디먼과 큐 관리자를 함께 연결하십시오. 다중 브로커를 함께 연결할 경우 루프를 작성할 수 있습니다. 주의하십시오. 발행물은 끊임없이 발견되지 않은 브로커의 루프 주변을 순환할 수 있습니다.

IBM WebSphere MQ에 브릿지된 디먼을 사용하는 몇 가지 이유는 다음과 같습니다.

WebSphere MQ에 대한 MQTT 클라이언트 연결 수 줄이기
디먼의 계층 구조를 사용하면 단일 큐 관리자가 한 번에 연결할 수 있는 수보다 더 많은 클라이언트를 WebSphere MQ에 연결할 수 있습니다.

MQTT 클라이언트와 WebSphere MQ 간에 메시지 저장 후 전달
클라이언트가 자체 저장소를 가지고 있지 않을 경우 클라이언트와 IBM WebSphere MQ 간 지속적인 연결 유지를 피하도록 저장 후 전달을 사용할 수 있습니다. MQTT 클라이언트와 WebSphere MQ 사이에 여러 유형의 연결을 사용할 수 있습니다(모니터링 및 제어 텔레메트리 개념과 시나리오 참조).

MQTT 클라이언트와 WebSphere MQ 간에 교환된 발행물 필터
보통 발행물은 로컬로 처리되는 메시지와 다른 애플리케이션을 포함하는 메시지로 나뉩니다. 로컬 발행물에는 센서와 작동 장치 간의 제어 플로우가 포함될 수 있으며 원격 발행물에는 자료, 상태 및 구성 명령에 대한 요청이 포함될 수 있습니다.

발행물의 토픽 공간 변경
다른 리스너 포트에 연결된 클라이언트의 토픽 문자열이 다른 하나와 상충되지 않도록 하십시오. 다음 예에서는 다른 건물에서 들어오는 미터 자료에 레이블을 지정하기 위해 디먼을 사용합니다. 다른 클라이언트 그룹의 토픽 공간 분리를 참조하십시오.

브릿지에 대한 토픽 설정의 예

원격 브로커에 모든 항목 발행 - 기본값 사용

기본 방향은 out이라 하며, 브릿지는 원격 브로커에 대해 토픽을 발행합니다. topic 매개변수는 토픽 필터를 사용하여 전파되는 토픽을 제어합니다.

브릿지는 MQTT 클라이언트 또는 다른 브로커에 의해 로컬 디먼으로 발행되는 모든 항목을 구독하기 위해 144 페이지의 그림 31의 topic 매개변수를 사용합니다. 브릿지는 브릿지에 의해 연결된 리모트 브로커에 토픽을 발행합니다.

```
connection Daemon1
topic #
```

그림 *31.* 원격 브로커에 모든 항목 발행

원격 브로커에 모든 항목 발행 - 명시

다음 코드 단편의 topic 설정은 기본값을 사용하는 것과 동일한 결과를 나타냅니다. 차이점은 **direction** 매개변수가 명확하다는 점뿐입니다. 로컬 브로커인 디먼을 구독하고 원격 브로커에 발행하려면 out 방향을 사용하십시오. 브릿지가 구독하는 로컬 디먼에서 작성된 발행물은 원격 브로커에서 발행됩니다.

```
connection Daemon1
topic # out
```

그림 *32.* 원격 브로커에 모든 항목 발행 - 명시

로컬 브로커에 모든 항목 발행

out 방향을 사용하는 대신 반대 방향인 in을 설정할 수 있습니다. 다음 코드 단편은 브릿지로 연결된 원격 브로커에서 발행되는 모든 항목을 구독하도록 브릿지를 구성합니다. 브릿지는 로컬 브로커 디먼에 토픽을 발행합니다.

```
connection Daemon1
topic # in
```

그림 *33.* 로컬 브로커에 모든 항목 발행

로컬 브로커의 내보내기 토픽에서 원격 브로커의 가져오기 토픽으로 모든 항목 발행

local_prefix와 **remote_prefix**라는 두 개의 추가 토픽 매개변수를 사용하여 이전 예의 토픽 필터 #을 수정하십시오. 한 매개변수는 구독에서 사용되는 토픽 필터를 수정하는 데 사용되고, 다른 매개변수는 발행물이 발행되는 토픽을 수정하는 데 사용됩니다. 그 영향은 한 브로커에서 사용된 토픽 문자열의 시작을 다른 브로커의 다른 토픽 문자열로 대체하는 것입니다.

토픽 명령의 방향에 따라 **local_prefix** 및 **remote_prefix**의 의미가 뒤바뀝니다. 방향이 기본값인 out인 경우 **local_prefix**는 토픽 구독의 일부로 사용되고 **remote_prefix**가 원격 구독에서 토픽 문자열의 **local_prefix** 부분을 대체합니다. 방향이 in인 경우 **remote_prefix**는 원격 구독의 일부가 되며 **local_prefix**는 토픽 문자열의 **remote_prefix** 부분을 대체합니다.

토픽 문자열의 첫 번째 파트는 토픽 공간을 정의하는 것으로 볼 수 있습니다. 토픽이 발행되는 토픽 공간을 변경하려면 추가 매개변수를 사용하십시오. 전파 중인 토픽이 대상 브로커의 다른 토픽과 충돌하지 않도록 하거나 마운트 지점 토픽 문자열을 제거하기 위해 이를 수행할 수 있습니다.

예를 들어 다음 코드 단편에서는 디먼의 토픽 문자열 export/#에 대한 모든 발행물이 원격 브로커의 import/#에 다시 발행됩니다.

```
topic # out export/ import/
```

그림 *34.* 로컬 브로커의 내보내기 토픽에서 원격 브로커의 가져오기 토픽으로 모든 항목 발행

원격 브로커의 내보내기 항목에서 로컬 브로커의 가져오기 항목으로 모든 항목 발행

다음 코드 단편에서는 반대 구성을 보여줍니다. 브릿지는 원격 브로커에서 export/# 토픽 문자열이 포함되어 발행된 모든 항목을 구독하고 이를 로컬 브로커의 import/#에 발행합니다.

```
connection Daemon1
topic # in import/ export/
```

그림 *35.* 원격 브로커의 내보내기 항목에서 로컬 브로커의 가져오기 항목으로 모든 항목 발행

1884/ 마운트 지점에서 원래 토픽 문자열이 있는 원격 브로커로 모든 항목 발행

다음 코드 단편에서는 브릿지가 로컬 디먼의 마운트 지점 1884/에 연결된 클라이언트가 발행하는 모든 항목을 구독합니다. 브릿지는 마운트 지점에 발행된 모든 항목을 원격 브로커에 발행합니다. 마운트 지점 문자열 1884/는 원격 브로커에 발행된 토픽에서 제거됩니다. *local_prefix*는 마운트 지점 문자열 1884/와 동일하며, *remote_prefix*는 빈 문자열입니다.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

그림 *36. 1884/* 마운트 지점에서 원래 토픽 문자열이 있는 원격 브로커로 모든 항목 발행

다른 디먼에 연결된 다른 클라이언트의 토픽 공간 분리

건물에 대한 미터 자료를 발행하기 위해 전력 미터용으로 애플리케이션이 작성된다고 가정합니다. 자료는 MQTT 클라이언트를 사용하여 동일한 건물에서 호스트된 디먼에 발행됩니다. 발행물을 위해 선택된 토픽은 power입니다. 동일한 애플리케이션이 많은 건물에 복잡하게 배치됩니다. 사이트 모니터링 및 데이터 저장을 위해 모든 건물의 자료는 브릿지 연결을 사용하여 집계됩니다. 연결은 건물 디먼을 중앙 위치의 WebSphere MQ로 링크합니다.

모든 건물에서 동일한 클라이언트 앱이 사용됩니다. 이 앱은 power라는 토픽을 발행합니다. 그러나 건물별로 데이터를 차별화해야 합니다. 이는 건물 번호를 토픽 이름의 접두부로 추가하는 각 건물의 디먼에 의해 수행됩니다. 첫 번째 건물의 브릿지는 접두부 meters/building01/를 사용하고 두 번째 건물에서는 meters/building02/라는 접두부를 사용합니다. 다른 건물의 자료도 동일한 패턴을 따릅니다. 따라서 WebSphere MQ 는 meters/building01/power와 같은 주제로 표시값을 수신합니다.

각 디먼의 구성 파일에는 다음 코드 단편의 패턴을 따르는 토픽 명령문이 있습니다.

```
connection Daemon1
topic power out "" meters/building01/
```

그림 *37.* 다른 디먼에 연결된 클라이언트의 토픽 공간 분리

이전 코드 단편에서는 비어 있는 문자열이 사용하지 않는 local_prefix 매개변수의 플레이스홀더입니다.

참고: 이 예는 다소 인위적이며 단지 예시를 위한 것입니다. 사실상 애플리케이션이 발행하는 토픽 공간은 구성 가능합니다.

동일한 디먼에 연결된 클라이언트의 토픽 공간 분리

단일 디먼이 모든 파워 미터를 연결하는 데 사용된다고 가정합니다. 애플리케이션에서 다른 포트에 연결하도록 구성할 수 있다고 가정하면 다음 코드 단편에서처럼 다른 건물에서 다른 리스너 포트로 미터를 연결하여 건물을 구분할 수 있습니다. 즉, 이 예에서는 마운트 지점이 사용되는 방법을 보여줍니다.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

그림 *38.* 동일한 디먼에 연결된 클라이언트의 토픽 공간 분리

두 방향으로 송신되는 발행물에 대해 다른 토픽 다시 맵핑

다음 코드 조각의 구성에서 브릿지는 원격 브로커의 단일 토픽 b 에 등록하고 b 에 대한 발행물을 로컬 디먼으로 전달하여 토픽을 a로 변경합니다. 브릿지는 로컬 브로커의 단일 토픽 x 에 등록하고 x 에 대한 발행물을 원격 브로커에 전달하여 토픽을 y로 변경합니다.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

그림 *39.* 두 방향으로 송신되는 발행물에 대해 다른 토픽 다시 맵핑

이 예에서 중요한 점은 다른 토픽이 두 브로커 모두에서 구독 및 발행된다는 점입니다. 두 브로커의 토픽 공간은 떨어져 있습니다.

두 방향으로 송신되는 발행물에 대해 동일한 토픽 다시 맵핑(루핑)

이전 예와 달리 일반적으로 147 페이지의 그림 40의 구성은 루프가 됩니다. 토픽 명령문 `topic "" in a b`에서는 브릿지가 원격으로 b를 구독하고 로컬로 a를 발행합니다. 다른 토픽 명령문에서는 브릿지가 로컬로 a를 구독하고 원격으로 b를 발행합니다. 147 페이지의 그림 41에 표시된 것처럼 동일한 구성을 작성할 수 있습니다.

일반적인 결과는 클라이언트가 b 에 원격으로 공개하는 경우 발행물이 a토픽에 대한 발행물로서 로컬 디먼으로 전송됩니다. 그러나 브릿지에 의해 a토픽의 로컬 디먼에 발행되는 경우 발행물은 브릿지가 로컬 토픽 a에 대해 작성한 구독과 일치합니다. 등록은 `topic "" out a b`입니다. 결과적으로 발행물은 b토픽에 대한 발행물로서 원격 브로커로 다시 전송됩니다. 이제 브릿지가 원격 토픽 b에 등록되고 순환이 다시 시작됩니다.

일부 브로커는 루프가 발생하지 않도록 하기 위해 루프 감지를 구현합니다. 그러나 루프 감지 메커니즘은 다른 유형의 브로커가 함께 브릿지될 때 작동해야 합니다. WebSphere MQ가 디바이스용 WebSphere MQ Telemetry 디먼에 브릿지될 경우 루프 감지가 작동하지 않습니다. 두 개의 디바이스용 IBM WebSphere MQ Telemetry 디먼이 함께 브릿지되는 경우에는 작동합니다. 기본적으로 루프 감지는 켜져 있습니다 (try_private 참조).

```
connection Daemon1
topic "" in a b
topic "" out a b
```

그림 *40. !*양방향으로 플로우되는 발행물에 대해 동일한 토픽을 다시 맵핑합니다.

```
connection Daemon1
topic "" both a b
```

그림 *41. !both*를 사용하여 양방향으로 플로우되는 발행물에 대해 동일한 주제를 다시 맵핑하십시오.

147 페이지의 그림 39의 구성은 147 페이지의 그림 40과 동일합니다.

IBM WebSphere MQ Telemetry daemon for devices 브릿지 연결의 가용성

사용 가능한 첫 번째 원격 브로커에 연결하도록 다중 IBM WebSphere MQ Telemetry daemon for devices 브릿지 연결 주소를 구성하십시오. 브로커가 멀티 인스턴스 큐 관리자인 경우 해당 TCP/IP 주소를 둘 다 제공하십시오. 사용 가능한 경우 1차 서버에 대해 연결 또는 다시 연결할 1차 연결을 구성하십시오.

연결 브릿지 매개변수 addresses는 TCP/IP 소켓 주소의 목록입니다. 브릿지는 연결에 성공할 때까지 각 주소에 대한 연결을 차례로 시도합니다. round_robin 및 start_type 연결 매개변수는 연결에 성공하고 난 후 주소 사용 방법을 제어합니다.

start_type이 auto, manual 또는 lazy인 경우, 이로 인해 연결이 실패할 경우, 브릿지가 재연결을 시도합니다. 이는 각 연결 시도 간에 약 20초 지연되며 차례로 각 주소를 사용합니다. `start_type`이 once인 경우, 이로 인해 연결이 실패할 경우, 브릿지는 자동으로 재연결을 시도하지 않습니다.

round_robin이 true이면 브릿지 연결 시도는 목록에 있는 첫 주소에서 시작하며 차례로 목록에 있는 각 주소를 시도합니다. 목록이 끝나면 첫 번째 주소에서 다시 시작합니다. 목록에 주소가 하나뿐인 경우 매 20초마다 다시 시도합니다.

round_robin이 false인 경우 1차 서버에서 호출되는 목록의 첫 번째 주소가 우선순위를 갖습니다. 1차 서버로의 연결을 위한 첫 번째 시도가 실패할 경우, 브릿지는 계속해서 백그라운드에서 1차 서버로 재연결을 시도합니다. 동시에 브릿지는 목록에 있는 다른 주소를 사용하여 연결을 시도합니다. 백그라운드가 1차 서버에 대한 연결 시도에 성공하면 브릿지가 현재 연결로부터 연결을 끊고 1차 서버 연결로 전환합니다.

연결이 자체적으로 끊어질 경우(예: **connection_stop** 명령에 의해) 연결이 재시작되면 동일한 주소를 다시 사용하도록 시도합니다. 연결 실패로 인해 또는 원격 브로커가 연결을 삭제하여 연결이 끊어질 경우 브릿지는 20초 동안 대기합니다. 그런 다음 목록에 있는 다음 주소로 연결을 시도하며, 목록에 주소가 하나뿐인 경우 동일한 주소로 연결을 시도합니다.

멀티 인스턴스 큐 관리자에 연결

멀티 인스턴스 큐 관리자 구성에서 큐 관리자는 다른 IP 주소를 가진 두 개의 다른 서버에서 실행됩니다. 일반적으로 텔레메트리 채널은 특정 IP 주소 없이 구성됩니다. 이러한 채널은 포트 번호로만 구성됩니다. 텔레메트리 채널이 시작되면 기본적으로 로컬 서버에서 사용 가능한 첫 번째 네트워크 주소를 선택합니다.

큐 관리자에서 사용된 두 개의 IP 주소로 브릿지 연결의 addresses 매개변수를 구성하십시오. round_robin을 true로 설정하십시오.

활성 큐 관리자 인스턴스가 실패하면 큐 관리자는 대기 인스턴스로 변경합니다. 디먼은 활성 인스턴스에 대한 연결이 중단되었음을 감지하고 대기 인스턴스로 다시 연결을 시도합니다. 이 경우 브릿지 연결용으로 구성된 주소 목록에 있는 다른 IP 주소를 사용합니다.

브릿지가 연결되는 큐 관리자는 여전히 동일한 큐 관리자입니다. 큐 관리자는 자체 상태를 복구합니다. cleansession이 False로 설정되면 브릿지 연결 세션이 장애 복구 이전과 동일한 상태로 복원됩니다. 지연 후 연결이 재개됩니다. "한 번 이상" 또는 "최대 한 번" 서비스 품질이 있는 메시지는 유실되지 않으며 등록은 계속 작동합니다.

재연결 횟수는 대기 인스턴스가 시작될 때 재시작하는 채널 및 클라이언트 수와 메시지가 발송되는 수에 따라 달라집니다. 브릿지 연결은 연결이 재설정되기 전에 여러 번 두 IP 주소 모두에 대해 다시 연결을 시도할 수 있습니다.

특정 IP 주소로 멀티 인스턴스 큐 관리자 텔레메트리 채널을 구성하지 마십시오. IP 주소는 하나의 서버에서만 유효합니다.

대체 고가용성 솔루션을 사용 중인 경우 이는 IP 주소를 관리하므로 특정 IP 주소로 텔레메트리 채널을 구성하는 것이 옳을 수 있습니다.

cleansession

브릿지 연결은 MQTT v3 클라이언트 세션입니다. 연결이 새 세션을 시작하는지 여부 또는 기존 세션을 복원하는지 여부를 제어할 수 있습니다. 기존 세션을 복원할 경우 브릿지 연결은 이전 세션의 구독 및 보유된 발행물을 보존합니다.

addresses에 다중 IP 주소가 나열되고 IP 주소가 다른 큐 관리자에 의해 호스트된 텔레메트리 채널 또는 다른 텔레메트리 디먼에 연결되는 경우 cleansession을 False로 설정하지 마십시오. 세션 상태는 큐 관리자 또는 디먼 간에 전송되지 않습니다. 다른 큐 관리자 또는 디먼에서 기존 세션을 재시작하려고 하면 새 세션이 시작됩니다. 인다우트 메시지가 유실되고 구독이 예상대로 작동하지 않을 수 있습니다.

알림

애플리케이션은 브릿지 연결이 알림을 사용하여 실행 중인지 여부를 계속해서 추적할 수 있습니다. 알림은 연결된 경우 1, 연결되지 않은 경우 0이라는 값을 가진 발행입니다. notification_topic 매개변수에 의해 정의된 *topicString* 에 공개됩니다. *topicString* 의 기본값은 $SYS/broker/connection/ *clientIdentifier*/state 입니다. 기본값 *topicString*에는 접두부 $SYS가 포함되어 있습니다. $SYS로 시작하는 토픽 필터를 정의하여 $SYS로 시작하는 토픽을 구독하십시오. 토픽 필터 #(모두 구독)은 디먼에서 $SYS로 시작하는 토픽을 구독하지 않습니다. $SYS를 애플리케이션 토픽 공간과 별개의 특수 시스템 토픽 공간을 정의하는 것으로 간주하십시오.

브릿지가 연결되거나 연결이 끊길 때 알림을 사용하여 IBM WebSphere MQ Telemetry daemon for devices가 MQTT 클라이언트에 알립니다.

keepalive_interval
keepalive_interval 브릿지 연결 매개변수는 TCP/IP ping을 원격 서버로 송신하는 브릿지 사이의 간격을 설정합니다. 기본 간격은 60초입니다. ping은 TCP/IP 세션이 원격 서버 또는 방화벽에 의해 닫히지 않도록 해주며, 연결 시 비활성 기간을 감지합니다.

clientid
브릿지 연결은 MQTT v3 클라이언트 세션으로, 브릿지 연결 매개변수 clientid에 의해 설정된 `clientIdentifier`를 가집니다. cleansession 매개변수를 false로 설정하여 이전 세션을 재개하기 위해 다시 연결을 시도할 경우 각 세션에 사용된 `clientIdentifier`는 동일해야 합니다. clientid의 기본값은 동일하게 남아 있는 *hostname.connectionName*입니다.

디바이스용 WebSphere MQ Telemetry 디먼의 설치, 확인, 구성 및 제어
디먼의 설치, 구성 및 제어는 파일 기반입니다.

디먼을 실행하려는 디바이스에 SDK(Software Development Kit)를 복사하여 디먼을 설치하십시오.

예를 들어, MQTT 클라이언트 유틸리티를 실행하고 발행/구독 브로커로 사용되는 디바이스용 WebSphere MQ Telemetry 디먼에 연결하십시오(디바이스용 WebSphere MQ Telemetry 디먼을 발행/구독 브로커로 사용 참조).

구성 파일을 작성하여 디먼을 구성하십시오(디바이스용 MQ Telemetry 디먼 구성 파일 참조).

`amqtdd.upd` 파일에서 명령을 작성하여 실행 중인 디먼을 제어하십시오. 5초마다 디먼이 파일을 읽고, 명령을 실행하고 해당 파일을 삭제합니다(디바이스용 WebSphere MQ Telemetry 디먼 명령 파일 참조).

디바이스용 WebSphere MQ Telemetry 디먼 리스너 포트
리스너 포트를 사용하여 디바이스용 WebSphere MQ Telemetry 디먼에 MQTT V3 클라이언트를 연결하십시오. 마운트 지점 및 최대 연결 수로 리스너 포트에 권한을 부여할 수 있습니다.

리스너 포트는 이 포트에 연결되어 있는 클라이언트의 MQTT 클라이언트 `connect(serverURI)` 메소드에 지정된 포트 번호를 따라야 합니다. 이는 클라이언트와 디먼 모두에서 기본값을 1883로 지정합니다.

디먼 구성 파일에서 글로벌 정의 `port`를 설정하면 디먼에 대한 기본 포트를 변경할 수 있습니다. `listener` 정의를 디먼 구성 파일에 추가하여 특정 포트를 설정할 수 있습니다.

기본 포트가 아닌 각 리스너 포트의 경우 마운트 지점을 지정하여 클라이언트를 분리할 수 있습니다. 마운트 지점이 있는 포트에 연결된 클라이언트는 다른 클라이언트와 분리됩니다. 150 페이지의 『디바이스용 WebSphere MQ Telemetry 디먼 마운트 지점』의 내용을 참조하십시오.

임의의 포트에 연결할 수 있는 클라이언트의 수를 제한할 수 있습니다. 기본 포트에 대한 연결을 제한하도록 글로벌 정의 `max_connections`를 설정하거나 각 리스너 포트에 `max_connections` 권한을 부여하십시오.

예

기본 포트를 1883에서 1880으로 변경하고 포트 1880 - 10000으로 연결을 제한하는 구성 파일의 예입니다. 포트 1884에 대한 연결은 1000으로 제한됩니다. 포트 1884에 연결된 클라이언트는 다른 포트에 연결된 클라이언트로부터 분리됩니다.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

디바이스용 WebSphere MQ Telemetry 디먼 마운트 지점

MQTT 클라이언트에서 사용하는 리스너 포트와 마운트 지점을 연관시켜 디바이스용 WebSphere MQ Telemetry 디먼에 연결할 수 있습니다. 마운트 지점은 다른 리스너 포트에 연결된 MQTT 클라이언트의 리스너 포트 하나를 사용하여 MQTT 클라이언트에 의해 교환된 발행물 및 구독을 격리합니다.

마운트 지점이 있는 리스너 포트에 연결된 클라이언트는 다른 리스너 포트에 연결된 클라이언트와 절대로 직접 토픽을 교환할 수 없습니다. 마운트 지점이 없는 리스너 포트에 연결된 클라이언트는 클라이언트의 토픽을 발행 또는 구독할 수 있습니다. 클라이언트는 마운트 지점을 통해 연결되어 있는지 여부를 인식하지 못합니다. 이는 클라이언트에 의해 작성된 토픽 문자열과 차이가 없습니다.

마운트 지점은 발행물과 구독의 토픽 문자열에 접두부로 지정되는 텍스트의 문자열입니다. 마운트 지점이 있는 리스너 포트에 연결된 클라이언트에 의해 작성된 모든 토픽 문자열에는 접두부가 지정됩니다. 텍스트 문자열은 리스터 포트에 연결된 클라이언트로 송신된 모든 토픽 문자열에서 제거됩니다.

리스너 포트에 마운트 지점이 없으면 포트에 연결된 클라이언트에 의해 작성되고 수신된 발행물 및 구독의 토픽 문자열이 대체되지 않습니다.

후미 문자 /가 포함된 마운트 지점 문자열을 작성하십시오. 해당 마운트 지점은 마운트 지점에 대한 토픽 트리의 상위 토픽입니다.

예

구성 파일에는 다음과 같은 리스너 포트가 포함되어 있습니다.

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

포트 1883에 연결된 클라이언트는 MyTopic에 대한 구독을 작성합니다. 디먼은 1883/MyTopic으로서 구독을 등록합니다. 포트 1883에 연결된 다른 클라이언트는 토픽 MyTopic에 메시지를 발행합니다. 디먼은 토픽 문자열을 1883/MyTopic으로 변경하고 구독과의 일치를 검색합니다. 포트 1883의 구독자는 원래 토픽 문자열 MyTopic이 있는 발행물을 수신합니다. 디먼이 토픽 문자열에서 마운트 지점 접두부를 제거했습니다.

포트 1884에 연결된 다른 클라이언트도 토픽 MyTopic을 발행합니다. 이 때는 디먼이 토픽을 1884/MyTopic으로 구독합니다. 다른 마운트 지점에는 다른 토픽 문자열이 있는 구독이 있으므로 포트 1883의 구독자는 해당 발행물을 수신하지 않습니다.

포트 1885에 연결된 클라이언트는 토픽 1883/MyTopic을 발행합니다. 디먼은 토픽 문자열을 변경하지 않습니다. 포트 1883의 구독자는 MyTopic에 대한 발행물을 수신합니다.

디바이스용 WebSphere MQ Telemetry 디먼 QoS(Quality of Service), 지속적 구독 및 보유된 발행

QoS(Quality of Service) 설정은 실행 중인 디먼에만 적용됩니다. 디먼이 중지되면 제어되는 방식으로든 아니면 실패로 인해서든 전달 중인 메시지의 상태가 손실됩니다. 디먼이 중지되면 최소 한 번 또는 최대 한 번이라는 메시지 전달을 보장할 수 없습니다. 디바이스용 WebSphere MQ Telemetry 디먼은 제한된 지속성을 지원합니다. 디먼이 종료될 경우 보유된 발행 및 구독을 저장하도록 **retained_persistence** 구성 매개변수를 설정하십시오.

WebSphere MQ와 달리 용 WebSphere MQ Telemetry 디먼은 지속적 데이터를 저널하지 않습니다. 세션 상태, 메시지 상태 및 보유된 발행은 트랜잭션 방식으로 저장되지 않습니다. 기본적으로 디먼은 중지 시 모든 데이터를 제거합니다. 정기적으로 체크포인트 구독 및 보유된 발행에 대한 옵션을 설정할 수 있습니다. 메시지 상태는 디먼이 중지되면 항상 손실됩니다. 보유되지 않는 모든 발행은 손실됩니다.

정기적으로 보유된 발행을 파일에 저장하려면 디먼 구성 옵션 Retained_persistence를 true로 설정하십시오. 디먼이 재시작되면 마지막으로 자동 저장된 보유된 발행물이 복원됩니다. 기본적으로 클라이언트가 작성한 보유 메시지는 디먼 재시작 시 복원되지 않습니다.

정기적으로 지속 세션에서 작성된 구독을 파일에 저장하려면 디먼 구성 옵션 Retained_persistence를 true로 설정하십시오. Retained_persistence가 true로 설정되고 CleanSession이 있는 세션에서 클

라이언트가 작성하는 구독이 false로 설정된 경우 "지속 세션"이 복원됩니다. 디먼이 재시작되면 구독을 복원하여 발행물 수신을 시작합니다. 클라이언트는 CleanSession을 false로 하여 다시 시작할 때 발행물을 수신합니다. 기본적으로 디먼이 중지하면 클라이언트 세션 상태가 저장되지 않습니다. 따라서 클라이언트가 CleanSession을 false로 설정하더라도 구독은 복원되지 않습니다.

Retained_persistence는 자동 저장 메커니즘입니다. 따라서 가장 최근에 보유된 발행물 또는 구독은 저장할 수 없습니다. 사용자는 보유된 발행물 및 구독이 저장되는 빈도를 변경할 수 있습니다. 구성 옵션 autosave_on_changes 및 autosave_interval을 사용하여 저장 사이의 간격 또는 저장 사이의 변경 수를 설정하십시오.

설정 지속성에 대한 구성의 예

```
# Sample configuration
# Daemon listens on port 1882 with persistence in /tmp
# Autosave every minute
port 1882
persistence_location /tmp/
retained_persistence true
autosave_on_changes false
autosave_interval 60
```

디바이스용 WebSphere MQ Telemetry 디먼 보안

디바이스용 WebSphere MQ Telemetry 디먼은 이에 연결하는 클라이언트를 인증하고, 신임 정보를 사용하여 다른 브로커에게 연결하며, 토픽에 대한 액세스를 제어합니다. 디먼이 제공하는 보안은 SSL 지원을 제공하지 않는 WebSphere MQ Telemetry C 클라이언트를 사용하여 빌드되도록 제한됩니다. 결과적으로 디먼에 대한 연결은 암호화되지 않으며 인증서를 사용하여 인증할 수 없게 됩니다.

기본적으로 보안은 켜져 있지 않습니다.

클라이언트 인증

MQTT 클라이언트는 메소드 MqttConnectOptions.setUserName 및 MqttConnectOptions.setPassword를 사용하여 사용자 이름 및 비밀번호를 설정할 수 있습니다.

비밀번호 파일의 항목에 대응하는 클라이언트에 의해 제공된 사용자 이름 및 비밀번호를 검사하여 디먼에 연결하는 클라이언트를 인증하십시오. 인증을 사용 가능하게 하려면 비밀번호 파일을 작성하고 디먼 구성 파일에서 password_file 매개변수를 설정하십시오(password_file 참조).

인증을 검사 중인 디먼에 연결하기 위해 사용자 이름 또는 비밀번호 없이 클라이언트 연결을 허용하도록 디먼 구성 파일의 allow_anonymous 매개변수를 설정하십시오(allow_anonymous 참조). 클라이언트가 사용자 이름 또는 비밀번호를 제공할 경우 password_file 매개변수가 설정되면 사용자 이름 또는 비밀번호가 비밀번호 파일에 대응하여 항상 검사됩니다.

특정 클라이언트에 대한 연결을 제한하려면 디먼 구성 파일의 clientid_prefixes 매개변수를 설정하십시오. 클라이언트는 clientid_prefixes 매개변수에 나열된 접두부 중 하나로 시작하는 clientIdentifiers가 있어야 합니다(clientid_prefixes 참조).

브릿지 연결 보안

디바이스용 각 WebSphere MQ Telemetry 디먼 브릿지 연결은 MQTT V3 클라이언트입니다. 디먼 구성 파일에서 브릿지 연결 매개변수로 각 브릿지 연결을 위한 사용자 이름 및 비밀번호를 설정할 수 있습니다(username 및 password 참조). 그러면 브릿지가 브로커에 대해 자체 인증할 수 있습니다.

토픽의 액세스 제어

클라이언트가 인증되는 중인 경우 디먼도 토픽에 대한 액세스 제어를 각 사용자에게 제공할 수 있습니다. 디먼은 클라이언트가 액세스 제어 파일에 액세스 토픽 문자열이 있는 발행 또는 구독 중 하나인 토픽과의 일치를 기반으로 액세스 제어 권한을 부여합니다(acl_file 참조).

액세스 제어 목록(ACL)은 두 부분으로 나눠집니다. 첫 번째 부분은 익명 클라이언트를 포함한 모든 클라이언트에 대한 액세스를 제어합니다. 두 번째 부분은 비밀번호 파일에 임의 사용자에 대한 섹션을 가집니다. 여기에는 각 사용자에 대한 특정 액세스 제어가 나열됩니다.

예

보안 매개변수는 다음 예에 표시됩니다.

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

그림 *42.* 디먼 구성 파일

```
Fred:Fredpassword
Barney:Barneypassword
```

그림 *43. Password file, passwords.txt*

```
topic home/public/#
topic read meters/#
user Fred
topic write meters/fred
topic home/fred/#
user Barney
topic write meters/barney
topic home/barney/#
```

그림 *44.* 액세스 제어 파일 *acl.txt*

MQTT 클라이언트 문제점 해결

MQTT 클라이언트를 실행하면서 발생하는 문제점을 해결하는 데 유용한 문제점 해결 태스크를 찾아봅니다.

관련 태스크

160 페이지의 『MQTT (Paho) Java 클라이언트 추적 및 디버깅』
기본 로거는 `java.util.logging` (JSR47) 라고 하는 표준 Java 로깅 기능을 사용합니다. 구성 파일을 사용하거나 프로그래밍 방식으로으로 이를 구성할 수 있습니다.

163 페이지의 『MQTT JavaScript 클라이언트 추적』
JavaScript 클라이언트를 사용하는 경우 연결된 클라이언트 오브젝트에서 메소드를 호출하도록 클라이언트 웹 애플리케이션을 변경하여 추적을 수집할 수 있습니다.

156 페이지의 『텔레메트리(MQXR) 서비스 추적』
다음 지시사항에 따라 텔레메트리 서비스 추적을 시작하고 추적을 제어하는 매개변수를 설정하고 추적의 출력을 찾으십시오.

158 페이지의 『MQTT v3 Java 클라이언트 추적』
MQTT Java 클라이언트 추적을 작성하고 그 결과물을 제어하려면 다음 지시사항을 따르십시오.

159 페이지의 『C용 MQTT 클라이언트 추적』
MQTT 클라이언트 C 앱을 추적하도록 환경 변수 `MQTT_C_CLIENT_TRACE`를 설정합니다.

168 페이지의 『문제점 해결: MQTT 클라이언트가 연결되지 않음』
텔레메트리(MQXR) 서비스에 연결되지 않는 MQTT 클라이언트 프로그램의 문제점을 해결합니다.

169 페이지의 『문제점 해결: MQTT 클라이언트 연결이 삭제됨』

클라이언트가 짧은 시간 또는 긴 시간 동안 연결하여 실행한 후에 예상치 못한 ConnectionLost 예외를 처리하는 원인을 알아냅니다.

170 페이지의 『문제점 해결: MQTT 애플리케이션에서 메시지가 손실됨』
메시지 손실로 인한 문제점을 해결하십시오. 메시지가 비지속적이거나 잘못된 위치로 전송되었거나 송신되지 않았습니까? 잘못 코드화된 클라이언트 프로그램에서는 메시지가 손실될 수 있습니다.

172 페이지의 『문제점 해결: 텔레메트리(MQXR) 서비스가 시작되지 않음』
텔레메트리(MQXR) 서비스가 시작되지 않는 문제를 해결합니다. WebSphere MQ Telemetry 설치를 검사하고 누락되거나, 이동되거나, 잘못된 권한을 가진 파일이 없는지 확인하십시오. 텔레메트리(MQXR) 서비스에서 사용하는 경로가 텔레메트리(MQXR) 서비스 프로그램을 찾는지 확인하십시오.

173 페이지의 『문제점 해결: 텔레메트리 서비스가 JAAS 로그인 모듈을 호출하지 않음』
텔레메트리(MQXR) 서비스가 JAAS 로그인 모듈을 호출 중이지 않은지 알아내고 문제점을 수정하도록 JAAS를 구성합니다.

176 페이지의 『문제점 해결: 디먼 시작 또는 실행』
디바이스용 IBM WebSphere MQ Telemetry 디먼 콘솔 로그를 참조하거나 추적을 켜거나 이 토픽의 증상 표를 사용하여 디먼에 관련된 문제점을 해결하십시오.

177 페이지의 『문제점 해결: MQTT 클라이언트가 디먼에 연결되지 않음』
클라이언트가 디먼에 연결되지 않거나 디먼이 다른 디먼 또는 WebSphere MQ Telemetry 채널에 연결되지 않습니다.

관련 참조

153 페이지의 『텔레메트리 로그, 오류 로그 및 구성 파일의 위치』
IBM WebSphere MQ Telemetry에서 사용되는 로그, 오류 로그 및 구성 파일을 찾으십시오.

155 페이지의 『MQTT v3 Java 클라이언트 이유 코드』
MQTT v3 Java 클라이언트 예외에서 이유 코드의 원인을 찾아보거나 행할 수 있습니다.

164 페이지의 『MQTT 클라이언트와 함께 SHA-2 암호 스위트를 사용하기 위한 시스템 요구사항』
Java 6의 경우 IBM, SR13 이상에서 SHA-2 암호 스위트를 사용하여 MQTT 채널 및 클라이언트 애플리케이션을 보호할 수 있습니다. 그러나 SHA-2 암호 스위트는 기본적으로 IBM, SR4 이상에서 Java 7까지 기본적으로 사용 가능하지 않으므로 이전 버전에서는 필수 스위트를 지정해야 합니다. 고유한 JRE를 포함하는 MQTT 클라이언트를 실행하는 경우 SHA-2 암호 스위트를 지원하는지 확인해야 합니다. 클라이언트 앱에서 SHA-2 암호 스위트를 사용하려면 클라이언트에서도 TLS(Transport Layer Security) 버전 1.2를 지원하는 값으로 SSL 컨텍스트를 설정해야 합니다.

164 페이지의 『SSL을 통한 모바일 메시징 웹 앱에 대한 브라우저 지원의 제한사항』
다른 플랫폼에서는 다른 브라우저 간에 기능의 차이가 있습니다. 이러한 차이를 이해하면 SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets를 사용하여 연결하도록 애플리케이션, 인증서 권한 (CAs) 및 클라이언트 인증서를 구성하는 데 도움이 됩니다.

텔레메트리 로그, 오류 로그 및 구성 파일의 위치

IBM WebSphere MQ Telemetry에서 사용되는 로그, 오류 로그 및 구성 파일을 찾으십시오.

참고: 이 예는 Windows용으로 코드화되었습니다. Linux 에서 예제를 실행하도록 구문을 변경하십시오.

서버 측 로그

IBM WebSphere MQ Telemetry의 설치 마법사는 해당 설치 로그에 메시지를 기록합니다.

```
WMQ program directory\mqxr
```

텔레메트리(MQXR) 서비스는 WebSphere MQ 큐 관리자 오류 로그에는 메시지를 기록하고 IBM WebSphere MQ 오류 디렉토리에는 FDC 파일을 기록합니다.

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

또한 텔레메트리(MQXR) 서비스에 대한 로그도 기록합니다. 이 로그는 서비스 시작 시의 특성과 MQTT 클라이언트에 대한 프록시 역할을 하면서 찾은 오류를 표시합니다. 예를 들어, 클라이언트가 작성하지 않은 구독을 구독 취소합니다. 로그 경로는 다음과 같습니다.

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

IBM WebSphere MQ Explorer에 의해 작성된 IBM WebSphere MQ 텔레메트리 샘플 구성은 **runMQXRService** 명령을 사용하여 텔레메트리 서비스를 시작합니다. **runMQXRService**는 *WMQ Telemetry install directory*\bin에 있습니다. 다음과 같은 경로에 기록합니다.

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

텔레메트리(MQXR) 서비스에 대해 구성된 경로를 표시하거나 텔레메트리(MQXR) 서비스를 시작하기 전에 초기화를 에코하려면 **runMQXRService**를 수정하십시오.

서버 측 구성 파일

텔레메트리 채널 및 텔레메트리(MQXR) 서비스

제한사항: 텔레메트리 채널 구성 파일의 형식, 위치, 컨텐츠 및 해석은 추후 릴리스에서 변경될 수 있습니다. 텔레메트리 채널을 구성하려면 IBM WebSphere MQ Explorer를 사용해야 합니다.

IBM WebSphere MQ Explorer는 mqxr_win.properties 파일 및 Linux의 mqxr_unix.properties 파일에 텔레메트리 구성을 저장합니다. 특성 파일은 텔레메트리 구성 디렉토리에 저장됩니다.

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

그림 *45. Windows*의 텔레메트리 구성 디렉토리

```
/var/mqm/qmgrs/qMgrName/mqxr
```

그림 *46. Linux* 의 텔레메트리 구성 디렉토리

JVM

java.properties파일에서 텔레메트리 (MQXR) 서비스에 인수로 전달되는 Java 특성을 설정하십시오. 파일의 특성은 텔레메트리(MQXR) 서비스를 실행하는 JVM에 직접 전달됩니다. 자바 명령행에서 추가 JVM 특성으로 전달된다. 명령행에 설정된 특성은 java.properties 파일로부터 명령행에 추가된 특성에 대해 우선순위를 갖습니다.

텔레메트리 구성과 동일한 폴더에서 java.properties 파일을 찾으십시오. 154 페이지의 그림 45 및 154 페이지의 그림 46의 내용을 참조하십시오.

각 특성을 별도의 행으로 지정하여 java.properties를 수정하십시오. 각 특성을 JVM에 인수로서 전달하고자 하는 양식에 정확히 맞춰 양식화하십시오. 예:

```
-Xmx1024m
-Xms1024m
```

JAAS

JAAS 구성 파일은 텔레메트리 채널 JAAS 구성 에 설명되어 있으며, 여기에는 IBM WebSphere MQ 텔레메트리와 함께 제공되는 샘플 JAAS 구성 파일 JAAS.config가 포함되어 있습니다.

JAAS를 구성하는 경우 대개 표준 JAAS 인증 프로시저를 대체하기 위해 사용자를 인증할 클래스를 작성하게 됩니다.

텔레메트리(MQXR) 서비스 클래스 경로에서 사용되는 경로에 Login 클래스를 포함시키려면 WebSphere MQ service.env 구성 파일을 제공하십시오.

service.env에서 JAAS LoginModule에 대한 클래스 경로를 설정하십시오. service.env에는 변수 %classpath%를 사용할 수 없습니다. service.env의 클래스 경로가 텔레메트리(MQXR) 서비스 정의에 이미 설정된 클래스 경로에 추가되었습니다.

`echo set classpath`를 `runMQXRService.bat`에 추가하여 텔레메트리(MQXR) 서비스에서 사용 중인 클래스 경로를 표시하십시오. 출력은 `mqxr.stdout`으로 송신됩니다.

`service.env` 파일의 기본 위치는 다음과 같습니다.

```
WMQ data directory\service.env
```

다음에서 각 큐 관리자에 대해 이러한 설정을 `service.env` 파일로 대체하십시오.

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

155 페이지의 그림 47 에서는 샘플 `LoginModule.class`를 사용하는 샘플 `service.env` 파일을 보여줍니다.

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

참고: `service.env`는 변수를 포함할 수 없습니다. *WMQ Install Directory*의 실제 값을 대체하십시오.

그림 *47. service.env*

추적

IBM 서비스 기술자가 추적을 구성해달라고 부탁할 경우가 있습니다. 156 페이지의 『텔레메트리(MQXR) 서비스 추적』의 내용을 참조하십시오. 구성된 추적의 매개변수는 다음 두 파일에 저장됩니다.

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

클라이언트 측 로그 파일

IBM WebSphere MQ 텔레메트리와 함께 제공된 Java SE MQTT 클라이언트의 기본 파일 지속성 클래스는 클라이언트 작업 디렉토리에 이름이 *clientIdentifier*-tcp*hostName*포트 또는 *clientIdentifier*-ssl*hostName*포트인 폴더를 작성합니다. 폴더 이름은 연결 시도에 사용되는 hostName 및 port를 알려줍니다.. 이 폴더에는 지속성 클래스가 저장한 메시지가 포함되어 있습니다. 전달이 완료되면 이 메시지는 삭제됩니다.

폴더는 새 세션이 있는 클라이언트가 종료되면 삭제됩니다.

클라이언트 추적이 켜져 있는 경우, 형식화되지 않은 로그는 기본적으로 클라이언트 작업 디렉토리에 저장됩니다. 추적 파일은 `mqtt-n.trc`라고 합니다.

클라이언트 측 구성 파일

자바 특성 파일을 사용하여 MQTT 자바 클라이언트에 대한 추적 및 SSL 특성을 설정하거나 특성을 프로그래밍 방식으로 설정한다. JVM `-D` 스위치를 사용하여 특성을 MQTT Java 클라이언트로 전달하십시오. 예를 들어,

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

158 페이지의 『MQTT v3 Java 클라이언트 추적』의 내용을 참조하십시오.MQTT 클라이언트 라이브러리에 대한 클라이언트 API 문서에 대한 링크는 MQTT 클라이언트 프로그래밍 참조를 참조하십시오.

MQTT v3 Java 클라이언트 이유 코드

MQTT v3 Java 클라이언트 예외에서 이유 코드의 원인을 찾아보거나 행할 수 있습니다.

표 *5. MQTT v3 Java* 클라이언트 이유 코드

이유 코드	가치	원인
REASON_CODE_BROKER_UNAVAILABLE	3	

표 5. *MQTT v3 Java* 클라이언트 이유 코드 *(계속)*

이유 코드	가치	원인
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	클라이언트가 이미 연결되어 있습니다.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	클라이언트의 연결이 이미 끊어졌습니다.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	MqttCallback의 메소드에서 MqttClient.disconnect 호출을 시도한 경우 처리됩니다.
REASON_CODE_CLIENT_DISCONNECTING	32102	클라이언트는 현재 연결을 끊는 중이므로 새 작업을 승인할 수 없습니다.
REASON_CODE_CLIENT_EXCEPTION	0	클라이언트가 예외를 발견했습니다.
REASON_CODE_CLIENT_NOT_CONNECTED	32104	클라이언트가 서버에 연결되지 않았습니다.
REASON_CODE_CLIENT_TIMEOUT	32000	서버 응답 대기 중에 클라이언트 제한시간이 초과되었습니다.
REASON_CODE_FAILED_AUTHENTICATION	4	사용자 이름 또는 비밀번호가 잘못되어 서버에 대한 인증에 실패했습니다.
REASON_CODE_INVALID_CLIENT_ID	2	제공한 클라이언트 ID를 서버가 거부했습니다.
REASON_CODE_INVALID_PROTOCOL_VERSION	1	요청된 프로토콜 버전이 서버에서 지원되지 않습니다.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	사용 가능한 새 메시지 ID가 없어 내부 오류가 발생했습니다.
REASON_CODE_NOT_AUTHORIZED	5	요청된 조작을 수행할 권한이 없습니다.
REASON_CODE_SERVER_CONNECT_ERROR	32103	서버에 연결할 수 없습니다.
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	서버 URI와 제공된 SocketFactory가 일치하지 않습니다.
REASON_CODE_SSL_CONFIG_ERROR	32106	SSL 구성 오류입니다.
REASON_CODE_UNEXPECTED_ERROR	6	예상치 못한 오류가 발생했습니다.

텔레메트리(MQXR) 서비스 추적

다음 지시사항에 따라 텔레메트리 서비스 추적을 시작하고 추적을 제어하는 매개변수를 설정하고 추적의 출력을 찾으십시오.

시작하기 전에

추적은 지원 기능입니다. IBM 서비스 엔지니어가 텔레메트리(MQXR) 서비스 추적을 요청하는 경우 다음 지시사항을 수행하십시오. 제품 문서에는 추적 파일의 형식이나 이를 사용하여 클라이언트를 디버그하는 방법이 나와 있지 않습니다.

이 태스크 정보

You can use the IBM WebSphere MQ **strmqtrc** and **endmqtrc** commands to start and stop IBM WebSphere MQ trace. **strmqtrc**는 텔레메트리(MQXR) 서비스에 대한 추적을 캡처합니다. **strmqtrc**를 사용하는 경우 텔레메트리 서비스 추적을 시작하기 전에 최대 몇 초 동안 지연이 발생할 수 있습니다. IBM WebSphere MQ 추적에 대한 추가적인 정보는 추적 사용을 참조하십시오. 또는 다음과 같은 프로시저를 사용하여 텔레메트리(MQXR) 서비스를 추적할 수 있습니다.

프로시저

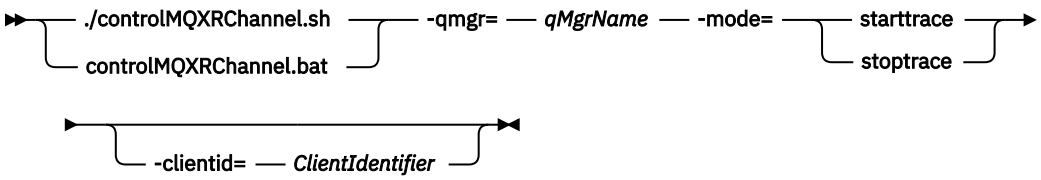
1. 추적의 크기 및 세부사항의 양을 제어하려면 추적 옵션을 사용하십시오. 옵션은 **strmqtrc** 또는 **controlMQXRChannel** 명령으로 시작된 추적에 적용됩니다.

   다음 파일의 추적 옵션을 설정하십시오.

   ```
   mqxrtrace.properties
   trace.config
   ```

   파일은 다음 디렉토리에 있습니다.

   - Windows에서 *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr입니다.
   - Linux에서 var/mqm/qmgrs/ *qMgrName*/mqxr입니다.
2. 다음 디렉토리에서 명령창을 여십시오.
   - Windows 시스템에서 *WebSphere MQ installation directory*\mqxr\bin 입니다.
   - Linux 시스템 /opt/mqm/mqxr/bin에서.
3. 다음 명령을 실행하여 SYSTEM.MQXR.SERVICE 추적을 실행하십시오.



필수 매개변수

qmgr=*qmgrName*
*qmgrName*을 큐 관리자 이름으로 설정하십시오.

mode=starttrace| stoptrace
추적을 시작하려면 starttrace 또는 추적을 종료하려면 stoptrace를 설정하십시오.

선택적 매개변수

clientid=*ClientIdentifier*
*ClientIdentifier*를 클라이언트의 ClientIdentifier로 설정하십시오. clientid가 단일 클라이언트에 대한 추적을 필터합니다. 다수의 클라이언트를 추적하려면 추적 명령을 여러 번 실행하십시오.

예를 들면, 다음과 같습니다.

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

결과

추적 출력을 보려면 다음 디렉토리로 이동하십시오.

- Windows 에서 *WebSphere MQ data directory*\trace입니다.
- Linux의 경우, /var/mqm/trace.

추적 파일은 mqxr_PPPPP.trc와 같이 이름 지정되며 PPPPP는 프로세스 ID입니다.

관련 참조
strmqtrc

MQTT v3 Java 클라이언트 추적

MQTT Java 클라이언트 추적을 작성하고 그 결과물을 제어하려면 다음 지시사항을 따르십시오.

시작하기 전에

이 주제는 IBM WebSphere MQ 버전 7.5.0.0에만 적용할 수 있습니다. 이후 버전에서의 Java 클라이언트 추적에 대해서는 160 페이지의 『MQTT (Paho) Java 클라이언트 추적 및 디버깅』의 내용을 참조하십시오.

추적은 지원 기능입니다. IBM 서비스 엔지니어가 MQTT Java 클라이언트를 추적해 달라고 요청할 경우 다음 지시사항을 따르십시오. 제품 문서에는 추적 파일의 형식이나 이를 사용하여 클라이언트를 디버그하는 방법이 나와 있지 않습니다.

추적은 WebSphere MQ Telemetry Java 클라이언트에 대해서만 작동합니다.

이 태스크 정보

참고: 이 예는 Windows용으로 코드화되었습니다. Linux 에서 예제를 실행하도록 구문을 변경하십시오.[2].

프로시저

1. 추적 구성을 포함하고 있는 Java 특성 파일을 작성하십시오.

   특성 파일에서 다음 선택적 특성을 지정하십시오. 특성 키가 두 번 이상 지정된 경우, 마지막 발생이 특성을 설정합니다.

   a) `com.ibm.micro.client.mqttv3.trace.outputName`

      추적 파일을 기록할 디렉토리입니다. 기본값은 클라이언트 작업 디렉토리입니다. 추적 파일은 `mqtt-`*`n`*`.trc`라고 합니다.

      ```
      java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
      ```

   b) `com.ibm.micro.client.mqttv3.trace.count`

      기록할 추적 파일의 수입니다. 기본값은 크기 제한이 없는 파일 하나입니다.

      ```
      java com.ibm.micro.client.mqttv3.trace.count=5
      ```

   c) `com.ibm.micro.client.mqttv3.trace.limit`

      기록할 파일의 최대 크기이며 기본값은 500000입니다. 한계는 둘 이상의 추적 파일이 요청되는 경우에만 적용됩니다.

      ```
      java com.ibm.micro.client.mqttv3.trace.limit=100000
      ```

   d) `com.ibm.micro.client.mqttv3.trace.client.`*`clientIdentifier`*`.status`

      클라이언트별로 추적을 켜거나 끄십시오. *`clientIdentifier`*`=*`인 경우 추적은 모든 클라이언트에 대해 켜지거나 꺼집니다. 기본적으로 추적은 모든 클라이언트에 대해 꺼져 있습니다.

      ```
      java com.ibm.micro.client.mqttv3.trace.client.*.status=on
      ```

      ```
      java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
      ```

2. 시스템 특성을 사용하여 추적 특성 파일을 JVM으로 전달하십시오.

[2] 자바는 올바른 경로 구분 기호를 사용합니다. You can code the delimiter in a property file as '/' or '\\'; '\' is the escape character

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. 클라이언트를 실행하십시오.
4. 추적 파일을 2진 인코딩에서 텍스트 또는 `.html`로 변환하십시오. 다음 명령을 사용하십시오.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o
outputFile] [-h] [-d
time]
```

여기서 인수는 다음과 같습니다.

-?
도움말을 표시함

-i traceFile
필수. 입력 파일에 들어갑니다(예: `mqtt-0.trc`).

-o outputFile
필수. 출력 파일을 정의합니다(예: `mqtt-0.trc.html` 또는 `mqtt-0.trc.txt`).

-h
HTML로 출력됩니다. 출력 파일 확장자가 `.html`이어야 합니다. 지정되지 않은 경우 일반 텍스트로 출력됩니다.

-d time
밀리초 단위의 시간 차이가 지정 시간 이상(>=)인 경우, *를 사용하여 행을 들여씁니다. HTML 출력에는 적용할 수 없습니다.

다음 예제는 추적 파일을 HTML 형식으로 출력합니다.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

두 번째 예제는 추적 파일을 일반 텍스트로 출력하며 차이가 50 밀리세컨드 이상인 연속 시간소인을 별표(*)를 사용하여 들여씁니다.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

최종 예제는 추적 파일을 일반 텍스트 형식으로 출력합니다.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

C용 MQTT 클라이언트 추적

MQTT 클라이언트 C 앱을 추적하도록 환경 변수 `MQTT_C_CLIENT_TRACE`를 설정합니다.

시작하기 전에

C용 MQTT 클라이언트 추적은 사전 빌드된 Windows와 Linux의 C용 MQTT 클라이언트, 사용자가 직접 빌드한 iOS 라이브러리에서 사용할 수 있습니다.

이 태스크 정보

환경 변수 `MQTT_C_CLIENT_TRACE`를 추적 출력을 포함할 파일의 경로로 설정하십시오. 추적 출력은 파일에 기록됩니다.

프로시저

C용 MQTT 클라이언트 앱을 실행하기 전에 `MQTT_C_CLIENT_TRACE=mqttcclient.log`를 설정하십시오.

a) 예를 들어 24 페이지의 『C용 MQTT 클라이언트 시작하기』의 샘플 스크립트를 다음과 같이 수정하십시오.

```
@echo off
setlocal
set MQTT_C_CLIENT_TRACE=mqttcclient.log
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3Sample.c" /link /
nologo ..\windows_ia32\mqttv3c.lib
set path=%path%;..\windows_ia32;
start "MQTT Subscriber" MQTTV3Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
MQTTV3Sample -b localhost -p 1883
pause
endlocal
```

b) %sdkroot%/sdk/client/c/samples 디렉토리에서 스크립트를 실행하십시오.

결과

추적 출력 파일은 다음과 같은 행으로 시작됩니다.

```
============================================================
                  Trace Output
============================================================
19700101 000000.000 (8084)  (1)> Socket_outInitialize:113
19700101 000000.000 (8084)   (2)> SocketBuffer_initialize:81
19700101 000000.000 (8084)   (2)< SocketBuffer_initialize:85
19700101 000000.000 (8084)  (1)< Socket_outInitialize:129
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> MQTTPersistence_create:43
19700101 000000.000 (8084)  (1)< MQTTPersistence_create:89 (0)
19700101 000000.000 (8084)  (1)> MQTTPersistence_initialize:104
19700101 000000.000 (8084)  (1)< MQTTPersistence_initialize:112 (0)
19700101 000000.000 (8084) (0)< MQTTClient_create:267 (0)
19700101 000000.000 (8084) (0)> MQTTClient_connect:701
19700101 000000.000 Connecting to serverURI localhost:1883
20130201 125912.234 (8084)  (1)> MQTTProtocol_connect:93
20130201 125912.234 (8084)   (2)> MQTTProtocol_addressPort:43
20130201 125912.234 (8084)   (2)< MQTTProtocol_addressPort:68
20130201 125912.234 (8084)   (2)> Socket_new:594
20130201 125912.234 New socket 1860 for localhost, port 1883
```

관련 태스크

24 페이지의 『C용 MQTT 클라이언트 시작하기』

C 소스를 컴파일할 수 있는 모든 플랫폼에서 샘플 C용 MQTT 클라이언트를 시작하고 실행할 수 있습니다. MQTT 서버로 IBM MessageSight 또는 IBM WebSphere MQ를 사용하여 C에 대해 샘플 MQTT 클라이언트를 실행할 수 있는지 확인하십시오.

MQTT (Paho) Java 클라이언트 추적 및 디버깅

기본 로거는 java.util.logging (JSR47) 라고 하는 표준 Java 로깅 기능을 사용합니다. 구성 파일을 사용하거나 프로그래밍 방식으로으로 이를 구성할 수 있습니다.

이 태스크 정보

참고: Paho Java 클라이언트는 IBM WebSphere MQ 버전 7.5.0.1 이상 버전에만 적용 가능합니다. IBM WebSphere MQ 버전 7.5.0.0에서 Java 클라이언트의 추적을 설명하는 정보는 158 페이지의 『MQTT v3 Java 클라이언트 추적』의 내용을 참조하십시오.

참고: 추적은 지원 기능입니다. IBM 서비스 엔지니어가 MQTT Java 클라이언트를 추적하도록 요청하는 경우 다음 지시사항을 따르십시오. 제품 문서에는 추적 파일의 형식이나 이를 사용하여 클라이언트를 디버그하는 방법이 나와 있지 않습니다. 추적은 IBM WebSphere MQ 텔레메트리 Java 클라이언트에 대해서만 작동합니다.

구성 파일을 사용하는 가장 간단한 방법은 `java.util.logging.config.file` 특성에 구성 파일의 이름을 지정하는 것입니다.

`org.eclipse.paho.client.mqttv3.logging` 패키지에서 작업 특성 파일 `jsr47min.properties`가 제공됩니다.

여러 가지 방법으로 JSR47 로그 기록 기능을 사용하여 다음을 수행할 수 있습니다.

- 선택한 패키지 세트에서 메시지 수집
- 특정 로그 레벨 이하에서 메시지 수집
- 로그 메시지의 여러 목적지 선택
- 파일에 기록하고 사용되는 파일의 크기 및 수를 제어하는 내장 로거를 제공하여
- 메모리에 기록하고 인메모리 메시지가 트리거를 기반으로 출력될 수 있도록 하는 내장 로거를 제공하여
- JSR47을 사용하여 MQTT 클라이언트 라이브러리를 사용하는 애플리케이션도 인스트루먼트화하는 경우 애플리케이션과 클라이언트 라이브러리의 메시지가 섞입니다.

디버그 정보 수집을 지원하기 위해 유틸리티 클래스가 제공됩니다. 이 클래스에는 이전에 설명한 로그 및 추적 메시지가 포함되어 있지만, Java 시스템 특성 및 Paho 클라이언트 내부의 변수 값과 같은 정보를 수집할 수 있다.

`org.eclipse.paho.client.mqttv3.util` 패키지의 일부인 공용 클래스 Debug에서 디버그 기능이 제공됩니다. 비동기 및 동기 MQTT 클라이언트 오브젝트 둘 다에서 getDebug() 메소드를 사용하여 디버그 인스턴스를 확보할 수 있습니다.

예를 들면, 다음과 같습니다.

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

dumpClientDebug() 메소드는 최대 양의 디버그 정보를 덤프합니다. 기록되는 전체 디버그 정보를 캡처하려면 로그 기능을 사용해야 합니다. 전체 디버그 정보를 캡처하려면 문제점이 발생함을 알 때, 예를 들어, 특정 예외가 발생한 후에 덤프 메소드를 호출하십시오.

프로시저

1. 구성 파일을 작성하거나 제공되는 jsr47min.properties를 사용하십시오.

   제공되는 특성 파일을 사용하는 경우 푸시 트리거가 올바른 오류 레벨로 설정되어 있는지 확인하십시오. 기본적으로는 심각 레벨 오류로 설정되어 있지만, 오류가 발생할 때까지 추적을 메모리에 유지하기 보다는 계속해서 추적을 파일에 기록해야 할 수도 있습니다. 이를 위해서는

   ```
   java.util.logging.MemoryHandler.push=SEVERE
   ```

   에 대하여

   ```
   java.util.logging.MemoryHandler.push=ALL
   ```

2. 시스템 특성을 사용하여 추적 구성 파일을 JVM에 전달하십시오.

   jsr4min.properties를 사용 중인 경우 이는 다음과 같습니다.

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. 클라이언트를 실행하십시오.

결과

예외 또는 문제점이 발생하면 Paho 디버그 클래스는 인메모리 추적을 구성된 파일 대상에 기록합니다.

추적은 생성 시 자동으로 파일에 기록되지 않고 푸시 트리거가 작동되거나 디버그 클래스로 인해 추적이 기록될 때만 파일에 기록됩니다. 후자의 경우 애플리케이션 코드 변경이 필요할 수도 있습니다.

각 행은 작성될 때 파일 핸들러에 기록됩니다. FileHandler를 구성하여 메시지가 출력되는 형식을 제어할 수 있습니다. Paho에서는 SimpleHandler보다는 많이 출력하고 JRE에서 제공하는 XMLHandler보다는 적게 출력하는 사용자 정의 파일 핸들러를 제공합니다. Paho 로그 형식기를 사용하는 추적 레코드의 양식은 다음과 같습니다.

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

예

작업 특성 파일 `jsr47min.properties`가 제공됩니다. 이 파일에는 Paho MQTT 클라이언트와 관련된 문제점을 해결하는 데 도움이 되는 추적을 수집하기 위한 제안된 구성이 포함되어 있습니다. 이 구성은 성능에 최소한의 영향을 미치면서 메모리에 계속해서 추적이 수집되도록 구성합니다. 푸시 트리거가 발생하거나 특정 푸시 요청이 있으면 인메모리 추적이 구성된 대상 핸들러로 푸시됩니다. 기본 푸시 트리거는 심각 레벨 메시지(연결이 끊어짐)입니다. 기본적으로 메모리에 수집된 추적은 이 시점에서 지정된 파일에 기록됩니다. 기본적으로 이 파일은 표준 `java.util.logging.FileHandler`입니다. Paho 디버그 클래스를 사용하여 메모리 추적을 해당 대상에 푸시할 수 있습니다.

JSR47에 대한 자세한 내용은 java.util.logging 패키지의 Javadoc을 참조하십시오.

```
# Loggers
# ----------------------------------------
# A memory handler is attached to the Paho packages
# and the level specified to collect all trace related
# to Paho packages.  This will override any root/global
# level handlers if set.
org.eclipse.paho.client.mqttv3.handlers=java.util.logging.MemoryHandler
org.eclipse.paho.client.mqttv3.level=ALL
# It is possible to set more granular trace on a per class basis e.g.
#org.eclipse.paho.client.mqttv3.internal.ClientComms.level=ALL

# Handlers
# ----------------------------------------
# Note: the target handler that is associated with the Memory Handler is not a root handler
# and hence not returned when getting the handlers from root. It appears accessing
# target handler programmatically is not possible as target is a private variable in
# class MemoryHandler
java.util.logging.MemoryHandler.level=FINEST
java.util.logging.MemoryHandler.size=10000
java.util.logging.MemoryHandler.push=SEVERE
java.util.logging.MemoryHandler.target=java.util.logging.FileHandler
#java.util.logging.MemoryHandler.target=java.util.logging.ConsoleHandler

# --- FileHandler ---
# Override of global logging level
java.util.logging.FileHandler.level=ALL

# Naming style for the output file:
# (The output file is placed in the directory
# defined by the "user.home" System property.)
# See java.util.logging for more options
java.util.logging.FileHandler.pattern=%h/paho%u.log

# Limiting size of output file in bytes:
java.util.logging.FileHandler.limit=200000

# Number of output files to cycle through, by appending an
# integer to the base file name:
java.util.logging.FileHandler.count=3

# Style of output (Simple or XML):
java.util.logging.FileHandler.formatter=org.eclipse.paho.client.mqttv3.logging.SimpleLogFormatte
r
```

다음에 수행할 작업

프로그래밍 방식으로 추적을 수집하기 위해 디버그 정보 수집을 지원하는 유틸리티 클래스가 제공됩니다. 이 클래스에는 이전에 설명한 로그 및 추적 메시지가 포함되어 있지만, Java 시스템 특성 및 Paho 클라이언트 내부의 변수 값과 같은 정보를 수집할 수 있다.

`org.eclipse.paho.client.mqttv3.util` 패키지의 일부인 공용 클래스 Debug에서 디버그 기능이 제공됩니다. 비동기 및 동기 MQTT 클라이언트 오브젝트 둘 다에서 getDebug() 메소드를 사용하여 디버그 인스턴스를 확보할 수 있습니다.

예를 들면, 다음과 같습니다.

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

dumpClientDebug() 메소드는 최대 양의 디버그 정보를 덤프합니다. 기록되는 전체 디버그 정보를 캡처하려면 로그 기능을 사용해야 합니다. 전체 디버그 정보를 캡처하려면 문제점이 발생함을 알 때, 예를 들어, 특정 예외가 발생한 후에 덤프 메소드를 호출하십시오.

MQTT JavaScript 클라이언트 추적

JavaScript 클라이언트를 사용하는 경우 연결된 클라이언트 오브젝트에서 메소드를 호출하도록 클라이언트 웹 애플리케이션을 변경하여 추적을 수집할 수 있습니다.

이 태스크 정보

추적을 수집하기 위해 다음 메소드를 사용할 수 있습니다.

- `client.startTrace()`는 클라이언트에 대한 추적을 시작합니다.
- `client.stopTrace()`는 클라이언트에 대한 추적을 중지합니다.
- `client.getTraceLog()`는 현재 추적 버퍼를 리턴합니다.

추적 버퍼를 출력하여 IBM Software Support에 보낼 수 있습니다. 여러 가지 방법으로 이를 수행할 수 있습니다. 예제에서는 추적을 시작한 후 출력을 콘솔 및 지정된 이메일 주소 모두에 전송하고 마지막으로 추적을 중지하는 방법을 보여줍니다.

```
client = new Messaging.Client(location.hostname, Number(location.port), "clientId");
// Start the client tracing, the trace records capture the method calls and network
//flows from now on.
client.startTrace();

client.onConnectionLost = onConnectionLost;
client.connect({onSuccess:onConnect});

function onConnect() {
  console.log("onConnect, will now disconnect then email Trace");
  client.disconnect();
};
function onConnectionLost(responseObject) {
  if (responseObject.errorCode !== 0)
    console.log("ConnectionLost:"+responseObject.errorMessage);
    console.log(client.getTraceLog());
    window.location="mailto:helpdesk@"+location.hostname+
        "?Subject=Web%20Messaging%20Utility%20Trace&body="+
        client.getTraceLog().join("%0A");
    client.stopTrace();
 };
```

샘플 출력:

```
Client.startTrace, "2013-10-03T10:58:10.531Z", "0.0.0.0",
Client.connect, {"keepAliveInterval":60,"cleanSession":true},, false,
Client._socket_send, {"type":1,"keepAliveInterval":60,"cleanSession":true,
    "clientId":"clientId"},
Client._on_socket_message, {},
Client._on_socket_message, {"type":2,"topicNameCompressionResponse":0,"returnCode":0},
Client.disconnect,Client._socket_send, {"type":14},
```

```
Client.getTraceLog, "2013-10-03T10:58:10.548Z",
Client.getTraceLog in flight messages,
```

V 7.5.0.2 MQTT 클라이언트와 함께 SHA-2 암호 스위트를 사용하기 위한 시스템 요구사항

Java 6의 경우 IBM, SR13 이상에서 SHA-2 암호 스위트를 사용하여 MQTT 채널 및 클라이언트 애플리케이션을 보호할 수 있습니다. 그러나 SHA-2 암호 스위트는 기본적으로 IBM, SR4 이상에서 Java 7까지 기본적으로 사용 가능하지 않으므로 이전 버전에서는 필수 스위트를 지정해야 합니다. 고유한 JRE를 포함하는 MQTT 클라이언트를 실행하는 경우 SHA-2 암호 스위트를 지원하는지 확인해야 합니다. 클라이언트 앱에서 SHA-2 암호 스위트를 사용하려면 클라이언트에서도 TLS(Transport Layer Security) 버전 1.2를 지원하는 값으로 SSL 컨텍스트를 설정해야 합니다.

Java 7의 경우 IBM, SR4 이상에서는 SHA-2 암호 스위트가 기본적으로 사용 가능합니다. IBM, SR13 및 이후 서비스 릴리스에서 Java 6의 경우 암호 스위트를 지정하지 않고 MQTT 채널을 정의하는 경우, 채널은 SHA-2 암호 스위트를 사용하여 클라이언트로부터의 연결을 승인하지 않습니다. SHA-2 암호 스위트를 사용하려면 채널 정의에서 필수 스위트를 지정해야 합니다.그러면 연결하기 전에 MQTT 서버에서 해당 스위트를 사용할 수 있습니다. 이는 지정된 스위트를 사용하는 클라이언트 앱만 이 채널에 연결할 수 있다는 것을 의미하기도 합니다.

Java용 MQTT 클라이언트에 이와 유사한 제한이 있습니다. 클라이언트 코드가 IBM의 Java 1.6 JRE에서 실행 중인 경우 필수 SHA-2 암호 스위트가 명시적으로 사용 가능해야 합니다. 이러한 스위트를 사용하기 위해서는 클라이언트에서 SSL 컨텍스트를 TLS(Transport Layer Security) 프로토콜 버전 1.2를 지원하는 값으로 설정해야 합니다. 예를 들면, 다음과 같습니다.

```
MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
java.util.Properties sslClientProps = new java.util.Properties();
sslClientProps.setProperty("com.ibm.ssl.keyStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.keyStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.trustStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.trustStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.protocol", "TLSv1.2");
sslClientProps.setProperty("com.ibm.ssl.enabledCipherSuites",
"SSL_RSA_WITH_AES_256_CBC_SHA256" );
mqttConnectOptions.setSSLProperties(sslClientProps);
```

2013년 6월에는 Internet Explorer 10이 JavaScript용 MQTT 메시징 클라이언트 와 함께 작동하고 TLS 1.2 프로토콜도 지원하므로 JavaScript 클라이언트와 SHA-2 연결을 설정하려는 경우에만 사용할 수 있는 유일한 브라우저입니다.

현재 지원되는 암호 스위트 목록은 관련 링크를 참조하십시오.

관련 개념

101 페이지의 『SSL을 사용하여 클라이언트 인증에 필요한 MQTT 클라이언트 구성』
SSL을 사용하여 MQTT 클라이언트를 인증하려면 클라이언트가 SSL을 통해 텔레메트리 채널에 연결해야 합니다. SSL 클라이언트를 인증하도록 구성된 텔레메트리 채널에 해당하는 TCP 포트를 지정해야 합니다.

103 페이지의 『SSL을 사용하여 채널 인증에 필요한 MQTT 클라이언트 구성』
SSL을 통해 텔레메트리 채널을 인증하려면 클라이언트가 SSL을 통해 텔레메트리 채널에 연결해야 합니다. 이 경우 SSL에 대해 구성된 텔레메트리 채널에 해당하는 포트를 지정해야 합니다. 구성에는 서버의 개인적으로 서명된 디지털 인증서가 있는 비밀번호 문구로 보호된 키 저장소가 포함되어야 합니다.

V 7.5.0.1 SSL을 통한 모바일 메시징 웹 앱에 대한 브라우저 지원의 제한사항

다른 플랫폼에서는 다른 브라우저 간에 기능의 차이가 있습니다. 이러한 차이를 이해하면 SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets를 사용하여 연결하도록 애플리케이션, 인증서 권한 (CAs) 및 클라이언트 인증서를 구성하는 데 도움이 됩니다.

SSL을 통해 JavaScript를 사용하는 모바일 메시징은 완전히 새로운 것이어서 다른 브라우저와 플랫폼 조합이 약간 다른 방식으로 다른 범위의 기능을 구현합니다. 다음 테이블에서는 각 브라우저 조합(Firefox, Chrome, Internet Explorer 및 Safari) 및 플랫폼 (Windows, Linux, Mac, iOS 및 Android)에 대해 현재 수행되거나 수행되지 않는 작업의 개요를 제공합니다.

표 *6.* 플랫폼 및 브라우저별 *SSL* 지원. 다음 테이블은 각 브라우저 및 플랫폼 조합에 대해 SSL 익명 및 비익명 연결이 지원되는지와 모든 인증 기관(CA) 및 클라이언트 인증서를 사용하여 브라우저에서 작업하는 범위를 지정합니다.

브라우저	SSL 지원(Y/N)	임의의 CA를 사용하여 SSL 작동(Y/N)	자세한 정보
Firefox 데스크탑.	SSL 익명 - 예 SSL 비익명 - 예	SSL 익명 - 예 SSL 비익명 - 예	브라우저에 CA 및 클라이언트 인증서 추가 Firefox는 자체 인증서 저장소를 사용합니다. CA 인증서를 가져오려면 **도구** > **옵션** > **고급** > **암호화** > **인증서 보기** > **권한** > **가져오기** 를 클릭하십시오. 클라이언트 인증서를 가져오려면 **도구** > **옵션** > **고급** > **암호화** > **인증서 보기** > **인증서** > **가져오기** 를 클릭하십시오. 보안 연결을 사용하려면 URL에 `https://`를 지정하십시오. Firefox는 인증서를 자동으로 선택하거나 매번 선택을 묻는 옵션을 제공합니다. 또한 Firefox는 SSL 3.0 또는 TLS 1.0 사용 옵션을 제공합니다. 둘 다 선택하십시오.
Chrome 데스크탑.	SSL 익명 - 예 SSL 비익명 - 예	SSL 익명 - 예 SSL 비익명 - 예	브라우저를 사용하여 다른 소프트웨어와 공유하는 운영 체제 인증서 저장소에 CA 및 클라이언트 인증서를 추가하십시오. CA 인증서를 가져오려면 **설정** > **고급 설정 표시** > **인증서 관리** > **신뢰할 수 있는 루트 인증 기관** > **가져오기** 를 클릭하십시오. 클라이언트 인증서를 가져오려면 **설정** > **고급 설정 표시** > **인증서 관리** > **개인용** > **가져오기** 를 클릭하십시오. 보안 연결을 사용하려면 URL에 `https://`를 지정하십시오. 크롬에서는 몇 가지 선택사항을 프롬프트합니다. 익명 또는 비익명 연결을 구성할지에 따라 올바른 하나를 선택하십시오.

표 *6.* 플랫폼 및 브라우저별 *SSL* 지원. 다음 테이블은 각 브라우저 및 플랫폼 조합에 대해 SSL 익명 및 비익명 연결이 지원되는지와 모든 인증 기관(CA) 및 클라이언트 인증서를 사용하여 브라우저에서 작업하는 범위를 지정합니다. *(계속)*

브라우저	SSL 지원(Y/N)	임의의 CA를 사용하여 SSL 작동(Y/N)	자세한 정보
Internet Explorer.	SSL 익명 - 예 SSL 비익명 - 예	SSL 익명 - 예 SSL 비익명 - 예	비익명 SSL 연결을 작성하는 경우 올바른 클라이언트 인증서를 선택하라는 프롬프트가 표시됩니다. Internet Explorer는 다른 소프트웨어와 공유하는 Windows 인증서 저장소를 사용합니다. CA 인증서를 가져오려면 **도구** > **인터넷 옵션** > **컨텐츠** > **인증서** > **신뢰할 수 있는 루트 인증 권한** > **가져오기** 를 클릭하십시오. 클라이언트 인증서를 가져오려면 **도구** > **인터넷 옵션** > **컨텐츠** > **인증서** > **개인용** > **가져오기** 를 클릭하십시오.
Safari 데스크탑.	SSL 익명 - 예 SSL 비익명 - 예	SSL 익명 - 예 SSL 비익명 - 예	브라우저를 사용하여 다른 소프트웨어와 공유하는 운영 체제 인증서 저장소에 CA 및 클라이언트 인증서를 추가하십시오.
Android의 Firefox	SSL 익명 - 예 SSL 비익명 - 예	SSL 익명 - 예 SSL 비익명 - 아니오	비익명: Firefox의 목록에 CA를 추가하기 위한 요구사항을 충족할 수 없어서 클라이언트 인증서가 작동하지 않습니다. 클라이언트 인증서를 가져오려면 **설정** > **보안** > **신임 스토리지**를 클릭하십시오. 목록에 있는 신뢰할 수 있는 CA에서 인증서를 서명한 경우 보안 연결을 작성할 수 있습니다.

표 6. 플랫폼 및 브라우저별 *SSL* 지원. 다음 테이블은 각 브라우저 및 플랫폼 조합에 대해 SSL 익명 및 비익명 연결이 지원되는지와 모든 인증 기관(CA) 및 클라이언트 인증서를 사용하여 브라우저에서 작업하는 범위를 지정합니다. *(계속)*

브라우저	SSL 지원(Y/N)	임의의 CA를 사용하여 SSL 작동(Y/N)	자세한 정보
Android의 Chrome	SSL 익명 - 예 SSL 비익명 - 예	SSL 익명 - 예 SSL 비익명 - 아니오	비익명: Chrome의 목록에 CA를 추가하기 위한 요구사항을 충족할 수 없어서 클라이언트 인증서가 작동하지 않습니다. **참고:** Google은 Chrome 버전 27에서 이를 지원할 계획입니다. 이는 버전 18 이후부터 열린 결함입니다. 클라이언트 인증서를 가져오려면 **설정** > **보안** > **신임 스토리지**를 클릭하십시오. 목록에 있는 신뢰할 수 있는 CA에서 인증서를 서명한 경우 보안 연결을 작성할 수 있습니다.
iOS의 Safari	SSL 익명 - 예 SSL 비익명 - 예	SSL 익명 - 예 SSL 비익명 - 아니오	비익명: CA 인증서가 동시에 설치된 경우에도 디바이스에서 클라이언트 인증서를 신뢰할 수 없습니다. Safari는 디바이스 인증서 저장소를 사용합니다. 이 저장소로 가져오려면 **설정** > **일반** > **프로파일**을 클릭하고 웹 페이지에서 CA 또는 클라이언트 인증서를 제공하거나 사용자 자신에게 이메일을 보내십시오.
iOS의 Chrome	SSL 익명 - 예 SSL 비익명 - 아니오	SSL 익명 - 아니오 SSL 비익명 - 아니오	익명: Apple 앱에서만 iOS 시스템 루트 저장소를 액세스할 수 있습니다. 따라서 Chrome은 인증서를 추가할 수 없는 자체 CA 목록을 사용해야 합니다. 비익명: 목록에 CA를 추가하기 위한 요구사항을 충족할 수 없어서 클라이언트 인증서가 작동하지 않습니다.

관련 태스크

74 페이지의 『SSL을 통해 JavaScript용 MQTT 메시징 클라이언트 및 WebSockets 연결』
SSL및 WebSocket protocol를 사용하는 JavaScript용 MQTT 메시징 클라이언트 샘플 HTML 페이지를 사용하여 웹 애플리케이션을 IBM WebSphere MQ 에 안전하게 연결하십시오.

관련 정보

Mozilla: (SSL) Firefox에서 Android CA 스토리지 또는 자체 스토리지를 사용합니까?

Chromium: 문제점 134418 - 클라이언트 인증서 지원 구현

ie10에서 신뢰할 수 없는 인증서가 있는 https 사이트를 열 수 없음

문제점 해결: MQTT 클라이언트가 연결되지 않음

텔레메트리(MQXR) 서비스에 연결되지 않는 MQTT 클라이언트 프로그램의 문제점을 해결합니다.

시작하기 전에

서버, 클라이언트 또는 연결 중 어디에 문제점이 발생합니까? C 또는 Java WebSphere MQTT 클라이언트를 사용하여 고유의 MQTT v3 프로토콜 처리 클라이언트 또는 MQTT 클라이언트 애플리케이션을 작성했습니까?

서버에서 WebSphere MQ Telemetry와 함께 제공되는 확인 애플리케이션을 실행하고 텔레메트리 채널과 텔레메트리(MQXR) 서비스가 올바로 실행 중인지 검사하십시오. 그런 다음 확인 애플리케이션을 클라이언트로 전송하고 거기서 확인 애플리케이션을 실행하십시오.

이 태스크 정보

MQTT 클라이언트가 연결되지 않는 이유 또는 텔레메트리 서버에 연결되지 않았다고 결론을 내릴 수 있는 이유가 많이 있습니다.

프로시저

1. 텔레메트리(MQXR) 서비스가 `MqttClient.Connect`에 리턴한 이유 코드로부터 끌어낼 수 있는 추론을 고려하십시오. 연결 실패 유형은 무엇입니까?

옵션	설명
REASON_CODE_INVALID_PROTOCOL_VERSION	소켓 주소가 텔레메트리 채널에 해당하는지, 다른 브로커에 대해 동일한 소켓 주소를 사용하지 않았는지 확인하십시오.
REASON_CODE_INVALID_CLIENT_ID	클라이언트 ID가 23바이트이하이고 `A-Z, a-z, 0-9, './_%` 범위의 문자만 포함하는지 확인하십시오.
REASON_CODE_INVALID_DESTINATION	클라이언트 ID가 큐 관리자 이름과 같지 않은지 확인하십시오.
REASON_CODE_SERVER_CONNECT_ERROR	텔레메트리(MQXR) 서비스 및 큐 관리자가 정상적으로 실행 중인지 확인하십시오. **netstat**를 사용하여 소켓 주소가 다른 애플리케이션에 할당되지 않았는지 확인하십시오.

   IBM WebSphere MQ Telemetry에서 제공하는 라이브러리 중 하나를 사용하지 않고 MQTT 클라이언트 라이브러리를 작성한 경우에는 `CONNACK` 리턴 코드를 살펴보십시오.

   이와 같은 세 가지 오류로부터 클라이언트가 텔레메트리(MQXR) 서비스에 연결되었지만 서비스에서 오류를 발견했음을 추론할 수 있습니다.
2. 텔레메트리(MQXR) 서비스가 응답하지 않는 경우 클라이언트가 생성하는 이유 코드에서 끌어낼 수 있는 추론을 고려하십시오.

옵션	설명
REASON_CODE_CLIENT_EXCEPTION **REASON_CODE_CLIENT_TIMEOUT**	서버에서 FDC 파일을 찾으십시오. 153 페이지의 『서버 측 로그』의 내용을 참조하십시오. 텔레메트리(MQXR) 서비스가 클라이언트의 제한시간 초과를 감지하면 이는 첫 번째 오류 데이터 캡처(FFDC) 파일을 작성합니다. 연결이 예상치 못하게 중단될 때마다 FDC 파일을이 작성합니다.

텔레메트리(MQXR) 서비스가 클라이언트에 응답하지 않았거나 클라이언트의 제한시간이 만기되었을 수 있습니다. WebSphere MQ 텔레메트리 Java 클라이언트는 애플리케이션이 무제한 제한시간을 설정한 경우에만 정지됩니다. 클라이언트는 진단되지 않는 연결 문제점이 발생하여 `MqttClient.Connect`의 제한시간 설정이 만기된 후 다음 예외 중 하나를 처리합니다.

연결 실패와 상관된 FDC 파일을 발견하지 않는 한 클라이언트가 서버에 연결을 시도했다고 추측할 수 없습니다.

a) 클라이언트가 연결 요청을 보냈는지 확인하십시오.

https://java.net/projects/tcpmon 에서 사용 가능한 **tcpmon**와 같은 도구를 사용하여 TCPIP 요청을 확인하십시오.

b) 클라이언트에 의해 사용된 리모트 소켓 주소가 텔레메트리 채널에 대해 정의된 소켓 주소와 일치합니까?

IBM WebSphere MQ 텔레메트리와 함께 제공된 Java SE MQTT 클라이언트의 기본 파일 지속성 클래스는 클라이언트 작업 디렉토리에 이름이 *clientIdentifier*-`tcp`*hostName*포트 또는 *clientIdentifier*-`ssl`*hostName*포트 인 폴더를 작성합니다. 폴더 이름은 연결 시도에 사용되는 `hostName` 및 `port`를 알려줍니다., 155 페이지의 『클라이언트 측 로그 파일』의 내용을 참조하십시오.

c) 리모트 서버 주소를 ping할 수 있습니까?

d) 서버의 **netstat**도 클라이언트가 연결 중인 포트에서 텔레메트리 채널을 실행 중임을 표시합니까?

3. 텔레메트리(MQXR) 서비스가 클라이언트 요청에서 문제점을 발견했는지 확인하십시오.

텔레메트리(MQXR) 서비스는 감지한 오류를 `mqxr.log`에 기록하고 큐 관리자는 `AMQERR01.LOG`에 오류를 기록합니다. 다음 내용을 참조하십시오.

4. 다른 클라이언트를 실행하여 문제점을 분리해 보십시오.

- 샘플 텔레메트리 채널을 사용하여 MQTT 샘플 애플리케이션을 실행하십시오.
- **wmqttSample** GUI 클라이언트를 실행하여 연결을 확인하십시오. SupportPac IA92를 다운로드하여 **wmqttSample** 를 얻으십시오.

  참고: 이전 버전의 IA92 에는 MQTT v3 Java 클라이언트 라이브러리가 포함되지 않습니다.

서버 플랫폼의 샘플 프로그램을 실행하여 네트워크 연결에 대한 불확실성을 제거한 다음 클라이언트 플랫폼에서 샘플을 실행하십시오.

5. 확인할 다른 사항은 다음과 같습니다.

a) 매우 많은 숫자의 MQTT 클라이언트가 동시에 연결하려고 시도 중입니까?

텔레메트리 채널에는 수신되는 연결의 백로그를 버퍼링하는 큐가 있습니다. 연결은 1초에 10,000개를 초과하여 처리됩니다. 백로그 버퍼의 크기는 IBM WebSphere MQ Explorer에서 텔레메트리 채널 마법사를 사용하여 구성 가능합니다. 기본 크기는 4096입니다. 백로그가 낮은 값으로 구성되지 않았는지 확인하십시오.

b) 텔레메트리(MQXR) 서비스 및 큐 관리자가 여전히 실행 중입니까?

c) 클라이언트가 TCP/IP 주소로 전환된 가용성이 높은 큐 관리자에 연결되었습니까?

d) 방화벽이 아웃바운드 또는 리턴 데이터 패킷을 선택적으로 필터링 중입니까?

문제점 해결: MQTT 클라이언트 연결이 삭제됨

클라이언트가 짧은 시간 또는 긴 시간 동안 연결하여 실행한 후에 예상치 못한 `ConnectionLost` 예외를 처리하는 원인을 알아냅니다.

시작하기 전에

MQTT 클라이언트가 연결되었습니다. 클라이언트가 오랜 시간 동안 작동 중일 수 있습니다. 클라이언트가 짧은 간격으로만 시작되는 경우 연결과 연결 삭제 사이의 시간이 짧을 수 있습니다.

성공적으로 작성한 연결에서 삭제된 연결을 구별하는 것은 어렵지 않으며 그런 다음 나중에 삭제됩니다. 삭제된 연결은 `MqttCallback.ConnectionLost` 메소드를 호출하는 MQTT 클라이언트에 의해 정의됩니다. 이 메소드는 연결이 성공적으로 설정된 후에만 호출됩니다. 증상은 부정적인 수신확인을 수신하거나 제한시간이 초과된 후 예외를 처리하는 `MqttClient.Connect`와 다릅니다.

MQTT 클라이언트 앱이 IBM WebSphere MQ에서 제공하는 MQTT 클라이언트 라이브러리를 사용하지 않는 경우 증상이 클라이언트에 따라 다릅니다. MQTT v3 프로토콜에서 증상은 서버에 대한 요청에 제때 응답하지 못하거나 TCP/IP 연결 장애가 발생하는 것입니다.

이 태스크 정보

MQTT 클라이언트는 긍정적인 연결 수신확인을 수신한 후에 발견한 서버 측 문제점에 대한 응답으로 throwable 예외와 함께 `MqttCallback.ConnectionLost`를 호출합니다. MQTT 클라이언트가 `MqttTopic.publish`와 `MqttClient.subscribe`에서 되돌아가는 경우 요청은 메시지 송수신을 담당하는 MQTT 클라이언트 스레드에 전송됩니다. 서버측 오류는 처리 가능한 예외를 `ConnectionLost` 콜백 메소드로 전달하여 비동기적으로 보고됩니다.

텔레메트리(MQXR) 서비스는 연결을 삭제하는 경우 항상 첫 번째 오류 데이터 캡처(FFDC) 파일을 작성합니다.

프로시저

1. 동일한 `ClientIdentifier`를 사용한 다른 클라이언트가 시작되었습니까?

   동일한 `ClientIdentifier`를 사용하여 두 번째 클라이언트가 시작되거나 동일한 클라이언트가 재시작되는 경우 첫 번째 클라이언트에 대한 첫 번째 연결이 삭제됩니다.
2. 클라이언트가 발행 또는 구독 권한이 없는 토픽에 액세스했습니까?

   텔레메트리 서비스가 `MQCC_FAIL`을 리턴하는 클라이언트 대신 조치를 수행하면 서비스가 클라이언트 연결을 삭제합니다.

   이유 코드는 클라이언트에 리턴되지 않습니다.

   - `mqxr.log` 파일과 `AMQERR01.LOG` 파일에서 클라이언트가 연결된 큐 관리자와 관련된 로그 메시지를 찾으십시오. 153 페이지의 『서버 측 로그』를 참조하십시오.
3. TCP/IP 연결이 삭제되었습니까?

   방화벽에는 TCP/IP 연결이 비활성이며 연결이 삭제되었음을 표시하는 낮은 제한시간 설정이 있을 수 있습니다.

   - `MqttConnectOptions.setKeepAliveInterval`을 사용하여 비활성 TCP/IP 연결 시간을 줄이십시오.

문제점 해결: MQTT 애플리케이션에서 메시지가 손실됨

메시지 손실로 인한 문제점을 해결하십시오. 메시지가 비지속적이거나 잘못된 위치로 전송되었거나 송신되지 않았습니까? 잘못 코드화된 클라이언트 프로그램에서는 메시지가 손실될 수 있습니다.

시작하기 전에

보낸 메시지가 손실되었다는 것을 얼만큼 확신합니까? 메시지가 수신되지 않았으므로 메시지가 손실되었다고 추측할 수 있습니까? 메시지가 발행인 경우, 손실된 메시지는 발행자가 송신한 메시지 또는 구독자에게 송신된 메시지 중 어느 것입니까? 또는 구독이 손실되었는데 브로커가 해당 구독에 대한 발행을 구독자에게 송신하지 않습니까?

솔루션이 분산된 발행/구독을 포함하고 있고 클러스터 또는 발행/구독 계층을 사용하는 경우, 결과적으로 메시지가 손실되는 많은 구성 문제가 있습니다.

메시지를 "적어도 한 번" 또는 "많아야 한 번"의 서비스 품질(QoS)로 보낸 경우 손실되었다고 생각하는 메시지가 사용자가 예상한 방식으로 전달되지 않았을 가능성이 있습니다. 메시지가 시스템에서 잘못 삭제되었을 가능성은 거의 없습니다. 예상한 발행 또는 구독을 작성하는 데 실패했을 수 있습니다.

손실된 메시지의 문제점을 판별하기 위한 가장 중요한 단계는 메시지가 손실되었는지 확인하는 것입니다. 시나리오를 다시 작성하고 더 많은 메시지가 손실되도록 하십시오. "적어도 한 번" 또는 "많아야 한 번" 서비스 품질(QoS)을 사용하여 시스템이 메시지를 제거하는 모든 경우를 제거하십시오.

이 태스크 정보

손실된 메시지를 진단하는 네 가지 근거가 있습니다.

1. "시작 후 삭제" 메시지가 디자인대로 작동함. 때때로 시스템이 "시작 후 삭제" 메시지가 시스템에 의해 제거됩니다.
2. 구성: 분산 환경에서 올바른 권한으로 직접 발행/구독을 설정하지 않습니다.
3. 클라이언트 프로그래밍 오류: 메시지 전달 책임이 IBM에서 작성한 코드에만 있는 것이 아닙니다.
4. 이와 같은 가능성을 모두 확인한 경우 IBM 서비스와 관련된 문제라고 결정할 수 있습니다.

프로시저

1. 손실된 메시지에 "시작 후 삭제" 서비스 품질(QoS)이 있는 경우 "적어도 한 번" 또는 "많아야 한 번" 서비스 품질(QoS)을 설정하십시오. 메시지가 손실되는지 다시 확인하십시오.
   - "실행 및 취소" 서비스 품질(QoS)과 함께 발송된 메시지는 수많은 환경에서 IBM WebSphere MQ에 의해 처리됩니다.
     - 통신이 손실되고 채널이 중지되었습니다.
     - 큐 관리자가 종료되었습니다.
     - 메시지 수가 너무 많습니다.
   - "시작 후 삭제" 메시지의 전달은 TCP/IP의 안정성에 따라 달라집니다. TCP/IP는 전달이 수신확인될 때까지 데이터 패킷을 계속 다시 송신합니다. TCP/IP 세션이 중단되면 서비스 품질(QoS)이 "시작 후 삭제"인 메시지가 손실됩니다. 세션은 클라이언트 또는 서버가 닫히거나 통신 문제점 또는 세션 연결을 끊는 방화벽으로 인해 중단될 수 있습니다.
2. "적어도 한 번" 또는 "많아야 한 번" 서비스 품질(QoS)인 미배달 메시지를 다시 전달하려면 클라이언트가 이전 세션을 재시작하는지 확인하십시오.

   a) 클라이언트 앱이 Java SE MQTT 클라이언트를 사용 중인 경우 `MqttClient.CleanSession` 를 `false` 로 설정하는지 확인하십시오.

   b) 다른 클라이언트 라이브러리를 사용 중인 경우, 세션이 올바르게 재시작되는지 확인하십시오.
3. 클라이언트 앱이 실수로 다른 세션을 시작하지 않고 동일한 세션을 재시작하는지 확인하십시오.

   동일한 세션을 다시 시작하려면 `cleanSession = false`, `Mqttclient.clientIdentifier` 및 `MqttClient.serverURI`가 이전 세션과 동일해야 합니다.
4. 세션이 너무 일찍 닫히는 경우, 클라이언트의 지속 저장소에서 메시지를 다시 송신할 수 있는지 확인하십시오.

   a) 클라이언트 앱이 Java SE MQTT 클라이언트를 사용 중인 경우, 메시지가 지속성 폴더에 저장되는지 확인하십시오. 155 페이지의 『클라이언트 측 로그 파일』 의 내용을 참조하십시오.

   b) 다른 클라이언트 라이브러리를 사용 중인 경우 또는 사용자의 자체 지속 메커니즘을 구현한 경우, 올바르게 작동하는지 검사하십시오.
5. 전달되기 전에 아무도 메시지를 삭제하지 않았는지 확인하십시오.

   MQTT 클라이언트에 전달되기를 기다리는 미배달 메시지는 `SYSTEM.MQTT.TRANSMIT.QUEUE`에 저장됩니다. 텔레메트리 서버로의 전달을 기다리는 메시지는 클라이언트 지속성 메커니즘을 통해 저장됩니다. MQTT 클라이언트의 메시지 지속성을 참조하십시오.
6. 클라이언트에 수신이 예상되는 발행에 대한 구독이 있는지 확인하십시오.

WebSphere MQ 탐색기를 사용하거나 **runmqsc** 또는 PCF 명령을 사용하여 등록을 나열하십시오. 모든 MQTT 클라이언트 구독은 이름이 지정되어 있습니다. 이름에는 다음 양식의 이름이 지정됩니다. *ClientIdentifier*:*Topic name*

7. 발행자에게 발행 권한이 있고 구독자에게 발행 토픽을 구독할 권한이 있는지 확인하십시오.

```
dspmqaut -m qMgr -n topicName -t topic -p user ID
```

클러스터된 발행/구독 시스템에서 구독자는 구독자가 연결되어 있는 큐 관리자의 토픽에 대해 권한이 부여되어야 합니다. 구독자에게는 발행이 발행된 큐 관리자에 있는 토픽을 구독할 권한이 필요하지 않습니다. 큐 관리자 사이의 채널은 프록시 구독 전달 및 발행물 전달을 위한 올바른 권한이 있어야 합니다.

동일한 구독을 작성하고 IBM WebSphere MQ Explorer를 사용하여 이에 발행하십시오. 클라이언트 유틸리티를 사용하여 애플리케이션 클라이언트 발행 및 구독을 시뮬레이션하십시오. IBM WebSphere MQ Explorer에서 유틸리티를 시작하고 사용자 ID를 클라이언트 앱에서 채택한 사용자 ID와 일치하도록 변경하십시오.

8. 구독자에게 발행물을 SYSTEM.MQTT.TRANSMIT.QUEUE에 넣을 권한이 있는지 확인하십시오.

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

9. IBM WebSphere MQ 지점간 애플리케이션이 SYSTEM.MQTT.TRANSMIT.QUEUE에 메시지를 넣을 권한이 있는지 확인하십시오.

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

MQTT 클라이언트로 메시지를 송신하도록 분산 큐잉 구성에서 "클라이언트에 직접 메시지 보내기" 를 참조하십시오.

문제점 해결: 텔레메트리(MQXR) 서비스가 시작되지 않음

텔레메트리(MQXR) 서비스가 시작되지 않는 문제를 해결합니다. WebSphere MQ Telemetry 설치를 검사하고 누락되거나, 이동되거나, 잘못된 권한을 가진 파일이 없는지 확인하십시오. 텔레메트리(MQXR) 서비스에서 사용하는 경로가 텔레메트리(MQXR) 서비스 프로그램을 찾는지 확인하십시오.

시작하기 전에

WebSphere MQ Telemetry 기능이 설치되어 있어야 합니다. IBM WebSphere MQ Explorer에는 **IBM WebSphere MQ > 큐 관리자 > *qMgr*이름 > 텔레메트리**에 텔레메트리 폴더가 있습니다. 이 폴더가 없으면 설치에 실패한 것입니다.

텔레메트리(MQXR) 서비스가 작성되어 있어야 시작할 수 있습니다. 텔레메트리 (MQXR) 서비스가 작성되지 않은 경우 **샘플 구성 정의 ...** 를 실행하십시오. Telemetry 폴더에 있습니다.

이전에 텔레메트리(MQXR) 서비스가 시작된 경우에는 추가 **Channels** 폴더와 **Channel Status** 폴더가 Telemetry 폴더 아래 작성됩니다. 텔레메트리 서비스 SYSTEM.MQXR.SERVICE는 **Services** 폴더에 있습니다. 시스템 오브젝트를 표시하는 탐색기 단일 선택 단추를 클릭하는 경우 표시됩니다.

SYSTEM.MQXR.SERVICE 을 마우스 오른쪽 단추로 눌러 서비스를 시작 및 중지하고 상태를 표시하며 사용자 ID에 서비스 시작 권한이 있는지 여부를 표시하십시오.

이 태스크 정보

SYSTEM.MQXR.SERVICE 텔레메트리(MQXR) 서비스를 시작하는 데 실패합니다. 시작 실패는 다음과 같은 두 가지 방식으로 표시됩니다.

1. 시작 명령이 즉시 실패합니다.
2. 시작 명령은 성공하지만 서비스가 즉시 중지됩니다.

프로시저

1. 서비스를 시작하십시오.

결과
서비스가 즉시 중지됩니다. 창에 오류 메시지가 표시됩니다. 예:

```
WebSphere MQ cannot process the request because the
executable specified cannot be started. (AMQ4160)
```

원인

설치 시 파일이 누락되었거나 설치된 파일에 대한 사용권한이 잘못 설정되었습니다.
IBM WebSphere MQ Telemetry 기능은 가용성이 높은 한 쌍의 큐 관리자 중 하나에만 설치됩니다. 큐 관리자 인스턴스가 대기 상태로 전환되는 경우 SYSTEM.MQXR.SERVICE를 시작하려고 시도합니다. 대기 상태에서는 텔레메트리(MQXR) 서비스가 설치되지 않으므로 서비스 시작 명령이 실패합니다.

조사
오류 로그를 조사하십시오. 153 페이지의 『서버 측 로그』의 내용을 참조하십시오.

조치
WebSphere MQ Telemetry 기능을 설치하거나 설치 제거한 후 다시 설치하십시오.

2. 서비스를 시작하고 30초 동안 기다린 후 탐색기를 새로 고치고 서비스 상태를 검사하십시오.

결과
서비스가 시작된 후 중지됩니다.

원인
SYSTEM.MQXR.SERVICE는 **runMQXRService** 명령을 시작했지만 명령이 실패했습니다.

조사

오류 로그를 조사하십시오. 153 페이지의 『서버 측 로그』의 내용을 참조하십시오.
정의된 샘플 채널에서만 문제점이 발생하는지 확인하십시오. *WMQ data directory*\Qmgrs*qMgrName*\mqxr\ 디렉토리의 컨텐츠를 백업한 후 지우십시오. 샘플 구성 마법사를 실행하고 서비스를 시작해 보십시오.

조치
권한 및 경로 문제점을 찾아보십시오.

문제점 해결: 텔레메트리 서비스가 JAAS 로그인 모듈을 호출하지 않음

텔레메트리(MQXR) 서비스가 JAAS 로그인 모듈을 호출 중이지 않은지 알아내고 문제점을 수정하도록 JAAS를 구성합니다.

시작하기 전에

WMQ installation directory\mqxr\samples\LoginModule.java를 수정하여 사용자 고유의 인증 클래스 *WMQ installation directory*\mqxr\samples\samples\LoginModule.class를 작성했습니다. 또는 사용자 고유의 JAAS 인증 클래스를 작성하여 선택한 디렉토리에 저장했습니다. 처음 텔레메트리(MQXR) 서비스를 테스트한 후 텔레메트리(MQXR) 서비스가 인증 클래스를 호출 중이지 않은 것으로 의심됩니다.

참고: WebSphere MQ에 적용되는 유지보수가 인증 클래스를 덮어쓸 가능성에 대비하여 보호 수단을 마련하십시오. WebSphere MQ 디렉토리 트리 내의 경로가 아닌 인증 클래스의 사용자 고유 경로를 사용하십시오.

이 태스크 정보

태스크는 시나리오를 사용하여 문제점 해결 방법을 설명합니다. 시나리오에서 security.jaas라는 패키지에는 JAASLogin.class라는 JAAS 인증 클래스가 포함되어 있습니다. 이 클래스는 C:\WMQTelemetryApps\security\jaas 경로에 저장되어 있습니다. IBM WebSphere MQ 텔레메트리에 대한 JAAS 구성에 대한 도움말은 텔레메트리 채널 JAAS 구성 을 참조하십시오. 174 페이지의 『JAAS 구성 예』의 예는 샘플 구성입니다.

프로시저

1. javax.security.auth.login.LoginException에서 발생한 예외에 대해 mqxr.log 를 참조하십시오.

   mqxr.log의 경로는 153 페이지의 『서버 측 로그』, 로그에 나열된 예외의 예는 176 페이지의 그림 54의 내용을 참조하십시오.
2. 174 페이지의 『JAAS 구성 예』에서 작업한 예와 비교하여 JAAS 구성을 수정하십시오.
3. 사용자의 로그인 클래스를 인증 패키지로 리팩토링한 후 샘플 JAASLoginModule로 바꾸고 동일한 경로를 사용하여 배치하십시오. loggedIn의 값을 true와 false 간에 전환하십시오.

   loggedIn이 true일 때 문제점이 사라지고 loggedIn이 false일 때 동일한 문제점이 나타나는 경우 문제점은 로그인 클래스에 있습니다.
4. 문제점이 인증 문제가 아니라 권한 부여 문제인지 확인하십시오.
   a) 고정된 사용자 ID를 사용하여 권한 검사를 수행하도록 텔레메트리 채널 정의를 변경하십시오. mqm 그룹의 구성원인 사용자 ID를 선택하십시오.
   b) 클라이언트 앱을 재실행하십시오.

      문제점이 해결되는 경우, 해결책은 권한 부여를 위해 전달되는 사용자 ID에 있습니다. 전달되는 사용자 이름이 무엇입니까? 해당 사용자 이름을 로그인 모듈에서 파일로 인쇄하십시오. IBM WebSphere MQ Explorer 또는 **dspmqauth**를 사용하여 해당 액세스 권한을 확인하십시오.

JAAS 구성 예

WebSphere MQ 탐색기에서 **새 텔레메트리 채널**을 사용하여 텔레메트리 채널을 구성합니다. 클라이언트는 포트 1884에서 JAASMCAUser 텔레메트리 채널에 연결됩니다. 174 페이지의 그림 48에는 텔레메트리 마법사에서 작성한 텔레메트리 특성 파일의 예가 표시되어 있습니다. 이 파일을 직접 편집하지 마십시오. 채널은 JAASConfig라는 구성을 사용하는 JAAS를 사용하여 인증합니다. 클라이언트가 인증되고 나면 사용자 ID Admin을 사용하여 IBM WebSphere MQ 오브젝트에 대한 해당 액세스를 권한 부여합니다.

```
com.ibm.mq.MQXR.channel/JAASMCAUser: \
com.ibm.mq.MQXR.Port=1884;\
com.ibm.mq.MQXR.JAASConfig=JAASConfig;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

그림 *48. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\mqxr_win.properties*

JAAS 구성 파일에는 Java 클래스 security.jaas.JAASLogin이름을 지정하는 JAASConfig 라는 스탠자가 있으며, JAAS 는 클라이언트를 인증하는 데 사용됩니다.

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

그림 *49. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config*

SYSTEM.MQTT.SERVICE가 시작되면 174 페이지의 그림 50의 경로를 CLASSPATH에 추가합니다.

```
CLASSPATH=C:\WMQTelemtryApps;
```

그림 *50. WMQ Installation directory\data\qmgrs\qMgrName\service.env*

175 페이지의 그림 51에는 텔레메트리(MQXR) 서비스에 대해 설정된 CLASSPATH에 추가된 174 페이지의 그림 50의 추가 경로가 표시되어 있습니다.

```
CLASSPATH=;C:\IBM\MQ\Program\mqxr\bin\\..\lib\MQXRListener.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\WMQCommonServices.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\objectManager.utils.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\com.ibm.micro.xr.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mq.jmqi.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mqjms.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mq.jar;
C:\WMQTelemtryApps;
```

그림 *51. runMQXRService.bat*의 *CLASSPATH* 출력

175 페이지의 그림 52의 출력에서는 텔레메트리(MQXR) 서비스가 174 페이지의 그림 48에 표시된 채널 정의로 시작되었다는 것을 보여줍니다.

```
21/05/2010 15:32:12 [main] com.ibm.mq.MQXRService.MQXRPropertiesFile
AMQXR2011I: Property com.ibm.mq.MQXR.channel/JAASMCAUser value
com.ibm.mq.MQXR.Port=1884;
com.ibm.mq.MQXR.JAASConfig=JAASConfig;
com.ibm.mq.MQXR.UserName=Admin;
com.ibm.mq.MQXR.StartWithMQXRService=true
```

그림 *52. WMQ Installation directory\data\qmgrs\qMgrName\errors\mqxr.log*

클라이언트 앱이 JAAS 채널에 연결되면 `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig`가 `jaas.config` 파일의 JAAS 스탠자 이름과 일치하지 않는 경우 연결에 실패하고 클라이언트가 예외를 처리하며 리턴 코드는 `0`입니다. 175 페이지의 그림 53의 내용을 참조하십시오. 연결되지 않은 클라이언트가 연결 끊기를 시도했기 때문에 두 번째 예외 `Client is not connected (32104)`가 처리되었습니다.

```
C:\WMQTelemetryApps>java com.ibm.mq.id.PubAsyncRestartable
Starting a clean session for instance "Admin_PubAsyncRestartab"
Publishing "Hello World Fri May 21 17:23:23 BST 2010" on topic "MQTT Example"
for client instance: "Admin_PubAsyncRestartab" using QoS=1 on address tcp://localhost:1884"
Userid: "Admin", Password: "Password"
Delivery token "528752516" has been received: false
Connection lost on instance "Admin_PubAsyncRestartab" with cause "MqttException"
MqttException (0) - java.io.EOFException
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:118)
        at java.lang.Thread.run(Thread.java:801)
Caused by: java.io.EOFException
        at java.io.DataInputStream.readByte(DataInputStream.java:269)
        at
com.ibm.micro.client.mqttv3.internal.wire.MqttInputStream.readMqttWireMessage(MqttInputStream.ja
va:56)
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:90)
        ... 1 more
Client is not connected (32104)
        at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33
)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:100)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:117)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.disconnect(ClientComms.java:229)
        at com.ibm.micro.client.mqttv3.MqttClient.disconnect(MqttClient.java:385)
        at com.ibm.mq.id.PubAsyncRestartable.main(PubAsyncRestartable.java:49)
```

그림 *53. com.ibm.mq.id.PubAsyncRestartable* 연결 중에 처리된 예외

`mqxr.log`에는 175 페이지의 그림 53에 표시된 추가 출력이 포함되어 있습니다.

`javax.security.auth.login.LoginException`을 처리하는 JAAS가 오류를 감지했으며 원인은 `No LoginModules configured for JAAS`입니다. 176 페이지의 그림 54에서 볼 수 있듯 잘못된 구성 이름으

로 인해 이 오류가 발생할 수 있습니다. JAAS 구성을 로드하는 중에 JAAS에 발생한 다른 문제점 때문일 수도 있습니다.

JAAS가 예외를 보고하지 않는 경우, JAAS는 이미 `JAASConfig` 스탠자에 이름이 지정된 `security.jaas.JAASLogin` 클래스를 로드한 것입니다.

```
21/05/2010 12:06:12 [ServerWorker0] com.ibm.mq.MQXRService.MQTTCommunications
AMQXR2050E: Unable to load JAAS config: JAASWrongConfig.
The following exception occurred javax.security.auth.login.LoginException:
No LoginModules configured for JAAS
```

그림 *54. mqxr.log - JAAS* 구성 로드 중에 오류 발생

문제점 해결: 디먼 시작 또는 실행

디바이스용 IBM WebSphere MQ Telemetry 디먼 콘솔 로그를 참조하거나 추적을 켜거나 이 토픽의 증상 표를 사용하여 디먼에 관련된 문제점을 해결하십시오.

프로시저

1. 콘솔 로그를 확인하십시오.

   디먼이 포그라운드에서 실행 중인 경우 콘솔 메시지가 터미널 창에 기록됩니다. 디먼이 백그라운드에서 시작된 경우에는 `stdout`를 경로 재지정한 위치에 콘솔이 있습니다.
2. 디먼을 다시 시작하십시오.

   디먼이 재시작될 때까지 구성 파일에 대한 변경사항이 활성화되지 않습니다.
3. 176 페이지의 표 7의 내용을 참고하십시오.

표 *7.* 증상 테이블

문제점	**제안된 솔루션**
Windows에서 디먼을 시작하면 다음과 같은 메시지가 표시됩니다. 시스템이 지정된 프로그램을 실행할 수 없음 또는 애플리케이션의 협력(side-by-side) 구성이 올바르지 않아 애플리케이션 시작에 실패했습니다.	Microsoft Visual C++ 2008 Redistributable Package를 설치하십시오.
둘 이상의 디먼 또는 MQTT 가능 서버가 브릿지를 통해 서로 연결되어 있고 프로세서가 과부하를 나타내고 있습니다.	하나 이상의 메시지가 한 서버에서 다른 서버로 반복적으로 전달되는 메시지 루프가 있는 것 같습니다. 구성 파일에서 토픽 매개변수를 조사하십시오. 가능한 경우 보다 구체적인 토픽을 사용하십시오. 연결 루프의 가장 일반적인 원인은 양방향의 폭넓은 와일드카드 문자입니다.
다른 MQTT 클라이언트가 연결할 수 있는 원격 MQTT 가능 서버에 브릿지가 연결할 수 없습니다.	원격 서버가 디바이스용 WebSphere MQ Telemetry 디먼이기도 한지 판별하려는 시도와 원격 서버가 호환되지 않을 수 있습니다. **try_private**을 `off`로 설정하여 메시지 루프를 제거하는 특수 처리를 사용 안함으로 설정하십시오.

표 7. 증상 테이블 (계속)	
문제점	**제안된 솔루션**
브릿지가 구성될 때 이 메시지가 인쇄됩니다. 경고: 연결이 소켓 1888의 첫 번째 패킷이 아닙니다. CONNACK을 가져왔습니다.	로컬 디먼으로 루프백하도록 브릿지를 구성했을 수 있습니다. 루프백은 지원되지 않습니다.

문제점 해결: MQTT 클라이언트가 디먼에 연결되지 않음

클라이언트가 디먼에 연결되지 않거나 디먼이 다른 디먼 또는 WebSphere MQ Telemetry 채널에 연결되지 않습니다.

이 태스크 정보

디먼에서 송수신하는 각 MQTT 패킷을 추적하십시오.

프로시저

디먼 구성 파일에서 **trace_output** 매개변수를 `protocol`로 설정하거나 `amqtdd.upd` 파일을 사용하여 디먼으로 명령을 송신하십시오.

`amqtdd.upd` 파일 사용에 대한 예제는 디바이스용 IBM WebSphere MQ 텔레메트리 디먼과 IBM WebSphere MQ 사이의 전송 메시지를 참조하십시오.

디먼은 프로토콜 설정을 사용하여 디먼이 송수신하는 각 MQTT 패킷을 설명하는 메시지를 콘솔에 인쇄합니다.

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

150-945
서울특별시 영등포구
국제금융로 10, 3IFC
한국 아이.비.엠 주식회사
U.S.A.

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

지적 재산권 라이센스 부여
2-31 Roppongi 3-chome, Minato-Ku
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 명시적 또는 묵시적인 일체의 보증 없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 본 프로그램에 관한 정보를 얻고자 하는 라이센스 사용자는 다음 주소로 문의하십시오.

서울특별시 영등포구
서울특별시 강남구 도곡동 467-12,
군인공제회관빌딩
한국 아이.비.엠 주식회사
U.S.A.

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등)하에서 사용될 수 있습니다.

이 정보에 기술된 라이센스가 부여된 프로그램 및 프로그램에 대해 사용 가능한 모든 라이센스가 부여된 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 단계의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한 일부 성능은 추정을

통해 추측되었을 수도 있으므로 실제 결과는 다를 수 있습니다. 이 책의 사용자는 해당 데이터를 본인의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 제품들을 테스트하지 않았으므로, 비IBM 제품과 관련된 성능의 정확성, 호환성 또는 기타 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM이 제시하는 방향 또는 의도에 관한 모든 언급은 특별한 통지 없이 변경될 수 있습니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위하여 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권 라이센스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 IBM에 추가 비용을 지불하지 않고 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이들 샘플 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 진술하지 않습니다.

이 정보를 소프트카피로 확인하는 경우에는 사진과 컬러 삽화가 제대로 나타나지 않을 수도 있습니다.

프로그래밍 인터페이스 정보

프로그래밍 인터페이스 정보는 본 프로그램과 함께 사용하기 위한 응용프로그램 소프트웨어 작성을 돕기 위해 제공됩니다.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of IBM WebSphere MQ.

그러나 본 정보에는 진단, 수정 및 성능 조정 정보도 포함되어 있습니다. 진단, 수정 및 성능 조정 정보는 응용프로그램 소프트웨어의 디버그를 돕기 위해 제공된 것입니다.

중요사항: 이 진단, 수정 및 튜닝 정보는 변경될 수 있으므로 프로그래밍 인터페이스로 사용하지 마십시오.

상표

IBM, IBM 로고, ibm.com®는 전세계 여러 국가에 등록된 IBM Corporation의 상표입니다. 현재 IBM 상표 목록은 웹 "저작권 및 상표 정보"(www.ibm.com/legal/copytrade.shtml) 에 있습니다. 기타 제품 및 서비스 이름은 IBM 또는 타사의 상표입니다.

Microsoft 및 Windows는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 The Open Group의 등록상표입니다.

Linux는 미국 또는 기타 국가에서 사용되는 Linus Torvalds의 등록상표입니다.

이 제품에는 Eclipse 프로젝트 (http://www.eclipse.org/) 에서 개발한 소프트웨어가 포함되어 있습니다.

Java 및 모든 Java 기반 상표와 로고는 Oracle 및/또는 그 계열사의 상표 또는 등록상표입니다.

부품 번호:

(1P) P/N: