7.5

Mobile Messaging and M2M

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen” auf Seite 199 gelesen werden.

Diese Ausgabe bezieht sich auf Version 7 Release 5 von IBM® WebSphere MQ und auf alle nachfolgenden Releases und Modifikationen, bis dieser Hinweis in einer Neuausgabe geändert wird.

Wenn Sie Informationen an IBMsenden, erteilen Sie IBM ein nicht ausschließliches Recht, die Informationen in beliebiger Weise zu verwenden oder zu verteilen, ohne dass eine Verpflichtung für Sie entsteht.

Inhaltsverzeichnis

Einführung in MQTT

Informationen zum Senden von Nachrichten zwischen mobilen Apps mithilfe des Telemetrietransports von MQ (MQTT). Das Protokoll ist für die Verwendung in drahtlosen Netzen und Netzen mit geringer Bandbreite vorgesehen. Eine mobile Anwendung, die MQTT verwendet, sendet und empfängt Nachrichten, indem sie eine MQTT-Bibliothek aufruft. Die Nachrichten werden über einen MQTT-Messaging-Server ausgetauscht. Der MQTT-Client und -Server führen die komplexe Aufgabe der Zustellung von Nachrichten auf zuverlässige Weise für die mobile App aus und halten die Kosten für das Netzmanagement niedrig.

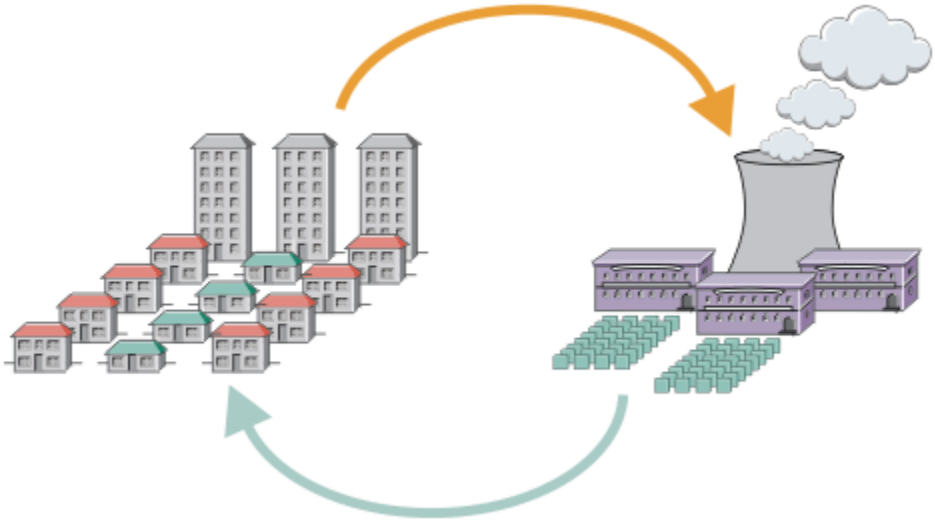
MQTT-Anwendungen werden auf mobilen Geräten wie Smartphones und Tablets ausgeführt. MQTT wird auch in der Telemetrie eingesetzt, um Daten von Sensoren zu empfangen und Sensoren über Fernzugriff zu steuern. Für mobile Geräte und Sensoren stellt MQTT ein hoch skalierbares Publish/Subscribe-Protokoll mit zuverlässiger Nachrichtenübermittlung bereit. Um MQTT-Nachrichten senden und empfangen zu können, fügen Sie eine MQTT-Clientbibliothek zu Ihrer Anwendung hinzu.

Die MQTT-Clientbibliothek ist klein. Sie funktioniert wie eine Mailbox, über die Nachrichten mit anderen MQTT-Anwendungen, die mit einem MQTT-Server verbunden sind, ausgetauscht werden. Indem sie Nachrichten senden, statt ständig mit einem Server verbunden zu sein, der auf eine Antwort wartet, verlängern MQTT-Anwendungen die Lebensdauer des Akkus. Die Bibliothek sendet Nachrichten über einen MQTT-Server, auf dem das MQTT version 3.1-Protokoll aktiv ist, an andere Geräte. Sie können Nachrichten an einen bestimmten Client senden oder über Publish/Subscribe-Messaging viele Geräte adressieren.

Die MQTT-Clientbibliotheken verbinden Anwendungen für mobile Geräte und Sensoren über das MQTT-Protokoll mit einem MQTT-Server.

IBM MessageSight und IBM WebSphere MQ sind MQTT-Server. Sie können große Volumen von MQTT-Clientanwendungen verbinden und sie können eine Verbindung zwischen MQTT- und IBM WebSphere MQ-Netzen herstellen. Siehe „Erste Schritte mit MQTT-Servern” auf Seite 142. IBM WebSphere MQ und IBM MessageSight können jeweils eine Bridge bilden zwischen externen Webanwendungen, die auf mobilen Einheiten und Sensoren ausgeführt werden, und anderen Arten von Publish/Subscribe- und Messaging-Anwendungen, die innerhalb des Unternehmens ausgeführt werden. Diese Bridgefunktion erleichtert die Umsetzung von "intelligenten Lösungen", in die mobile Geräte und Sensoren einbezogen sind.

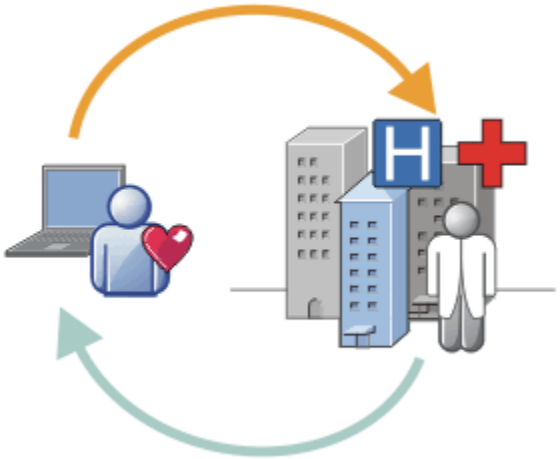
Intelligente Lösungen erschließen die Fülle von Informationen, die im Internet verfügbar ist, für Anwendungen auf mobilen Geräten und Sensoren. Zwei Beispiele für intelligente Anwendungen, die auf Telemetrie basieren, sind intelligente Elektrizität und intelligente Gesundheitsdienste.



- Eine MQTT-Nachricht mit Energieverbrauchsdaten, die an den Serviceanbieter gesendet wird.
- Eine Telemetrieanwendung sendet Steuerbefehle, die auf einer Analyse von Energieverbrauchsdaten basieren.
- Weitere Informationen finden Sie unter Telemetrieszenario: Überwachung und Steuerung des Energieverbrauchs in Privathaushalten.

Abbildung 1. Intelligente Elektrizitätsmessung

Abbildung 2. Intelligente Gesundheitsüberwachung



- Eine Telemetrieanwendung sendet Gesundheitsdaten an ein Krankenhaus oder einen Arzt.
- Auf Basis einer Analyse der Gesundheitsdaten können MQTT-Alarmnachrichten oder eine Rückmeldung gesendet werden.
- Weitere Informationen finden Sie unter Telemetrieszenario: Überwachung nicht stationärer Patienten.

Sie können MQTT in kompakte Endgeräte einbauen, indem Sie eine eigene App für das MQTT protocol schreiben. Dazu stellt IBM Clientbibliotheken bereit, die Apps unterstützen, die über MQTT ausgeführt werden. Siehe „Erste Schritte mit MQTT-Clients” auf Seite 11. IBM stellt Clientbibliotheken für iOS-Apps und für Android-Apps V 7.5.0.1 sowie einen JavaScript-Browser-Client für plattformunabhängige Web-Apps bereit. V 7.5.0.1 Die JavaScript -Clientseiten stellen über WebSocketseine Verbindung zu IBM MessageSight und IBM WebSphere MQ mit dem Protokoll MQTT her. IBM stellt auch MQTT-Beispiel-Apps für C und Java unter Linux® und Windows bereit.

Die C- und Java-Bibliotheken können unter iOS, Android und Windows sowie auf einer Reihe von UNIX and Linux-Plattformen ausgeführt werden. Sie können den C-Quellcode für die MQTT-Clientbibliothek auf andere Plattformen übertragen. Die MQTT-Clientbibliotheken für C und Java sind mit einer Open-Source-Lizenz über das Eclipse Paho-Projekt verfügbar. Weitere Informationen finden Sie im Abschnitt Eclipse Paho. Die MQTT protocolspezifikation ist eine offene Spezifikation und auf der Website MQTT.org verfügbar.

MQTT protocol

Das MQTT protocol ist schlank in dem Sinne, dass Clients kompakt sind und die Netzbandbreite effizient genutzt wird. Das MQTT-Protokoll unterstützt die zuverlässige Nachrichtenübermittlung und Übertragungen ohne Empfangsbestätigungen (Fire-and-Forget). Im Protokoll wird die Nachrichtenübermittlung von der Anwendung entkoppelt. Das Ausmaß der Entkopplung in einer Anwendung hängt davon ab, wie ein MQTT-Client und MQTT-Server geschrieben sind. Entkoppelte Übermittlung bedeutet, dass eine Anwendung nicht von einer Serververbindung abhängig ist und nicht auf Nachrichten warten muss. Das Interaktionsmodell ist mit dem der E-Mail vergleichbar, jedoch optimiert für die Anwendungsprogrammierung.

Das MQTT V3.1-Protokoll ist veröffentlicht. (siehe MQTT V3.1-Protokollspezifikation). Die Spezifikation führt eine Reihe von Unterscheidungsmerkmalen für das Protokoll auf:

- Es ist ein Publish/Subscribe-Protokoll.

  Zusätzlich zur Bereitstellung einer Eins-zu-viele-Verteilung entkoppelt Publish/Subscribe Anwendungen. Beide Funktionen sind in Anwendungen mit vielen Clients nützlich.
- Es ist vollständig unabhängig vom Nachrichteninhalt.
- Es wird über TCP/IP ausgeführt, das grundlegende Netzkonnektivität bereitstellt.
- Es bietet drei Servicequalitätsstufen für die Nachrichtenübermittlung:

  "Höchstens einmal"
  Nachrichten werden unter bestmöglicher Ausnutzung der Leistungsmöglichkeiten des zugrunde liegenden IP-Netzes übermittelt. Nachrichtenverluste sind möglich.

  Verwenden Sie diese Servicequalitätsstufe zum Beispiel bei der Übertragung von Umgebungssensordaten. Es ist kein Problem, wenn ein einzelner gemessener Wert verloren geht, wenn kurz darauf bereits der nächste Wert veröffentlicht wird.

  "At least once (Mindestens einmal)"
  Es wird sichergestellt, dass Nachrichten ankommen, wobei doppelte Nachrichten jedoch nicht ausgeschlossen werden können.

  "Genau einmal"
  Es wird sichergestellt, dass Nachrichten genau einmal ankommen.

  Verwenden Sie diese Servicequalitätsstufe zum Beispiel bei Gebührenabrechnungssystemen. Wenn Nachrichten doppelt ankommen oder verloren gehen, können Unstimmigkeiten oder falsche Gebührenabrechnungen die Folge sein.
- Es ist wirtschaftlich hinsichtlich der Art und Weise der Verwaltung des Nachrichtenflusses im Netz. Beispielsweise hat der Header eine feste Länge von nur 2 Bytes und der interne Protokollaustausch ist auf ein Minimum reduziert, um den Netzverkehr zu verringern.
- Es besitzt eine "Last Will and Testament"-Funktion, die Subskribenten über die abnormale Trennung eines Clients vom MQTT-Server informiert. Siehe „Veröffentlichung "Last Will and Testament"" auf Seite 134.

MQTT version 3.1 wird von IBM IBM WebSphere MQ und IBM MessageSight unterstützt. MQTT wird über TCP/IP implementiert. Für Nicht-TCP/IP-Netze ist eine andere Version des Protokolls, MQTT-S, verfügbar. Weitere Informationen finden Sie im Abschnitt MQTT-S version 1.2-Spezifikation.

MQTT-Communitys

IBM betreibt eine IBM Developer Messaging Community für MQTT-Entwickler, die Anwendungen für IBM MessageSight und IBM WebSphere MQ schreiben.

Ein guter Ort, um sich über Implementierungen und Erweiterungen des MQTT-Protokolls zu informieren und auszutauschen, ist MQTT.org.

MQTT ist ein Open-Source-Eclipse-Projekt, das zum Eclipse Technology Project gehört. Die Paho-Community entwickelt Open-Source-Clients und -Server. Weitere Informationen finden Sie im Abschnitt Eclipse Paho.

Einführung in MQTT

Informationen zum Senden von Nachrichten zwischen mobilen Apps mithilfe des Telemetrietransports von MQ (MQTT). Das Protokoll ist für die Verwendung in drahtlosen Netzen und Netzen mit geringer Bandbreite vorgesehen. Eine mobile Anwendung, die MQTT verwendet, sendet und empfängt Nachrichten, indem sie eine MQTT-Bibliothek aufruft. Die Nachrichten werden über einen MQTT-Messaging-Server ausgetauscht. Der MQTT-Client und -Server führen die komplexe Aufgabe der Zustellung von Nachrichten auf zuverlässige Weise für die mobile App aus und halten die Kosten für das Netzmanagement niedrig.

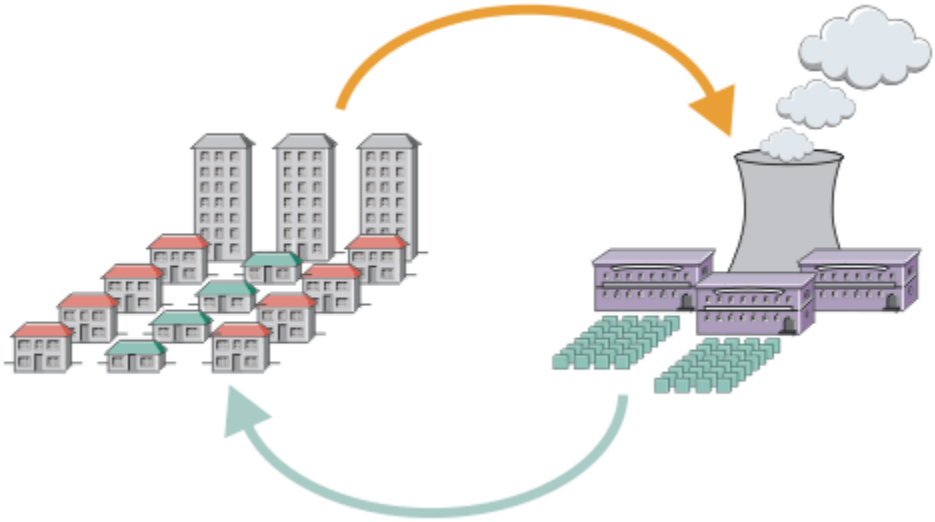
MQTT-Anwendungen werden auf mobilen Geräten wie Smartphones und Tablets ausgeführt. MQTT wird auch in der Telemetrie eingesetzt, um Daten von Sensoren zu empfangen und Sensoren über Fernzugriff zu steuern. Für mobile Geräte und Sensoren stellt MQTT ein hoch skalierbares Publish/Subscribe-Protokoll mit zuverlässiger Nachrichtenübermittlung bereit. Um MQTT-Nachrichten senden und empfangen zu können, fügen Sie eine MQTT-Clientbibliothek zu Ihrer Anwendung hinzu.

Die MQTT-Clientbibliothek ist klein. Sie funktioniert wie eine Mailbox, über die Nachrichten mit anderen MQTT-Anwendungen, die mit einem MQTT-Server verbunden sind, ausgetauscht werden. Indem sie Nachrichten senden, statt ständig mit einem Server verbunden zu sein, der auf eine Antwort wartet, verlängern MQTT-Anwendungen die Lebensdauer des Akkus. Die Bibliothek sendet Nachrichten über einen MQTT-Server, auf dem das MQTT version 3.1-Protokoll aktiv ist, an andere Geräte. Sie können Nachrichten an einen bestimmten Client senden oder über Publish/Subscribe-Messaging viele Geräte adressieren.

Die MQTT-Clientbibliotheken verbinden Anwendungen für mobile Geräte und Sensoren über das MQTT-Protokoll mit einem MQTT-Server.

IBM MessageSight und IBM WebSphere MQ sind MQTT-Server. Sie können große Volumen von MQTT-Clientanwendungen verbinden und sie können eine Verbindung zwischen MQTT- und IBM WebSphere MQ-Netzen herstellen. Siehe „Erste Schritte mit MQTT-Servern” auf Seite 142. IBM WebSphere MQ und IBM MessageSight können jeweils eine Bridge bilden zwischen externen Webanwendungen, die auf mobilen Einheiten und Sensoren ausgeführt werden, und anderen Arten von Publish/Subscribe- und Messaging-Anwendungen, die innerhalb des Unternehmens ausgeführt werden. Diese Bridgefunktion erleichtert die Umsetzung von "intelligenten Lösungen", in die mobile Geräte und Sensoren einbezogen sind.

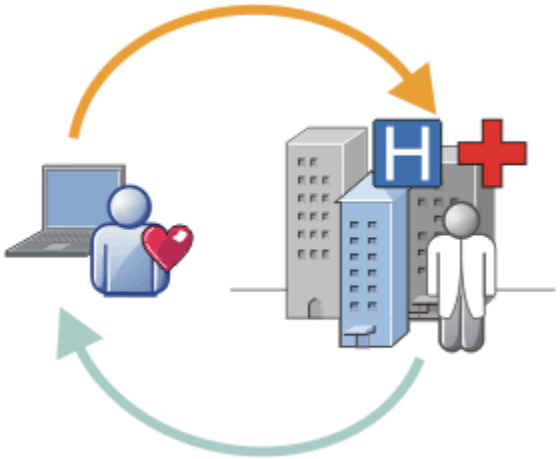
Intelligente Lösungen erschließen die Fülle von Informationen, die im Internet verfügbar ist, für Anwendungen auf mobilen Geräten und Sensoren. Zwei Beispiele für intelligente Anwendungen, die auf Telemetrie basieren, sind intelligente Elektrizität und intelligente Gesundheitsdienste.



- Eine MQTT-Nachricht mit Energieverbrauchsdaten, die an den Serviceanbieter gesendet wird.
- Eine Telemetrieanwendung sendet Steuerbefehle, die auf einer Analyse von Energieverbrauchsdaten basieren.
- Weitere Informationen finden Sie unter Telemetrieszenario: Überwachung und Steuerung des Energieverbrauchs in Privathaushalten.

Abbildung 3. Intelligente Elektrizitätsmessung

Abbildung 4. Intelligente Gesundheitsüberwachung



- Eine Telemetrieanwendung sendet Gesundheitsdaten an ein Krankenhaus oder einen Arzt.
- Auf Basis einer Analyse der Gesundheitsdaten können MQTT-Alarmnachrichten oder eine Rückmeldung gesendet werden.
- Weitere Informationen finden Sie unter Telemetrieszenario: Überwachung nicht stationärer Patienten.

Sie können MQTT in kompakte Endgeräte einbauen, indem Sie eine eigene App für das MQTT protocol schreiben. Dazu stellt IBM Clientbibliotheken bereit, die Apps unterstützen, die über MQTT ausgeführt werden. Siehe „Erste Schritte mit MQTT-Clients” auf Seite 11. IBM stellt Clientbibliotheken für iOS-Apps und für Android-Apps V 7.5.0.1 sowie einen JavaScript-Browser-Client für plattformunabhängige Web-Apps bereit. V 7.5.0.1 Die JavaScript -Clientseiten stellen über WebSocketseine Verbindung zu IBM MessageSight und IBM WebSphere MQ mit dem Protokoll MQTT her. IBM stellt auch MQTT-Beispiel-Apps für C und Java unter Linux und Windows bereit.

Die C- und Java-Bibliotheken können unter iOS, Android und Windows sowie auf einer Reihe von UNIX and Linux-Plattformen ausgeführt werden. Sie können den C-Quellcode für die MQTT-Clientbibliothek auf andere Plattformen übertragen. Die MQTT-Clientbibliotheken für C und Java sind mit einer Open-Source-Lizenz über das Eclipse Paho-Projekt verfügbar. Weitere Informationen finden Sie im Abschnitt Eclipse Paho. Die MQTT protocolspezifikation ist eine offene Spezifikation und auf der Website MQTT.org verfügbar.

MQTT protocol

Das MQTT protocol ist schlank in dem Sinne, dass Clients kompakt sind und die Netzbandbreite effizient genutzt wird. Das MQTT-Protokoll unterstützt die zuverlässige Nachrichtenübermittlung und Übertragungen ohne Empfangsbestätigungen (Fire-and-Forget). Im Protokoll wird die Nachrichtenübermittlung von der Anwendung entkoppelt. Das Ausmaß der Entkopplung in einer Anwendung hängt davon ab, wie ein MQTT-Client und MQTT-Server geschrieben sind. Entkoppelte Übermittlung bedeutet, dass eine Anwendung nicht von einer Serververbindung abhängig ist und nicht auf Nachrichten warten muss. Das Interaktionsmodell ist mit dem der E-Mail vergleichbar, jedoch optimiert für die Anwendungsprogrammierung.

Das MQTT V3.1-Protokoll ist veröffentlicht. (siehe MQTT V3.1-Protokollspezifikation). Die Spezifikation führt eine Reihe von Unterscheidungsmerkmalen für das Protokoll auf:

- Es ist ein Publish/Subscribe-Protokoll.

  Zusätzlich zur Bereitstellung einer Eins-zu-viele-Verteilung entkoppelt Publish/Subscribe Anwendungen. Beide Funktionen sind in Anwendungen mit vielen Clients nützlich.
- Es ist vollständig unabhängig vom Nachrichteninhalt.
- Es wird über TCP/IP ausgeführt, das grundlegende Netzkonnektivität bereitstellt.
- Es bietet drei Servicequalitätsstufen für die Nachrichtenübermittlung:

  "Höchstens einmal"
  Nachrichten werden unter bestmöglicher Ausnutzung der Leistungsmöglichkeiten des zugrunde liegenden IP-Netzes übermittelt. Nachrichtenverluste sind möglich.

  Verwenden Sie diese Servicequalitätsstufe zum Beispiel bei der Übertragung von Umgebungssensordaten. Es ist kein Problem, wenn ein einzelner gemessener Wert verloren geht, wenn kurz darauf bereits der nächste Wert veröffentlicht wird.

  "At least once (Mindestens einmal)"
  Es wird sichergestellt, dass Nachrichten ankommen, wobei doppelte Nachrichten jedoch nicht ausgeschlossen werden können.

  "Genau einmal"
  Es wird sichergestellt, dass Nachrichten genau einmal ankommen.

  Verwenden Sie diese Servicequalitätsstufe zum Beispiel bei Gebührenabrechnungssystemen. Wenn Nachrichten doppelt ankommen oder verloren gehen, können Unstimmigkeiten oder falsche Gebührenabrechnungen die Folge sein.
- Es ist wirtschaftlich hinsichtlich der Art und Weise der Verwaltung des Nachrichtenflusses im Netz. Beispielsweise hat der Header eine feste Länge von nur 2 Bytes und der interne Protokollaustausch ist auf ein Minimum reduziert, um den Netzverkehr zu verringern.
- Es besitzt eine "Last Will and Testament"-Funktion, die Subskribenten über die abnormale Trennung eines Clients vom MQTT-Server informiert. Siehe „Veröffentlichung "Last Will and Testament"" auf Seite 134.

MQTT version 3.1 wird von IBM IBM WebSphere MQ und IBM MessageSight unterstützt. MQTT wird über TCP/IP implementiert. Für Nicht-TCP/IP-Netze ist eine andere Version des Protokolls, MQTT-S, verfügbar. Weitere Informationen finden Sie im Abschnitt MQTT-S version 1.2-Spezifikation.

MQTT-Communitys

IBM betreibt eine IBM Developer Messaging Community für MQTT-Entwickler, die Anwendungen für IBM MessageSight und IBM WebSphere MQ schreiben.

Ein guter Ort, um sich über Implementierungen und Erweiterungen des MQTT-Protokolls zu informieren und auszutauschen, ist MQTT.org.

MQTT ist ein Open-Source-Eclipse-Projekt, das zum Eclipse Technology Project gehört. Die Paho-Community entwickelt Open-Source-Clients und -Server. Weitere Informationen finden Sie im Abschnitt Eclipse Paho.

Erste Schritte mit MQTT-Clients

Sie können mit der Entwicklung einer mobilen App oder einer Machine-to-Machine (M2M)-App beginnen, indem Sie eine Beispiel-App für den MQTT-Client erstellen und ausführen, die eine MQTT-Clientbibliothek verwendet. Die Beispiel-Apps und zugehörigen Clientbibliotheken im Mobile Messaging und M2M Client-Pack von IBM verfügbar. Es gibt Versionen der Apps und Clientbibliotheken, die in Java, in JavaScriptund in C geschrieben wurden. Sie können diese Apps auf den meisten Plattformen und Geräten ausführen, einschließlich Android -Geräten und -Produkten von Apple.

Vorbereitende Schritte

Um eine App erstellen und ausführen zu können, wird etwas Erfahrung in der Erstellung von Apps für das Zielgerät oder die Zielplattform sowie in der verwendeten Programmiersprache vorausgesetzt. Normalerweise ist ein wenig Erfahrung ausreichend, um eine Beispiel-App auf dem ausgewählten Gerät oder der ausgewählten Plattform ausführen zu können.

Wenn Sie einen auf Unternehmen abgestimmten MQTT-Server wie beispielsweise IBM WebSphere MQ oder IBM MessageSight verwenden, können Sie Informationen aus Ihrer Beispiel-App mit Ihren bestehenden Unternehmens-Apps austauschen.

Informationen zu diesem Vorgang

Sie haben folgende Ziele:

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.
2. Laden Sie Mobile Messaging und M2M Client-Packherunter.
3. Erstellen Sie für Ihr Zielgerät oder Ihre Zielplattform die Beispiel-Apps aus dem Clientsoftwarepaket.
4. Überprüfen Sie, ob sich die Beispiele wie erwartet verhalten, indem Sie sie mit dem MQTT-Server verbinden.

Als Ergebnis der Erstellung und des Tests der Beispiel-Apps für Ihr Gerät oder Ihre Plattform richten Sie eine funktionierende Entwicklungsumgebung zum Erstellen eigener Clientanwendungen ein.

Im Mobile Messaging und M2M Client-Pack ist das MQTT-SDK enthalten. Dieses Software-Development-Kit (SDK) stellt die folgenden Ressourcen bereit:

- Beispiel-Apps für den MQTT-Client, die in den Programmiersprachen Java, JavaScript und C geschrieben sind.
- MQTT-Clientbibliotheken, die diese Client-Apps unterstützen und deren Ausführung auf den meisten Plattformen und Geräten ermöglichen.

Darüber hinaus enthält das Software-Development-Kit den Quellcode für den MQTT-Client für C. Durch die Anpassung dieses Quellcodes können Sie MQTT-Clientbibliotheken in der Programmiersprache C für andere Plattformen erstellen. Informationen hierzu finden Sie im Abschnitt „MQTT-Client für C-Bibliotheken erstellen” auf Seite 32. Der Quellcode für den MQTT-Client für C ist auch mit einer Open-Source-Lizenz von Eclipse Paho verfügbar.

Prozedur

In den folgenden Artikeln werden Sie durch die plattformspezifischen Schritte für die Erstellung und Ausführung einer MQTT-Beispiel-App auf einem Desktop-Computer oder auf einer mobilen Einheit für Android oder von Apple geführt:

- „Erste Schritte mit dem MQTT-Client für Java” auf Seite 12
- „Erste Schritte mit dem MQTT-Client für Java unter Android” auf Seite 19
- V 7.5.0.1 „Erste Schritte mit dem MQTT-Messaging-Client für JavaScript” auf Seite 25
- „Erste Schritte mit dem MQTT-Client für C” auf Seite 27
- „Erste Schritte mit dem MQTT-Client für C unter iOS” auf Seite 49

Nächste Schritte

Um eine neue MQTT-Anwendung entwickeln zu können, müssen Sie folgende Kenntnisse besitzen oder sich aneignen:

- Programmierung in der Sprache, die für das Gerät oder die Plattform erforderlich ist.
- Programmierung für das Zielgerät oder die Zielplattform
- Entwurf von Publish/Subscribe-Anwendungen
- Entwurf von Programmen für das MQTT-Programmiermodell
- Entwurf von Programmen zur Ausführung auf dem ausgewählten mobilen Gerät
- Verwendung von SSL und JAAS zum Schützen von Programmen

Sie benötigen keine Netzprogrammierkenntnisse, um einen MQTT-Client mit einem anderen Gerät oder einer anderen Anwendung zu verbinden, da MQTT ein Messaging- und Warteschlangensystem ist. Die MQTT-Clientbibliotheken verwalten die Netzverbindungen für Ihre Anwendung.

Sie haben zwei Möglichkeiten, Ihren MQTT-Client in vorhandene Unternehmensanwendungen zu integrieren. Sie können die MQTT-Publish/Subscribe-Themen mit (zum Beispiel) einer IBM WebSphere MQ- oder JMS-Anwendung gemeinsam nutzen oder einen eigenen Integrationsadapter als anderen MQTT-Client schreiben.

Heute bereits verfügbare Informationsquellen:

- Anwendungen für WebSphere MQ Telemetry entwickeln
- MQTT.org
- Eclipse Paho

Zugehörige Konzepte

„Erste Schritte mit MQTT-Servern” auf Seite 142

Erste Schritte mit dem MQTT-Client für Java

Verwenden Sie IBM MessageSight oder IBM WebSphere MQ als MQTT -Server, um den MQTT -Client für Java -Beispielanwendungen betriebsbereit zu machen. Die Beispielanwendungen verwenden eine Clientbibliothek aus dem MQTT-Software-Development-Kit (SDK) von IBM. Die `SampleAsyncCallBack` -Beispielanwendung ist ein Modell zum Schreiben von MQTT -Anwendungen für Android und andere ereignisgesteuerte Betriebssysteme.

Vorbereitende Schritte

- Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.. Siehe Systemvoraussetzungen für IBM Mobile Messaging und M2M Client-Pack.
- Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.

Informationen zu diesem Vorgang

Mithilfe dieser Aufgabe soll überprüft werden, ob Sie eine Beispielanwendung des MQTT-Clients für Java erstellen und ausführen können und ob Sie die Anwendung mit IBM WebSphere MQ oder IBM MessageSight als MQTT version 3-Server verbinden und Nachrichten austauschen können.

Folgen Sie den Anweisungen in diesem Abschnitt, um die Beispielanwendung über die Eclipse-Workbench oder über eine Befehlszeile auszuführen. Die Schritte im Beispiel gelten für Windows. Mit geringfügigen Änderungen kann die Beispielanwendung auf allen Plattformen ausgeführt werden, die JSE 1.5 oder höher unterstützen.

Sie können Anwendungen auf demselben Server wie IBM WebSphere MQ ausführen, auf dem die Umgebung für die Ausführung von Anwendungen, die Verbindungen mit IBM WebSphere MQ herstellen, für Sie konfiguriert ist. Folgen Sie den Anweisungen im Abschnitt „MQTT-Service über die Befehlszeile konfigurieren" auf Seite 146, um IBM WebSphere MQ mit der Option IBM WebSphere MQ Telemetry unter Windows oder Linux zu installieren und zu konfigurieren. Wenn die Umgebung installiert und konfiguriert ist, führen Sie die Beispielanwendung `MQTTV3Sample` aus, um die Installation zu überprüfen.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

   Der Server muss das MQTT version 3.1 -Protokoll unterstützen. Alle MQTT-Server von IBM führen dies aus, einschließlich IBM WebSphere MQ und IBM MessageSight. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern" auf Seite 142.
2. Optional: Konfigurieren Sie den MQTT-Server.
   - Sie müssen in IBM WebSphere MQ eine der folgenden Aufgaben ausführen, um einen Warteschlangenmanager einzurichten und seinen Telemetrieservice (MQXR) zu konfigurieren:
     - „MQTT-Service über die Befehlszeile konfigurieren" auf Seite 146
     - „MQTT-Service über IBM WebSphere MQ Explorer konfigurieren" auf Seite 149
   - Ziehen Sie bei anderen Servern die Serverdokumentation hinzu. Für den Really Small Message Broker sind keine Konfigurationsschritte erforderlich. Weitere Informationen finden Sie im Abschnitt Really Small Message Broker.
3. Laden Sie Mobile Messaging und M2M Client-Pack herunter und installieren Sie das MQTT SDK.

   Es gibt kein Installationsprogramm; Sie müssen lediglich die heruntergeladene Datei dekomprimieren.
   a. Laden Sie Mobile Messaging und M2M Client-Packherunter.
   b. Erstellen Sie einen Ordner, in dem Sie das Software-Development-Kit (SDK) installieren werden.

      Geben Sie dem Ordner zum Beispiel den Namen `MQTT`. Der Pfad zu diesem Ordner wird hier als *sdkroot*bezeichnet.
   c. Entpacken Sie den Inhalt der komprimierten Datei Mobile Messaging und M2M Client-Pack in *sdkroot*. Die Erweiterung erstellt eine Verzeichnisbaumstruktur, die bei *sdkroot*\SDKbeginnt.
4. Installieren Sie ein Java Development Kit (JDK) Version 6 oder höher.

   Da Sie eine Java -App entwickeln for Android, muss JDK aus Oraclestammen. Sie können das JDK unter Java SE-Downloads abrufen.
5. Kompilieren Sie einen oder mehrere der MQTT-Clients für Java-Beispielanwendungen und führen Sie sie aus:
   - „Paho-Beispielprogramme über die Befehlszeile kompilieren und ausführen" auf Seite 14
   - „Java-Beispiel-Apps des MQTT-Clients über Eclipse kompilieren und ausführen" auf Seite 16
   - „Erste Schritte mit dem MQTT-Client für Java unter Android" auf Seite 19

   Folgende Java-Beispiel-Apps des MQTT-Clients sind im SDK enthalten:

MQTTV3Sample
Das Beispiel ist auch in IBM WebSphere MQ enthalten und stellt eine Verknüpfung mit dem Paket `com.ibm.micro.client.mqttv3.jar` her.

Sample
`Sample` befindet sich im Paket Paho und ist mit dem Paket `org.eclipse.paho.client.mqttv3` verknüpft. Die Anwendung entspricht `MQTTV3Sample`; sie wartet, bis jede einzelne MQTT-Aktion beendet ist.

SampleAsyncWait
`SampleAsyncWait` ist im Paket `org.eclipse.paho.client.mqttv3` enthalten. Es verwendet die asynchrone MQTT-API und wartet auf einen anderen Thread, bis eine Aktion beendet wird. Der Hauptthread kann andere Arbeiten erledigen, bis er mit dem Thread synchron ist, der auf die Beendigung der MQTT-Aktion wartet.

SampleAsyncCallBack
`SampleAsyncCallBack` ist im Paket `org.eclipse.paho.client.mqttv3` enthalten. Die asynchrone MQTT-API wird aufgerufen. Die asynchrone API wartet nicht, bis MQTT die Verarbeitung eines Aufrufs beendet, sondern kehrt zur Anwendung zurück. Die Anwendung fährt mit anderen Aufgaben fort und wartet dann auf das nächste zu verarbeitende Ereignis. MQTT sendet eine Ereignisbenachrichtigung an die Anwendung, sobald es die Verarbeitung beendet hat. Die ereignisgesteuerte MQTT-Schnittstelle ist an das Service- und Aktivitätsprogrammiermodell von Android und anderen ereignisgesteuerten Betriebssystemen angepasst.

Betrachten Sie zum Beispiel, wie `mqttExerciser` mithilfe des Service- und Aktivitätsprogrammiermodells MQTT in Android integriert.

mqttExerciser
`mqttExerciser` ist ein Beispielprogramm für Android. Da es auf andere Weise erstellt und ausgeführt wird, wird es gesondert beschrieben. Weitere Informationen finden Sie unter „Erste Schritte mit dem MQTT-Client für Java unter Android" auf Seite 19.

Ergebnisse

Sie haben die MQTT-Java-Beispielanwendungen, die mit IBM WebSphere MQ oder IBM MessageSight als MQTT-Server verbunden werden, kompiliert und ausgeführt.

Nächste Schritte

Lesen Sie die Javadoc-Referenzinformationen (siehe Schritt „3" auf Seite 17 im Abschnitt „Java-Beispiel-Apps des MQTT-Clients über Eclipse kompilieren und ausführen" auf Seite 16). Öffnen Sie alternativ die Javadoc-HTML-Dateien, die sich im Verzeichnis `SDK\clients\java\doc\javadoc` im Mobile Messaging und M2M Client-Pack befinden.

Paho-Beispielprogramme über die Befehlszeile kompilieren und ausführen

Kompilieren und führen Sie die Paho Beispielanwendung `Sample.java` über die Befehlszeile aus. Das Beispiel ist im MQTT-SDK enthalten. Das Beispiel wird mit den MQTT Paho -Clientbibliotheken im Paket `org.eclipse.paho.client.mqttv3` erstellt. Es veranschaulicht einen MQTT-Bereitsteller und -Subskribenten. Zwei andere Paho-Beispiele in demselben Verzeichnis können auf dieselbe Weise erstellt und ausgeführt werden. Sie unterscheiden sich dadurch, dass sie die MQTT-Bibliothek asynchron aufrufen.

Vorbereitende Schritte

Führen Sie die Schritte „1" auf Seite 13 bis „4" auf Seite 13 in der Hauptaufgabe aus, um einen MQTT-Server zu konfigurieren und das Mobile Messaging und M2M Client-Pack herunterzuladen.

Informationen zu diesem Vorgang

Kompilieren Sie `Sample.java` aus dem Unterverzeichnis `SDK\clients\java\samples` für SDK-Clientbeispiele und führen Sie es aus. Der Java-Code ist im Verzeichnis `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` enthalten.

Vorgehensweise

1. Erstellen Sie ein Script im Clientbeispielverzeichnis, um `Sample` auf Ihrer ausgewählten Plattform zu kompilieren und auszuführen.

   Mit folgendem Script wird das Beispiel unter Windows kompiliert und ausgeführt.

```
@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\sam
ple\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.sam
ple.mqttv3app.Sample -b localhost -p 1883
pause
endlocal
```

   Abbildung 5. `Sample.java` kompilieren und ausführen

2. Führen Sie das Script aus.

   Ergebnisse:

```
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
```

   Abbildung 6. `MQTTV3Sample`-Subskribent

```
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
```

   Abbildung 7. `MQTTV3Sample`-Bereitsteller

   Wenn Sie die aktive Subskribentenanwendung verlassen, können Sie denselben Subskribenten nicht erneut ausführen. Die Client-ID des neuen Subskribenten ist mit der des alten Subskribenten identisch. Es können nicht zwei MQTT-Clients mit derselben Client-ID gleichzeitig ausgeführt werden. Sie können die Option `-i` angeben, um die Client-ID so festzulegen, dass Sie verschiedene Subskribenten gleichzeitig ausführen können.

   Wenn Sie denselben Client erneut ausführen, können Sie die Option `-c` nutzen, um den Client zu starten und `cleansession` dabei auf 'false' zu setzen. Mithilfe der Option können Sie das Verhalten von unterbrochenen Clientsitzungen untersuchen.

3. Beenden Sie den Subskribenten, indem Sie die Eingabetaste drücken oder das Fenster schließen.

Nächste Schritte

Erstellen Sie Scripts zum Kompilieren und Ausführen anderer Beispiele im Unterverzeichnis `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app`. Kopieren Sie das Script in Abbildung 5 auf Seite 15 und ersetzen Sie `Sample` durch `SampleAsyncWait` oder `SampleAsyncCallBack`. Die anderen Beispiele sind asynchrone Versionen des synchronen Programms `Sample`.

SampleAsyncWait
SampleAsyncWait ist im Paket org.eclipse.paho.client.mqttv3 enthalten. Es verwendet die asynchrone MQTT-API und wartet auf einen anderen Thread, bis eine Aktion beendet wird. Der Hauptthread kann andere Arbeiten erledigen, bis er mit dem Thread synchron ist, der auf die Beendigung der MQTT-Aktion wartet.

SampleAsyncCallBack
SampleAsyncCallBack ist im Paket org.eclipse.paho.client.mqttv3 enthalten. Die asynchrone MQTT-API wird aufgerufen. Die asynchrone API wartet nicht, bis MQTT die Verarbeitung eines Aufrufs beendet, sondern kehrt zur Anwendung zurück. Die Anwendung fährt mit anderen Aufgaben fort und wartet dann auf das nächste zu verarbeitende Ereignis. MQTT sendet eine Ereignisbenachrichtigung an die Anwendung, sobald es die Verarbeitung beendet hat. Die ereignisgesteuerte MQTT-Schnittstelle ist an das Service- und Aktivitätsprogrammiermodell von Android und anderen ereignisgesteuerten Betriebssystemen angepasst.

Betrachten Sie zum Beispiel, wie mqttExerciser mithilfe des Service- und Aktivitätsprogrammiermodells MQTT in Android integriert.

Die asynchronen Beispiele zeigen, wie die Zeit, die eine MQTT-Anwendung blockiert ist, während sie auf den MQTT-Client wartet, verkürzt werden kann. Es ist wichtig, blockierende Aufrufe im Hauptthread zu vermeiden, um die Reaktionsfähigkeit und die Lebensdauer des Akkus in einer mobilen Umgebung zu verbessern.

Die Beispiele zeigen zwei Muster für den Aufruf asynchroner Schnittstellen im MQTT-Client.

1. SampleAsyncWait wird nicht blockiert, während MQTT auf Netzinteraktionen wartet.
2. SampleAsyncCallBack wird nicht blockiert, während es darauf wartet, dass der MQTT-Client irgendwelche Aktionen beendet. Letzteres ist notwendig, wenn eine JavaScript-Seite den MQTT-Client über einen Browser aufruft. JavaScript-Seiten dürfen nicht blockiert werden. Antworten auf Aktionen müssen an den Haupt-Browser-Thread zurückgesendet werden, der den MQTT-Ereignishandler aufruft, den Sie für die Verarbeitung der Benachrichtigung schreiben.

Java-Beispiel-Apps des MQTT-Clients über Eclipse kompilieren und ausführen

Kompilieren und führen Sie die Java -Beispielapps für den MQTT -Client aus, die sich in Mobile Messaging und M2M Client-Packbefinden. Sie veranschaulichen einen MQTT-Bereitsteller und -Subskribenten.

Informationen zu diesem Vorgang

Kompilieren und führen Sie die MQTT Java -Beispiele Sampleund MQTTV3Sample in Eclipseaus. Sample befindet sich im Unterverzeichnis für *sdkroot*\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app SDK-Clients und MQTTV3Sample.java im Verzeichnis *sdkroot*\SDK\clients\java\samples.

Vorgehensweise

1. Laden Sie Eclipse IDE for Java Developers herunter.
2. Erstellen Sie ein Java -Projekt mit dem Namen MQTT Samples in Eclipse.
   a) **Datei** > **Neu** > **Java-Projekt** und Typ MQTT Samples. Klicken Sie auf **Weiter**.

      Prüfen Sie, ob die korrekte JRE-Version oder eine höhere Version verwendet wird.JSE muss Version 1.5 oder höher aufweisen.
   b) Klicken Sie im Fenster **Java Settings** (Java-Einstellungen) auf **Link additional source folders** (Zusätzliche Quellenordner verknüpfen).
   c) Navigieren Sie zu dem Verzeichnis, in dem Sie den MQTT-Java-SDK-Ordner installiert haben. Wählen Sie den Ordner *sdkroot*\SDK\clients\java\samples aus und klicken Sie auf **OK** > **Weiter** > **Fertigstellen**.

d) Klicken Sie im Fenster **Java Settings** (Java-Einstellungen) auf **Libraries** > **Add External Jars** (Bibliotheken > Externe JAR-Dateien hinzufügen).

e) Navigieren Sie zu dem Verzeichnis, in dem Sie den MQTT-Java-SDK-Ordner installiert haben. Suchen Sie den Ordner *sdkroot*\SDK\clients\java und wählen Sie die Dateien `org.eclipse.paho.client.mqttv3.jar` und `com.ibm.micro.client.mqttv3.jar` aus; klicken Sie auf **Öffnen** > **Fertigstellen**.

Das Beispiel `MQTTV3Sample.java` stellt eine Verknüpfung mit `com.ibm.micro.client.mqttv3.jar` und die Beispiele in der Verzeichnisstruktur paho stellen eine Verknüpfung mit `org.eclipse.paho.client.mqttv3.jar` her. Der `com.ibm.micro.client.mqttv3.jar` wird beibehalten, sodass vorhandene MQTT -Anwendungen weiterhin unverändert erstellt und ausgeführt werden.

Das Projekt `MQTT Samples` wird mit Warnungen, aber ohne Fehler erstellt.

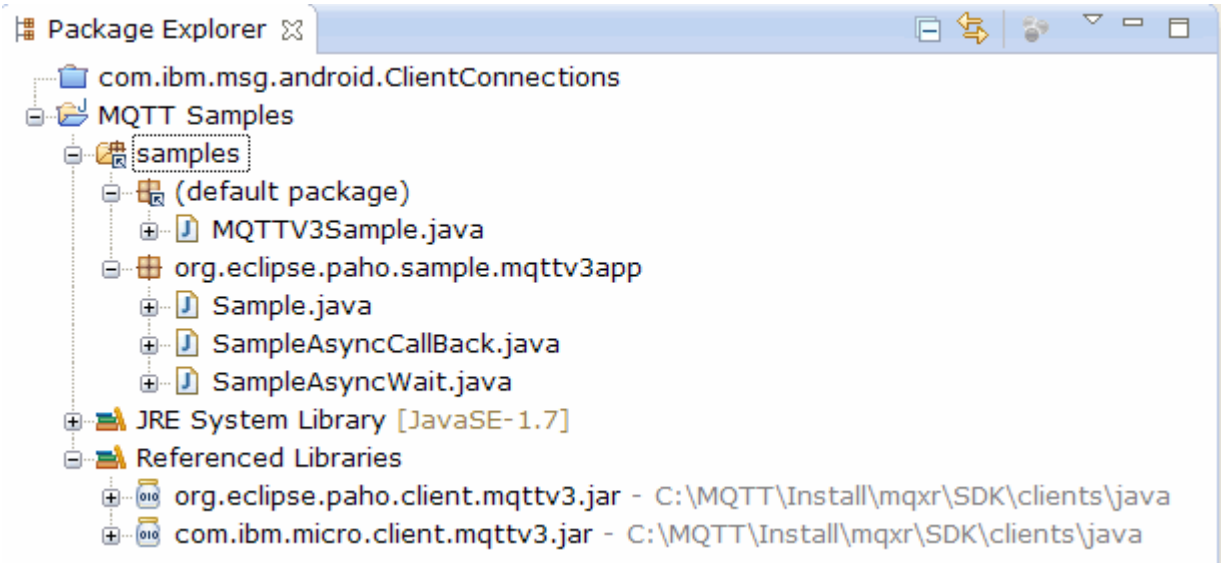


Abbildung 8. Projekt des MQTT-Clients für Java

3. Optional: Installieren Sie die Javadoc für den MQTT-Client.

Wenn das Javadoc des MQTT -Clients installiert ist, beschreibt der Java-Editor die MQTT -Klassen in der Kurzinfo.

a) Öffnen Sie **Paketexplorer** > **Referenzierte Bibliotheken** in Ihrem Java-Projekt. Klicken Sie mit der rechten Maustaste auf `org.eclipse.paho.client.mqttv3.jar` **> Eigenschaften**.

b) Klicken Sie im Eigenschaftennavigator auf **Javadoc Location (Javadoc-Position)**.

c) Klicken Sie auf **Javadoc-URL** > **Durchsuchen** auf der Seite **Javadoc-Position** und suchen Sie den Ordner SDK\clients\java\doc\javadoc **> OK**.

d) Klicken Sie auf **Validate** > **OK** (Prüfen).

Sie werden aufgefordert, einen Browser zum Anzeigen der Dokumentation zu öffnen.

e) Wiederholen Sie diesen Vorgang für die Datei `com.ibm.micro.client.mqttv3.jar`.

4. Erstellen Sie eine Bereitsteller- und Subskribentenlaufzeitkonfiguration, um die Anwendung `mqttv3app.Sample` auszuführen.

a) Klicken Sie mit der rechten Maustaste auf die Klasse **Sample** und klicken Sie auf **Ausführen als** > **Ausführungskonfigurationen**.

b) Klicken Sie mit der rechten Maustaste auf **Java-Anwendung** > **Neu** und geben Sie den Namen `SampleSubscriber`ein.

c) Klicken Sie auf die Argumenteregisterkarte, geben Sie die Programmargumente ein und klicken Sie dann auf **Apply (Anwenden)**.

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

d) Wiederholen Sie den letzten Schritt, um eine `SamplePublisher`-Konfiguration zu erstellen, ohne dabei den Parameter `-a subscribe` anzugeben.

5. Führen Sie den `mqttv3app.Sample`-Subskribenten und anschließend den Bereitsteller aus.
   a) Klicken Sie auf **Run** > **Run configurations** (Ausführen > Konfigurationen ausführen).
   b) Klicken Sie auf **SampleSubscriber** > **Ausführen**.

      Öffnen Sie die Ansicht **Console (Konsole)**. Der Subskribent wartet auf eine Veröffentlichung.

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

   c) Klicken Sie auf **SamplePublisher** > **Ausführen**.

      Öffnen Sie die Ansicht **Console (Konsole)**. Sie sehen die Veröffentlichung, die vom Bereitsteller erstellt wird:

      ```
      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) Wechseln Sie in den Konsolenansichten zur Subskribentenkonsole.

      Das Symbol für den Wechsel zwischen Konsolen ist  .

      Der Subskribent hat die Veröffentlichung empfangen:

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

6. Optional: Erstellen Sie eine Bereitsteller- und Subskribentenlaufzeitkonfiguration für `MQTTV3Sample`.
   a) Klicken Sie mit der rechten Maustaste auf die Klasse **MQTTV3Sample** und klicken Sie auf **Ausführen als** > **Ausführungskonfigurationen**.
   b) Klicken Sie mit der rechten Maustaste auf **Java-Anwendung** > **Neu** und geben Sie den Namen `MQTTV3SampleSubscriber`ein.
   c) Klicken Sie auf die Argumenteregisterkarte, geben Sie die Programmargumente ein und klicken Sie dann auf **Apply (Anwenden)**.

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

   d) Wiederholen Sie den letzten Schritt, um eine `MQTTV3SamplePublisher`-Konfiguration zu erstellen, ohne dabei den Parameter `-a subscribe` anzugeben.

7. Optional: Führen Sie den `MQTTV3Sample`-Subskribenten und anschließend den Bereitsteller aus.
   a) Klicken Sie auf **Run** > **Run configurations** (Ausführen > Konfigurationen ausführen).
   b) Klicken Sie auf **MQTTV3SampleSubscriber** > **Ausführen**.

      Öffnen Sie die Ansicht **Console (Konsole)**. Der Subskribent wartet auf eine Veröffentlichung.

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

   c) Klicken Sie auf **MQTTV3SamplePublisher** > **Ausführen**.

      Öffnen Sie die Ansicht **Console (Konsole)**. Sie können die Veröffentlichung sehen, die vom Bereitsteller erstellt wird.

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

d) Wechseln Sie in den Konsolenansichten zur Subskribentenkonsole.

Das Symbol für den Wechsel zwischen Konsolen ist  .

Der Subskribent hat die Veröffentlichung empfangen:

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

Erste Schritte mit dem MQTT-Client für Java unter Android

Sie können eine MQTT-Clientbeispiel Java -App für Android installieren, die Nachrichten mit einem MQTT-Server austauscht. Diese App verwendet eine Clientbibliothek aus dem MQTT-Software-Development-Kit von IBM. YouSie können die App entweder selbst erstellen oder eine vorgefertigte Beispiel-App herunterladen.

Vorbereitende Schritte

- Informationen zu unterstützten Plattformen und Referenzplattformen für den MQTT-Client finden Sie im Abschnitt Systemvoraussetzungen für IBM Mobile Messaging und M2M Client-Pack.
- Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.
- Die MQTT-Client Beispiel-App kann auf Ice Cream Sandwich (Android 4.0) und höher ausgeführt werden. Diese Version von Android bringt auch eine höhere Anzeigeauflösung auf Tablet-Computern.

Informationen zu diesem Vorgang

Die MQTT-Clientbeispiel Java -App für Android heißt "`mqttExerciser`". Diese App verwendet eine Clientbibliothek aus dem Software-Development-Kit für MQTT und tauscht Nachrichten mit einem MQTT-Server aus.

Sie können die Beispielapp selbst erstellen und anschließend aus Eclipse als `mqttExerciser.apk`exportieren oder die vordefinierte Beispielapp verwenden, die als Datei `mqttExerciser.apk` im Ordner *`sdkroot`*`\SDK\clients\android\samples\apks` der Mobile Messaging und M2M Client-Packverfügbar ist. Wenn Sie die App selbst erstellen, wird die Entwicklungsumgebung, die Sie erstellen, angepasst, um mobiles Messaging in Ihre for Android-Apps einzuschließen. Das müsste Ihnen helfen, wenn Sie damit beginnen, mobiles Messaging in Ihre eigenen Apps einzuschließen.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

   Der Server muss das MQTT version 3.1 -Protokoll unterstützen. Alle MQTT-Server von IBM führen dies aus, einschließlich IBM WebSphere MQ und IBM MessageSight. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern” auf Seite 142.
2. Die richtigen Tools abrufen.

   Installieren Sie ein Java Development Kit (JDK) Version 6 oder höher. Da Sie eine Java -App entwickeln for Android, muss JDK aus Oraclestammen. Sie können das JDK unter Java SE-Downloads abrufen.

   Sie benötigen auch eine Eclipse-Entwicklungsumgebung. Diese muss Eclipse 3.6.2 (Helios) oder höher sein. Eclipse muss eine Java-Compiler-Version von mindestens 6 aufweisen, um mit Ihrem Java Development Kit übereinzustimmen. Sie können all dies über die Eclipse Foundationabrufen.

   Schließlich benötigen Sie noch das Android-Software-Development-Kit. Dieses können Sie unter Android-SDK abrufen abrufen.
3. Laden Sie Mobile Messaging und M2M Client-Pack herunter und installieren Sie das MQTT SDK.

   Es gibt kein Installationsprogramm; Sie müssen lediglich die heruntergeladene Datei dekomprimieren.

a. Laden Sie Mobile Messaging und M2M Client-Packherunter.

b. Erstellen Sie einen Ordner, in dem Sie das Software-Development-Kit (SDK) installieren werden.

   Geben Sie dem Ordner zum Beispiel den Namen `MQTT`. Der Pfad zu diesem Ordner wird hier als *`sdkroot`*bezeichnet.

c. Entpacken Sie den Inhalt der komprimierten Datei Mobile Messaging und M2M Client-Pack in *`sdkroot`*. Die Erweiterung erstellt eine Verzeichnisbaumstruktur, die bei *`sdkroot`*`\SDK`beginnt.

4. Optional: mqttExerciser-Beispiel-App erstellenfor Android.

   Konfigurieren Sie die Tools Eclipse und Android und importieren und erstellen Sie das `mqttExerciser` -Projekt aus dem MQTT SDK.

   Anmerkung: Wenn Sie dies nicht sofort ausführen möchten, können Sie die vorgefertigte Beispiel-App verwenden, die als Datei `mqttExerciser.apk` im Ordner *`sdkroot`*`\SDK\clients\android\samples\apks` des MQTT-SDK verfügbar ist.

   a) Starten Sie die Eclipse-Entwicklungsumgebung mit der JRE aus dem JDK.

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

   b) Wählen Sie eine Gruppe von Paketen und Plattformen aus dem Android-SDK aus und installieren Sie sie.

      Auf der Webseite Adding Platforms and Packages (Plattformen und Pakete hinzufügen) finden Sie eine Liste der von Google empfohlenen Plattformen und Pakete.

      Anmerkung: Die Software-Development-Kit-Plattform muss die Android-API-Version 16 oder höher aufweisen. Das Projekt kann mit älteren API-Versionen nicht erfolgreich kompiliert werden.

   c) Fügen Sie das ADT-Plug-in (ADT = Android Development Tools) zu Eclipse hinzu.

   d) Importieren Sie das `mqttExerciser`-Beispiel-App-Projekt in Eclipse und korrigieren Sie die Fehler.

      i) Importieren Sie das Beispiel-App-Projekt aus dem MQTT SDK im Pfad *`sdkroot`*`\SDK\clients\android\samples\mqttExerciser`.

         In der Ansicht **Problems** (Probleme) werden zahlreiche Buildfehler aufgelistet. Sie beheben die Buildfehler in den nächsten Schritten.

      ii) Kopieren Sie die Bibliothek `org.eclipse.paho.client.mqttv3.jar` in den Ordner **libs** im Projekt Android . Windows Unter Windowsist dies beispielsweise der Ordner *`sdkroot`*`\SDK\clients\java` . Es wird ein **Dateivorgang**-Fenster angezeigt. Akzeptieren Sie die Auswahl **Dateien kopieren** und klicken Sie anschließend auf **OK**.

      iii) Klicken Sie mit der rechten Maustaste auf den Projektordner `com.ibm.msg.android`; klicken Sie auf **Android-Tools ...** > **Unterstützungsbibliothek hinzufügen ...**. Lesen und akzeptieren Sie die Lizenzbedingungen und klicken Sie anschließend auf **Install (Installieren)**.

      iv) Klicken Sie mit der rechten Maustaste auf den Projektordner `com.ibm.msg.android`; klicken Sie auf **Android-Tools ...** > **Projekteigenschaften korrigieren**.

      v) Wenn der Arbeitsbereich dennoch etwa 84 Fehler aufweist, die auf das Überschreiben einer Superklassenmethode hinweisen, ist die Compiler-Compliance-Version wahrscheinlich auf 1.5 oder niedriger gesetzt. Android SDK Version 16 erwartet, dass die Compiler-Compliance-Version nicht höher als 1.5. ist. Führen Sie folgende Schritte aus, um die verbleibenden Fehler zu korrigieren:

         a) Überprüfen Sie Ihr Android SDK und die entsprechenden Eclipse-Plugins und aktualisieren Sie sie falls notwendig auf Android SDK Version 17.

         b) Klicken Sie mit der rechten Maustaste auf den Projektordner **com.ibm.msg.android** und wählen Sie dann **Eigenschaften** > **Java-Compiler**aus. Überprüfen Sie die Compiler-Compliance-Version, setzen Sie sie mindestens auf 1.6 und und erstellen Sie anschließend den Arbeitsbereich erneut.

Das Projekt wird mit einigen Warnungen, aber ohne Fehler erstellt.

5. Installieren und starten Sie die MQTT-Clientbeispiel Java -App auf einem Android-Gerät.

   Weitere Informationen finden Sie auf der `developer.android.com`-Seite Ausführung Ihrer App.

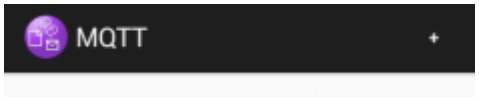
   Wenn Sie die App selbst als Eclipse-Projekt erstellen, können Sie die App über Eclipse starten.

   Wenn Ihnen die Anwendungspaketdatei (APK-Datei) `mqttExerciser.apk` zur Verfügung steht, können Sie diese außerhalb von Eclipse mithilfe des Installationsbefehls Android Debug Bridge (ADB) installieren. In diesem Befehl wird die Speicherposition der APK-Datei als Argument verwendet. Wenn Sie die vorgefertigte Beispiel-App verwenden, lautet die Speicherposition *sdkroot*`\SDK\clients\android\samples\apks\mqttExerciser.apk`.

6. Verwenden Sie die mqttExerciser-Beispiel-App for Android, um sich mit einem Thema zu verbinden, ein Thema zu subskribieren und zu veröffentlichen.

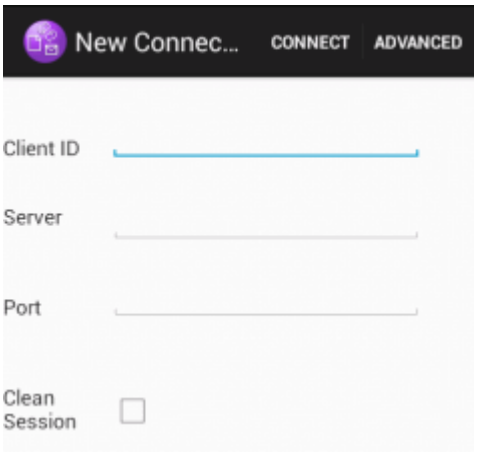
   a) Öffnen Sie die MQTT-Clientbeispiel Java -App für Android.

      Auf Ihrem Android-Gerät wird folgendes Fenster geöffnet:

   b) Stellen Sie eine Verbindung zu einem MQTT -Server her.

      i) Klicken Sie auf das Pluszeichen (**+**), um eine neue MQTT-Verbindung zu öffnen.

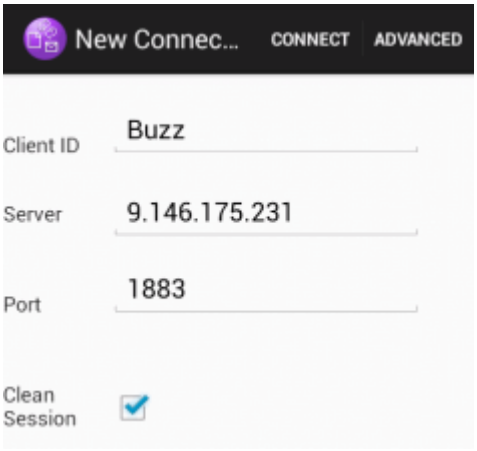
      ii) Geben Sie im Feld **Client-ID** eine eindeutige ID ein. Haben Sie etwas Geduld, die Tastatureingabe ist möglicherweise etwas langsam.

      iii) Geben Sie im Feld **Server** die IP-Adresse Ihres MQTT-Servers ein.

      Dies ist der Server, den Sie im ersten Hauptschritt ausgewählt haben. Die IP-Adresse darf nicht `127.0.0.1` lauten.

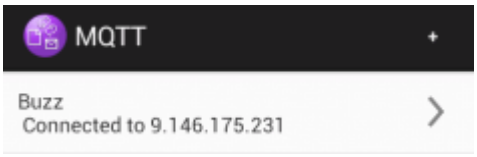
      iv) Geben Sie die Portnummer der MQTT-Verbindung ein.

      Die Standardportnummer für eine normale MQTT -Verbindung ist 1883.



v) Klicken Sie auf **Connect** (Verbinden).

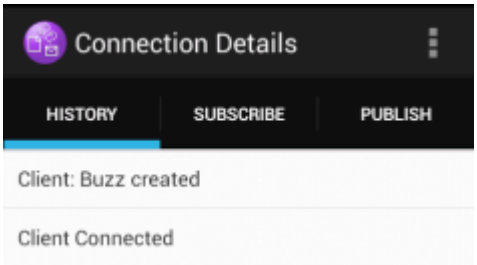
Wird die Verbindung erfolgreich hergestellt, sehen Sie eine "Connecting"-Nachricht und danach folgendes Fenster:



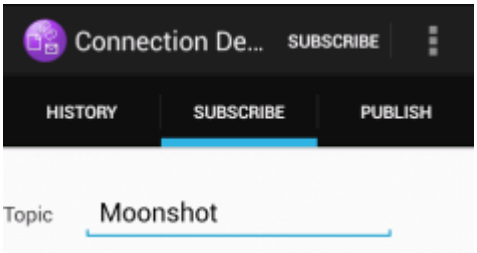
c) Subskribieren Sie ein Thema.

i) Klicken Sie auf die Nachricht **Connected** (Verbunden).

Es wird das Fenster **Connection Details** (Verbindungsdetails) mit dem Verlaufsprotokoll angezeigt:



ii) Klicken Sie auf die Registerkarte **Subscribe** (Subskribieren) und geben Sie eine Topic-Zeichenfolge ein.

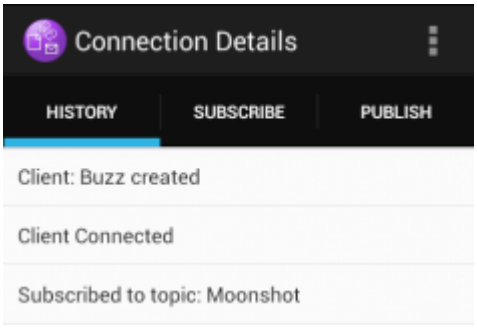


iii) Klicken Sie auf die Aktion **Subscribe** (Subskribieren).

Die Nachricht "Subscribed" (Subskribiert) erscheint für kurze Zeit.

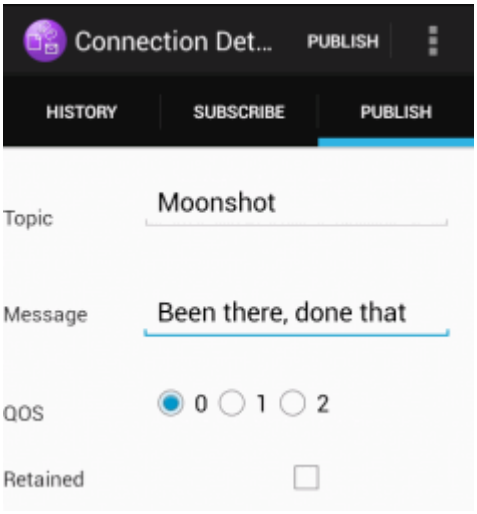
iv) Klicken Sie auf die Registerkarte **History** (Protokoll).

Die Subskription ist jetzt im Protokoll enthalten:



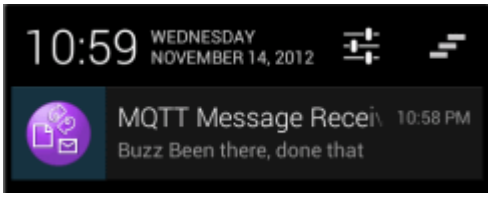
d) Veröffentlichen Sie jetzt im selben Thema.

i) Klicken Sie auf die Registerkarte **Publish** (Veröffentlichen) und geben Sie dieselbe Themenzeichenfolge ein, die Sie für die Subskription verwendet haben. Geben Sie eine Nachricht ein.

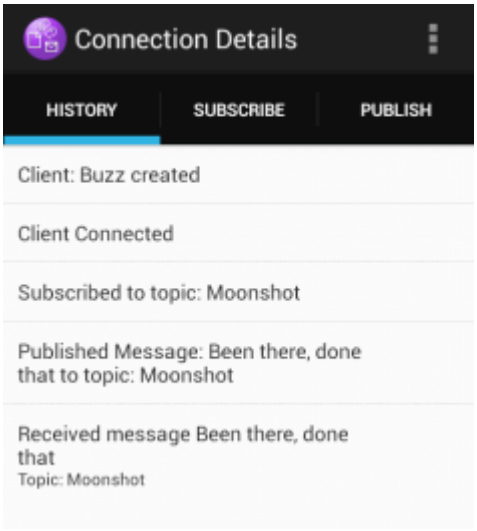


ii) Klicken Sie auf die Aktion **Publish** (Veröffentlichen).

Für kurze Zeit werden zwei Nachrichten angezeigt, zunächst "Published" (Veröffentlicht), danach "Subscribed" (Subskribiert). Die Veröffentlichung wird im Statusbereich angezeigt (ziehen Sie die Trennlinie nach unten, um das Statusfenster zu öffnen).



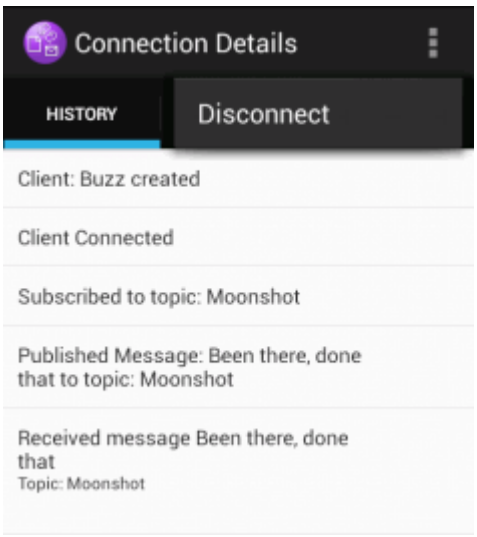
iii) Klicken Sie auf die Registerkarte **History**, um das vollständige Protokoll anzuzeigen.



e) Trennen Sie die Verbindung zur Clientinstanz.

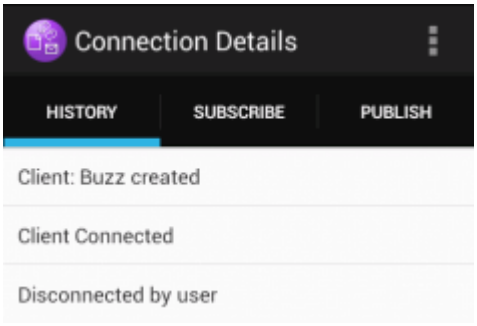
i) Klicken Sie auf das Menüsymbol in der Aktionsleiste. 

Die MQTT-Clientbeispiel Java -App für Android fügt die Schaltfläche **Disconnect** (Verbindung trennen) zum MQTT-Fenster **Connection Details** (Verbindungsdetails) hinzu.

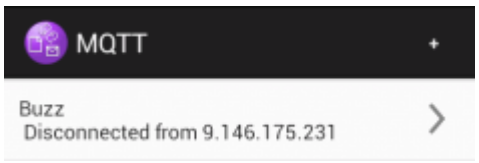


ii) Klicken Sie auf **Disconnect** (Verbindung trennen).

Der Verbindungsstatus ändert sich; die Verbindung ist getrennt:



f) Klicken Sie auf **Back** (Zurück), um zur Liste mit den MQTT-Clientbeispiel Java -App-Sitzungen zurückzukehren.



- Klicken Sie auf das Pluszeichen, um eine neue MQTT-Clientbeispiel Java -App-Sitzung zu starten.
- Klicken Sie auf den getrennten Client, um die Verbindung wiederherzustellen.
- Klicken Sie auf **Back** (Zurück), um zum Launchpad zurückzukehren.

g) Klicken Sie auf die Aufgabenschaltfläche, um aktive Apps aufzulisten. Suchen Sie die MQTT-Clientbeispiel Java -App. Schieben Sie das Symbol vom Bildschirm, um die App zu schließen.

Nächste Schritte

Wenn Sie die Beispiel-App selbst erstellt haben, können Sie mit der Entwicklung Ihrer eigenen Android-Apps beginnen, die MQTT-Bibliotheken zum Austausch von Nachrichten aufrufen. Sie können Ihre Android -Apps anhand der Klassen in `mqttExerciser`modellieren. Um das Beispiel zu studieren, müssen Sie die Javadoc für die Klassen in `com.ibm.msg.android` und `com.ibm.msg.android.service` im Projekt `mqttExerciser` generieren.

Zugehörige Informationen

Projekte aus Eclipse mit ADT verwalten

Erste Schritte mit dem MQTT-Messaging-Client für JavaScript

Für die ersten Schritte mit dem MQTT-Messaging-Client für JavaScript können Sie die Beispielhomepage für den Messaging-Client anzeigen und die Ressourcen durchsuchen, zu denen Links vorhanden sind. Um diese Homepage anzuzeigen, konfigurieren Sie einen MQTT-Server für die Annahme von Verbindungen von den Beispielseiten des MQTT-Messaging-Clients JavaScript und geben anschließend die URL, die Sie auf dem Server konfiguriert haben, in einen Web-Browser ein. Der MQTT-Messaging-Client für JavaScript wird automatisch auf Ihrem Gerät gestartet und die Beispielhomepage für den Messaging-Client wird angezeigt. Diese Seite enthält Verknüpfungen zu Dienstprogrammen, einer Dokumentation zur Programmierschnittstelle, einem Lernprogramm und weiteren hilfreichen Informationsquellen.

Vorbereitende Schritte

Zur erweiterten Verwendung oder zur Verwendung in der Produktion sollten Sie die Beispielhomepage für den Messaging-Client umgestalten oder entfernen. Beachten Sie, dass für Benutzerschnittstellen, die aus dem Beispielcode resultieren, nicht gewährleistet ist, dass sie mit Standards oder Anforderungen für Eingabehilfen konform sind.

Sie benötigen einen MQTT -Server zur Unterstützung von MQTT-Messaging-Client für JavaScript. Dieser Server muss das MQTT V3.1 -Protokoll über WebSocketsunterstützen. IBM MessageSight und IBM WebSphere MQ Version 7.5.0, Fix Pack 1 und höhere Versionen unterstützen MQTT protocol über WebSockets. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern” auf Seite 142. Informationen zur Installation einer kostenlosen 90-Tage-Testversion von IBM WebSphere MQ finden Sie im Abschnitt „IBM WebSphere MQ installieren” auf Seite 144.

Das WebSocket protocol wurde kürzlich eingerichtet. Falls es zwischen Ihrem Client und dem Server eine Firewall gibt, stellen Sie sicher, dass sie den WebSockets-Datenverkehr nicht blockiert. Ebenso gilt: Wenn Ihr Browser die WebSocket protocol noch nicht unterstützt[1]können Sie das Clientdienstprogramm oder die Lernprogramme, die auf der Beispielhomepage für den Messaging-Client verfügbar sind, nicht verwenden. In Tabelle 1 auf Seite 26 sind die Browser aufgelistet, deren aktuellste Versionen getestet wurden und mit dem Messaging-Client kompatibel sind.

[1] Insbesondere, wenn der Standard RFC 6455 (WebSocket) nicht unterstützt wird.

Tabelle 1. Unterstützte Browser zur Verwendung mit dem MQTT-Messaging-Client für JavaScript			
Android	**iOS**	**Linux**	**Windows**
Firefox for Android 19.0 und höher Chrome for Android 25.0 und höher	Safari 6.0 und höher Chrome 14.0 und höher	Firefox 6.0 und höher Chrome 14.0 und höher	Firefox 6.0 und höher Chrome 14.0 und höher

Informationen zu diesem Vorgang

Die meisten Schritte in dieser Aufgabe betreffen die Konfiguration des MQTT-Servers. Für den Zugriff auf den Messaging-Client für JavaScript ist lediglich ein Browser erforderlich, der das WebSocket protocol unterstützt.

Führen Sie in IBM WebSphere MQ die Schritte zum Aktivieren von IBM WebSphere MQ Telemetry durch Erstellung der Beispielkanäle aus. Stellen Sie eine Verbindung mit dem standardmäßigen MQTT-WebSockets-Beispielkanal an Port 1883 her. Die URL der Beispielhomepage des Messaging-Clients lautet `http://`*`hostname`*`:1883` on IBM WebSphere MQ.

In IBM MessageSight müssen Sie das Gerät installieren und konfigurieren, den Messaging-Hub für die Annahme von Verbindungen konfigurieren und einen MQTT-WebSockets-Endpunkt erstellen.

Vorgehensweise

1. Laden Sie Mobile Messaging und M2M Client-Pack herunter und wählen Sie einen MQTT-Server aus, mit dem Sie die Client-App verbinden können.

   Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Clients” auf Seite 11.
2. Konfigurieren Sie Ihren MQTT-Server, sodass er Verbindungen von den HTML-Beispielseiten für den MQTT-Messaging-Client für JavaScript akzeptiert.
   - Unter IBM WebSphere MQ:
     - Wenn Sie bereits über einen IBM WebSphere MQ-Warteschlangenmanager verfügen, der für MQTT konfiguriert ist, ändern Sie das Protokoll in der Kanaldefinition so, dass sowohl MQTT als auch HTTP unterstützt werden. Siehe **`ALTER CHANNEL`**.
     - Führen Sie eine der folgenden Aufgaben aus, um einen IBM WebSphere MQ-Warteschlangenmanager zu erstellen und den MQTT-WebSockets-Beispielendpunkt zu konfigurieren:
       - „MQTT-Service über die Befehlszeile konfigurieren” auf Seite 146
       - „MQTT-Service über IBM WebSphere MQ Explorer konfigurieren” auf Seite 149
3. Öffnen Sie auf Ihrem Gerät einen Web-Browser.
4. Geben Sie die URL der Beispielhomepage für den Messaging-Client ein.
   - Unter IBM WebSphere MQist dies `http://`*`hostname`*`:1883`
   - Unter IBM MessageSightist dies `http://`*`hostname`*`:`*`port`*

     Dabei ist *`hostname`* der DNS-Name oder die IP-Adresse des Ethernet-Sockets, den Sie auf Ihrer IBM MessageSight -Appliance als Endpunkt konfiguriert haben, zu dem der Client eine Verbindung herstellen soll, und *`port`* ist die TCP/IP-Portnummer, die Sie dem Endpunkt für den Client zugewiesen haben.

   Die Beispielhomepage für den Messaging-Client wird angezeigt.

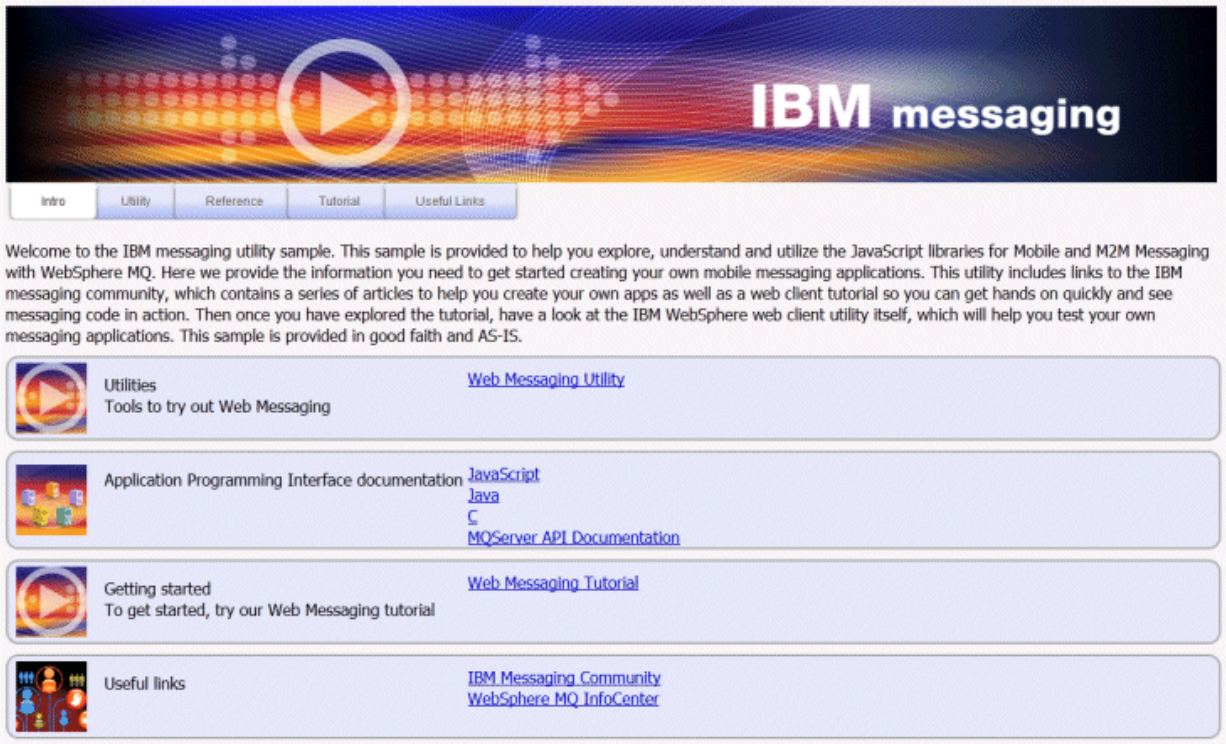


Abbildung 9. Beispielhomepage für den MQTT-Messaging-Client für JavaScript

Ergebnisse

Sie haben einen MQTT-Kanal für WebSockets konfiguriert.

Klicken Sie in der Beispielhomepage für den MQTT-Messaging-Client für JavaScript auf **Web Messaging Utility** (Web-Messaging-Dienstprogramm), um die verschiedenen Funktionen in der Messaging-Client-API auszuprobieren. Sie können beispielsweise eine Verbindung zum Warteschlangenmanager herstellen, Nachrichten subskribieren und anschließend einige Nachrichten veröffentlichen. Sie können auch auf **Web Messaging Tutorial** (Lernprogramm für Web-Messaging) klicken, um zu erfahren, wie eine Webseite erstellt wird, die die API des MQTT-Messaging-Clients für JavaScript aufruft.

Zugehörige Konzepte

„MQTT-Messaging-Client für JavaScript und Web-Apps” auf Seite 120

„Vorgehensweise bei der Programmierung von Messaging-Apps in JavaScript” auf Seite 124

Zugehörige Tasks

„MQTT-Messaging-Client für JavaScript über SSL und WebSockets verbinden” auf Seite 80

Verbinden Sie Ihre Web-App sicher mit IBM WebSphere MQ, indem Sie die HTML-Beispielseiten des MQTT-Messaging-Clients für MQTT-Messaging-Client für JavaScript> mit SSL und dem WebSocket protocol verwenden.

Erste Schritte mit dem MQTT-Client für C

Erstellen Sie den MQTT-Beispielclient für C auf einer beliebigen Plattform, auf der Sie die C-Quelle kompilieren können. Prüfen Sie, ob Sie den MQTT-Beispielclient für C mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen können.

Vorbereitende Schritte

- Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.
- Für den unterstützten und den MQTT-Referenzclient für C-Plattformen. (siehe Systemvoraussetzungen für IBM Mobile Messaging und M2M Client-Pack).

Informationen zu diesem Vorgang

Gehen Sie wie hier beschrieben vor, um den MQTT-Beispielclient für C unter Windows über die Befehlszeile oder über Microsoft Visual Studio 2010 zu kompilieren und auszuführen. Im Befehlszeilenbeispiel wird Microsoft Visual Studio 2010 auch zum Kompilieren des Clients verwendet. Ändern Sie die Befehlszeilenscripts, wenn Sie das Beispiel auf anderen Plattformen kompilieren und ausführen möchten.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

   Der Server muss das MQTT version 3.1 -Protokoll unterstützen. Alle MQTT-Server von IBM führen dies aus, einschließlich IBM WebSphere MQ und IBM MessageSight. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern” auf Seite 142.
2. Installieren Sie auf Ihrer Entwicklungsplattform eine C-Entwicklungsumgebung.

   Für die Makefiles in den Beispielen dieses Abschnitts werden die folgenden Tools verwendet:

   - 

iOS: Apple Mac mit OS X 10.8.2 mit den iOS-Entwicklungstools von Xcode.
   - 

Linux: GCC Version 4.4.6 aus Red Hat® Enterprise Linux Version 6.2.

     Die unterstützte Mindestversion der C-Bibliothek `glibc` ist 2.12 und die des Linux-Kernels ist 2.6.32.
   - 

Microsoft Windows: Visual Studio Version 10.0
3. Laden Sie Mobile Messaging und M2M Client-Pack herunter und installieren Sie das MQTT SDK.

   Es gibt kein Installationsprogramm; Sie müssen lediglich die heruntergeladene Datei dekomprimieren.

   a. Laden Sie Mobile Messaging und M2M Client-Packherunter.
   b. Erstellen Sie einen Ordner, in dem Sie das Software-Development-Kit (SDK) installieren werden.

      Geben Sie dem Ordner zum Beispiel den Namen `MQTT`. Der Pfad zu diesem Ordner wird hier als *`sdkroot`*bezeichnet.
   c. Entpacken Sie den Inhalt der komprimierten Datei Mobile Messaging und M2M Client-Pack in *`sdkroot`*. Die Erweiterung erstellt eine Verzeichnisbaumstruktur, die bei *`sdkroot`*`\SDK`beginnt.
4. Optional: Führen Sie die Schritte in „MQTT-Client für C-Bibliotheken erstellen” auf Seite 32aus.

   Führen Sie diesen Schritt nur aus, wenn das Mobile Messaging und M2M Client-Pack die C-Clientbibliothek für Ihre Zielplattform nicht enthält.
5. Kompilieren Sie die C-Beispielapp `MQTTV3Sample.c`für den MQTT -Client und führen Sie sie aus.

   - Führen Sie in der Befehlszeile die im Abschnitt „C-Beispiel-App des MQTT-Clients über die Befehlszeile kompilieren und ausführen” auf Seite 29 beschriebenen Schritte aus.
   - Führen Sie in einer integrierten Entwicklungsumgebung (IDE) die im Abschnitt „C-Beispiel-App des MQTT-Clients in Microsoft Visual Studio kompilieren und ausführen” auf Seite 30 beschriebenen Schritte aus.

C-Beispiel-App des MQTT-Clients über die Befehlszeile kompilieren und ausführen

Sie können die C-Beispiel-App des MQTT-Clients über die Befehlszeile kompilieren und ausführen. Das Beispiel ist im MQTT-SDK enthalten. Es veranschaulicht einen MQTT-Bereitsteller und -Subskribenten.

Vorbereitende Schritte

Install a C development environment; for example the Microsoft Visual Studio 2010 as used in the example.

Informationen zu diesem Vorgang

Kompilieren Sie das C-Beispiel `MQTTV3Sample`im Unterverzeichnis *sdkroot*`\SDK\clients\c\samples`des SDK-Clients und führen Sie es aus.

Vorgehensweise

Erstellen Sie ein Script im Clientbeispielverzeichnis, um `Sample` auf Ihrer ausgewählten Plattform zu kompilieren und auszuführen.

Mit folgendem Script wird das Beispiel auf einer Windows-32-Bit-Plattform kompiliert und ausgeführt. Die Erstellung erfolgte mit Microsoft Visual Studio 2010. Führen Sie das Script im Unterverzeichnis *sdkroot*`\SDK\clients\c\samples` aus.

```
@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 ..\wind
ows_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
```

Ergebnisse

Der Bereitsteller und der Subskribent von Veröffentlichungen schreiben Ausgaben in ihre Befehlsfenster:

```
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 . . .
```

Abbildung 10. Ausgabe vom Bereitsteller

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

Abbildung 11. Ausgabe vom Subskribenten

C-Beispiel-App des MQTT-Clients in Microsoft Visual Studio kompilieren und ausführen

Sie können die C-Beispiel-App des MQTT-Clients in Microsoft Visual Studio kompilieren und ausführen. Das Beispiel ist im Mobile Messaging und M2M Client-Pack enthalten. Es veranschaulicht einen MQTT-Bereitsteller und -Subskribenten.

Vorbereitende Schritte

Im Beispiel wird Microsoft Visual Studio 2010 verwendet. Sie können andere C-Entwicklungsumgebungen unter Windows und auf anderen Plattformen verwenden, z. B. Eclipse IDE for C/C++ Developers.

Informationen zu diesem Vorgang

Kompilieren Sie das C-Beispiel `MQTTV3Sample` mit Microsoft Visual Studio 2010 und führen Sie es aus. `MQTTV3Sample.c` befindet sich im Unterverzeichnis *`sdkroot`*`\SDK\clients\c\samples`des SDK-Clients.

Vorgehensweise

1. Starten Sie Microsoft Visual Studio.
2. Erstellen Sie ein neues Projekt auf Basis des vorhandenen Codes.
   a) Klicken Sie auf **Datei** > **Neu** > **Projekt aus vorhandenem Code**.
   b) Wählen Sie **Visual C++** als zu erstellenden Projekttyp aus.
   c) Klicken Sie auf **Weiter**.
3. Geben Sie die Parameter im Fenster **Projektspeicherort und Quelldateien** an.
   a) Klicken Sie auf **Durchsuchen** und suchen Sie das Verzeichnis *`sdkroot`*`\SDK\clients\c\samples` .
   b) Geben Sie dem Projekt den Namen `MQTTV3Sample`.
   c) Klicken Sie auf **Weiter**.
4. Wählen Sie **Konsolenanwendungsprojekt** in der Liste **Projekttyp** aus. Klicken Sie auf **Fertigstellen**
5. Konfigurieren Sie nur die Debugkonfiguration.

   Microsoft Visual Studio erstellt standardmäßig sowohl eine Release- als auch eine Debugkonfiguration. Im Lernprogramm konfigurieren Sie die Debugkonfiguration. Inaktivieren Sie die Option **Build** für die Releasekonfiguration, um Buildfehler zu unterdrücken.

   a) Klicken Sie auf **Projekt** > **MQTTV3Sample Eigenschaften** > **Konfigurationsmanager**. Wählen Sie **Release** als **Konfiguration der aktuellen Projektmappe** aus und inaktivieren Sie **Build**.
   b) Wählen Sie **Debug** als **Konfiguration der aktiven Lösung** > **Schließen**aus.

      Sie müssen die Debugkonfiguration in allen folgenden Schritten ändern.
6. Ändern Sie die **C/C++**-Einstellungen auf den **MQTTV3Sample-Eigenschaftenseiten**.
   a) Öffnen Sie im Fenster **MQTTV3Sample Eigenschaftenseiten Konfigurationseigenschaften** > **C/C++** > **Allgemein**.
   b) Klicken Sie in der Liste der allgemeinen Eigenschaften auf **Weitere Include-Verzeichnisse**, fügen Sie Ihren Verzeichnispfad zu *`sdkroot`*`\SDK\clients\c\include` hinzu und klicken Sie auf **Anwenden**.
7. Ändern Sie die **Linker**-Einstellungen.
   a) Öffnen Sie **Konfigurationseigenschaften** > **Linker** > **Allgemein**.
   b) Klicken Sie in der Liste der allgemeinen Eigenschaften auf **Zusätzliche Bibliotheksverzeichnisse**und fügen Sie Ihren Verzeichnispfad zu *`sdkroot`*`\SDK\clients\c\windows_ia32` hinzu
   c) Klicken Sie in der Liste mit den Linker-Eigenschaften auf **Befehlszeile**. Geben Sie `mqttv3c.lib` im Dateneingabebereich **Zusätzliche Optionen** ein und klicken Sie auf **Übernehmen**.

8. Entfernen Sie die Quellendatei `MQTTV3SSample.c` aus dem Projekt.
   a) Öffnen Sie den Ordner **MQTTV3sample** > **Quellendateien** im Fenster **Projektmappen-Explorer** .
   b) Klicken Sie mit der rechten Maustaste auf **MQTTV3SSample.c** > **Aus Projekt ausschließen**
9. Erstellen Sie das Projekt `MQTTV3Sample`.
   a) Klicken Sie mit der rechten Maustaste auf das Projekt **MQTTV3sample** im Projektmappen-Explorer und klicken Sie auf **Erstellen**.

      Der Build wird ohne Fehler erstellt.
10. Fügen Sie zwei neue Projekte hinzu, um **`MQTTV3Sample`** sowohl als Subskribent- als auch als Bereitsteller-Laufzeitinstanz auszuführen.

    Das Bereitsteller- und das Subskribent-Projekt müssen die Befehle zum Ausführen von **`MQTTV3Sample`** enthalten. Sie werden nicht erstellt und enthalten keinen Code.
    a) Klicken Sie im **Lösungsexplorer**mit der rechten Maustaste auf **Lösung ` MQTTV3Sample `** > **Hinzufügen** > **Neues Projekt**.
    b) Geben Sie `Subscriber` in das Feld **Name** ein. Lassen Sie **Win32-Konsolenanwendung** aktiviert. Klicken Sie auf **OK**.

       Der **Win32-Anwendungsassistent** wird gestartet.
    c) Klicken Sie in **Win32-Anwendungsassistent** auf **Weiter**. Aktivieren Sie **Leeres Projekt** > **Fertigstellen**.
    d) Wiederholen Sie diese Schritte, um ein Bereitsteller-Projekt hinzuzufügen.
    e) Klicken Sie mit der rechten Maustaste auf das Subskribent-Projekt und klicken Sie auf **Als Startprojekt festlegen**.

    Das Fenster **Projektmappen-Explorer** wird in Abbildung 12 auf Seite 31 gezeigt.

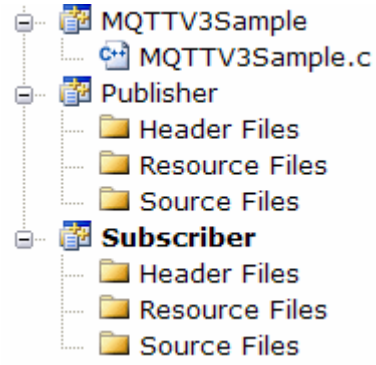


Abbildung 12. Projektmappe MQTTV3Sample

11. Konfigurieren Sie die Subskribent-Eigenschaftenseiten.
    a) Klicken Sie im Solution-Explorer mit der rechten Maustaste auf **Subskribent Eigenschaften** > **Konfigurationseigenschaften** > **Eigenschaften** > **Debugging**

       Überprüfen Sie, ob der Fenstertitel **Subskribent-Eigenschaftenseiten** lautet.
    b) Klicken Sie auf **Umgebung**. Geben Sie `path=%path%;`*`sdkroot`*`\SDK\clients\c\windows_ia32` ein und klicken Sie auf **Anwenden**.

       Ändern Sie *`sdkroot`* entsprechend Ihrer Umgebung.
    c) Klicken Sie auf **Befehle** und ersetzen Sie `$(TargetPath)` durch den Pfad des Moduls `MQTTV3Sample`.

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

       Ändern Sie *`sdkroot`* entsprechend Ihrer Umgebung.
    d) Klicken Sie auf **Befehlsargumente** , geben Sie `-a subscribe -b localhost -p 1883` ein und klicken Sie auf **Anwenden**.

12. Konfigurieren Sie die Bereitsteller-Eigenschaftenseiten.

    Tipp: Sie können zwischen den Projekteigenschaftenseiten wechseln, indem Sie im Fenster **Projektmappen-Explorer** auf die Projekte klicken.

    a) Wiederholen Sie die für den Subskribenten ausgeführten Schritte für den Bereitsteller.

    Das Befehlsargument ist `-b localhost -p 1883` .
13. Stoppen Sie den Erstellungsprozess, mit dem die Projekte 'Bereitsteller' und 'Subskribent' erstellt werden.
    a) Klicken Sie auf den Eigenschaftenseiten eines der Projekte auf **Konfigurationsmanager** und inaktivieren Sie **Build** für Bereitsteller und Subskribent sowohl in der Release- als auch in der Debugkonfiguration. Klicken Sie auf **Schließen** .
14. Führen Sie das Beispiel aus.
    a) Klicken Sie auf **F5**, um den Subskribenten zu starten.
    b) Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf **Bereitsteller** und klicken Sie dann auf **Debug** > **Neue Instanz starten**.

Ergebnisse

Der Bereitsteller und der Subskribent schreiben Ausgaben in Befehlsfenster. Visual Studio schließt das Bereitsteller-Fenster. Schauen Sie sich das Bereitsteller-Fenster an, das in der folgenden Abbildung gezeigt wird, und schließen Sie danach das Bereitsteller-Fenster.

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

Abbildung 13. Ausgabe vom Subskribenten

Nächste Schritte

Erstellen Sie den asynchronen Bereitsteller und Subskribenten und führen Sie sie aus.Die Beispiele sind `MQTTV3ASample.c` und `MQTTV3ASSample.c` in *`sdkroot`*`\SDK\clients\c\samples`.

MQTT-Client für C-Bibliotheken erstellen

Führen Sie die hier genannten Schritte aus, um den MQTT-Client für C-Bibliotheken zu erstellen. Der Abschnitt enthält auch die Kompilierungs- und Verknüpfungsschalter für eine Reihe von Plattformen sowie Beispiele für die Erstellung von Bibliotheken unter iOS und Windows.

Vorbereitende Schritte

1. Erstellen Sie die C-Clientbibliothek nur, wenn es nötig ist. Verknüpfen Sie die vorgefertigten Clientbibliotheken im SDK (Software-Development-Kit) im Unterverzeichnis `SDK\clients\c`, wenn eine davon mit Ihrer Zielplattform übereinstimmt.
2. Konfigurieren Sie einen MQTT-Server, um die Bibliothek zu testen, die Sie mit der MQTT-Client - Beispielapp für C erstellen. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern” auf Seite 142. Prüfen Sie die Serverkonfiguration, indem Sie eine der Beispiel-Apps des MQTT-Clients ausführen.
3. Wenn Sie eine sichere Version der C-Bibliothek erstellen, die SSL (Secure Sockets Layer) unterstützt, müssen Sie auch die OpenSSL-Bibliothek erstellen. (siehe „OpenSSL-Paket erstellen” auf Seite 47).

   Wichtig: Download und Umverteilung des OpenSSL -Pakets unterliegen strengen Import-und Exportbestimmungen sowie Open-Source-Lizenzbedingungen. Lesen Sie sorgfältig die Einschränkungen und Warnungen, bevor Sie entscheiden, ob Sie das Paket herunterladen.

Informationen zu diesem Vorgang

Folgen Sie den Anweisungen im Abschnitt „OpenSSL-Paket erstellen” auf Seite 47, um die OpenSSL-Bibliothek herunterzuladen und zu erstellen. Sie müssen die OpenSSL-Instanz erstellen, wenn Sie eine sichere Version des MQTT-Clients für die C-Bibliothek erstellen möchten. OpenSSL ist nicht erforderlich, um eine nicht gesicherte Version der MQTT-Bibliothek zu erstellen. Die Schritte schließen Beispiele für die Erstellung der Bibliothek unter iOS und Windows ein.

Erstellen Sie die MQTT-Clientbibliothek für C, indem Sie C-Entwicklungsbibliothekstools und das MQTT-Software-Development-Kit (SDK) auf Ihre Buildplattform herunterladen. Schreiben Sie eine Makefile, um die Bibliothek für Ihre Zielplattform zu erstellen, und verwenden Sie dabei die Optionen, die im Abschnitt „MQTT -Buildoptionen für verschiedene Plattformen” auf Seite 34 dokumentiert sind. Sie finden die plattformspezifischen Schritte für die Erstellung und Ausführung einer Makefile hier:

- iOS „MQTT-Clientbibliotheken für C auf einem Apple Mac zur Verwendung für iOS-Geräte erstellen” auf Seite 35
- Windows „MQTT-Bibliotheken unter Windows erstellen” auf Seite 41

Vorgehensweise

1. Installieren Sie auf Ihrer Entwicklungsplattform eine C-Entwicklungsumgebung.

   Für die Makefiles in den Beispielen dieses Abschnitts werden die folgenden Tools verwendet:

   - iOS iOS: Apple Mac mit OS X 10.8.2 mit den iOS-Entwicklungstools von Xcode.
   - Linux Linux: GCC Version 4.4.6 aus Red Hat Enterprise Linux Version 6.2.

     Die unterstützte Mindestversion der C-Bibliothek `glibc` ist 2.12 und die des Linux-Kernels ist 2.6.32.
   - Windows Microsoft Windows: Visual Studio Version 10.0
2. Laden Sie Mobile Messaging und M2M Client-Pack herunter und installieren Sie das MQTT SDK.

   Es gibt kein Installationsprogramm; Sie müssen lediglich die heruntergeladene Datei dekomprimieren.

   a. Laden Sie Mobile Messaging und M2M Client-Packherunter.

   b. Erstellen Sie einen Ordner, in dem Sie das Software-Development-Kit (SDK) installieren werden.

      Geben Sie dem Ordner zum Beispiel den Namen `MQTT`. Der Pfad zu diesem Ordner wird hier als *`sdkroot`*bezeichnet.

   c. Entpacken Sie den Inhalt der komprimierten Datei Mobile Messaging und M2M Client-Pack in *`sdkroot`*. Die Erweiterung erstellt eine Verzeichnisbaumstruktur, die bei *`sdkroot`*`\SDK`beginnt.
3. Erweitern Sie den Quellcode für die MQTT-Clientbibliothek für C.

   Die komprimierte Quellcodedatei ist *`sdkroot`*`\SDK\clients\c\source.zip`.
4. Optional: Erstellen Sie OpenSSL.

   (siehe „OpenSSL-Paket erstellen” auf Seite 47).
5. Erstellen Sie die MQTT-Clientbibliothek für C.

   Die Befehle und Optionen für das Erstellen der Bibliotheken werden unter „MQTT -Buildoptionen für verschiedene Plattformen” auf Seite 34 aufgelistet.

   Führen Sie die Schritte in den folgenden Beispielen aus, um eine Makefile zum Erstellen des MQTT-Clients für die C-Bibliotheken für Ihre Zielplattform zu schreiben.

   - „MQTT-Clientbibliotheken für C auf einem Apple Mac zur Verwendung für iOS-Geräte erstellen” auf Seite 35
   - „MQTT-Bibliotheken unter Windows erstellen” auf Seite 41

MQTT -Buildoptionen für verschiedene Plattformen

In der folgenden Tabelle werden die Compiler- und Buildoptionen für die Erstellung des MQTT-Clients für die C-Bibliotheken auf verschiedenen Plattformen aufgelistet.

Tabelle 2. MQTT -Buildoptionen für verschiedene Plattformen

Plattform	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	-m 64
Linux x86-64		-fPIC -Os -Wall -I *MQTTCLIENT_DIR*		
Linux x86-32				-m 32
Linux -ARM (glibc)	arm-linux-gcc			
Linux -ARM (uclibc)	arm-unknown-linux -uclibcgnueabi-gcc			
Windows 32 Bit	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)	

Tabelle 2. MQTT -Buildoptionen für verschiedene Plattformen (Forts.)

Plattform	Compiler	Compiler Options	Linker Options	Extra Options
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 *MQTT-Clientbibliotheken für C auf einem Apple Mac zur Verwendung für iOS-Geräte erstellen*

Führen Sie die hier beschriebenen Schritte aus, um eine Makefile für die Erstellung der MQTT-Clientbibliotheken für C auf einem Apple Mac zu schreiben, damit diese später für iOS-Geräte verwendet werden können.

Vorbereitende Schritte

1. Installieren Sie Erstellungstools, entwickeln Sie die Makefile und führen Sie sie auf einem Apple Mac mit OS X 10.8.2 oder höher aus.
2. Installieren Sie die Befehlszeilentools für Xcode, die das Programm **make** enthalten. Die Befehlszeilentools können Sie von der Website für Xcode herunterladen.

Informationen zu diesem Vorgang

Erstellen Sie eine Makefile zur Erstellung der MQTT-Clientbibliotheken für C für ein iPhone oder iPad mit einem ARMv7- oder ARMv7s-Prozessor und für den iPhone-Simulator, der mit einem i386-64-Bit-Prozessor ausgeführt wird. Informationen finden Sie unter Liste von iOS-Geräten.

Tipp: Unter „MQTTios.mak-Makefile-Liste" auf Seite 39 wird die vollständige Makefile aufgelistet.

1. Kopieren Sie die Liste in eine Datei.
2. Ersetzen Sie das führende Zeichen in jeder Zeile, die auf ein Ziel folgt, durch ein Tabulatorzeichen (siehe Schritt „8" auf Seite 38).
3. Führen Sie die Datei mit dem Befehl aus, der in Schritt „9" auf Seite 39 der Prozedur genannt wird.

Vorgehensweise

1. Laden Sie die iOS -Entwicklungstools herunter und installieren Sie sie.
   a. Melden Sie sich mit einer Benutzer-ID an, die Verwaltungsberechtigungen besitzt.
   b. Überprüfen Sie, ob Ihr Apple Mac Version 10.8.2 oder höher hat.
   c. Rufen Sie die Website Xcode auf, um Xcode aus dem Mac App Store herunterzuladen.
   d. Installieren Sie Xcode, die Befehlszeilenumgebung und den Simulator.

   Wenn im Mac App Store mehrere Versionen des Simulators angeboten werden, wählen Sie die Version aus, die mit der iOS-Version kompatibel ist, für die Ihre App vorgesehen ist.
2. Erstellen Sie die Makefile `MQTTios.mak`

   Fügen Sie einen Prolog hinzu:

   ```
   # Die Buildausgabe wird im aktuellen Verzeichnis erzeugt.
   # MQTTCLIENT_DIR muss auf das Basisverzeichnis verweisen, das den MQTT-Clientquellcode ent
   hält.
   # Der Standardwert MQTTCLIENT_DIR ist das aktuelle Verzeichnis.
   # TOOL_DIR ist /Applications/Xcode.app/Contents/Developer/Platforms
   # Das standardmäßige OPENSSL_DIR ist sdkroot/openssl, relativ zu sdkroot/sdk/clients/c/
   mqttv3c/src
   # OPENSSL_DIR muss auf das Basisverzeichnis verweisen, das den OpenSSL -Build enthält.
   # Beispiel: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
   ```
3. Position des MQTT -Quellcodes festlegen.

   Führen Sie die Makefile in demselben Verzeichnis wie die MQTT -Quellendateien aus oder setzen Sie den Befehlszeilenparameter `MQTTCLIENT_DIR` :

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

   Fügen Sie folgende Zeilen zur Makefile hinzu:

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

   Im Beispiel wird VPATH auf das Verzeichnis gesetzt, in dem **make** nach Quellendateien sucht, die nicht explizit angegeben sind, z. B. nach allen Headerdateien, die im Build erforderlich sind.
4. Optional: Legen Sie die Position der OpenSSL -Bibliotheken fest.

   Dieser Schritt ist erforderlich, um die SSL-Versionen des MQTT -Clients für C-Bibliotheken zu erstellen.

   Setzen Sie den Standardpfad zu den OpenSSL -Bibliotheken auf dasselbe Verzeichnis, in dem Sie das MQTT SDK erweitert haben. Legen Sie andernfalls `OPENSSL_DIR` als Befehlszeilenparameter fest.

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

Tipp: *OpenSSL* ist das Verzeichnis OpenSSL , das alle Unterverzeichnisse OpenSSL enthält. Sie müssen die Verzeichnisstruktur möglicherweise vom Ort ihrer Dekomprimierung verschieben, weil sie leere, übergeordnete Verzeichnisse enthält, die nicht benötigt werden.

5. Legen Sie die Verzeichnisse der Entwicklungstools fest.

   Wenn Sie Xcode in einem anderen Verzeichnis installiert haben, legen Sie `TOOL_DIR` über die Befehlszeile fest.

```
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. Wählen Sie alle Quellendateien aus, die zum Erstellen der einzelnen MQTT -Bibliotheken erforderlich sind. Legen Sie außerdem den Namen und die Position der zu erstellenden MQTT-Bibliothek fest.

   Fügen Sie die folgende Zeile zur Makefile hinzu, um alle MQTT -Quellendateien aufzulisten:

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

   Welche Quellendateien erforderlich sind, hängt davon ab, ob Sie eine synchrone oder asynchrone Bibliothek erstellen und ob die Bibliothek SSL einschließt oder nicht.

   Fügen Sie abhängig von den zu erstellenden Zielen eine oder mehrere der folgenden Zeilen hinzu. Die gemeinsam genutzten Bibliotheken werden im Verzeichnis `darwin_x86_64` erstellt.

   - Synchron, nicht gesichert:

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

   - Synchron, gesichert:

```
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S} .a
```

   - Asynchron, nicht gesichert:

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

   - Asynchron, gesichert:

```
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS} .a
```

7. Definieren Sie den Compiler und Compileroptionen.

   Sehen Sie sich die Optionen für verschiedene Plattformen an, die in MQTT-Buildoptionen für verschiedene Plattformengezeigt werden.

   a) Legen Sie Gnu-Projekt C und C++ (**gcc**) als Compiler fest.

      Wählen Sie drei Cross-Compiler zum Erstellen der Bibliothek für unterschiedliche Geräte und den iPhone-Simulator aus:

```
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) Fügen Sie die Compileroptionen hinzu.

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

c) Fügen Sie die Include-Pfade hinzu.

```
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. Definieren Sie die Erstellungsziele.

Tipp: Jede nachfolgende Zeile, die die Implementierung eines Ziels definiert, muss mit einem Tabulatorzeichen beginnen.

a) Definieren Sie das Ziel **all**.

Das Ziel "all" erstellt alle Bibliotheken.

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

Wird es als Erstes angegeben, ist es das Standardziel.

a) Erstellen Sie die synchrone, nicht gesicherte Bibliothek `libmqttv3c.a`.

```
${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 -create $@.armv7 $@.armv7s $@.i386 -output $@
```

Die Anweisung `rm *.o` löscht alle Objektdateien, die für jede Bibliothek erstellt werden. `lipo` verknüpft alle drei Bibliotheken in einer Datei.

b) Erstellen Sie die asynchrone, nicht gesicherte Bibliothek `libmqttv3a.a`.

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

Die Anweisung `rm *.o` löscht alle Objektdateien, die für jede Bibliothek erstellt werden. `lipo` verknüpft alle drei Bibliotheken in einer Datei.

c) Erstellen Sie die synchrone, gesicherte Bibliothek `libmqttv3cs.a`.

```
${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 -create $@.armv7 $@.armv7s $@.i386 -output $@
```

Die Anweisung `rm *.o` löscht alle Objektdateien, die für jede Bibliothek erstellt werden. `lipo` verknüpft alle drei Bibliotheken in einer Datei.

d) Erstellen Sie die asynchrone, gesicherte Bibliothek `libmqttv3as.a`.

```
${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL -DUSE_NA
MED_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_NA
MED_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_NA
MED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o
$@.i386 *.o
    rm *.o
    Lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
```

Die Anweisung `rm *.o` löscht alle Objektdateien, die für jede Bibliothek erstellt werden. `lipo` verknüpft alle drei Bibliotheken in einer Datei.

e) Definieren Sie das Ziel **`clean`**.

Das Ziel "clean" entfernt alle Dateien und Verzeichnisse, die von der Makefile generiert werden.

```
.PHONIE: sauber
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

9. Führen Sie die Makefile aus.

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

Ergebnisse

Die folgenden Dateien werden im Verzeichnis *sdkroot/sdk/clients/c/mqttv3c/src*/darwin_x86_64 erstellt.

```
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-Makefile-Liste

```
# Die Buildausgabe wird im aktuellen Verzeichnis erzeugt.
# MQTTCLIENT_DIR muss auf das Basisverzeichnis verweisen, das den MQTT-Clientquellcode enthält.
```

```
# Der Standardwert MQTTCLIENT_DIR ist das aktuelle Verzeichnis.
# TOOL_DIR ist /Applications/Xcode.app/Contents/Developer/Platforms
# Das standardmäßige OPENSSL_DIR ist sdkroot/openssl, relativ zu sdkroot/sdk/clients/c/
mqttv3c/src
# OPENSSL_DIR muss auf das Basisverzeichnis verweisen, das den OpenSSL -Build enthält.
# Beispiel: make -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 = ${filter-out ${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 = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S} .a
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 = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS} .a

# 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 -create $@.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 -DNO
SIGPIPE
    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 -DNO
SIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    Lipo -create $@.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 -DNOSIG
PIPE
    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 -DNO
SIGPIPE
    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 -DNOSIG
PIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
    rm *.o
    Lipo -create $@.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_SEMA
PHORES -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_SE
MAPHORES -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_SEMA
PHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
    rm *.o
    Lipo -create $@.armv7 $@.armv7s $@.i386 -output $@

.PHONIE: sauber
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

Windows *MQTT-Bibliotheken unter Windows erstellen*

Führen Sie die hier genannten Schritte aus, um eine Makefile für die Erstellung der MQTT-Clientbibliotheken für C unter Windows zu schreiben.

Vorbereitende Schritte

1. Installieren Sie bei Bedarf eine Version von **Make** auf Ihrer Build-Workstation, die mit Makefiles kompatibel ist, die für Gnu make geschrieben wurden. Laden Sie andernfalls Gnu make herunter und erstellen Sie es. Weitere Informationen finden Sie im Abschnitt Gnu Make. Die Website Make for Windowsbietet eine installierbare Version von **Make** for Windows.
2. Sie benötigen außerdem Linux-Befehle für Windows, damit Sie das Ziel 'clean' im Makefile-Beispiel verwenden können. Linux-Befehle für Windows können Sie von Websites wie beispielsweise Cygwin abrufen.

Informationen zu diesem Vorgang

Erstellen Sie eine Makefile, die MQTT-Clientbibliotheken für C für die 32-Bit-Version von Windows erstellt.

Tipp: Unter „MQTTwin.mak-Makefile-Liste” auf Seite 45 wird die vollständige Makefile aufgelistet.

1. Kopieren Sie die Liste in eine Datei.
2. Ersetzen Sie das führende Zeichen in jeder Zeile, die auf ein Ziel folgt, durch ein Tabulatorzeichen (siehe Schritt „7” auf Seite 44).
3. Führen Sie die Datei mit dem Befehl aus, der in Schritt „9” auf Seite 45 der Prozedur genannt wird.

Vorgehensweise

1. Erstellen Sie die Makefile `MQTTwin.mak`

   Fügen Sie einen Prolog hinzu:

```
# Die Buildausgabe wird im aktuellen Verzeichnis erzeugt.
# MQTTCLIENT_DIR muss auf das Basisverzeichnis verweisen, das den MQTT-Clientquellcode ent
hält.
# Der Standardwert MQTTCLIENT_DIR ist das aktuelle Verzeichnis.
# OPENSSL_DIR ist sdkroot\openSSL, relativ zu sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR muss auf das Basisverzeichnis verweisen, das den OpenSSL -Build enthält.
# Beispiel: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Legen Sie die Erstellungsumgebung fest. Beispiel:
# %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. Position des MQTT -Quellcodes festlegen.

   Führen Sie die Makefile in demselben Verzeichnis wie die MQTT -Quellendateien aus oder setzen Sie den Befehlszeilenparameter `MQTTCLIENT_DIR` :

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

   Fügen Sie folgende Zeilen zur Makefile hinzu:

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

   Im Beispiel wird VPATH auf das Verzeichnis gesetzt, in dem **make** nach Quellendateien sucht, die nicht explizit angegeben sind, z. B. nach allen Headerdateien, die im Build erforderlich sind.

3. Optional: Legen Sie die Position der OpenSSL -Bibliotheken fest.

   Dieser Schritt ist erforderlich, um die SSL-Versionen des MQTT -Clients für C-Bibliotheken zu erstellen.

   Setzen Sie den Standardpfad zu den OpenSSL -Bibliotheken auf dasselbe Verzeichnis, in dem Sie das MQTT SDK erweitert haben. Legen Sie andernfalls `OPENSSL_DIR` als Befehlszeilenparameter fest.

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

   Tipp: *OpenSSL* ist das Verzeichnis OpenSSL , das alle Unterverzeichnisse OpenSSL enthält. Sie müssen die Verzeichnisstruktur möglicherweise vom Ort ihrer Dekomprimierung verschieben, weil sie leere, übergeordnete Verzeichnisse enthält, die nicht benötigt werden.

4. Wählen Sie alle Quellendateien aus, die zum Erstellen der einzelnen MQTT -Bibliotheken erforderlich sind. Legen Sie außerdem den Namen und die Position der zu erstellenden MQTT-Bibliothek fest.

   Fügen Sie die folgende Zeile zur Makefile hinzu, um alle MQTT -Quellendateien aufzulisten:

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

   Welche Quellendateien erforderlich sind, hängt davon ab, ob Sie eine synchrone oder asynchrone Bibliothek erstellen und ob die Bibliothek SSL einschließt oder nicht.

   Fügen Sie abhängig von den zu erstellenden Zielen eine oder mehrere der folgenden Zeilen hinzu. Die gemeinsam genutzten Bibliotheken und Manifeste werden im Verzeichnis `windows_ia32` erstellt.

   - Synchron, nicht gesichert:

     ```
     MQTTLIB = mqttv3c
     MQTTDLL = windows_ia32/${MQTTLIB} .dll
     SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c,
     ${ALL_SOURCE_FILES}}
     MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource: ${MQTTDLL}\; 2
     ```

   - Synchron, gesichert:

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

- Asynchron, nicht gesichert:

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

- Asynchron, gesichert:

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

5. Definieren Sie den Compiler und Compileroptionen.

   Sehen Sie sich die Optionen für verschiedene Plattformen an, die in MQTT-Buildoptionen für verschiedene Plattformengezeigt werden.

   a) Legen Sie Microsoft Visual C++ als Compiler fest.

   ```
   CC = cl
   ```

   b) Fügen Sie die Vorprozessoroptionen hinzu.

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

   c) Fügen Sie die Compileroptionen hinzu.

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

   d) Fügen Sie die Include-Pfade hinzu.

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

   e) Optional: Fügen Sie eine Vorprozessoroption hinzu, wenn Sie eine sichere Bibliothek erstellen.

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

   f) Optional: Fügen Sie die OpenSSL-Headerdateien hinzu, wenn Sie eine sichere Bibliothek erstellen.

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

   Tipp: Die Headerdateien befinden sich im Verzeichnis `${OPENSSL_DIR}/inc32/openssl`, aber die Datei `ssl.h` ist in "`openssl/ssl.h`" eingeschlossen.

6. Legen Sie den Linker und Linkeroptionen fest.

   a) Legen Sie Microsoft Visual C++ als Linker fest.

   ```
   LD = link
   ```

   b) Fügen Sie die Linkeroptionen hinzu.

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

   c) Fügen Sie die Bibliothekspfade hinzu.

   ```
   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) Fügen Sie die temporären Ausgabedateien hinzu.

   ```
   IMP = /implib: ${@:.dll=.lib}
   LIBPDB = /pdb: ${@:.dll=.pdb}
   LIBMAP = /map: ${@:.dll=.map}
   ```

   e) Optional: Fügen Sie die OpenSSL-Bibliotheken hinzu, wenn Sie eine sichere Bibliothek erstellen.

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

f) Optional: Fügen Sie den OpenSSL-Bibliothekspfad hinzu, wenn Sie eine sichere Bibliothek erstellen.

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

7. Definieren Sie die vier Buildziele.

a) Definieren Sie das Ziel **all**.

Tipp: Jede nachfolgende Zeile, die die Implementierung eines Ziels definiert, muss mit einem Tabulatorzeichen beginnen.

Das Ziel "all" erstellt alle Bibliotheken.

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

b) Erstellen Sie die synchrone, nicht gesicherte Bibliothek `mqttv3c.dll`.

```
${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}
```

Die Anweisung `-rm ${CURDIR}/MQTTAsync.obj` löscht jede `MQTTAsync.obj`-Datei, die für ein früheres Ziel erstellt wurde. `MQTTAsync.obj` und `MQTTClient.obj` schließen sich gegenseitig aus.

c) Erstellen Sie die asynchrone, nicht gesicherte Bibliothek `mqttv3a.dll`.

```
${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}
```

Die Anweisung `-rm ${CURDIR}/MQTTClient.obj` löscht jede `MQTTClient.obj`-Datei, die für ein früheres Ziel erstellt wurde. `MQTTAsync.obj` und `MQTTClient.obj` schließen sich gegenseitig aus.

d) Erstellen Sie die synchrone, gesicherte Bibliothek `mqttv3cs.dll`.

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

Die Anweisung `-rm ${CURDIR}/MQTTAsync.obj` löscht jede `MQTTAsync.obj`-Datei, die für ein früheres Ziel erstellt wurde. `MQTTAsync.obj` und `MQTTClient.obj` schließen sich gegenseitig aus.

e) Erstellen Sie die asynchrone, gesicherte Bibliothek `mqttv3as.dll`.

```
${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}
```

Die Anweisung `-rm $(CURDIR}/MQTTClient.obj` löscht jede `MQTTClient.obj`-Datei, die für ein früheres Ziel erstellt wurde. `MQTTAsync.obj` und `MQTTClient.obj` schließen sich gegenseitig aus.

f) Definieren Sie das Ziel **clean**.

Das Ziel "clean" entfernt alle Dateien und Verzeichnisse, die von der Makefile generiert werden.

```
.PHONIE: sauber
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

8. Legen Sie den Windows-Pfad zur Ausführung der Makefile fest.

Passen Sie die Teile in Kursivschrift an Ihre Installation an.

a) Legen Sie die Microsoft Visual Studio-Umgebung fest.

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

b) Legen Sie die Variable Path fest, um das Make-Programm und die Linux -Befehlsumgebung einzuschließen.

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

9. Führen Sie die Makefile aus.

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

Tipp: Das Dateitrennzeichen muss ein Schrägstrich sein; umgekehrte Schrägstriche sind nicht zulässig.

Ergebnisse

Die folgenden Dateien werden im Verzeichnis *sdkroot**SDK**clients**c**mqttv3c**src*\windows_ia32 erstellt.

```
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-Makefile-Liste

```
# Die Buildausgabe wird im aktuellen Verzeichnis erzeugt.
# MQTTCLIENT_DIR muss auf das Basisverzeichnis verweisen, das den MQTT-Clientquellcode enthält.
# Der Standardwert MQTTCLIENT_DIR ist das aktuelle Verzeichnis.
# OPENSSL_DIR ist sdkroot\openSSL, relativ zu sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR muss auf das Basisverzeichnis verweisen, das den OpenSSL -Build enthält.
# Beispiel: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Legen Sie die Erstellungsumgebung fest. Beispiel:
# %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 = ${filter-out ${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 = ${filter-out ${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 = ${filter-out ${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 = /implib: ${@:.dll=.lib}
LIBPDB = /pdb: ${@:.dll=.pdb}
LIBMAP = /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} ${LIBMAP} ${LINKFLAGS} ${IMP} ${LIBPDB} ${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}

.PHONIE: sauber
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

OpenSSL-Paket erstellen

Erstellen Sie das Paket OpenSSL , bevor Sie die sicheren MQTT -Clientbibliotheken für C, `mqttv3cs` und `mqttv3a`erstellen. Dabei werden die Bibliotheken, die zum Erstellen einer sicheren Version der MQTT-Clientbibliothek für C erforderlich sind, und das OpenSSL-Zertifikatverwaltungstool erstellt.

Vorbereitende Schritte

1. iOS Die iOS-Makefile-Anpassung betrifft Zielgeräte mit iOS6. Die Anpassung kann für frühere oder spätere Versionen von iOS unterschiedlich sein.
2. Windows Die Windows-Makefile-Anpassung betrifft 32-Bit-Windows.

Informationen zu diesem Vorgang

Laden Sie das OpenSSL-Paket und alle Softwarevoraussetzungen herunter und installieren Sie sie. Passen Sie die OpenSSL-Makefiles an und erstellen Sie OpenSSL-Bibliotheken für Ihre Zielplattform. Unter Windows und Linux erstellt Make auch das OpenSSL-Tool zur Schlüsselerstellung und -verwaltung.

Vorgehensweise

1. Installieren Sie das OpenSSL-Paket.
   a) Laden Sie das OpenSSL-Paket von OpenSSL herunter.

      Wichtig: Download und Umverteilung des OpenSSL -Pakets unterliegen strengen Import-und Exportbestimmungen sowie Open-Source-Lizenzbedingungen. Lesen Sie sorgfältig die Einschränkungen und Warnungen, bevor Sie entscheiden, ob Sie das Paket herunterladen.

   b) Entpacken Sie den Inhalt der komprimierten Datei in *`sdkroot`*.

      Suchen Sie auf der Registerkarte **News** der OpenSSL-Website nach der Speicherposition für den Download des neuesten Pakets. Das Paket ist eine komprimierte TAR-Datei mit der Erweiterung `tar.gz`. When expanded, the package creates a top-level folder `openssl`*`version`*; for example *`openssl-1.0.1c`*. Die Beispiele beziehen sich auf den Pfad zum Ordner als %openssl% unter Windows und $openssl unter iOS. Unter Windowsist %openssl% beispielsweise *`sdkroot`*\openssl-1.0.1c.

      Tipp: Überprüfen Sie den Verzeichnispfad, der beim Entpacken des OpenSSL-Pakets erstellt wird. Einige Pakete haben doppelte Ebenen des Ordners `openssl`*`version`* .

2. Windows
   Optional: Laden Sie unter Windows Perl herunter und installieren Sie es. (siehe perl.org).

   Im Beispiel wurde Perl von ActivePerl Downloads heruntergeladen.

3. iOS
   Optional: Erstellen Sie unter iOS drei weitere Verzeichnisse.

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

   Unter iOS müssen Sie das OpenSSL-Paket für drei verschiedene Hardwareplattformen erstellen.

4. Generieren Sie die OpenSSL-Makefile zum Erstellen des OpenSSL-Pakets für Ihre Hardware und Ihr Betriebssystem.
   a) Öffnen Sie ein Befehlsfenster im Verzeichnis %openssl% oder $openssl.
   b) Führen Sie den Perl-Befehl **`Configure`** mit den geeigneten Parametern aus.

   - 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

Passen Sie unter iOS die generierte OpenSSL-Makefile für unterschiedliche Apple-Geräte an.

a) Erstellen Sie drei Kopien der generierten Makefile $openssl/Makefile.

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

b) Ändern Sie in jeder Makefile die Anweisung "CC=gcc".

Die Anweisung CC=gcc steht in Zeile 62 oder in deren Nähe. Ersetzen Sie die Anweisung durch folgende Befehle:

$openssl/Makefile_armv7

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

$openssl/Makefile_armv7s

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

$openssl/Makefile_i386

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

c) Ändern Sie in jeder Makefile die Anweisung "CFLAG=...".

Die Anweisung steht in Zeile 63 oder in deren Nähe (zur besseren Lesbarkeit auf drei Zeilen verteilt):

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

Die angezeigte Position des SDK ist von Ihrer Auswahl bei der Xcode-Installation abhängig. Die Version des SDK hängt von der Betriebssystemversion ab, für die Sie die Makefile erstellen.

iPhone-Simulator

Die iPhone-Simulator-Makefile heißt $openssl/Makefile_i386.

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimula
tor.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

Die iOS-Makefiles heißen $openssl/Makefile_arm7 und $openssl/Makefile_arm7s.

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.plat
form/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. Führen Sie die generierte Makefile aus.

- Windows

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

- iOS Auf 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
```

Ergebnisse

Der Build generiert die gemeinsam genutzten Bibliotheken, Bibliotheks- und Headerdateien, die zum Erstellen sicherer Versionen der MQTT-Clientbibliothek für C erforderlich sind.

Erste Schritte mit dem MQTT-Client für C unter iOS

Lernen Sie, wie Sie iOS-Anwendungen dazu bringen, Nachrichten mit einem MQTT-Server auszutauschen. Zur Verwendung auf iOS-Geräten (d. h. iPhone und iPad) müssen Sie die MQTT-Clientbibliothek für C aus dem Quellcode erstellen, der als Teil des MQTT-Software-Development-Kits bereitgestellt wird.

Vorbereitende Schritte

1. Link zum iOS Dev Center und wissen, wie Anwendungen für iOSentwickelt werden.
2. Beschaffen Sie einen Apple Mac mit OS X 10.8.2 oder höher, um die integrierte Entwicklungsumgebung Xcode auszuführen.
3. Optional: Konfigurieren Sie eine C-Entwicklungsumgebung unter Windows oder Linux. Es ist hilfreich, die C-Beispiel-Apps des MQTT-Clients unter Windows oder Linux zu erstellen und auszuführen, bevor Sie eine MQTT-iOS-App entwickeln. Alternativ können Sie auch den Beispielquellcode studieren, ohne die Beispiele zu erstellen.
4. Informationen zu unterstützten Plattformen und Referenzplattformen für den MQTT-Client für C finden Sie im Abschnitt Systemvoraussetzungen für IBM Mobile Messaging und M2M Client-Pack.
5. Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.

Informationen zu diesem Vorgang

Die Vorgehensweise besteht aus folgenden Schritten:

1. Machen Sie sich mit der Programmierung für MQTT vertraut, indem Sie die Beispiel-Apps des MQTT-Clients und MQTT-Clientbibliotheken für C studieren, erstellen und ausführen.
2. Installieren Sie die Xcode-Entwicklungsumgebung für iOS on Apple Mac.
3. Führen Sie die Aufgabe „MQTT-Client für C-Bibliotheken erstellen” auf Seite 32 aus, um die Bibliotheken des MQTT-Clients für C für iOS-Geräte zu erstellen.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

Der Server muss das MQTT version 3.1 -Protokoll unterstützen. Alle MQTT-Server von IBM führen dies aus, einschließlich IBM WebSphere MQ und IBM MessageSight. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern" auf Seite 142.

2. Laden Sie Mobile Messaging und M2M Client-Pack herunter und installieren Sie das MQTT SDK.

   Es gibt kein Installationsprogramm; Sie müssen lediglich die heruntergeladene Datei dekomprimieren.

   a. Laden Sie Mobile Messaging und M2M Client-Packherunter.

   b. Erstellen Sie einen Ordner, in dem Sie das Software-Development-Kit (SDK) installieren werden.

      Geben Sie dem Ordner zum Beispiel den Namen `MQTT`. Der Pfad zu diesem Ordner wird hier als *`sdkroot`*bezeichnet.

   c. Entpacken Sie den Inhalt der komprimierten Datei Mobile Messaging und M2M Client-Pack in *`sdkroot`*. Die Erweiterung erstellt eine Verzeichnisbaumstruktur, die bei *`sdkroot`*\SDKbeginnt.

3. Optional: Machen Sie sich mit der MQTT-API vertraut, indem Sie sich mit der MQTT-Client - Beispiellapp für C eingehender beschäftigen.

   a) Erstellen Sie die synchrone MQTT Client-Beispiel-C-App `MQTTV3sample.c` für Windows oder Linux. Weitere Informationen finden Sie unter „Erste Schritte mit dem MQTT-Client für C" auf Seite 27.

   b) Stellen Sie eine Verbindung zu einem MQTT version 3-Server her und führen Sie Veröffentlichungen und Subskriptionen für Themen auf dem Server durch.

   c) Untersuchen Sie den Quellcode und die MQTT -API-Dokumentation. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

      Lernen Sie, wie MQTT-Clients erstellt und wiederaufgenommen und wie Veröffentlichungen und Subskriptionen für MQTT-Themen durchgeführt werden, indem Sie das synchrone Beispiel studieren. Das synchrone Beispiel ist einfacher als das asynchrone Beispiel. Wenn Sie bisher noch nicht für MQTT programmiert haben, schreiben Sie ein synchrones MQTT-Programm, um sich mit dem MQTT-Programmiermodell und der Anwendungsprogrammierschnittstelle (API) vertraut zu machen.

   d) Erstellen Sie die asynchrone MQTT-Clientbibliothek für C unter Windows oder Linux. Weitere Informationen finden Sie unter „MQTT-Client für C-Bibliotheken erstellen" auf Seite 32.

   e) Erstellen Sie die asynchrone C-Beispiel-App des MQTT-Clients für Veröffentlichungen und Subskriptionen und führen Sie sie aus.

   f) Studieren Sie den Quellcode der asynchronen C-Beispiel-App des MQTT-Clients und die MQTT-Referenzliteratur.

      Sie müssen die asynchrone Schnittstelle verwenden, um eine MQTT-Anwendung für ein mobiles Gerät zu schreiben. Gut geschriebene Apps, die die asynchrone Schnittstelle aufrufen, sind reaktionsfähiger und verlängern die Lebensdauer des Akkus mehr als Apps, die für die synchrone Schnittstelle geschrieben werden.

      Die asynchrone Schnittstelle kennt zwei Stufen der Asynchronität:

      i) In der ersten Stufe wird die Blockierung der Anwendung aufgehoben, während die MQTT-Clientbibliothek auf Veröffentlichungen vom Server wartet.

      ii) In der zweiten Stufe wird die Blockierung der Anwendung aufgehoben, während die Clientbibliothek eine Verbindung zum Server herstellt, Subskriptionen erstellt und Veröffentlichungen sendet.

4. Laden Sie die iOS -Entwicklungstools herunter und installieren Sie sie.

   a. Melden Sie sich mit einer Benutzer-ID an, die Verwaltungsberechtigungen besitzt.

   b. Überprüfen Sie, ob Ihr Apple Mac Version 10.8.2 oder höher hat.

   c. Rufen Sie die Website Xcode auf, um Xcode aus dem Mac App Store herunterzuladen.

   d. Installieren Sie Xcode, die Befehlszeilenumgebung und den Simulator.

Wenn im Mac App Store mehrere Versionen des Simulators angeboten werden, wählen Sie die Version aus, die mit der iOS-Version kompatibel ist, für die Ihre App vorgesehen ist.

5. Erstellen Sie die MQTT-Clientbibliotheken für C unter iOS. Weitere Informationen finden Sie unter „MQTT-Client für C-Bibliotheken erstellen” auf Seite 32.

Nächste Schritte

1. Prüfen Sie die erstellten MQTT-Clientbibliotheken für C:
   a. Kompilieren Sie mithilfe der Xcode-Entwicklungsumgebung die asynchrone MQTT-Client - Beispiellapp für C und stellen Sie eine Verbindung mit der nicht gesicherten, asynchronen MQTT-Clientbibliothek für C her.
   b. Führen Sie mithilfe der Xcode-Entwicklungsumgebung die asynchrone C-Beispiel-App des MQTT-Clients auf einem iOS-Gerät aus. Verbinden Sie das Beispiel mit dem MQTT version 3-Server, den Sie konfiguriert haben (siehe „MQTT-Service über die Befehlszeile konfigurieren” auf Seite 146).
2. Erstellen Sie eine C-App des MQTT-Clients für iOS. Die Codierungsbeispiele für die asynchrone C-Anwendung können sich als hilfreich erweisen. Die Beispiele sind `MQTTV3ASample.c` und `MQTTV3ASSample.c` in *`sdkroot`*`\SDK\clients\c\samples`. Beginnen Sie zur Übung damit, dass Sie das MQTT-Publish/Subscribe-Beispiel implementieren.

   Tipp: Um eine Vorstellung davon zu bekommen, wie die App aussieht und was sie tut, können Sie sich Screenshots der MQTT-Client - Beispielapp für C anschauen. Weitere Informationen finden Sie unter „Erste Schritte mit dem MQTT-Client für Java unter Android” auf Seite 19.

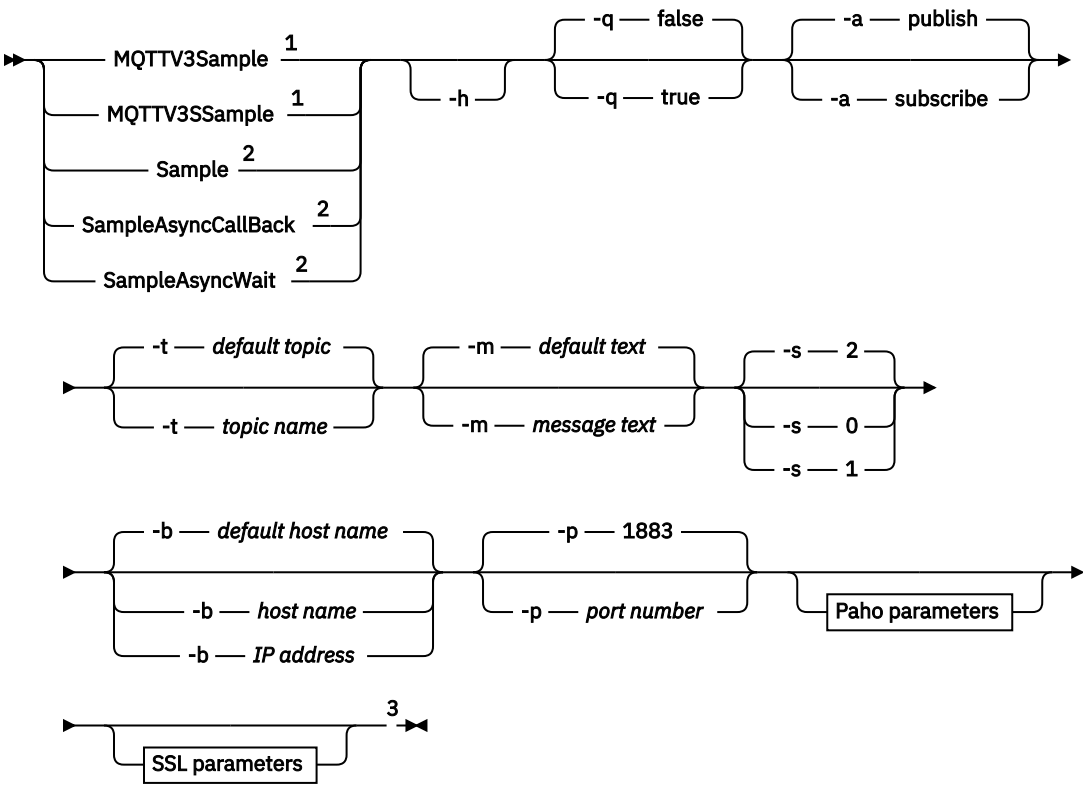
MQTT-Befehlszeilenbeispielprogramme

Syntax und Parameter der MQTT-Befehlszeilenbeispielprogramme

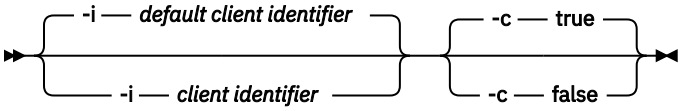
Verwendungszweck

Durchführung von Veröffentlichungen und Subskriptionen für ein Thema

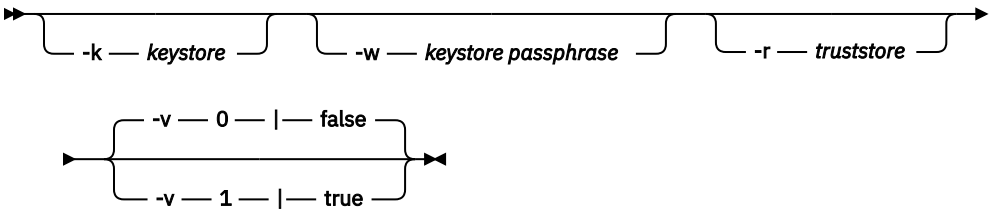
Syntax



Paho parameters



SSL parameters



Anmerkungen:

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

Parameter

`-h`
Der Hilfetext wird ausgedruckt und das Script wird beendet.

`-q`
Einstellen des Standardmodus 'false' anstatt des Befehlszeilenmodus.

`-a publish|subscribe`
Statt Veröffentlichung als Standardaktion vorauszusetzen, wird `publish` oder `subscribe` als Aktion festgelegt.

`-t` *Themaname*
Veröffentlichen oder subskribieren Sie *`topic name`*, anstatt das Standardtopic zu veröffentlichen oder zu subskribieren. Es gibt folgende Standardthemen:

Paho-Beispiele

Veröffentlichen
`Sample/Java/v3`

Abonnieren
`Sample/#`

IBM WebSphere MQ-Beispiele

Veröffentlichen
`MQTTV3Sample/Java/v3` oder `MQTTV3Sample/C/v3`

Abonnieren
`MQTTV3Sample/#`

`-m` *Nachrichtentext*
Veröffentlichen Sie *`message text`* , anstatt den Standardtext zu senden. Der Standardtext ist entweder "`Message from MQTTv3 C client`"oder "`Message from MQTTv3 Java client`"

`-s 0|1|2`
Statt die Standardservicequalität 2 zu verwenden, wird die Servicequalität (QoS) eingestellt.

`-b` *Hostname*
Stellen Sie eine Verbindung zu *`host name`* oder IP-Adresse her, anstatt eine Verbindung zum Standardhostnamen herzustellen. Der Standardhostname für die Paho-Beispiele lautet `m2m.eclipse.org`. Für die IBM WebSphere MQ-Beispiele ist es `localhost`.

-p ***Portnummer***
Verwenden Sie den Port *port number* anstelle des Standardports 1883.

Paho-Parameter

-i ***Client-ID***
Setzen Sie die Client-ID auf *client identifier*. Die Standardclient-ID ist `SampleJavaV3_"+action`, wobei `action` für `publish` oder `subscribe`besteht.

-c **`true`|`false`**
Legt die Markierung für die Bereinigungssitzung fest. Der Standardwert ist `true`: Subskriptionen sind nicht permanent.

SSL-Parameter

-k ***Keystore***
Legen Sie den Pfad zu dem Keystore mit dem privaten Schlüssel, der den Client identifiziert, für *keystore*fest. Der Speicher für die C-Beispiele ist eine PEM-Datei (Privacy-Enhanced Mail). Für die Java-Beispiele ist es ein Java-Keystore (JKS).

-w ***Keystore-Kennphrase***
Legen Sie die Kennphrase fest, um den Client für den Zugriff auf den Keystore für *keystore passphrase*zu berechtigen.

-r ***Truststore***
Setzen Sie den Pfad zum Keystore mit den öffentlichen Schlüsseln von MQTT -Servern, denen der Client vertraut, auf *truststore*. Der Keystore ist eine PEM-Datei (Privacy-Enhanced Mail). Der Speicher für die C-Beispiele ist eine PEM-Datei (Privacy-Enhanced Mail). Für die Java-Beispiele ist es ein Java-Keystore (JKS).

-v **`0`|`false`|`1`|`true`**
Setzen Sie die Prüfoption auf `1`|`true`, wenn ein Serverzertifikat erforderlich sein soll. Der Standardwert ist `0`|`false`: Das Serverzertifikat wird nicht überprüft. Der SSL-Kanal wird immer verschlüsselt.

Setzen Sie die Option auf `0`|`1` für C-Programme und `true`|`false` für Java -Programme.

Zugehörige Tasks

„Erste Schritte mit dem MQTT-Client für Java” auf Seite 12
Verwenden Sie IBM MessageSight oder IBM WebSphere MQ als MQTT -Server, um den MQTT -Client für Java -Beispielanwendungen betriebsbereit zu machen. Die Beispielanwendungen verwenden eine Clientbibliothek aus dem MQTT-Software-Development-Kit (SDK) von IBM. Die `SampleAsyncCallBack` -Beispielanwendung ist ein Modell zum Schreiben von MQTT -Anwendungen für Android und andere ereignisgesteuerte Betriebssysteme.

„Erste Schritte mit dem MQTT-Client für C” auf Seite 27
Erstellen Sie den MQTT-Beispielclient für C auf einer beliebigen Plattform, auf der Sie die C-Quelle kompilieren können. Prüfen Sie, ob Sie den MQTT-Beispielclient für C mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen können.

„MQTT-Client für C-Bibliotheken erstellen” auf Seite 32
Führen Sie die hier genannten Schritte aus, um den MQTT-Client für C-Bibliotheken zu erstellen. Der Abschnitt enthält auch die Kompilierungs- und Verknüpfungsschalter für eine Reihe von Plattformen sowie Beispiele für die Erstellung von Bibliotheken unter iOS und Windows.

MQTT-Sicherheit

Es gibt drei grundlegende Konzepte für die MQTT-Sicherheit: Identität, Authentifizierung und Autorisierung. Identität bezieht sich auf die Benennung des Clients, der autorisiert und dem Berechtigung erteilt wird. Authentifizierung bezieht sich auf die Überprüfung der Identität des Clients und Autorisierung bezieht sich auf die Verwaltung der Berechtigungen, die dem Client erteilt werden.

Sicherheitsbeispiele

- „Sichere MQTT-Clientbeispiel Java -App erstellen und ausführen” auf Seite 57
- „Verbindung zur Java-Beispiel-App des MQTT -Clients unter Android über SSL herstellen” auf Seite 65
- „Java-App eines MQTT -Clients mit JAAS authentifizieren” auf Seite 75
- V 7.5.0.1 „MQTT-Messaging-Client für JavaScript über SSL und WebSockets verbinden” auf Seite 80
- „Sichere MQTT-Client - Beispielapp für C erstellen und ausführen” auf Seite 88

ID

Identifizieren Sie einen MQTT-Client durch seine Client-ID, Benutzer-ID oder sein öffentliches digitales Zertifikat. Eines dieser Attribute definiert die Clientidentität. Ein MQTT-Server authentifiziert das Zertifikat, das vom Client gesendet wird, mithilfe des SSL-Protokolls oder die Clientidentität mithilfe eines Kennworts, das vom Client festgelegt wird. Der Server steuert, auf welche Ressourcen der Client auf Basis der Clientidentität zugreifen kann.

Der MQTT-Server identifiziert sich selbst mit seiner IP-Adresse und seinem digitalen Zertifikat gegenüber dem Client. Der MQTT-Client authentifiziert mithilfe des SSL-Protokolls das vom Server gesendete Zertifikat. In einigen Fällen verwendet er den DNS-Namen des Servers, um zu verifizieren, dass der Server, der ihm das Zertifikat gesendet hat, als der Zertifikatseigner registriert ist.

Sie haben folgende Möglichkeiten, die Identität des Clients festzulegen:

Client-ID
Die Klasse `MqttClient` (`MQTTClient_create` bzw. `MQTTAsync_create` in C) legt die Client-ID fest. Rufen Sie den Klassenkonstruktor auf, um die Client-ID als Parameter anzugeben oder eine nach dem Zufallsprinzip generierte Client-ID zurückzugeben.Die Client-ID muss unter allen Clients, die sich mit dem Server verbinden, eindeutig sein, und sie darf nicht identisch mit dem Namen des Warteschlangenmanagers auf dem Server sein. Alle Clients müssen eine Client-ID haben, auch wenn sie nicht für die Identitätsprüfung verwendet wird. Weitere Informationen finden Sie unter „Client-ID” auf Seite 132.

Benutzer-ID
Die `MqttClient`-Klasse (`MQTTClient_create` oder `MQTTAsync_create` in C) legt die Client-Benutzer-ID als Attribut von `MqttConnectOptions` (`MqttClient_ConnectOptions` in C) fest. Die Benutzer-ID muss für den Client nicht eindeutig sein.

Digitales Clientzertifikat
Das digitale Clientzertifikat wird im Client-Keystore gespeichert. Die Speicherposition des Keystores ist vom Client abhängig:

- **Java**

  Legen Sie die Position und die Eigenschaften des Client-Keystores fest, indem Sie die Methode `setSSLProperties` von `MqttConnectOptions` aufrufen und die Keystore-Eigenschaften übergeben. Weitere Informationen finden Sie unter SSL-Änderungen an Example.java. Das Tool **keytool** verwaltet Java-Schlüssel und -Keystores.

- **C**

  `MQTTClient_create` oder `MQTTAsync_create` legt die Keystore-Eigenschaften als Attribute von `MQTTClient_SSLOptions ssl_opts`fest. Das Tool **openSSL** erstellt und verwaltet die Schlüssel und Keystores, auf die der MQTT -Client für C zugreift.

- **Android**

  Verwalten Sie den Keystore eines Android-Geräts über das Menü **Einstellungen** > **Sicherheit**. Laden Sie neue Zertifikate von der SD-Karte.

Legen Sie die Identität des Servers fest, indem Sie seinen privaten Schlüssel im Server-Keystore speichern:

IBM WebSphere MQ

Der Keystore des MQTT-Servers ist ein Attribut des Telemetriekanals, mit dem der Client verbunden ist.

Legen Sie die Keystore-Position und die Attribute mit IBM WebSphere MQ Exploreroder mit dem Befehl **DEFINE CHANNEL** fest (siehe DEFINE CHANNEL (MQTT)). Ein Keystore kann von mehreren Kanälen gemeinsam genutzt werden.

Authentifizierung

Ein MQTT-Client kann den MQTT-Server, mit dem er eine Verbindung herstellt, und der Server kann den Client, der eine Verbindung mit ihm herstellt, authentifizieren.

Ein Client authentifiziert einen Server mithilfe des SSL-Protokolls. Ein MQTT-Server authentifiziert einen Client mithilfe des SSL-Protokolls oder eines Kennworts (oder beides).

Wenn der Client den Server authentifiziert, aber der Server nicht den Client, wird der Client häufig als anonymer Client bezeichnet. Es ist üblich, eine anonyme Clientverbindung über SSL herzustellen und den Client dann mithilfe eines Kennworts, das von der SSL-Sitzung verschlüsselt wird, zu authentifizieren. Aufgrund der problematischen Zertifikatsverteilung und -verwaltung wird ein Client häufiger mithilfe eines Kennworts als eines Clientzertifikats authentifiziert. Clientzertifikate werden vorwiegend in hochwertigen Geräten wie Geldausgabeautomaten und Systemen für bargeldlose Bezahlung mit Chipkarte und Geheimzahl sowie in kundenspezifischen Geräten wie intelligenten Stromzählern verwendet.

Serverauthentifizierung durch einen Client

Ein MQTT-Client prüft, ob er mit dem richtigen Server verbunden ist, indem er das Serverzertifikat mithilfe des SSL-Protokolls authentifiziert. Diese Form der Verifizierung ist Ihnen vertraut, wenn Sie über das HTTPS-Protokoll auf eine Website zugreifen.

Der Server sendet sein öffentliches Zertifikat, das von einer Zertifizierungsstelle signiert wurde, an den Client. Der Client prüft mithilfe des öffentlichen Schlüssels der Zertifizierungsstelle die Signatur der Zertifizierungsstelle auf dem Serverzertifikat. Außerdem überprüft er, ob das Zertifikat aktuell ist. Anhand dieser Prüfungen wird festgestellt, ob das Zertifikat gültig ist.

Zertifikate einer Zertifizierungsstelle, oft auch Stammzertifikate genannt, werden im Client-Truststore gespeichert:

- **Java**

  Rufen Sie die Methode `setSSLProperties` von `MqttConnectOptions` auf und übergeben Sie die Truststore-Eigenschaften, um die Speicherposition und Eigenschaften des Client-Truststores festzulegen. Weitere Informationen finden Sie unter SSL-Änderungen an Example.java. Verwalten Sie Zertifikate und Truststores mit dem Tool **keytool**.

- **C**

  `MQTTClient_create` oder `MQTTAsync_create` legen Sie die Truststore-Eigenschaften als Attribute von `MQTTClient_SSLOptions ssl_opts`fest. Verwalten Sie Zertifikate und Truststores mit dem Tool **openSSL**.

- **Android**

  Verwalten Sie den Truststore eines Android-Geräts über das Menü **Einstellungen** > **Sicherheit**. Laden Sie neue Stammzertifikate von der SD-Karte.

Clientauthentifizierung durch einen Server

Ein MQTT-Server prüft, ob er mit dem richtigen Client verbunden ist, indem er das Clientzertifikat mithilfe des SSL-Protokolls oder die Clientidentität mithilfe eines Kennworts authentifiziert.

Er authentifiziert den Client mithilfe des Kennworts, das der Client in einem MQTT protocolheader an den Server sendet. Der Server kann wählen, ob er die Client-ID, Benutzer-ID oder das Zertifikat mithilfe des Kennworts authentifiziert. Dies ist vom Server abhängig. Üblicherweise authentifiziert der Server die Benutzer-ID. Überprüfen Sie Kennwörter über eine SSL-Verbindung, die durch eine

Verifizierung des Servers gesichert ist, um zu verhindern, dass Kennwörter als Klartext gesendet werden.

- **IBM WebSphere MQ**

  IBM WebSphere MQ authentifiziert ein Clientzertifikat mithilfe des SSL-Protokolls. Speichern Sie Stammzertifikate im Keystore von IBM WebSphere MQ Telemetry. Ein Clientzertifikat kann nur im Rahmen einer gegenseitigen SSL-Authentifizierung authentifiziert werden. Das heißt, Sie müssen dem Client das öffentliche Serverzertifikat und dem Server das öffentliche Clientzertifikat zur Verfügung stellen.

  IBM WebSphere MQ Telemetry nutzt für das eigene private und öffentliche Zertifikat sowie für andere öffentliche Zertifikate wie die Stammzertifikate von einer Zertifizierungsstelle denselben Speicher.

  Legen Sie die Keystore-Position und die Attribute mit IBM WebSphere MQ Exploreroder mit dem Befehl **DEFINE CHANNEL** fest (siehe DEFINE CHANNEL (MQTT)). Ein Keystore kann von mehreren Kanälen gemeinsam genutzt werden.

  IBM WebSphere MQ authentifiziert die Client-Benutzer-ID oder die Client-ID, indem es den Java Authentication and Authorization Service (JAAS) aufruft.

  Konfigurieren Sie JAAS in einer `MQXRConfig`-Konfigurationszeilengruppe, die in der Datei `jaas.config` gespeichert ist. Die Datei wird im Verzeichnis `qmgrs\`*`QmgrName`*`\mqxr` im Datenpfad IBM WebSphere MQ gespeichert.

  Überprüfen Sie die Authentizität des Clients, indem Sie eine `login`-Methode für das Modul `JAASLoginModule` schreiben. Weitere Informationen finden Sie unter „JAAS-Konfiguration für Telemetriekanal” auf Seite 118.

  IBM WebSphere MQ Telemetry übergibt die Methode `JAASLoginModule.login` an die folgenden Parameter:

  – Benutzer-ID
  – Passwort
  – Client-ID
  – Netzwerk-ID
  – Kanalname
  – ValidPrompts

Berechtigung

Die Autorisierung ist nicht Bestandteil des MQTT protocols. Sie wird von MQTT-Servern bereitgestellt. Was autorisiert wird, hängt davon ab, was der Server tut. MQTT-Server sind Publish/Subscribe-Broker, die anhand geeigneter MQTT-Berechtigungsregeln steuern, welche Clients Verbindungen zum Server herstellen können und für welche Themen ein Client Veröffentlichungen oder Subskriptionen durchführen kann. Falls ein MQTT-Client den Server verwalten kann, wird durch zusätzliche Berechtigungsregeln gesteuert, welche Clients unterschiedliche Aspekte des Servers verwalten können.

Die Anzahl der möglichen Clients ist so groß, dass es nicht möglich ist, jeden Client separat zu autorisieren. Ein MQTT-Server hat die Möglichkeit, Clients nach Profilen zu Gruppen zusammenzufassen.

Die Identität eines Clients ist in Bezug auf Zugriff und Autorisierung nichts, was für einen MQTT-Client eindeutig ist. Verwechseln Sie die Identität eines Clients nicht mit der Client-ID. Sie können identisch sein, unterscheiden sich aber in der Regel. Beispielsweise haben Sie wahrscheinlich einen Benutzernamen, der für mehrere Services gültig ist, wobei einige dieser Services im Rahmen des "Single Sign-on" zusammenarbeiten. Ein MQTT-Server auf Unternehmensebene ruft normalerweise einen Berechtigungsservice auf, der gemeinsame Identitäten und Berechtigungen für unterschiedliche Anwendungen anbietet.

IBM WebSphere MQ

IBM WebSphere MQ verfügt über einen modular aufgebauten Berechtigungsservice. Der Standardberechtigungsservice, der unter Windows und Linux bereitgestellt wird, ist der Objektberechtigungsmanager. Siehe Zugriff auf Objekte mithilfe des OAM auf UNIX-, Linux -und Windows-Systemen steuern. Er ordnet Betriebssystembenutzer-IDs und Gruppen Operationen für IBM WebSphere MQ-Objekte, z. B. Themen und Warteschlangen, zu.

Sie können einen Telemetriekanal konfigurieren, um mit einer festgelegten Benutzer-ID auf IBM WebSphere MQ zuzugreifen. So ist auch der Beispielkanal konfiguriert. Sie können auch mit der vom MQTT -Client festgelegten Benutzer-ID auf IBM WebSphere MQ zugreifen. Im Abschnitt MQTT-Clients für den Zugriff auf WebSphere MQ-Objekte autorisieren wird beschrieben, wie IBM WebSphere MQ Telemetry für eine einfache, mittlere und differenzierte Clientzugriffssteuerung konfiguriert werden kann.

Zugehörige Tasks

„Sichere MQTT-Client - Beispielapp für C erstellen und ausführen” auf Seite 88
Auf der Basis eines Windows-Beispiels können Sie die sichere C-Beispiel-App erstellen und auf einem beliebigen Betriebssystem ausführen, für das Sie die C-Quelle kompilieren können. Prüfen Sie, ob Sie die C-Beispiel-App mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen können.

„Sichere MQTT-Clientbeispiel Java -App erstellen und ausführen” auf Seite 57
Auf der Basis eines Windows-Beispiels können Sie die sichere Java-Beispiel-App erstellen und mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen.Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.

„MQTT-Messaging-Client für JavaScript über SSL und WebSockets verbinden” auf Seite 80
Verbinden Sie Ihre Web-App sicher mit IBM WebSphere MQ, indem Sie die HTML-Beispielseiten des MQTT-Messaging-Clients für MQTT-Messaging-Client für JavaScript> mit SSL und dem WebSocket protocol verwenden.

„Verbindung zur Java-Beispiel-App des MQTT -Clients unter Android über SSL herstellen” auf Seite 65
Bereiten Sie den Android-MQTT-Beispielclient, der über SSL mit IBM WebSphere MQ verbunden wird, für die Ausführung vor.

„Java-App eines MQTT -Clients mit JAAS authentifizieren” auf Seite 75
An dieser Stelle erfahren Sie, wie Sie einen Client mit JAAS authentifizieren können. Führen Sie die Schritte in dieser Task aus, um das Beispielprogramm `JAASLoginModule.java` zu ändern und IBM WebSphere MQ für die Authentifizierung einer MQTT -Client-Java-App mit JAASzu konfigurieren.

Sichere MQTT-Clientbeispiel Java -App erstellen und ausführen

Auf der Basis eines Windows-Beispiels können Sie die sichere Java-Beispiel-App erstellen und mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen.Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.

Vorbereitende Schritte

1. Sie müssen über Zugriff auf einen MQTT version 3.1 -Server verfügen, der MQTT protocol über SSL unterstützt.
2. Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.
3. Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.. Siehe Systemvoraussetzungen für IBM Mobile Messaging und M2M Client-Pack.
4. Die SSL-Kanäle müssen gestartet sein.

Informationen zu diesem Vorgang

In diesem Artikel wird veranschaulicht, wie Sie die sichere MQTT-Clientbeispiel Java -App unter Windows über die Befehlszeile kompilieren und auszuführen können.

Sichern Sie den SSL-Kanal entweder mit Schlüsseln, die eine Zertifizierungsstelle signiert hat, oder mit selbst signierten Schlüsseln.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

   Der Server muss das MQTT version 3.1 -Protokoll über SSL unterstützen. Alle MQTT-Server von IBM führen dies aus, einschließlich IBM WebSphere MQ und IBM MessageSight. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern" auf Seite 142.
2. Optional: Installieren Sie ein Java Development Kit (JDK) ab Version 7.

   Version 7 ist erforderlich, um den Befehl **keytool** zum Zertifizieren von Zertifikaten auszuführen. Wenn Sie keine Zertifikate zertifizieren müssen, brauchen Sie JDK Version 7 nicht.
3. Laden Sie Mobile Messaging und M2M Client-Pack herunter und installieren Sie das MQTT SDK.

   Es gibt kein Installationsprogramm; Sie müssen lediglich die heruntergeladene Datei dekomprimieren.

   a. Laden Sie Mobile Messaging und M2M Client-Packherunter.

   b. Erstellen Sie einen Ordner, in dem Sie das Software-Development-Kit (SDK) installieren werden.

      Geben Sie dem Ordner zum Beispiel den Namen MQTT. Der Pfad zu diesem Ordner wird hier als *sdkroot*bezeichnet.

   c. Entpacken Sie den Inhalt der komprimierten Datei Mobile Messaging und M2M Client-Pack in *sdkroot*. Die Erweiterung erstellt eine Verzeichnisbaumstruktur, die bei *sdkroot*\SDKbeginnt.
4. Erstellen Sie die Scripts und führen Sie sie aus, um Schlüsselpaare und Zertifikate zu generieren und IBM WebSphere MQ als MQTT -Server zu konfigurieren.

   Führen Sie die Schritte in „Schlüssel und Zertifikate generieren" auf Seite 98 aus, um die Scripts zu erstellen und auszuführen. Die Scripts sind auch im Abschnitt „Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows" auf Seite 60 aufgelistet.
5. Überprüfen Sie, ob die SSL-Kanäle aktiv und wie erwartet konfiguriert sind.

   Geben Sie in IBM WebSphere MQ folgenden Befehl in einem Befehlsfenster ein:

   - 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. Erstellen Sie die Scripts zum Erstellen und Ausführen der sicheren MQTT-Clientbeispiel Java -App.

   a) Erstellen Sie das Script `ssjavaclient.bat` und führen Sie es aus, um einen SSL-Kanal zu testen, der mit selbst signierten Zertifikaten gesichert ist.

   b) Erstellen Sie das Script `cajavaclient.bat` und führen Sie es aus, um einen SSL-Kanal zu testen, der mit Zertifikaten gesichert ist, die von einer Zertifizierungsstelle signiert wurden.

Scripts zur Ausführung des sicheren MQTT-Java-Clients

Führen Sie die Scripts im Abschnitt „Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows" auf Seite 60 aus, bevor Sie diese Scripts ausführen.

Sicherer MQTT Java-Client mit selbst signierten Zertifikaten.
Führen Sie dieses Script mit den selbst signierten Zertifikaten aus, die Sie durch Ausführung des Scripts `sscerts.bat` erstellt haben.

```
@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\sam
ple\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.sam
ple.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.sam
ple.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w %cltjkskeystorepass%
-r %cltsrvjkstruststore% -v true
pause
endlocal
```

Abbildung 14. `ssjavaclient.bat`

Führen Sie den sicheren MQTT Java-Client mit den von einer Zertifizierungsstelle signierten Zertifikaten aus.

Führen Sie dieses Script mit den von einer Zertifizierungsstelle signierten Zertifikaten aus, die Sie durch Ausführung des Scripts `cacerts.bat` erstellt haben.

```
@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\sam
ple\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.sam
ple.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.sam
ple.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w %cltjkskeystorepass%
-r %cltcajkstruststore% -v true
pause
endlocal
```

Abbildung 15. `cajavaclient.bat`

Zugehörige Konzepte

„MQTT-Sicherheit” auf Seite 53

Es gibt drei grundlegende Konzepte für die MQTT-Sicherheit: Identität, Authentifizierung und Autorisierung. Identität bezieht sich auf die Benennung des Clients, der autorisiert und dem Berechtigung erteilt

wird. Authentifizierung bezieht sich auf die Überprüfung der Identität des Clients und Autorisierung bezieht sich auf die Verwaltung der Berechtigungen, die dem Client erteilt werden.

Zugehörige Tasks

„Schlüssel und Zertifikate generieren” auf Seite 98
Folgen Sie dieser Vorgehensweise, um Schlüssel und Zertifikate für Java- und C-Clients, einschließlich Android- und iOS-Apps, sowie für den IBM WebSphere MQ-Server und den IBM MessageSight-Server zu generieren.

„Verbindung zur Java-Beispiel-App des MQTT -Clients unter Android über SSL herstellen” auf Seite 65
Bereiten Sie den Android-MQTT-Beispielclient, der über SSL mit IBM WebSphere MQ verbunden wird, für die Ausführung vor.

„Java-App eines MQTT -Clients mit JAAS authentifizieren” auf Seite 75
An dieser Stelle erfahren Sie, wie Sie einen Client mit JAAS authentifizieren können. Führen Sie die Schritte in dieser Task aus, um das Beispielprogramm `JAASLoginModule.java` zu ändern und IBM WebSphere MQ für die Authentifizierung einer MQTT -Client-Java-App mit JAASzu konfigurieren.

Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows

** **

Die Beispielbefehlsdateien erstellen die Zertifikate und Zertifikatsspeicher wie in den Taskschritten beschrieben. Darüber hinaus wird der Warteschlangenmanager des MQTT-Clients im Beispiel so eingerichtet, dass er den Zertifikatsspeicher des Servers verwendet. Im Beispiel wird der Warteschlangenmanager gelöscht und neu erstellt, indem das Script `SampleMQM.bat` aufgerufen wird, das mit IBM WebSphere MQbereitgestellt wird.

`initcert.bat`

`initcert.bat` legt die Namen und Pfade zu Zertifikaten und anderen Parametern fest, die von den Befehlen **`keytool`** und **openSSL** benötigt werden. Die Einstellungen werden in den Kommentaren direkt im Script beschrieben.

```
@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

Die Befehle im Script `cleancert.bat` löschen den Warteschlangenmanager des MQTT-Clients, um sicherzustellen, dass der Serverzertifikatsspeicher nicht gesperrt wird. Dann löschen sie alle Keystores und Zertifikate, die von den Beispielsicherheitsscripts erstellt wurden.

```
@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

Die Befehle des Scripts `genkeys.bat` erstellen Schlüsselpaare für Ihre private Zertifizierungsstelle, den Server und einen Client.

```
@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 %cltjkskey
store% -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 %srvjkskey
```

```
store% -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%  -key
store %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg %algo
rithm% -validity %validity%
```

sscerts.bat

Mit den Befehlen des Scripts `sscerts.bat` werden die selbst signierten Client- und Serverzertifikate aus den Keystores exportiert. Danach wird das Serverzertifikat in den Truststore des Clients und das Clientzertifikat in den Keystore des Servers importiert. Der Server hat keinen Truststore. Die Befehle erstellen aus dem JKS-Truststore des Clients einen Client-Truststore im PEM-Format.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore% -store
pass %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% -store
pass %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-sig
ned 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 au
thentication)
@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% -destkey
store %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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Das Script importiert das Stammzertifikat der Zertifizierungsstelle in die privaten Keystores. Das Stammzertifikat der Zertifizierungsstelle wird für die Erstellung der Schlüsselkette zwischen dem Stammzertifikat und dem signierten Zertifikat benötigt. Das Script `cacerts.bat` exportiert die Cli-

ent- und die Serverzertifikatsanforderung aus deren Keystores. Das Script signiert die Zertifikatsanforderung mit dem Schlüssel der privaten Zertifizierungsstelle aus dem Keystore `cajkskeystore.jks` und importiert die signierten Zertifikate dann zurück in die Keystores, aus denen die Anforderungen gestellt wurden. Durch den Import wird die Zertifizierungskette zum Stammzertifikat der Zertifizierungsstelle erstellt. Das Script erstellt aus dem Client-JKS-Truststore einen Client-Truststore im PEM-Format.

```
@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%, %cltjkskey
store%, %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% -sto
repass %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% -sto
repass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained authenti
cation)
%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% -key
store %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% -key
store %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 %cltjkskeystore
pass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store: %cltcapemtrust
store%
@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 %cltcajkstrusts
torepass%  -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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

Dieses Script listet die Keystores und Zertifikate im Zertifikatsverzeichnis auf. Danach erstellt es den MQTT-Beispielwarteschlangenmanager und konfiguriert die sicheren Telemetriekanäle.

```
@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%') MCAU
SER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%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%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%) MCAUSER(%mcau
ser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Verbindung zur Java-Beispiel-App des MQTT -Clients unter Android über SSL herstellen

Bereiten Sie den Android-MQTT-Beispielclient, der über SSL mit IBM WebSphere MQ verbunden wird, für die Ausführung vor.

Vorbereitende Schritte

In diesem Artikel wird vorausgesetzt, dass Sie mindestens die Android-API der Version 14 (ICS 4.0) ausführen. Frühere Versionen hatten einen Keystore, es konnten jedoch nur System-Apps darauf zugreifen.

1. Sie müssen über Zugriff auf einen MQTT version 3.1 -Server verfügen, der MQTT protocol über SSL unterstützt.

2. Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.
3. Wenn Sie die Verbindung auf einem älteren Android-Gerät testen, benötigen Sie möglicherweise eine SD-Karte, um das Zertifikat an das Gerät zu übertragen.
4. Wenn Sie die Verbindung auf einer virtuellen Android-Einheit testen, konfigurieren Sie eine SD-Karte für die virtuelle Einheit.
5. Die SSL-Kanäle müssen gestartet sein.

Informationen zu diesem Vorgang

Führen Sie diese Aufgabe aus, um die MQTT-Clientbeispiel Java -App für Android über SSL auszuführen. Bei einer erfolgreichen SSL-Verbindung wird ein sicherer verschlüsselter Kanal zwischen Ihrem Android-Gerät und dem MQTT-Server erstellt. Die Identität des Servers wird authentifiziert.

Bei Android können Sie den Server mit SSL authentifizieren. Sie können auch das Gerät authentifizieren, auch wenn die Beispiel-App dies nicht unterstützt. Verwenden Sie zur Authentifizierung des Geräts entweder die KeyChain-API oder verwenden Sie JAAS, um die Client-ID, die Client-IP-Adresse, oder den Benutzernamen und das Kennwort, die von der MQTT-Android-App bereitgestellt werden, zu authentifizieren.

Alle X.509-Zertifikate, die Sie im Android-Truststore installieren, müssen von einer Zertifizierungsstelle signiert werden. Im dem Beispiel erstellen Sie eine Zertifizierungsstelle, die das Zertifikat signiert, das Sie in Ihrem Android-Gerät installieren. Zahlreiche Stammzertifikate werden in Android-Geräten vorinstalliert.

Sie müssen eine Sperre auf Ihrem Android-Gerät installieren, bevor Sie ein vertrauenswürdiges Zertifikat installieren. Die Sperre verhindert, dass jemand ohne Ihr Wissen Zertifikate auf dem Gerät installiert.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

   Der Server muss das MQTT version 3.1 -Protokoll über SSL unterstützen. Alle MQTT-Server von IBM führen dies aus, einschließlich IBM WebSphere MQ und IBM MessageSight. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern” auf Seite 142.
2. Führen Sie die Beispiel-App "`MQTTExerciser`" des MQTT-Clients for Android auf einem nicht gesicherten MQTT-Kanal aus. Weitere Informationen finden Sie unter „Erste Schritte mit dem MQTT-Client für Java unter Android” auf Seite 19.

   Sie verwenden die App erneut, um den sicheren Kanal zu testen.

   Wenn Sie eine virtuelle Android-Einheit gestartet haben, lassen Sie sie aktiviert.
3. Optional: Installieren Sie ein Java Development Kit (JDK) ab Version 7.

   Version 7 ist erforderlich, um den Befehl **`keytool`** zum Zertifizieren von Zertifikaten auszuführen. Wenn Sie keine Zertifikate zertifizieren müssen, brauchen Sie JDK Version 7 nicht.
4. Erstellen Sie die Scripts und führen Sie sie aus, um Schlüsselpaare und Zertifikate zu generieren und IBM WebSphere MQ als MQTT -Server zu konfigurieren.

   Führen Sie die Schritte in „Schlüssel und Zertifikate generieren” auf Seite 98 aus, um die Scripts zu erstellen und auszuführen. Die Scripts sind auch im Abschnitt „Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows” auf Seite 70 aufgelistet.

   Sie benötigen das öffentliche Zertifikat einer Zertifizierungsstelle und den Server-Keystore. Sie benötigen keine Clientzertifikate oder Zertifikate im Format `.pem` oder `.p12`.
5. Überprüfen Sie, ob die SSL-Kanäle aktiv und wie erwartet konfiguriert sind.

   Geben Sie in IBM WebSphere MQ folgenden Befehl in einem Befehlsfenster ein:

- 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. Installieren Sie das Zertifikat einer Zertifizierungsstelle im Android-Truststore.

   Im Beispiel handelt es sich bei der Zertifizierungsstellendatei um `cacert.cer`.

   a) Benennen Sie das Zertifikat in `cacert.crt` um.

   b) Kopieren Sie das Zertifikat in den internen Stammspeicher oder auf die SD-Karte.

      Öffnen Sie bei einer aktiven virtuellen Einheit Eclipse oder führen Sie die Android Debug Bridge (ADB) aus, um das Zertifikat in die virtuelle Einheit zu kopieren:

      Eclipse

      i) Führen Sie Eclipse aus und öffnen Sie die DDMS-Perspektive.

      ii) Öffnen Sie in der Hauptansicht das Fenster **File Explorer** (Dateiexplorer).

      iii) Öffnen Sie das Verzeichnis `mnt/sdcard`.

      iv) Ziehen Sie die Datei `cacert.crt` in das Verzeichnis `mnt/sdcard`.

      ADB

      i) Open a command window, and set its current directory to `android-sdk\platform-tools` in the android installation directory; for example, `C:\Program Files\Android\android-sdk\platform-tools`.

      ii) Kopieren Sie das Zertifikat in das Verzeichnis `mnt/sdcard`:

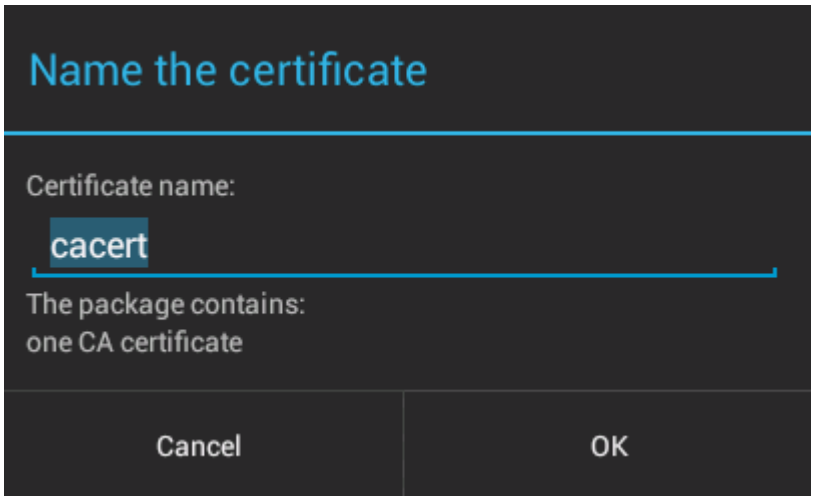
```
adb push %cacert% /mnt/sdcard/cacert.crt
```

7. Installieren Sie das Zertifikat im Zertifikatstruststore auf dem Android-Gerät.

   Das Zertifikat muss über eine Nutzungseinschränkungsklausel mit dem Wert `Subject Type=CA` verfügen.

   a) Entsperren Sie das Gerät und klicken Sie auf die Schaltfläche **widgets** (Widgets).

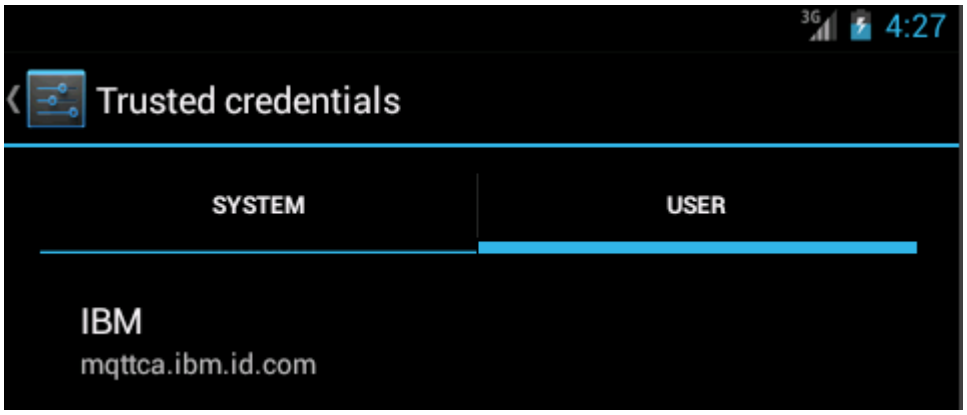
   b) Klicken Sie auf **Einstellungen** > **Sicherheit** > **Speicher für Berechtigungsnachweise** > **Von SD-Karte installieren**.



   c) Bestätigen Sie, dass der Name der Zertifikatsdatei korrekt ist, und klicken Sie auf **OK**.

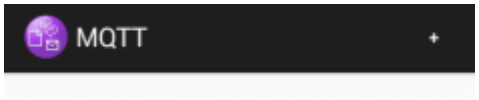
Anmerkung: Wenn Sie keine Sperre für das Gerät definiert haben, werden Sie jetzt von Android aufgefordert, eine Sperre einzurichten.

8. Bestätigen Sie, dass das Zertifikat auf dem Gerät installiert ist.
   a) Klicken Sie auf **Vertrauenswürdige Berechtigungsnachweise** > **Benutzer** und warten Sie einige Minuten, bis Ihr Zertifikat in der Liste der Benutzerzertifikate angezeigt wird.

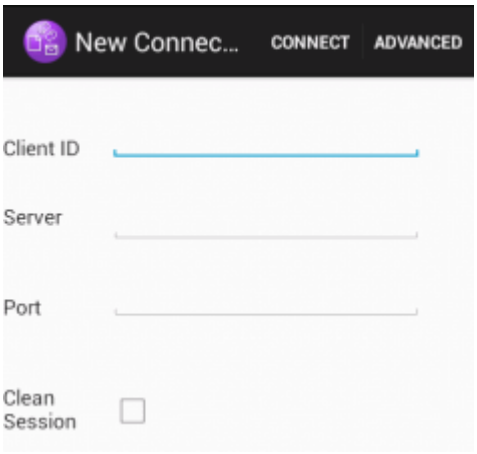


9. Führen Sie die App `MQTTExerciser` erneut aus und stellen Sie eine Verbindung zu einem MQTT-Kanal aus, den Sie für anonyme SSL-Clients konfiguriert haben.
   a) Öffnen Sie die MQTT-Clientbeispiel Java -App für Android.

   Auf Ihrem Android-Gerät wird folgendes Fenster geöffnet:



   b) Stellen Sie eine Verbindung zu einem MQTT -Server her.
      i) Klicken Sie auf das Pluszeichen (**+**), um eine neue MQTT-Verbindung zu öffnen.

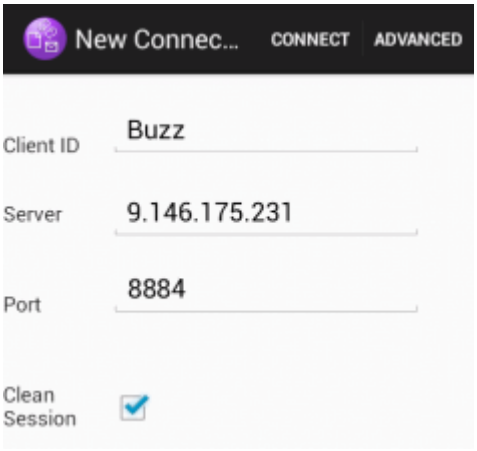


      ii) Geben Sie im Feld **Client-ID** eine eindeutige ID ein. Haben Sie etwas Geduld, die Tastatureingabe ist möglicherweise etwas langsam.
      iii) Geben Sie im Feld **Server** die IP-Adresse Ihres MQTT-Servers ein.

      Dies ist der Server, den Sie im ersten Hauptschritt ausgewählt haben. Die IP-Adresse darf nicht `127.0.0.1` lauten.

      iv) Geben Sie die Portnummer der MQTT-Verbindung ein.

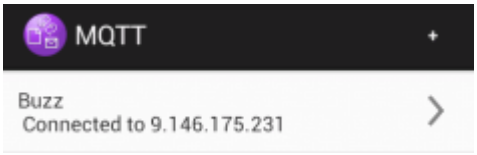
      Setzen Sie die Portnummer auf 8884. Dies wird durch die Variable `%sslportopt%` in den Beispielscripts festgelegt. Diese Portnummer ist die Portnummer des MQTT-Kanals, den Sie durch Ausführung der Beispielscripts in Schritt „4" auf Seite 66 für anonyme SSL-Clients konfiguriert haben.



v) Klicken Sie auf die Registerkarte **Advanced** (Erweitert) und wählen Sie die Option **SSL** aus. Klicken Sie auf **Speichern**.

vi) Klicken Sie auf **Connect** (Verbinden).

Wird die Verbindung erfolgreich hergestellt, sehen Sie eine "Connecting"-Nachricht und danach folgendes Fenster:



Ergebnisse

Die App `MQTTExerciser` braucht etwas länger, um eine Verbindung herzustellen und Nachrichten auszutauschen, verhält sich jedoch ansonsten nicht anders, als wenn eine unsichere Verbindung mit ihr hergestellt wird.

Zugehörige Konzepte

„MQTT-Sicherheit” auf Seite 53
Es gibt drei grundlegende Konzepte für die MQTT-Sicherheit: Identität, Authentifizierung und Autorisierung. Identität bezieht sich auf die Benennung des Clients, der autorisiert und dem Berechtigung erteilt wird. Authentifizierung bezieht sich auf die Überprüfung der Identität des Clients und Autorisierung bezieht sich auf die Verwaltung der Berechtigungen, die dem Client erteilt werden.

Zugehörige Tasks

„Schlüssel und Zertifikate generieren” auf Seite 98
Folgen Sie dieser Vorgehensweise, um Schlüssel und Zertifikate für Java- und C-Clients, einschließlich Android- und iOS-Apps, sowie für den IBM WebSphere MQ-Server und den IBM MessageSight-Server zu generieren.

„Sichere MQTT-Clientbeispiel Java -App erstellen und ausführen” auf Seite 57
Auf der Basis eines Windows-Beispiels können Sie die sichere Java-Beispiel-App erstellen und mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen.Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.

„Java-App eines MQTT -Clients mit JAAS authentifizieren” auf Seite 75
An dieser Stelle erfahren Sie, wie Sie einen Client mit JAAS authentifizieren können. Führen Sie die Schritte in dieser Task aus, um das Beispielprogramm `JAASLoginModule.java` zu ändern und IBM WebSphere MQ für die Authentifizierung einer MQTT -Client-Java-App mit JAASzu konfigurieren.

Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows

** **

Die Beispielbefehlsdateien erstellen die Zertifikate und Zertifikatsspeicher wie in den Taskschritten beschrieben. Darüber hinaus wird der Warteschlangenmanager des MQTT-Clients im Beispiel so eingerichtet, dass er den Zertifikatsspeicher des Servers verwendet. Im Beispiel wird der Warteschlangenmanager gelöscht und neu erstellt, indem das Script `SampleMQM.bat` aufgerufen wird, das mit IBM WebSphere MQbereitgestellt wird.

`initcert.bat`

`initcert.bat` legt die Namen und Pfade zu Zertifikaten und anderen Parametern fest, die von den Befehlen **`keytool`** und **openSSL** benötigt werden. Die Einstellungen werden in den Kommentaren direkt im Script beschrieben.

```
@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

Die Befehle im Script `cleancert.bat` löschen den Warteschlangenmanager des MQTT-Clients, um sicherzustellen, dass der Serverzertifikatsspeicher nicht gesperrt wird. Dann löschen sie alle Keystores und Zertifikate, die von den Beispielsicherheitsscripts erstellt wurden.

```
@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

Die Befehle des Scripts `genkeys.bat` erstellen Schlüsselpaare für Ihre private Zertifizierungsstelle, den Server und einen Client.

```
@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 %cltjkskey
store% -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 %srvjkskey
store% -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%  -key
store %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg %algo
rithm% -validity %validity%
```

sscerts.bat

Mit den Befehlen des Scripts `sscerts.bat` werden die selbst signierten Client- und Serverzertifikate aus den Keystores exportiert. Danach wird das Serverzertifikat in den Truststore des Clients und das Clientzertifikat in den Keystore des Servers importiert. Der Server hat keinen Truststore. Die Befehle erstellen aus dem JKS-Truststore des Clients einen Client-Truststore im PEM-Format.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
```

```
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore% -store
pass %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% -store
pass %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-sig
ned 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 au
thentication)
@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% -destkey
store %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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Das Script importiert das Stammzertifikat der Zertifizierungsstelle in die privaten Keystores. Das Stammzertifikat der Zertifizierungsstelle wird für die Erstellung der Schlüsselkette zwischen dem Stammzertifikat und dem signierten Zertifikat benötigt. Das Script `cacerts.bat` exportiert die Client- und die Serverzertifikatsanforderung aus deren Keystores. Das Script signiert die Zertifikatsanforderung mit dem Schlüssel der privaten Zertifizierungsstelle aus dem Keystore `cajkskeystore.jks` und importiert die signierten Zertifikate dann zurück in die Keystores, aus denen die Anforderungen gestellt wurden. Durch den Import wird die Zertifizierungskette zum Stammzertifikat der Zertifizierungsstelle erstellt. Das Script erstellt aus dem Client-JKS-Truststore einen Client-Truststore im PEM-Format.

```
@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%, %cltjkskey
store%, %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% -sto
repass %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% -sto
repass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained authenti
cation)
%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% -key
store %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% -key
store %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 %cltjkskeystore
pass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store: %cltcapemtrust
store%
@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 %cltcajkstrusts
torepass%  -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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

Dieses Script listet die Keystores und Zertifikate im Zertifikatsverzeichnis auf. Danach erstellt es den MQTT-Beispielwarteschlangenmanager und konfiguriert die sicheren Telemetriekanäle.

```
@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%') MCAU
SER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%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%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%) MCAUSER(%mcau
ser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Java-App eines MQTT -Clients mit JAAS authentifizieren

An dieser Stelle erfahren Sie, wie Sie einen Client mit JAAS authentifizieren können. Führen Sie die Schritte in dieser Task aus, um das Beispielprogramm `JAASLoginModule.java` zu ändern und IBM WebSphere MQ für die Authentifizierung einer MQTT -Client-Java-App mit JAASzu konfigurieren.

Vorbereitende Schritte

1. Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.. Siehe Systemvoraussetzungen für IBM Mobile Messaging und M2M Client-Pack.
2. Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.
3. Sie müssen Zugriff auf die `MQXR JAASLoginModule` -und `JAASPrincipal` Java -Beispiele in einer IBM WebSphere MQ -Installation haben. Die Beispiele befinden sich im Pfad `%MQ_FILE_PATH%\mqxr\samples.`
4. Führen Sie die Schritte unter Windows oder Linux aus. Die Beispiele stammen aus Windows.
5. Um Schritt „1” auf Seite 76 ausführen zu können, müssen Sie über die Berechtigung zum Erstellen des Warteschlangenmanagers `MQXR_SAMPLE_QM` in IBM WebSphere MQ verfügen.

Informationen zu diesem Vorgang

In der Task geben Sie die MQTT `Sample` -Clientidentifikationsparameter aus Ihrer Version von `JAASLoginModule`aus. Das Ausgeben der Clientparameter führt dazu, dass das Beispielprogramm `JAASLoginModule` geändert wird und IBM WebSphere MQ für das Laden Ihrer Version von `JAASLoginModule` konfiguriert wird.

Vorgehensweise

1. Führen Sie die Schritte unter „Java-Beispiel-Apps des MQTT-Clients über Eclipse kompilieren und ausführen” auf Seite 16 aus, um den Paho-MQTT-Sample-Client auszuführen.

   Ihr Ziel ist es, eine Entwicklungsumgebung zum Entwickeln und Testen der JAAS-Authentifizierung vorzubereiten. Sie benötigen eine Java-Entwicklungsumgebung, um das JAAS-Authentifizierungsmodul anzupassen. Im Beispiel führen Sie den Paho-Beispielclient für Java aus, um Ihre JAAS-Konfiguration zu testen. Verwenden Sie der Einfachheit halber dieselbe Entwicklungsumgebung, um sowohl den Beispielclient als auch das JAAS-Beispielanmeldemodul zu ändern. Sie können Ihr JAAS-Anmeldemodul auch mit dem MQTT-Client für C oder einem beliebigen anderen MQTT-Client testen.

2. Optional: Fügen Sie zu dem MQTT-Paho-Beispiel jeweils einen Parameter für den Benutzernamen und das Kennwort hinzu.

   Anmerkung: Wenn Ihr Paho-Client für Java die Parameter für den Benutzernamen und das Kennwort enthält, ist dieser Schritt nicht erforderlich. Überprüfen Sie die Download-Site auf eine Aktualisierung. Schauen Sie auf der Website Downloads für IBM Messaging-Community nach, ändern Sie andernfalls Ihre Kopie von `Sample.java`.

   a) Öffnen Sie den Paketexplorer im Paket `org.eclipse.paho.sample.mqttv3app` im Paho-Beispielprojekt.

   b) Klicken Sie mit der rechten Maustaste auf `Sample.java` und wählen Sie **Kopieren** > **Einfügen** aus. Geben Sie im Fenster **Namenskonflikt** den Namen `SampleForJAAS` ein.

   c) Fügen Sie die folgenden Codezeilen zur Methode `main` hinzu.

      i) Deklarieren Sie unter der Zeile "`boolean ssl = false;`" die Variablen `userName` und `password`:

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

      Damit die Kompatibilität mit älteren MQTT-Servern gewahrt bleibt, legen Sie die Parameter für das Kennwort und den Benutzernamen standardmäßig nicht fest.

      ii) Analysieren Sie nach der Zeile "`case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;`" die beiden neuen Eingabeparameter:

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

      iii) Fügen Sie vor der Zeile "`if (action.equals("publish")) {`" `userName` und `password` zu den Argumenten des Konstruktors `Sample` hinzu:

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

   d) Fügen Sie `userName` und `password` zum Konstruktor `Sample` hinzu.

      Folgenden Eintrag:

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

      In:

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

   e) Fügen Sie die folgenden Codezeilen zum Konstruktor `Sample` hinzu.

      Legen Sie nach der Zeile "`conOpt.setCleanSession(clean);`" die Variablen `userName` und `password` im Objekt `conOpt` in der Methode `Sample` fest:

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

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

f) Ändern Sie in den Methoden `publish` und `subscribe` die folgenden Codezeilen:

Ändern Sie die Zeile "`client.connect();`" in

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

3. Erstellen Sie ein Java-Projekt, `JAASSample`, für Ihr JAAS-Beispiel.
   a) Öffnen Sie in Ihrem Eclipse-Arbeitsbereich den Assistenten **New Java Project** (Neues Java-Projekt): Klicken Sie auf **File** > **New** > **Java project** (Datei > Neu > Java-Projekt).
   b) Geben Sie im Feld **Project name** (Projektname) `JAASSample` ein.
   c) Wählen Sie in den JRE-Optionen `J2SE-1.5` als JRE für die Ausführungsumgebung aus und klicken Sie auf **Next** (Weiter).

      Die JRE muss mit der JRE übereinstimmen, die Ihr IBM WebSphere MQ-Server ausführt. IBM WebSphere MQ Version 7.5 führt `J2SE-1.5`aus.
   d) Klicken Sie im Fenster **Java Settings** (Java-Einstellungen) auf **Libraries** (Bibliotheken) und klicken Sie auf **Add External JARS...** (Externe JAR-Dateien hinzufügen). Navigieren Sie zum`%MQ_FILE_PATH%\mqxr\lib` und wählen Sie **MQXR.jar**aus; klicken Sie auf **Fertigstellen**.
4. Importieren Sie die Klassen JAAS samples `JAASLoginModule` und `JAASPrincipal` .
   a) Klicken Sie mit der rechten Maustaste auf das Projekt `JAASSample` im Paketexplorer **Importieren ...** > **Allgemein** > **Dateisystem** und klicken Sie auf **Weiter**.
   b) Navigieren Sie zu `%MQ_FILE_PATH%\mqxr\samples` und aktivieren Sie `JAASLoginModule.java` und `JAASPrincipal.java`. Klicken Sie auf **Finish** (Fertigstellen).
   c) Wählen Sie beide Java -Dateien im Paketexplorer aus und klicken Sie mit der rechten Maustaste darauf. **Refactoring ...** > **Verschieben**.
   d) Überprüfen Sie im Fenster **Move** (Verschieben), ob `JAASSample` als Ziel für beide Elemente ausgewählt ist, und klicken Sie auf **Create Package...** (Paket erstellen...).
   e) Geben Sie in das Feld **Name** im Assistenten **New Java Package** (Neues Java-Paket) `samples` ein. Klicken Sie auf **Finish** > **OK** (Fertigstellen > OK).

   Eclipse erstellt die importierten Java-Klassen mit einer Reihe von Warnungen zu nicht verwendeten Werten.
5. Benennen Sie die Klasse `JAASLoginModule` um.

   Benennen Sie die Klasse so um, dass sie leichter von der Beispielklasse `JAASLoginModule` zu unterscheiden ist, die im Lieferumfang von IBM WebSphere MQ enthalten ist.
   a) Klicken Sie im Paketexplorer mit der rechten Maustaste auf `JAASloginModule.java` **Refactoring ...** > **Umbenennen**.
   b) Ändern Sie im Fenster **Rename Compilation Unit** (Kompiliereinheit umbenennen) das Feld **New name** (Neuer Name) von `JAASloginModule` in `MyJAASloginModule`. Klicken Sie auf **Finish** (Fertigstellen).
6. Ändern Sie die Klasse `MyJAASloginModule` so, dass der Inhalt der Callbackfelder ausgegeben wird.
   a) Fügen Sie die folgende Codezeile zu `MyJAASloginModule.java`hinzu.

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

      Platzieren Sie die Zeilen vor der Anweisung "`if (true) loggedIn = true;`".
   b) Drücken Sie `CTRL+Shift+O` , um Importe zu reorganisieren und die Datei zu speichern.
7. Benennen Sie `JAASPrincipal` in `MyJAASPrincipal` um.

Benennen Sie die Klasse um, um eine Verwechslung mit der Beispielklasse JAASPrincipal zu vermeiden. Lassen Sie im Beispiel den Inhalt der Klasse MyJAASPrincipal unverändert.

8. Erteilen Sie der Benutzer-ID, die die Warteschlangenmanagerprozesse ausführt, Lese- und Ausführungsberechtigungen (read und execute) für Ihre JAAS-Klassen.
   a) Öffnen Sie im Windows-Explorer Ihr Eclipse-Arbeitsbereichsverzeichnis. Im Beispiel wird die Arbeitsbereichsposition Eclipse durch die Variable Eclipse , *workspace_loc*, dargestellt.
   b) Navigieren Sie zu dem Verzeichnis, das Ihre Klassen MyJAASLoginModule und MyJAASPrincipal enthält.

      Der Verzeichnispfad lautet *workspace_loc*\JAASSample\bin\samples .
   c) Wählen Sie beide Klassen aus und klicken Sie anschließend mit der rechten Maustaste darauf. Klicken Sie auf **Eigenschaften** und anschließend auf die Registerkarte **Sicherheit** im Fenster **Eigenschaften**.
   d) Klicken Sie auf **Hinzufügen ...**. Geben Sie den Objektnamen mqmein und klicken Sie auf **Namen überprüfen** , um ihn zu überprüfen; klicken Sie auf **OK**.
   e) Wählen Sie in der Liste **Gruppen- oder Benutzernamen** den Namen **mqm** aus und aktivieren Sie in der Liste mit Berechtigungen für mqm **Lesen und Ausführen** und **Lesen**. Klicken Sie auf 'OK'.
9. Konfigurieren Sie IBM WebSphere MQ für die Ausführung Ihrer Klasse MyJAASLoginModule.
   a) Fügen Sie die Datei service.env zur IBM WebSphere MQ-Konfiguration hinzu, um die Klassenpfade zum Laden Ihrer Klasse MyJAASLoginModule zu definieren.

      Erstellen Sie eine Datei service.env mit der folgenden Klassenpfadanweisung in Ihrem Verzeichnis *WMQ_DATA_PATH* :

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

      Dabei ist *user.dir* das Verzeichnisstammverzeichnis für die Klassendateien, die in IhremEclipse -Arbeitsbereich kompiliert werden. Das Verzeichnis *WMQ_DATA_PATH* enthält das Verzeichnis qmgrs . Weitere Informationen finden Sie unter Zusätzliche Umgebungsvariablen.

      Tipp: CLASSPATH=*user.dir*\JAASSample\bin ist möglicherweise die einzige Anweisung in der Datei service.env .
   b) Fügen Sie eine Zeilengruppe, MyJAASStanza, zur Datei jaas.config hinzu, um Ihre Klasse MyJAASLoginModule relativ zu den Klassenpfaden in der Datei service.env anzugeben.

      jaas.config befindet sich im Verzeichnis mqxr des Warteschlangenmanagers *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.

      Die Zeilengruppe lautet wie folgt:

      ```
      MyJAASStanza {
        samples.MyJAASLoginModule required debug=true;
      };
      ```
   c) Konfigurieren Sie einen IBM WebSphere MQ Telemetry-Kanal mit dem Namen Ihrer JAAS-Konfigurationszeilengruppe.

      Führen Sie den folgenden Befehl über ein Befehlsfenster aus:

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

      Der Befehl bindet den Kanal MyJAAS an MyJAASStanza in der Datei jaas.config. Wenn die Option MCAUSER nicht angegeben wird oder die Option USECLTID in der Kanaldefinition angegeben wird, autorisiert der Kanal den Zugriff auf Warteschlangenmanagerressourcen mit dem Benutzernamen, der durch das MQTT-Clientprogramm angegeben wird. Im Beispiel wird der Benutzername, der durch den Client angegeben wird, auf "Guest" gesetzt. Die vorhandenen Berechtigungen für Guest, die durch die Befehlsdatei SampleMQM festgelegt werden, werden in diesem Beispiel ebenfalls verwendet.

10. Starten Sie den IBM WebSphere MQ Telemetry-Service erneut, um die neuen Konfigurationsdaten zu lesen.
    Um den IBM WebSphere MQ Telemetry-Service erneut zu starten, starten Sie den Warteschlangenmanager oder den Service über IBM WebSphere MQ Explorer oder führen Sie die folgenden Befehle für die Beispielkonfiguration aus:

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

11. Führen Sie das Programm `Sample` aus.

    Um eine Ausführungskonfiguration für `SampleForJAAS` zu konfigurieren, verwenden Sie dieselbe Vorgehensweise wie in Schritt „1” auf Seite 76, mit folgenden Änderungen:

    a) Setzen Sie die Portnummer entsprechend der MQTT-Kanalkonfiguration auf `1890`.

    b) Fügen Sie die Parameter `-u Guest -z password`zu den Kennwörtern auf der Registerkarte **(x)= Arguments** ((x) = Argumente) für die Subskribenten- und Bereitstellerkonfigurationen, die Sie für das Programm `Sample` erstellt haben.

    Die Beispielprogramme werden ohne Änderungen an der Ausgabe ausgeführt, mit der Ausnahme, dass die Portnummer jetzt `1890` und nicht `1883` ist.

    Öffnen Sie im Verzeichnis *WMQ_DATA_PATH*`\Qmgrs\MQXR_SAMPLE_QM` die Datei `mqxr.stdout` . Die Ausgabe von `MyJAASLoginModule` wird in `mqxr.stdout` geschrieben:

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

Nächste Schritte

Wenn Ihr Beispiel nicht funktioniert, lesen Sie den Abschnitt zur Fehlerbehebung für JAAS („Problembehebung: Das JAAS-Anmeldemodul wird vom Telemetrieservice nicht aufgerufen” auf Seite 193) und versuchen Sie, die folgenden Tipps zur Fehlerbehebung anzuwenden.

1. Fügen Sie `-verbose` zu den Parametern in *WMQ_DATA_PATH*`\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties`hinzu. In diesem Protokoll können Sie sehen, ob Ihre Klasse erfolgreich geladen wurden.

   Die Ausgabe wird in *WMQ_DATA_PATH*`\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr`geschrieben.
2. Suchen Sie in *WMQ_DATA_PATH*`\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log` nach Ausnahmebedingungen, die in `MyJAASLoginModule`ausgelöst wurden. Wenn Sie beispielsweise versuchen, ein Nullkennwort (`password`) auszugeben, das ein Zeichenbereich und keine Zeichenfolge ist, wird eine Ausnahmebedingung ausgelöst.
3. Suchen Sie in *WMQ_DATA_PATH*`\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log`. Wenn der Benutzername in `userName` nicht berechtigt ist, auf Warteschlangenmanagerressourcen zuzugreifen, und der Kanal ohne die Option `MCAUSER` oder mit der Option `USECLTID` konfiguriert ist, werden Fehler hier dokumentiert.
4. Überprüfen Sie, ob der Name der Zeilengruppe in *WMQ_DATA_PATH*`\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config` mit dem Namen in dem MQTT -Kanal übereinstimmt, der für den Port konfiguriert ist, zu dem der `Sample` -Client eine Verbindung herzustellen versucht.
5. Überprüfen Sie, ob der Pfad in der Zeilengruppe dem Pfad zur Klasse `MyJAASLoginModule` in Eclipse entspricht. Beispiel:

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

6. Um auszuschließen, dass der Fehler im Klassenpfad in der Datei `service.env` in *WMQ_DATA_PATH* liegt, ändern Sie die Zeile "`set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%`" in `%MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT` , um Ihren Klassenpfad einzuschließen. Sie können den Klassenpfad auch zurückmelden. Der Klassenpfad enthält jedoch nicht den Klassenpfad, der in der Datei `service.env` festgelegt ist, daher funktioniert dies nur, wenn Sie die Datei `controlMQXR.BAT` ändern.

Zugehörige Konzepte

„MQTT-Sicherheit" auf Seite 53
Es gibt drei grundlegende Konzepte für die MQTT-Sicherheit: Identität, Authentifizierung und Autorisierung. Identität bezieht sich auf die Benennung des Clients, der autorisiert und dem Berechtigung erteilt wird. Authentifizierung bezieht sich auf die Überprüfung der Identität des Clients und Autorisierung bezieht sich auf die Verwaltung der Berechtigungen, die dem Client erteilt werden.

„JAAS-Konfiguration für Telemetriekanal" auf Seite 118
Konfigurieren Sie JAAS für die Authentifizierung des vom Client gesendeten `Benutzernamens` .

Zugehörige Tasks

„Problembehebung: Das JAAS-Anmeldemodul wird vom Telemetrieservice nicht aufgerufen" auf Seite 193
Stellen Sie fest, ob Ihr JAAS-Anmeldemodul tatsächlich nicht vom Telemetrieservice (MQXR) aufgerufen wird, und konfigurieren Sie JAAS entsprechend, um das Problem zu beheben.

„Sichere MQTT-Clientbeispiel Java -App erstellen und ausführen" auf Seite 57
Auf der Basis eines Windows-Beispiels können Sie die sichere Java-Beispiel-App erstellen und mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen.Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.

„Verbindung zur Java-Beispiel-App des MQTT -Clients unter Android über SSL herstellen" auf Seite 65
Bereiten Sie den Android-MQTT-Beispielclient, der über SSL mit IBM WebSphere MQ verbunden wird, für die Ausführung vor.

Zugehörige Informationen

Zusätzliche Umgebungsvariablen

MQTT-Messaging-Client für JavaScript über SSL und WebSockets verbinden

Verbinden Sie Ihre Web-App sicher mit IBM WebSphere MQ, indem Sie die HTML-Beispielseiten des MQTT-Messaging-Clients für MQTT-Messaging-Client für JavaScript> mit SSL und dem WebSocket protocol verwenden.

Vorbereitende Schritte

1. Sie müssen Zugriff auf einen MQTT version 3-Server haben, der das MQTT protocol über WebSockets unterstützt.
2. Der Browser muss SSL und das WebSocket protocol unterstützen. Siehe „Einschränkung der Browser-Unterstützung für Web-Apps für Mobile Messaging über SSL" auf Seite 181.
3. Die SSL-Kanäle müssen gestartet sein.

Informationen zu diesem Vorgang

Führen Sie diese Aufgabe aus, um die Beispielseiten des MQTT-Messaging-Clients für JavaScript über SSL auszuführen. In der Aufgabe werden Sie an Abschnitt „Schlüssel und Zertifikate generieren" auf Seite 98 weitergeleitet, um Zertifikate zu erstellen und IBM WebSphere MQ zu konfigurieren.

Sichern Sie den SSL-Kanal entweder mit Schlüsseln, die eine Zertifizierungsstelle signiert hat, oder mit selbst signierten Schlüsseln.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

Der Server muss das MQTT protocol über sichere WebSockets unterstützen.

- Für IBM MessageSight und Releases von IBM WebSphere MQ Version 7.5.0.1 und höher ist dies der Fall.

2. Optional: Installieren Sie ein Java Development Kit (JDK) Version 7 oder höher.

   Wenn Sie ein Testsystem einrichten und selbst signierte Zertifikate verwenden möchten, müssen Sie Ihre Zertifikate mit dem Befehl **keytool** von JDK Version 7 zertifizieren. Wenn Sie ein Produktionssystem einrichten und Zertifikatssignieranforderungen an eine externe Zertifizierungsstelle senden, ist JDK Version 7 nicht erforderlich.

3. Erstellen Sie die Scripts und führen Sie sie aus, um Schlüsselpaare und Zertifikate zu generieren und IBM WebSphere MQ als MQTT -Server zu konfigurieren.

   Führen Sie die Schritte in „Schlüssel und Zertifikate generieren” auf Seite 98 aus, um die Scripts zu erstellen und auszuführen. Die Scripts sind auch im Abschnitt „Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows” auf Seite 83 aufgelistet.

   Der allgemeine Name des Serverzertifikats muss mit dm DNS-Namen des Serverkanals übereinstimmen. Einige Browser akzeptieren Zertifikate, die eine Liste mit allgemeinen Namen enthalten, z. B.:

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

   Andere Browser akzeptieren nur einen einzigen allgemeinen Namen, z. B. Firefox bis Version 18. In späteren Versionen mag das anders sein.

4. Überprüfen Sie, ob die SSL-Kanäle aktiv und wie erwartet konfiguriert sind.

   Geben Sie in IBM WebSphere MQ folgenden Befehl in einem Befehlsfenster ein:

   - 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
     ```

5. Installieren Sie die Zertifikate im Browser-Zertifikatsspeicher.

   Wählen Sie zum Beispiel eines der folgenden Serverzertifikate aus:

   a. Für ein selbst signiertes Serverzertifikat das Zertifikat `srvcertselfsigned.cer`
   b. Für ein Serverzertifikat, das von Ihrer privaten Zertifizierungsstelle signiert ist, das Zertifikat `cacert.cer`
   c. Überprüfen Sie für ein Serverzertifikat, das von einer externen Zertifizierungsstelle signiert ist, ob das Stammzertifikat der Zertifizierungsstelle bereits im Zertifikatsspeicher installiert ist.

   Installieren Sie je nach dem Support, der in Ihrem Browser verfügbar ist, `cacert.cer` in der Liste 'Trusted Root Certification Authorities' (Anerkannte Root-Zertifizierungsstellen). Siehe „Einschränkung der Browser-Unterstützung für Web-Apps für Mobile Messaging über SSL” auf Seite 181.

6. Optional: Authentifizieren Sie den Client.

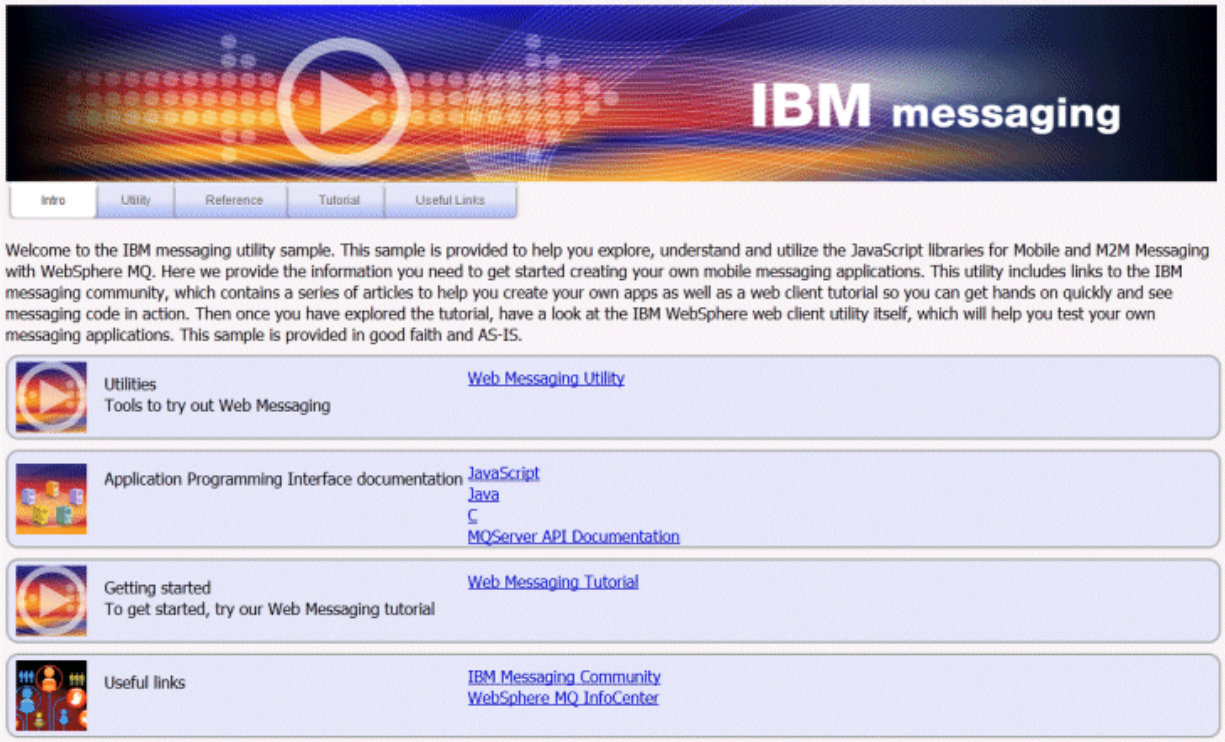
   Im Beispiel installieren Sie `cacert.cer`, um den Server zu authentifizieren und den Kanal zu verschlüsseln, aber nicht zur Authentifizierung des Clients. Um den Client zu authentifizieren, müssen Sie den Client-Keystore (`cltkeystore.p12`) im Browser installieren. Nicht alle Browser unterstützen die Authentifizierung von Clients. Siehe „Einschränkung der Browser-Unterstützung für Web-Apps für Mobile Messaging über SSL” auf Seite 181.

7. Stellen Sie eine Verbindung zum sicheren WebSockets-Kanal her.

Öffnen Sie Ihren Browser und geben Sie die URL des WebSockets-Kanals in der Adresszeile ein; im Beispiel ist dies folgende URL:

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

IBM WebSphere MQ antwortet mit der ersten Seite der Beispielseiten des MQTT-Messaging-Clients JavaScript.



Wenn die Verbindung fehlschlägt und Sie die Beispielscripts zum Konfigurieren des MQTT-Beispielwarteschlangenmanagers ausgeführt haben, versuchen Sie, eine Verbindung zum normalen WebSockets-Kanal an Port 1886 herzustellen. Durch einen Erfolg an Port 1886 wird der Fehler auf die SSL-Verbindung eingegrenzt.

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

Zugehörige Konzepte

„MQTT-Messaging-Client für JavaScript und Web-Apps" auf Seite 120

„Vorgehensweise bei der Programmierung von Messaging-Apps in JavaScript" auf Seite 124

Zugehörige Tasks

„Schlüssel und Zertifikate generieren" auf Seite 98

Folgen Sie dieser Vorgehensweise, um Schlüssel und Zertifikate für Java- und C-Clients, einschließlich Android- und iOS-Apps, sowie für den IBM WebSphere MQ-Server und den IBM MessageSight-Server zu generieren.

„Erste Schritte mit dem MQTT-Messaging-Client für JavaScript" auf Seite 25

Für die ersten Schritte mit dem MQTT-Messaging-Client für JavaScript können Sie die Beispielhomepage für den Messaging-Client anzeigen und die Ressourcen durchsuchen, zu denen Links vorhanden sind. Um diese Homepage anzuzeigen, konfigurieren Sie einen MQTT-Server für die Annahme von Verbindungen von den Beispielseiten des MQTT-Messaging-Clients JavaScript und geben anschließend die URL, die Sie auf dem Server konfiguriert haben, in einen Web-Browser ein. Der MQTT-Messaging-Client für JavaScript wird automatisch auf Ihrem Gerät gestartet und die Beispielhomepage für den Messaging-Client wird angezeigt. Diese Seite enthält Verknüpfungen zu Dienstprogrammen, einer Dokumentation zur Programmierschnittstelle, einem Lernprogramm und weiteren hilfreichen Informationsquellen.

Zugehörige Verweise

„Einschränkung der Browser-Unterstützung für Web-Apps für Mobile Messaging über SSL” auf Seite 181
Die verschiedenen Browser zeigen auf den einzelnen Plattformen unterschiedliches Verhalten. Diese Unterschiede sollten Sie bei der Konfiguration Ihrer Apps, Zertifizierungsstellen (CAs) und Clientzertifikate für die Verbindung mit MQTT-Messaging-Client für JavaScript über SSL und WebSockets berücksichtigen.

Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows

** **

Die Beispielbefehlsdateien erstellen die Zertifikate und Zertifikatsspeicher wie in den Taskschritten beschrieben. Darüber hinaus wird der Warteschlangenmanager des MQTT-Clients im Beispiel so eingerichtet, dass er den Zertifikatsspeicher des Servers verwendet. Im Beispiel wird der Warteschlangenmanager gelöscht und neu erstellt, indem das Script `SampleMQM.bat` aufgerufen wird, das mit IBM WebSphere MQbereitgestellt wird.

`initcert.bat`

`initcert.bat` legt die Namen und Pfade zu Zertifikaten und anderen Parametern fest, die von den Befehlen **`keytool`** und **openSSL** benötigt werden. Die Einstellungen werden in den Kommentaren direkt im Script beschrieben.

```
@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

Die Befehle im Script `cleancert.bat` löschen den Warteschlangenmanager des MQTT-Clients, um sicherzustellen, dass der Serverzertifikatsspeicher nicht gesperrt wird. Dann löschen sie alle Keystores und Zertifikate, die von den Beispielsicherheitsscripts erstellt wurden.

```
@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

Die Befehle des Scripts `genkeys.bat` erstellen Schlüsselpaare für Ihre private Zertifizierungsstelle, den Server und einen Client.

```
@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 %cltjkskey
store% -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 %srvjkskey
store% -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%  -key
store %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg %algo
rithm% -validity %validity%
```

sscerts.bat

Mit den Befehlen des Scripts `sscerts.bat` werden die selbst signierten Client- und Serverzertifikate aus den Keystores exportiert. Danach wird das Serverzertifikat in den Truststore des Clients und das

Clientzertifikat in den Keystore des Servers importiert. Der Server hat keinen Truststore. Die Befehle erstellen aus dem JKS-Truststore des Clients einen Client-Truststore im PEM-Format.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore% -store
pass %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% -store
pass %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-sig
ned 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 au
thentication)
@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% -destkey
store %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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Das Script importiert das Stammzertifikat der Zertifizierungsstelle in die privaten Keystores. Das Stammzertifikat der Zertifizierungsstelle wird für die Erstellung der Schlüsselkette zwischen dem Stammzertifikat und dem signierten Zertifikat benötigt. Das Script `cacerts.bat` exportiert die Client- und die Serverzertifikatsanforderung aus deren Keystores. Das Script signiert die Zertifikatsanforderung mit dem Schlüssel der privaten Zertifizierungsstelle aus dem Keystore `cajkskeystore.jks` und importiert die signierten Zertifikate dann zurück in die Keystores, aus denen die Anforderungen gestellt wurden. Durch den Import wird die Zertifizierungskette zum Stammzertifikat der Zertifizierungsstelle erstellt. Das Script erstellt aus dem Client-JKS-Truststore einen Client-Truststore im PEM-Format.

```
@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%, %cltjkskey
store%, %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% -sto
repass %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% -sto
repass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained authenti
cation)
%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% -key
store %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% -key
store %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 %cltjkskeystore
pass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store: %cltcapemtrust
store%
@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 %cltcajkstrusts
torepass%  -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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

Dieses Script listet die Keystores und Zertifikate im Zertifikatsverzeichnis auf. Danach erstellt es den MQTT-Beispielwarteschlangenmanager und konfiguriert die sicheren Telemetriekanäle.

```
@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%') MCAU
SER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) | runmqsc %qm% >> %mqlog%
```

V7.5.0.1

```
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%) MCAUSER(%mcau
ser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Sichere MQTT-Client - Beispielapp für C erstellen und ausführen

Auf der Basis eines Windows-Beispiels können Sie die sichere C-Beispiel-App erstellen und auf einem beliebigen Betriebssystem ausführen, für das Sie die C-Quelle kompilieren können. Prüfen Sie, ob Sie die C-Beispiel-App mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen können.

Vorbereitende Schritte

1. Sie müssen über Zugriff auf einen MQTT version 3.1 -Server verfügen, der MQTT protocol über SSL unterstützt.
2. Wenn sich zwischen Client und Server eine Firewall befindet, stellen Sie sicher, dass der MQTT -Datenverkehr nicht blockiert wird.
3. Binärversionen der Bibliotheken des Clients für C werden für eine Reihe von Betriebssystemen bereitgestellt. Für einige dieser Betriebssysteme wird die sichere Version des Clients nicht als Binärdatei bereitgestellt. Bei diesen Betriebssystemen müssen Sie den Anweisungen im Abschnitt „MQTT-Client für C-Bibliotheken erstellen” auf Seite 32 folgen.
4. Zur Fehlerbehebung werden Sie möglicherweise vom IBM Support aufgefordert, den MQTT-Client für C auf einer Referenzplattform auszuführen.
5. Die SSL-Kanäle müssen gestartet sein.

Eine Übersicht über unterstützte Plattformen und Referenzplattformen finden Sie unter Systemvoraussetzungen für IBM Mobile Messaging und M2M Client-Pack. Ausführliche Informationen dazu, was für

den C-Client unterstützt wird, finden Sie in den relevanten Abschnitten unter System Requirements for WebSphere MQ V7.5 Telemetry.

Informationen zu diesem Vorgang

In diesem Artikel wird veranschaulicht, wie Sie die sichere MQTT-Client - Beispielapp für C unter Windows über die Befehlszeile kompilieren und auszuführen können. Dabei wird Microsoft Visual Studio 2010 zum Kompilieren des Clients verwendet. Sie können die Befehlszeilenscripts ändern, wenn Sie die Beispiel-App auf anderen Betriebssystemen, z. B. Linux oder iOS, kompilieren und ausführen möchten.

Anmerkung:

Die Windows-Scripts, die in diesem Artikel bereitgestellt werden, setzen voraus, dass Sie das gesamte OpenSSL-Paket von der Quelle aus erstellen. Wenn Sie die vorkompilierten Bibliotheken verwenden möchten, die IBM bereitstellt, möchten Sie möglicherweise auch lieber ein vorkompiliertes binäres Release von OpenSSL verwenden. Vorkompilierte Bibliotheken sind für die Verwendung mit iOS nicht verfügbar.

Sichern Sie den SSL-Kanal entweder mit Schlüsseln, die eine Zertifizierungsstelle signiert hat, oder mit selbst signierten Schlüsseln.

Vorgehensweise

1. Wählen Sie einen MQTT -Server aus, mit dem Sie die Client-App verbinden können.

   Der Server muss das MQTT version 3.1 -Protokoll über SSL unterstützen. Alle MQTT-Server von IBM führen dies aus, einschließlich IBM WebSphere MQ und IBM MessageSight. Weitere Informationen finden Sie unter „Erste Schritte mit MQTT-Servern” auf Seite 142.
2. Optional: Installieren Sie ein Java Development Kit (JDK) ab Version 7.

   Version 7 ist erforderlich, um den Befehl **keytool** zum Zertifizieren von Zertifikaten auszuführen. Wenn Sie keine Zertifikate zertifizieren müssen, brauchen Sie JDK Version 7 nicht.
3. Installieren Sie auf Ihrer Entwicklungsplattform eine C-Entwicklungsumgebung.

   Für die Makefiles in den Beispielen dieses Abschnitts werden die folgenden Tools verwendet:

   - iOS: Apple Mac mit OS X 10.8.2 mit den iOS-Entwicklungstools von Xcode.
   - Linux: GCC Version 4.4.6 aus Red Hat Enterprise Linux Version 6.2.

     Die unterstützte Mindestversion der C-Bibliothek `glibc` ist 2.12 und die des Linux-Kernels ist 2.6.32.
   - Windows: Microsoft Windows: Visual Studio Version 10.0
4. Laden Sie Mobile Messaging und M2M Client-Pack herunter und installieren Sie das MQTT SDK.

   Es gibt kein Installationsprogramm; Sie müssen lediglich die heruntergeladene Datei dekomprimieren.

   a. Laden Sie Mobile Messaging und M2M Client-Packherunter.

   b. Erstellen Sie einen Ordner, in dem Sie das Software-Development-Kit (SDK) installieren werden.

      Geben Sie dem Ordner zum Beispiel den Namen `MQTT`. Der Pfad zu diesem Ordner wird hier als *sdkroot*bezeichnet.

   c. Entpacken Sie den Inhalt der komprimierten Datei Mobile Messaging und M2M Client-Pack in *sdkroot*. Die Erweiterung erstellt eine Verzeichnisbaumstruktur, die bei *sdkroot*\SDKbeginnt.
5. Optional: Führen Sie die Schritte in „MQTT-Client für C-Bibliotheken erstellen” auf Seite 32aus.

   Führen Sie diesen Schritt nur aus, wenn das MQTT-SDK (Software-Development-Kit) nicht die Bibliothek des sicheren C-Clients für Ihr Zielbetriebssystem beinhaltet.

   - Windows: Bei den Bibliotheken handelt es sich um `mqttv3cs.lib` für die Kompilierung und um `mqttv3cs.dll` für die Ausführung.

- Linux Bei der Bibliothek handelt es sich um `libmqttv3cs.so`.
- iOS Bei der Bibliothek handelt es sich um `libmqttv3cs.a`.

6. Erstellen Sie die Scripts und führen Sie sie aus, um Schlüsselpaare und Zertifikate zu generieren und IBM WebSphere MQ als MQTT -Server zu konfigurieren.

   Führen Sie die Schritte in „Schlüssel und Zertifikate generieren" auf Seite 98 aus, um die Scripts zu erstellen und auszuführen. Die Scripts sind auch im Abschnitt „Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows" auf Seite 92 aufgelistet.
7. Überprüfen Sie, ob die SSL-Kanäle aktiv und wie erwartet konfiguriert sind.

   Geben Sie in IBM WebSphere MQ folgenden Befehl in einem Befehlsfenster ein:

   - 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. Erstellen Sie die Scripts zum Erstellen und Ausführen der sicheren MQTT-Client - Beispielapp für C.
   a) Erstellen Sie das Script `sscclient.bat` und führen Sie es aus, um einen SSL-Kanal zu testen, der mit selbst signierten Zertifikaten gesichert ist.
   b) Erstellen Sie das Script `cacclient.bat` und führen Sie es aus, um einen SSL-Kanal zu testen, der mit Zertifikaten gesichert ist, die von einer Zertifizierungsstelle signiert wurden.

Ergebnisse

Die Ergebnisse ähneln denen bei der Ausführung des nicht gesicherten Clients.

```
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
```

Abbildung 16. Sicherer Subskribent

```
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 . . .
```

Abbildung 17. Sicherer Bereitsteller

Scripts zur Ausführung der sicheren MQTT-Client - Beispielapp für C

Führen Sie die Scripts im Abschnitt „Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows” auf Seite 92 aus, bevor Sie diese Scripts ausführen.

Sicherer MQTT-Client - Beispielapp für C mit selbst signierten Zertifikaten.

Führen Sie dieses Script mit den selbst signierten Zertifikaten aus, die Sie durch Ausführung des Scripts `sscerts.bat` erstellt haben.

```
@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 ..\wind
ows_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 %cltpemkey
store% -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 %cltpemkey
store% -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
```

Abbildung 18. `sscclient.bat`

Sichere C-Beispiel-App des MQTT-Clients mit von einer Zertifizierungsstelle signierten Zertifikaten ausführen.

Führen Sie dieses Script mit den von einer Zertifizierungsstelle signierten Zertifikaten aus, die Sie durch Ausführung des Scripts `cacerts.bat` erstellt haben.

```
@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 ..\wind
ows_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 %cltpemkey
store% -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 %cltpemkey
store% -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
```

Abbildung 19. `cacclient.bat`

Zugehörige Konzepte

„MQTT-Sicherheit” auf Seite 53

Es gibt drei grundlegende Konzepte für die MQTT-Sicherheit: Identität, Authentifizierung und Autorisierung. Identität bezieht sich auf die Benennung des Clients, der autorisiert und dem Berechtigung erteilt wird. Authentifizierung bezieht sich auf die Überprüfung der Identität des Clients und Autorisierung bezieht sich auf die Verwaltung der Berechtigungen, die dem Client erteilt werden.

Zugehörige Tasks

„Schlüssel und Zertifikate generieren” auf Seite 98

Folgen Sie dieser Vorgehensweise, um Schlüssel und Zertifikate für Java- und C-Clients, einschließlich Android- und iOS-Apps, sowie für den IBM WebSphere MQ-Server und den IBM MessageSight-Server zu generieren.

Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows

Beispiel

Die Beispielbefehlsdateien erstellen die Zertifikate und Zertifikatsspeicher wie in den Taskschritten beschrieben. Darüber hinaus wird der Warteschlangenmanager des MQTT-Clients im Beispiel so eingerichtet, dass er den Zertifikatsspeicher des Servers verwendet. Im Beispiel wird der Warteschlangenmanager gelöscht und neu erstellt, indem das Script `SampleMQM.bat` aufgerufen wird, das mit IBM WebSphere MQbereitgestellt wird.

`initcert.bat`

`initcert.bat` legt die Namen und Pfade zu Zertifikaten und anderen Parametern fest, die von den Befehlen **keytool** und **openSSL** benötigt werden. Die Einstellungen werden in den Kommentaren direkt im Script beschrieben.

```
@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

Die Befehle im Script `cleancert.bat` löschen den Warteschlangenmanager des MQTT-Clients, um sicherzustellen, dass der Serverzertifikatsspeicher nicht gesperrt wird. Dann löschen sie alle Keystores und Zertifikate, die von den Beispielsicherheitsscripts erstellt wurden.

```
@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

Die Befehle des Scripts `genkeys.bat` erstellen Schlüsselpaare für Ihre private Zertifizierungsstelle, den Server und einen Client.

```
@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 %cltjkskey
store% -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 %srvjkskey
store% -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%  -key
store %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg %algo
rithm% -validity %validity%
```

sscerts.bat

Mit den Befehlen des Scripts `sscerts.bat` werden die selbst signierten Client- und Serverzertifikate aus den Keystores exportiert. Danach wird das Serverzertifikat in den Truststore des Clients und das Clientzertifikat in den Keystore des Servers importiert. Der Server hat keinen Truststore. Die Befehle erstellen aus dem JKS-Truststore des Clients einen Client-Truststore im PEM-Format.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore% -store
pass %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% -store
pass %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-sig
ned 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 au
thentication)
```

```
@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% -destkey□
store %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 %cltjkskeystore□
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Das Script importiert das Stammzertifikat der Zertifizierungsstelle in die privaten Keystores. Das Stammzertifikat der Zertifizierungsstelle wird für die Erstellung der Schlüsselkette zwischen dem Stammzertifikat und dem signierten Zertifikat benötigt. Das Script `cacerts.bat` exportiert die Client- und die Serverzertifikatsanforderung aus deren Keystores. Das Script signiert die Zertifikatsanforderung mit dem Schlüssel der privaten Zertifizierungsstelle aus dem Keystore `cajkskeystore.jks` und importiert die signierten Zertifikate dann zurück in die Keystores, aus denen die Anforderungen gestellt wurden. Durch den Import wird die Zertifizierungskette zum Stammzertifikat der Zertifizierungsstelle erstellt. Das Script erstellt aus dem Client-JKS-Truststore einen Client-Truststore im PEM-Format.

```
@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%, %cltjkskey□
store%, %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% -sto□
repass %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% -sto□
repass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained authenti□
cation)
%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% -key
store %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% -key
store %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 %cltjkskeystore
pass%
```

```
@rem
@echo ________________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store: %cltcapemtrust
store%
@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 %cltcajkstrusts
torepass%  -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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

Dieses Script listet die Keystores und Zertifikate im Zertifikatsverzeichnis auf. Danach erstellt es den MQTT-Beispielwarteschlangenmanager und konfiguriert die sicheren Telemetriekanäle.

```
@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%') MCAU
SER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
```

```
SER(%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%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%) MCAUSER(%mcau
ser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Schlüssel und Zertifikate generieren

Folgen Sie dieser Vorgehensweise, um Schlüssel und Zertifikate für Java- und C-Clients, einschließlich Android- und iOS-Apps, sowie für den IBM WebSphere MQ-Server und den IBM MessageSight-Server zu generieren.

Vorbereitende Schritte

1. Sie müssen eine Kopie des Befehls **keytool** besitzen. Nicht alle Versionen von **keytool** unterstützen die Konvertierung von Keystores aus dem Java-Keystore (JKS) in PKCS (Public Key Cryptographic System) oder die Signatur von Zertifikatsanforderungen. Im Beispiel wird der Befehl **keytool** in JDK Version 7.0 verwendet, das beide Leistungsmerkmale unterstützt.
2. Wenn Sie beabsichtigen, für den Client für C Schlüssel und Zertifikate zu generieren, die das Format Privacy-Enhanced Mail (PEM) haben, müssen Sie eine Kopie des Befehls **openSSL** besitzen. Folgen Sie den Schritten im Abschnitt „MQTT-Client für C-Bibliotheken erstellen” auf Seite 32, um das openSSL-Paket zu erstellen.
3. Ändern Sie die Parameterwerte im Script `initcert.bat`, um sie Ihren Anforderungen anzupassen. Insbesondere sollten Sie die Kennwortparameter ausschließen, um zu verhindern, dass Kennwörter eingetragen werden. Der Befehl **keytool** fordert Sie auf, fehlende Kennwörter einzugeben.

Informationen zu diesem Vorgang

Sie benötigen Schlüssel und Zertifikate, um sichere SSL-Verbindungen zwischen MQTT-Clients und -Servern herzustellen. In diesem Abschnitt werden zwei verschiedene Methoden zum Erstellen der erforderlichen Schlüssel und Zertifikate beschrieben: selbst signiert und von der eigenen Zertifizierungsstelle signiert. Welche Methode Sie verwenden, hängt davon ab, wie Sie Keystores und Zertifikate verwalten wollen.

Wenn Sie Zertifikate verwenden möchten, die eine externe Zertifizierungsstelle signiert hat, führen Sie nicht den Signierungsschritt in `cacerts.bat` aus, sondern senden Sie stattdessen die Zertifikatsanforderungen an eine externe Zertifizierungsstelle. Die Zertifizierungsstelle gibt zusätzlich zum signierten Zertifikat möglicherweise ein Zwischenzertifikat und ein Stammzertifikat zurück. Folgen Sie bezüglich der Installation der zurückgegebenen Zertifikate den Anweisungen der externen Zertifizierungsstelle.

Der IBM WebSphere MQ-Server sucht nur in dem Zertifikatsspeicher, den Sie in den Telemetriekanalkonfigurationsparametern angeben, nach Zertifikaten. Er sucht nicht zusätzlich im JSE-Speicher `cacerts`. Ein Java-Client sucht im von Ihnen angegebenen Truststore nach Zertifikaten. Wenn Sie keinen Truststore angeben, sucht er im Speicher `cacerts` im JSE-Verzeichnis `jre\lib\security`. Android-Clients suchen im vordefinierten Zertifikatsspeicher auf dem Android-Gerät nach Zertifikaten. C-Client-Apps und iOS-Apps suchen nur in den von der Anwendung angegebenen Zertifikatsspeichern.

Android- und Java-Clients durchsuchen einen vorkonfigurierten Truststore nach vertrauenswürdigen Zertifikaten. CA-Stammzertifikate werden im vertrauenswürdigen Android -Zertifikatsspeicher und im JSE `jre\lib\security\cacerts` -Speicher gespeichert. Wenn das Stammzertifikat der Zertifizierungsstelle, die das Serverzertifikat zertifiziert hat, bereits im vorkonfigurierten Truststore installiert ist, definieren Sie keinen Client-Truststore. Die einzige erforderliche Konfiguration ist die des TCP/IP-Ports für den gesicherten MQTT-Serverkanal.

Die Tools zum Erstellen von Schlüsseln und Zertifikaten und Verwalten der unterschiedlichen Formate sind nicht einfach. Es müssen zahlreiche Parameter verwaltet werden und für **openSSL** sind eine Konfigurationsdatei (`openssl.cnf`) und Befehlszeilenparameter erforderlich. Keines der Tools bietet alle Funktionen, die zur Verwaltung von Schlüsseln und Zertifikaten für Anwendungen, die sowohl in C als auch in Java ausgeführt werden, erforderlich sind. Telemetriekanäle in IBM WebSphere MQ erfordern einen JKS-Keystore, weshalb in den Beispielen hauptsächlich die Java-Zertifikattools **ikeyman** und **keytool** verwendet werden. Allerdings unterstützen die Java-Tools nicht das Format PEM, das für C-Client-Apps erforderlich ist. Um Keystores im Format PEM zu erstellen, müssen Sie das Tool **openSSL** ausführen. Das Tool **openSSL** konvertiert Keystores aus dem Format PKCS12 in das Format PEM und **keytool** konvertiert Keystores zwischen den Formaten JKS und PKCS12. Eine `openssl.cnf`-Datei ist für die Keystore-Konvertierung nicht erforderlich. Sie benötigen **openSSL** nur, wenn Sie planen, C-Client-Apps oder iOS-Apps zu erstellen. Wenn Sie lieber mit **openSSL** arbeiten, können Sie es zum Signieren von Zertifikaten verwenden, statt Zertifikate mithilfe von **keytool** zu signieren.

Vorgehensweise

1. Öffnen Sie ein Befehlsfenster, um die folgenden Scripts auszuführen.
2. Erstellen Sie das Script `initcert.bat` und führen Sie es aus, um die Parameter festzulegen, die zum Ausführen der sicheren MQTT -Beispielclients erforderlich sind.
3. Erstellen Sie das Script `cleancert.bat` und führen Sie es aus, um die Umgebung zu bereinigen und für die Erstellung neuer Keystores und Zertifikate vorzubereiten.
4. Erstellen Sie das Script `genkeys.bat` und führen Sie es aus, um die erforderlichen Schlüsselpaare zu generieren.
5. Wählen Sie eine der folgenden Optionen aus:
   - Erstellen Sie das Script `sscerts.bat` und führen Sie es aus, um selbst signierte Zertifikate zu erstellen.
   - Erstellen Sie das Script `cacerts.bat` und führen Sie es aus, um von einer Zertifizierungsstelle signierte Zertifikatsketten zu erstellen.
6. Erstellen Sie das Script `mqcerts.bat` und führen Sie es aus, um den Warteschlangenmanager `MQXR_SAMPLE_QM` zu erstellen und seine Telemetriekanäle zu konfigurieren.

Zugehörige Tasks

„Sichere MQTT-Client - Beispielapp für C erstellen und ausführen” auf Seite 88
Auf der Basis eines Windows-Beispiels können Sie die sichere C-Beispiel-App erstellen und auf einem beliebigen Betriebssystem ausführen, für das Sie die C-Quelle kompilieren können. Prüfen Sie, ob Sie die C-Beispiel-App mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen können.

„Sichere MQTT-Clientbeispiel Java -App erstellen und ausführen” auf Seite 57
Auf der Basis eines Windows-Beispiels können Sie die sichere Java-Beispiel-App erstellen und mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen.Sie können eine MQTT-Client für Java -App auf jeder Plattform mit JSE 1.5 oder höher ausführen, die "Java kompatibel"ist.

Beispielscripts für die Konfiguration von SSL-Zertifikaten für Windows

Beispiel

Die Beispielbefehlsdateien erstellen die Zertifikate und Zertifikatsspeicher wie in den Taskschritten beschrieben. Darüber hinaus wird der Warteschlangenmanager des MQTT-Clients im Beispiel so eingerichtet, dass er den Zertifikatsspeicher des Servers verwendet. Im Beispiel wird der Warteschlangenmanager gelöscht und neu erstellt, indem das Script `SampleMQM.bat` aufgerufen wird, das mit IBM WebSphere MQbereitgestellt wird.

initcert.bat

initcert.bat legt die Namen und Pfade zu Zertifikaten und anderen Parametern fest, die von den Befehlen **keytool** und **openSSL** benötigt werden. Die Einstellungen werden in den Kommentaren direkt im Script beschrieben.

```
@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

Die Befehle im Script `cleancert.bat` löschen den Warteschlangenmanager des MQTT-Clients, um sicherzustellen, dass der Serverzertifikatsspeicher nicht gesperrt wird. Dann löschen sie alle Keystores und Zertifikate, die von den Beispielsicherheitsscripts erstellt wurden.

```
@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

Die Befehle des Scripts `genkeys.bat` erstellen Schlüsselpaare für Ihre private Zertifizierungsstelle, den Server und einen Client.

```
@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 %cltjkskey
store% -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 %srvjkskey
store% -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%  -key
store %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg %algo
rithm% -validity %validity%
```

sscerts.bat

Mit den Befehlen des Scripts `sscerts.bat` werden die selbst signierten Client- und Serverzertifikate aus den Keystores exportiert. Danach wird das Serverzertifikat in den Truststore des Clients und das Clientzertifikat in den Keystore des Servers importiert. Der Server hat keinen Truststore. Die Befehle erstellen aus dem JKS-Truststore des Clients einen Client-Truststore im PEM-Format.

```
@rem
@echo ------------------------------------------------------------------------
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore% -store
pass %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% -store
pass %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-sig
ned 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 au
thentication)
@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% -destkey
store %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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Das Script importiert das Stammzertifikat der Zertifizierungsstelle in die privaten Keystores. Das Stammzertifikat der Zertifizierungsstelle wird für die Erstellung der Schlüsselkette zwischen dem Stammzertifikat und dem signierten Zertifikat benötigt. Das Script `cacerts.bat` exportiert die Client- und die Serverzertifikatsanforderung aus deren Keystores. Das Script signiert die Zertifikatsanforderung mit dem Schlüssel der privaten Zertifizierungsstelle aus dem Keystore `cajkskeystore.jks` und importiert die signierten Zertifikate dann zurück in die Keystores, aus denen die Anforderungen gestellt wurden. Durch den Import wird die Zertifizierungskette zum Stammzertifikat der Zertifizierungsstelle erstellt. Das Script erstellt aus dem Client-JKS-Truststore einen Client-Truststore im PEM-Format.

```
@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%, %cltjkskey
store%, %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% -sto
repass %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% -sto
repass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained authenti
cation)
```

```
%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% -key
store %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% -key
store %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 %cltjkskeystore
pass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store: %cltcapemtrust
store%
@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 %cltcajkstrusts
torepass%  -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 %cltjkskeystore
pass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

Dieses Script listet die Keystores und Zertifikate im Zertifikatsverzeichnis auf. Danach erstellt es den MQTT-Beispielwarteschlangenmanager und konfiguriert die sicheren Telemetriekanäle.

```
@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%') MCAU
SER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%') MCAU
SER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%) MCAUSER(%mcau
ser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

MQTT-Clientidentifikation, Autorisierung und Authentifizierung

Der Telemetrieservice (MQXR) veröffentlicht oder subskribiert unter Verwendung von MQTT-Kanälen WebSphere MQ-Themen für MQTT-Clients. Der WebSphere MQ-Administrator konfiguriert die MQTT-Kanalidentität, die für die WebSphere MQ-Berechtigung verwendet wird. Der Administrator kann eine gemeinsame Identität für den Kanal definieren oder den Benutzernamen oder den ClientIdentifier eines Clients verwenden, der mit dem Kanal verbunden ist.

Der Telemetrieservice (MQXR) kann den Client über den Benutzernamen authentifizieren, der vom Client bereitgestellt wird, oder indem er ein Clientzertifikat verwendet. Der Benutzername wird mit einem Kennwort authentifiziert, das vom Client bereitgestellt wird.

Zusammenfassen: Die Client-ID ist die Auswahl der Cliendentität. Je nach Kontext wird der Client durch die Client-ID , Benutzername , eine vom Administrator erstellte gemeinsame Cliendentität oder ein Clientzertifikat identifiziert. Die für die Authentizitätsprüfung verwendete Client-ID muss nicht mit der ID identisch sein, die für die Autorisierung verwendet wird.

MQTT-Clientprogramme legen die Angaben für den Benutzernamen und das Kennwort fest, die an den Server unter Verwendung eines MQTT-Kanals gesendet werden. Sie können auch die SSL-Eigenschaften festlegen, die zur Verschlüsselung und Authentifizierung der Verbindung erforderlich sind. Der Administrator entscheidet, ob und wie der MQTT-Kanal authentifiziert wird.

Zur Autorisierung eines MQTT-Clients für den Zugriff auf IBM WebSphere MQ-Objekte müssen Sie die Client-ID oder den Benutzernamen des Clients oder eine allgemeine Client-ID autorisieren. Wenn Sie einem Client die Verbindung mit IBM WebSphere MQ ermöglichen möchten, berechtigen Sie den Benutzernamen oder verwenden Sie ein Clientzertifikat. Konfigurieren Sie JAAS, um den Benutzernamen zu authentifizieren, und konfigurieren Sie SSL, um ein Clientzertifikat zu authentifizieren.

Wenn Sie ein Kennwort beim Client festlegen, verschlüsseln Sie die Verbindung entweder über ein virtuelles privates Netz (VPN) oder konfigurieren Sie den MQTT-Kanal für die Verwendung von SSL, um das Kennwort nicht öffentlich zu machen.

Es ist schwierig, Clientzertifikate zu verwalten. Aus diesem Grund wird die Kennwortauthentifizierung häufig zur Authentifizierung von Clients verwendet, wenn die Risiken, die mit der Kennwortauthentifizierung verknüpft sind, akzeptabel sind.

Wenn es eine sichere Möglichkeit gibt, das Clientzertifikat zu verwalten und zu speichern, ist es möglich, sich auf die Zertifikatsauthentifizierung zu verlassen. Es ist jedoch selten der Fall, dass Zertifikate

sicher in den Typen von Umgebungen verwaltet werden können, in denen Telemetrie verwendet wird. Stattdessen wird die Authentifizierung von Einheiten, die Clientzertifikate verwenden, durch die Authentifizierung von Clientkennwörtern auf dem Server ergänzt. Aufgrund der zusätzlichen Komplexität ist die Verwendung von Clientzertifikaten auf hochsensible Anwendungen beschränkt. Die Verwendung von zwei Formen der Authentifizierung wird als Zwei-Faktor-Authentifizierung bezeichnet. Sie müssen einen der Faktoren kennen, z. B. ein Kennwort, und die anderen Faktoren, wie z. B. ein Zertifikat, haben.

Bei einer hochempfindlichen Anwendung, wie z. B. einer Chip-und-Pin-Vorrichtung, wird die Vorrichtung während der Fertigung gesperrt, um Manipulationen an der internen Hardware und Software zu verhindern. Ein anerkantes, zeitbegrenztes Clientzertifikat wird in die Einheit kopiert. Die Einheit wird an der Position implementiert, an der sie verwendet werden soll. Die weitere Authentifizierung wird jedes Mal durchgeführt, wenn die Einheit verwendet wird, entweder mit einem Kennwort oder einem anderen Zertifikat von einer Smart-Card.

MQTT-Clientidentität und Autorisierung

Verwenden Sie die `Client-ID`, den `Benutzernamen` oder eine allgemeine Clientidentität für die Autorisierung des Zugriffs auf WebSphere MQ-Objekte.

Der IBM WebSphere MQ-Administrator hat drei Möglichkeiten zur Auswahl der Identität des MQTT-Kanals. Er trifft diese Auswahl beim Definieren oder Ändern des MQTT-Kanals, der vom Client verwendet wird. Die Identität dient dazu, den Zugriff auf IBM WebSphere MQ-Themen zu berechtigen. Es sind folgende Auswahlmöglichkeiten verfügbar:

1. Die Client-ID.
2. Eine Identität, die der Administrator für den Kanal zur Verfügung stellt.
3. Der vom MQTT-Client übergebene `Benutzername`.

Der Benutzername (`Username`) ist ein Attribut der Klasse 'MqttConnectOptions'. Es muss festgelegt werden, bevor der Client eine Verbindung zum Service herstellt. Sein Standardwert lautet null.

Wählen Sie mit dem IBM WebSphere MQ-Befehl **`setmqaut`** aus, für welche Objekte und welche Aktionen die Identität berechtigt sein soll, die dem MQTT-Kanal zugeordnet ist. Geben Sie beispielsweise Folgendes ein, um die Kanalidentität `MQTTClient`, die vom Administrator des Warteschlangenmanagers `QM1` bereitgestellt wird, zu autorisieren:

```
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
```

Zugehörige Informationen

MQTT-Clients für den Zugriff auf WebSphere MQ-Objekte autorisieren

MQTT-Clientauthentifizierung mithilfe eines Kennworts

Authentifizieren Sie den `Benutzernamen` mithilfe des Clientkennworts. Sie können den Client mit einer anderen Identität gegenüber der Identität authentifizieren, die verwendet wird, um den Client zu autorisieren, Themen zu veröffentlichen und zu subskribieren.

Der Telemetrieservice (MQXR) verwendet JAAS, um den Client `benutzername` zu authentifizieren. JAAS verwendet das vom MQTT-Client übergebene `Kennwort`.

Der IBM WebSphere MQ-Administrator entscheidet, ob der `Benutzername` authentifiziert wird oder ob überhaupt keine Authentifizierung stattfindet. Hierfür konfiguriert er den MQTT-Kanal, mit dem sich ein Client verbindet. Clients können verschiedenen Kanälen zugeordnet werden, und jeder Kanal kann so konfiguriert werden, dass er seine Clients auf unterschiedliche Weise authentifiziert. Mit JAAS können Sie konfigurieren, welche Methoden den Client authentifizieren müssen und die optional den Client authentifizieren können.

Die Auswahl der Identität für die Authentifizierung hat keinen Einfluss auf die Auswahl der Identität für die Berechtigung. Es kann sein, dass Sie eine gemeinsame Identität für die Berechtigung für den Verwaltungskomfort einrichten möchten, aber jeder Benutzer authentifiziert wird, um diese Identität zu

verwenden. In der folgenden Prozedur werden die Schritte zur Authentifizierung einzelner Benutzer für die Verwendung einer gemeinsamen Identität beschrieben:

1. Der IBM WebSphere MQ -Administrator setzt die MQTT-Kanalidentität mithilfe von IBM WebSphere MQ Explorer auf einen beliebigen Namen, z. B. `MQTTClientUser`.
2. Der IBM WebSphere MQ-Administrator berechtigt `MQTTClient`, Veröffentlichungen und Subskriptionen für alle Themen durchzuführen:

```
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. Der Entwickler der MQTT-Client-App erstellt ein `MqttConnectOptions`-Objekt und legt die Angaben für `Benutzername` und `Kennwort` fest, bevor eine Verbindung zum Server hergestellt wird.
4. Der Sicherheitsentwickler erstellt ein JAAS `LoginModule` zur Authentifizierung des `Benutzernamens` mit dem `Kennwort` und schließt es in die JAAS -Konfigurationsdatei ein.
5. Der IBM WebSphere MQ-Administrator konfiguriert unter Verwendung von JAAS den MQTT-Kanal für die Authentifizierung des Benutzernamens (`UserName`) des Clients.

MQTT-Clientauthentifizierung mithilfe von SSL

Verbindungen zwischen dem MQTT-Client und dem Warteschlangenmanager werden immer vom MQTT-Client eingeleitet. Der MQTT-Client ist immer der SSL-Client. Die Clientauthentifizierung für den Server und die Serverauthentifizierung des MQTT-Clients sind beide optional.

Wenn Sie dem Client ein privates signiertes digitales Zertifikat bereitstellen, können Sie den MQTT-Client bei IBM WebSphere MQauthentifizieren. Der IBM WebSphere MQ-Administrator kann MQTT-Clients auch zwingen, sich selbst über SSL beim Warteschlangenmanager zu authentifizieren. Eine Clientauthentifizierung kann nur im Rahmen einer gegenseitigen Authentifizierung angefordert werden.

Als Alternative zu SSL können bestimmte virtuelle private Netze (Virtual Private Network, VPN), z. B. IPsec, die Endpunkte einer TCP/IP-Verbindung authentifizieren. In einem VPN wird jedes IP-Paket, das im Netz übertragen wird, verschlüsselt. Sobald eine solche VPN-Verbindung hergestellt ist, haben Sie ein vertrauenswürdiges Netz aufgebaut. MQTT-Clients können auch mit TCP/IP über das VPN mit Telemetriekanälen verbunden werden.

Die Clientauthentifizierung über SSL setzt voraus, dass der Client über einen geheimen Schlüssel verfügt. Der geheime Schlüssel ist im Falle eines selbst signierten Zertifikats der private Schlüssel des Clients bzw. andernfalls ein Schlüssel, der von einer Zertifizierungsstelle ausgestellt wird. Mit dem Schlüssel wird das digitale Zertifikat des Clients signiert. Jede Person im Besitz des entsprechenden öffentlichen Schlüssels kann das digitale Zertifikat verifizieren. Zertifikate können vertrauenswürdig sein oder - falls sie verkettet sind - durch eine Zertifikatskette bis zu einem Trusted-Root-Zertifikat zurückverfolgt werden. Bei der Clientverifizierung werden alle Zertifikate in der vom Client bereitgestellten Zertifikatskette an den Server gesendet. Der Server überprüft die Zertifikatskette, bis er ein Zertifikat findet, dem er vertraut. Das vertrauenswürdige Zertifikat ist entweder das öffentliche Zertifikat, das auf Basis eines selbst signierten Zertifikats generiert wird, oder ein Stammzertifikat, das für gewöhnlich von einer Zertifizierungsstelle ausgestellt wird. In einem letzten (optionalen) Schritt kann das vertrauenswürdige Zertifikat mit einer aktuellen Zertifikatswiderrufsliste abgeglichen werden.

Das vertrauenswürdige Zertifikat kann von einer Zertifizierungsstelle ausgegeben und bereits im JRE-Zertifikatsspeicher enthalten sein. Es kann sich aber auch um ein selbst signiertes Zertifikat oder um jedes Zertifikat handeln, das dem Keystore des Telemetriekanals als vertrauenswürdiges Zertifikat hinzugefügt wurde.

Anmerkung: Der Telemetriekanal verfügt über eine Kombination aus Keystore/Truststore, die sowohl private Schlüssel für einen oder mehrere Telemetriekanäle enthält, als auch öffentliche Zertifikate, die zur Authentifizierung von Clients erforderlich sind. Da ein SSL-Kanal einen Keystore haben muss und dies dieselbe Datei wie der Truststore des Kanals ist, wird der JRE-Zertifikatsspeicher niemals referenziert. Dies führt dazu, dass Sie das Stammzertifikat in den Keystore für den Kanal stellen müssen, wenn für eine Authentifizierung eines Clients ein Stammzertifikat einer Zertifizierungsstelle erforderlich ist. Dies gilt selbst dann, wenn sich das Stammzertifikat der Zertifizierungsstelle bereits im JRE-Zertifikatsspeicher befindet. Der JRE-Zertifikatsspeicher wird niemals referenziert.

Überlegen Sie sich, welchen Sicherheitsbedrohungen die Clientauthentifizierung entgegenwirken soll, und welche Rollen der Client und Server bei der Abwendung der Sicherheitsbedrohungen spielen. Die Authentifizierung des Clientzertifikats reicht allein nicht aus, um unbefugten Zugriff auf ein System zu verhindern. Wenn eine dritte Person über das Clientgerät verfügt, handelt das Clientgerät nicht unbedingt mit der Berechtigung des Zertifikatseigners. Verlassen Sie sich bei der Abwehr von unerwünschten Attacken niemals auf eine einzige Maßnahme. Verwenden Sie mindestens eine aus zwei Faktoren bestehende Authentifizierung und sichern Sie dies durch den Besitz eines Zertifikats ab, für das nicht öffentliche Informationen bekannt sein müssen. Verwenden Sie beispielsweise JAAS und authentifizieren Sie den Client mittels eines vom Server ausgestellten Kennworts.

Die größte Sicherheitsbedrohung für das Clientzertifikat ist, dass es in falsche Hände gelangt. Das Zertifikat befindet sich in einem kennwortgeschützten Keystore beim Client. Wie wird es in den Keystore gestellt? Wie erhält der MQTT-Client das Kennwort für den Keystore? Wie sicher ist der Kennwortschutz? Telemetriegeräte sind häufig einfach zu entfernen und können dann in Ruhe manipuliert werden. Muss das Gerät manipulationssicher sein? Die Verteilung und der Schutz clientseitiger Zertifikate gilt als besonders schwierig; diese Aufgabe wird als das Schlüsselverwaltungsproblem bezeichnet.

Eine weitere große Bedrohung besteht darin, dass das Gerät unbeabsichtigt für den Zugriff auf Server missbraucht wird. Wird die MQTT-Anwendung beispielsweise manipuliert, kann unter Verwendung der authentifizierten Clientidentität eine Schwachstelle in der Serverkonfiguration ausgenutzt werden.

Zur Authentifizierung eines MQTT-Clients mit SSL müssen Sie den Telemetriekanal und den Client konfigurieren.

-
-

MQTT-Clientkonfiguration für die Clientauthentifizierung mithilfe von SSL

Zur Authentifizierung des MQTT-Clients mithilfe von SSL verbindet sich der Client über SSL mit einem Telemetriekanal. Er muss einen TCP-Port angeben, der einem Telemetriekanal entspricht, welcher für die Authentifizierung von SSL-Clients konfiguriert wurde.

Beispiel beim Client:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

Die Java Virtual Machine des Clients muss die standardmäßige Socket-Factory aus Java Secure Socket Extension (JSEE) verwenden. Wenn Sie Java ME verwenden, müssen Sie sicherstellen, dass das JSSE-Paket geladen ist. Wenn Sie Java SE verwenden, ist JSSE seit Java Version 1.4.1in der JRE enthalten.

Vor der Herstellung einer SSL-Verbindung müssen einige SSL-Eigenschaften festgelegt werden. Sie können die Eigenschaften entweder festlegen, indem Sie sie mit dem Switch `-D` an die JVM übergeben, oder Sie können die Eigenschaften mit der Methode `MqttConnectionOptions.setSSLProperties` definieren.

Wenn Sie eine vom Standard abweichende Socket-Factory laden, indem Sie die Methode `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)` aufrufen, wird die Art und Weise, wie SSL-Einstellungen an das Netzsocket übergeben werden, anwendungsdefiniert.

Fügen Sie das digitale Zertifikat des Clients, das entweder unter Verwendung des privaten Schlüssels des Clients oder durch eine Zertifizierungsstelle signiert wurde, zum kennwortgeschützten Keystore auf dem Client hinzu. Wenn das Zertifikat über eine Schlüsselkette verfügt, können Sie die Zertifikate aus der Schlüsselkette zum Store hinzufügen. Wenn der Server das Clientzertifikat verifiziert, verwendet er die vom Client gesendeten Zertifikate für den Abgleich mit den Zertifikaten in seinem Keystore. Er sucht nach der ersten Übereinstimmung in der Schlüsselkette mit einem Zertifikat, das ihm vorliegt. Die übrigen Glieder der Schlüsselkette werden ignoriert.

Der MQTT-Client sendet alle Zertifikate in seinem Keystore an den Server. Wenn der Server eine der vom Client gesendeten Schlüsselketten authentifiziert, wird der Client authentifiziert.

Sie können für die Clientauthentifizierung auch SSL-Cipher-Suites verwenden.Es folgt eine alphabetische Liste der SSL Cipher-Suites, die momentan unterstützt werden:

- 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

V 7.5.0.2 Wenn Sie beabsichtigen, SHA-2-Cipher-Suites zu verwenden, lesen Sie die Informationen unter „Systemvoraussetzungen für Verwendung von SHA-2-CipherSuites mit MQTT-Clients” auf Seite 180.

Zugehörige Konzepte

„MQTT-Clientkonfiguration für die Kanalauthentifizierung mithilfe von SSL” auf Seite 111
Zur Authentifizierung des Telemetriekanals mithilfe von SSL muss sich der Client über SSL mit dem Telemetriekanal verbinden. Er muss einen TCP-Port angeben, der einem Telemetriekanal entspricht, welcher für SSL konfiguriert wurde. Die Konfiguration muss einen Keystore umfassen, der durch eine Kennphrase geschützt ist und das privat signierte digitale Zertifikat des Servers enthält.

Authentifizierung des Telemetriekanals über SSL

Verbindungen zwischen dem MQTT-Client und dem Warteschlangenmanager werden immer vom MQTT-Client eingeleitet. Der MQTT-Client ist immer der SSL-Client. Die Clientauthentifizierung für den Server und die Serverauthentifizierung des MQTT-Clients sind beide optional.

Der Client versucht immer, den Server zu authentifizieren, es sei denn, der Client wurde für die Verwendung einer CipherSpec konfiguriert, die anonyme Verbindungen unterstützt. Schlägt die Authentifizierung fehl, wird keine Verbindung hergestellt.

Als Alternative zu SSL können bestimmte virtuelle private Netze (Virtual Private Network, VPN), z. B. IPsec, die Endpunkte einer TCP/IP-Verbindung authentifizieren. In einem VPN wird jedes IP-Paket, das im Netz übertragen wird, verschlüsselt. Sobald eine solche VPN-Verbindung hergestellt ist, haben Sie ein vertrauenswürdiges Netz aufgebaut. MQTT-Clients können auch mit TCP/IP über das VPN mit Telemetriekanälen verbunden werden.

Bei der Serverauthentifizierung mit SSL wird der Server, an den Sie vertrauliche Informationen senden möchten, mithilfe von Secure Sockets Layer (SSL) authentifiziert. Der Client führt die Prüfungen durch, die mit den vom Server gesendeten Zertifikaten übereinstimmen, mit Zertifikaten, die in seinem Truststore oder in seinem JRE- `cacerts` Speicher gespeichert sind.

Der JRE-Zertifikatsspeicher ist eine JKS-Datei, `cacerts`. Sie befindet sich im Pfad `JRE Install-Path\lib\security\`. Sie wird mit dem Standardkennwort `changeit` installiert. Sie können Zertifikate, denen Sie vertrauen, entweder im JRE-Zertifikatsspeicher oder im Truststore des Clients speichern. Sie können jedoch nicht beide Speicher verwenden. Verwenden Sie den Truststore des Clients, wenn Sie die öffentlichen Zertifikate, denen der Client vertraut, getrennt von den Zertifikaten aufbewahren möchten, die von anderen Java-Anwendungen verwendet werden. Verwenden Sie den JRE-Zertifikatsspeicher, wenn Sie einen allgemeinen Zertifikatsspeicher für alle Java-Anwendungen nutzen möchten, die auf dem Client ausgeführt werden. Wenn Sie sich für die Verwendung des JRE-Zertifikatsspeichers entscheiden, überprüfen Sie die darin enthaltenen Zertifikate, um sicherzustellen, dass diese wirklich vertrauenswürdig sind.

Sie können die JSSE-Konfiguration ändern, indem Sie einen anderen Trust-Provider bereitstellen. Ein Trust-Provider kann so angepasst werden, dass er verschiedene Zertifikatsprüfungen durchführt. In einigen OGSi-Umgebungen, die den MQTT-Client verwendet haben, stellt die Umgebung einen unterschiedlichen Trust-Provider bereit.

Zur Authentifizierung eines Telemetriekanals mit SSL müssen Sie den Server und den Client konfigurieren.

•

Zugehörige Konzepte

„MQTT-Clientkonfiguration für die Kanalauthentifizierung mithilfe von SSL” auf Seite 111
Zur Authentifizierung des Telemetriekanals mithilfe von SSL muss sich der Client über SSL mit dem Telemetriekanal verbinden. Er muss einen TCP-Port angeben, der einem Telemetriekanal entspricht, welcher für SSL konfiguriert wurde. Die Konfiguration muss einen Keystore umfassen, der durch eine Kennphrase geschützt ist und das privat signierte digitale Zertifikat des Servers enthält.

MQTT-Clientkonfiguration für die Kanalauthentifizierung mithilfe von SSL

Zur Authentifizierung des Telemetriekanals mithilfe von SSL muss sich der Client über SSL mit dem Telemetriekanal verbinden. Er muss einen TCP-Port angeben, der einem Telemetriekanal entspricht, welcher für SSL konfiguriert wurde. Die Konfiguration muss einen Keystore umfassen, der durch eine Kennphrase geschützt ist und das privat signierte digitale Zertifikat des Servers enthält.

Beispiel beim Client:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

Die Java Virtual Machine des Clients muss die standardmäßige Socket-Factory aus Java Secure Socket Extension (JSEE) verwenden. Wenn Sie Java ME verwenden, müssen Sie sicherstellen, dass das JSSE-Paket geladen ist. Wenn Sie Java SE verwenden, ist JSSE seit Java Version 1.4.1in der JRE enthalten.

Vor der Herstellung einer SSL-Verbindung müssen einige SSL-Eigenschaften festgelegt werden. Sie können die Eigenschaften entweder festlegen, indem Sie sie mit dem Switch `-D` an die JVM übergeben, oder Sie können die Eigenschaften mit der Methode `MqttConnectionOptions.setSSLProperties` definieren.

Wenn Sie eine vom Standard abweichende Socket-Factory laden, indem Sie die Methode `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)` aufrufen, wird die Art und Weise, wie SSL-Einstellungen an das Netzsocket übergeben werden, anwendungsdefiniert.

Codieren Sie den Client für die Verbindung mit dem Telemetriekanal unter Verwendung von SSL und konfigurieren Sie ihn auf eine der drei folgenden Arten, einem Serverzertifikat zu vertrauen:

Verwendung eines Serverzertifikats, das von einer anerkannten Zertifizierungsstelle signiert wurde und sich im Speicher `cacerts` befindet.

Keine weitere Konfiguration, wenn der Server alle temporären Schlüssel in der Zertifikatskette sendet. Es wird empfohlen, die Zertifikate im Speicher `cacerts` der Client-JRE zu prüfen und das Kennwort für den Speicher `cacerts` zu ändern.

Sonstige Zertifikate

Speichern Sie die Zertifikate, denen Sie vertrauen, im Truststore beim Client. Sie müssen mindestens eines der Zertifikate in der Zertifikatskette im Truststore speichern. Legen Sie die Truststore-Parameter in `MqttConnectionOptions.SSLProperty` fest.

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

Verwendung eines angepassten Trust-Managers

Implementieren Sie einen Trust-Provider und übergeben Sie ihm den Namen des verwendeten Algorithmus. Legen Sie den Namen der Providerklasse und den zu verwendenden Algorithmus in `MqttConnectionOptions.SSLProperty` fest.

- `com.ibm.ssl.trustStoreProvider`
- `com.ibm.ssl.trustStoreManager`

Alternativ können für die Kanalauthentifizierung auch SSL Cipher-Suites verwendet werden.Es folgt eine alphabetische Liste der SSL Cipher-Suites, die momentan unterstützt werden:

- `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 Wenn Sie beabsichtigen, SHA-2-Cipher-Suites zu verwenden, lesen Sie die Informationen unter „Systemvoraussetzungen für Verwendung von SHA-2-CipherSuites mit MQTT-Clients” auf Seite 180.

Zugehörige Konzepte

„MQTT-Clientkonfiguration für die Clientauthentifizierung mithilfe von SSL” auf Seite 108
Zur Authentifizierung des MQTT-Clients mithilfe von SSL verbindet sich der Client über SSL mit einem Telemetriekanal. Er muss einen TCP-Port angeben, der einem Telemetriekanal entspricht, welcher für die Authentifizierung von SSL-Clients konfiguriert wurde.

Datenschutz auf Telemetriekanälen

Die Vertraulichkeit von MQTT-Veröffentlichungen, die in beiden Richtungen über Telemetriekanäle gesendet werden, wird dadurch sichergestellt, dass Übertragungen über die Verbindung mit SSL verschlüsselt werden.

MQTT-Clients, die Verbindungen mit Telemetriekanälen herstellen, verwenden SSL, um die Vertraulichkeit von Veröffentlichungen, die über die Kanäle übertragen werden, durch eine symmetrische Verschlüsselung sicherzustellen. Da die Endpunkte nicht authentifiziert sind, können Sie die Kanalverschlüsselung nicht allein vertrauen. Kombinieren Sie den Schutz der Privatsphäre mit dem Server oder der gegenseitigen Authentifizierung.

Als Alternative zu SSL können bestimmte virtuelle private Netze (Virtual Private Network, VPN), z. B. IPsec, die Endpunkte einer TCP/IP-Verbindung authentifizieren. In einem VPN wird jedes IP-Paket, das im Netz übertragen wird, verschlüsselt. Sobald eine solche VPN-Verbindung hergestellt ist, haben Sie ein vertrauenswürdiges Netz aufgebaut. MQTT-Clients können auch mit TCP/IP über das VPN mit Telemetriekanälen verbunden werden.

Informationen zu einer typischen Konfiguration mit Kanalverschlüsselung und Serverauthentifizierung finden Sie im Abschnitt „Authentifizierung des Telemetriekanals über SSL” auf Seite 110.

Wird eine SSL-Verbindung ohne Authentifizierung des Servers verschlüsselt, ist die Verbindung für Man-in-the-Middle-Attacken anfällig. Obwohl die Informationen, die Sie austauschen, vor Lauschangriffen geschützt sind, wissen Sie nicht, mit wem Sie sie austauschen. Wenn Sie das Netz nicht steuern, sind Sie einem Benutzer ausgesetzt, der Ihre IP-Übertragungen abfängt und als Endpunkt maskiert wird.

Sie können eine verschlüsselte SSL-Verbindung aufbauen (ohne Authentifizierung des Servers), indem Sie eine Diffie-Hellman-Schlüsselaustausch-CipherSpec, die anonymes SSL unterstützt, einsetzen. Der geheime Master-Schlüssel, den Client und Server gemeinsam nutzen und mit dem SSL-Übertragungen verschlüsselt werden, wird erstellt, ohne dass ein privat signiertes Serverzertifikat ausgetauscht wird.

Da anonyme Verbindungen unsicher sind, verwenden SSL-Implementierungen standardmäßig keine anonymen CipherSpecs. Wenn eine Clientanforderung für eine SSL-Verbindung von einem Telemetriekanal akzeptiert wird, muss der Kanal über einen durch eine Kennphrase geschützten Schlüsselspeicher verfügen. Da SSL-Implementierungen keine anonymen CipherSpecs verwenden, muss der Schlüsselspeicher standardmäßig ein privat signiertes Zertifikat enthalten, das der Client authentifizieren kann.

Wenn Sie anonyme CipherSpecs verwenden, muss der Server-Keystore vorhanden sein, aber er muss keine privat signierten Zertifikate enthalten.

Eine weitere Möglichkeit, eine verschlüsselte Verbindung herzustellen, besteht darin, den Trust-Provider auf dem Client durch Ihre eigene Implementierung zu ersetzen. Ihr Trust-Provider würde das Serverzertifikat nicht authentifizieren, aber die Verbindung würde verschlüsselt sein.

SSL-Konfiguration der MQTT-Clients und Telemetriekanäle

MQTT-Clients und der WebSphere MQ Telemetry-Service (MQXR) verwenden JSSE (Java Secure Socket Extension), um Telemetriekanäle über SSL zu verbinden. MQTT-Clients in der Programmiersprache C und der WebSphere MQ Telemetry-Dämon für Geräte unterstützen kein SSL.

Konfigurieren Sie SSL für die Authentifizierung des Telemetriekanals und des MQTT-Clients sowie die Verschlüsselung der Übertragung von Nachrichten zwischen Clients und dem Telemetriekanal.

Als Alternative zu SSL können bestimmte virtuelle private Netze (Virtual Private Network, VPN), z. B. IPsec, die Endpunkte einer TCP/IP-Verbindung authentifizieren. In einem VPN wird jedes IP-Paket, das im Netz übertragen wird, verschlüsselt. Sobald eine solche VPN-Verbindung hergestellt ist, haben Sie ein vertrauenswürdiges Netz aufgebaut. MQTT-Clients können auch mit TCP/IP über das VPN mit Telemetriekanälen verbunden werden.

Sie können die Verbindung zwischen einem Java-MQTT-Client und einem Telemetriekanal für die Verwendung des SSL-Protokolls über TCP/IP konfigurieren. Was gesichert wird, hängt davon ab, wie Sie SSL für die Verwendung von JSSE konfigurieren. Beginnend mit der sichersten Konfiguration können Sie drei verschiedene Sicherheitsstufen konfigurieren:

1. Erlauben Sie nur vertrauenswürdigen MQTT-Clients die Herstellung einer Verbindung. Verbinden Sie einen MQTT-Client nur mit einem vertrauenswürdigen Telemetriekanal. Verschlüsseln Sie Nachrichten zwischen dem Client und dem Warteschlangenmanager; siehe „MQTT-Clientauthentifizierung mithilfe von SSL" auf Seite 107.
2. Verbinden Sie einen MQTT-Client nur mit einem vertrauenswürdigen Telemetriekanal. Verschlüsseln Sie Nachrichten zwischen dem Client und dem Warteschlangenmanager; siehe „Authentifizierung des Telemetriekanals über SSL" auf Seite 110.
3. Verschlüsseln Sie Nachrichten zwischen dem Client und dem Warteschlangenmanager; siehe „Datenschutz auf Telemetriekanälen" auf Seite 113.

JSSE-Konfigurationsparameter

Durch die Modifizierung der JSSE-Parameter können Sie die Art ändern, wie eine SSL-Verbindung konfiguriert ist. Die JSSE-Konfigurationsparameter sind in drei Gruppen angeordnet:

1. IBM WebSphere MQ Telemetriekanal
2. MQTT-Java-Client
3. JRE

Konfigurieren Sie die Telemetriekanalparameter mithilfe von IBM WebSphere MQ Explorer. Legen Sie die MQTT-Java-Clientparameter im Attribut `MqttConnectionOptions.SSLProperties` fest. Ändern Sie die JRE-Sicherheitsparameter, indem Sie Dateien im JRE-Sicherheitsverzeichnis sowohl auf dem Client als auch auf dem Server bearbeiten.

IBM WebSphere MQ Telemetry-Kanal

Legen Sie alle SSL-Parameter des Telemetriekanals mithilfe von WebSphere MQ Explorer fest.

`ChannelName`

`ChannelName` ist ein erforderlicher Parameter auf allen Kanälen.

Der Kanalname identifiziert den Kanal, der einer bestimmten Port-Nummer zugeordnet ist. Geben Sie den Kanälen einen Namen, um die Verwaltung von MQTT-Clientgruppen zu vereinfachen.

`PortNumber`

`PortNumber` ist ein optionaler Parameter auf allen Kanälen. Bei TCP-Kanälen hat er standardmäßig den Wert 1883, bei SSL-Kanälen den Wert 8883.

Die TCP/IP-Anschlussnummer, die diesem Kanal zugeordnet ist. MQTT-Clients werden durch die Angabe des Ports verbunden, der für den Kanal definiert ist. Wenn der Kanal über SSL-Eigenschaften verfügt, muss sich der Client unter Verwendung des SSL-Protokolls verbinden. Beispiel:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

`KeyFileName`

`KeyFileName` ist ein erforderlicher Parameter für SSL-Kanäle. Er muss für TCP-Kanäle weggelassen werden.

`KeyFileName` ist der Pfad zum Java-Keystore mit den digitalen Zertifikaten, die Sie angeben. Verwenden Sie JKS, JCEKS oder PKCS12 als den Typ des Schlüsselspeichers auf dem Server.

Geben Sie den Keystore-Typ mithilfe einer der folgenden Dateierweiterungen an:

```
.jks
.jceks
.p12
.pkcs12
```

Es wird angenommen, dass ein Schlüsselspeicher mit einer anderen Dateierweiterung ein JKS-Keystore ist.

Sie können einen Typ von Schlüsselspeicher auf dem Server mit anderen Keystoretypen auf dem Client kombinieren.

Speichern Sie das private Zertifikat des Servers in den Schlüsselspeicher. Das Zertifikat wird als Serverzertifikat bezeichnet. Das Zertifikat kann selbst signiert sein oder Teil einer Zertifikatskette sein, die von einer Signaturberechtigung signiert ist.

Wenn Sie eine Zertifikatskette verwenden, legen Sie die zugeordneten Zertifikate in den Serverschlüsselspeicher.

Das Serverzertifikat und alle Zertifikate in seiner Zertifikatskette werden an Clients gesendet, um die Identität des Servers zu authentifizieren.

Wenn Sie `ClientAuth` auf `Required` gesetzt haben, muss der Keystore alle Zertifikate enthalten, die für die Authentifizierung des Clients erforderlich sind. Der Client sendet ein selbst signiertes Zertifikat oder eine Zertifikatskette, und der Client wird durch die erste Überprüfung dieses Materials anhand eines Zertifikats im Keystore authentifiziert. Mit Hilfe einer Zertifikatskette kann ein Zertifikat viele Clients prüfen, auch wenn sie mit unterschiedlichen Clientzertifikaten ausgegeben werden.

PassPhrase

`PassPhrase` ist ein erforderlicher Parameter für SSL-Kanäle. Er muss für TCP-Kanäle weggelassen werden.

Der Verschlüsselungstext wird zum Schutz des Keystores verwendet.

ClientAuth

`ClientAuth` ist ein optionaler SSL-Parameter. Der Standardwert ist keine Clientauthentifizierung. Er muss für TCP-Kanäle weggelassen werden.

Legen Sie `ClientAuth` fest, wenn der Telemetrieservice (MQXR) den Client authentifizieren soll, bevor er dem Client die Verbindung zum Telemetriekanal ermöglicht.

Wenn Sie `ClientAuth` festlegen, muss sich der Client unter Verwendung von SSL mit dem Server verbinden und den Server authentifizieren. Als Antwort auf die Einstellung von `ClientAuth` sendet der Client sein digitales Zertifikat an den Server und alle anderen Zertifikate in seinem Keystore. Das digitale Zertifikat wird als Clientzertifikat bezeichnet. Diese Zertifikate werden für Zertifikate authentifiziert, die im Channel-Keystore und im JRE- `cacerts` speicher gespeichert sind.

CipherSuite

`CipherSuite` ist ein optionaler SSL-Parameter. Standardmäßig werden alle aktivierten CipherSpecs probiert. Er muss für TCP-Kanäle weggelassen werden.

Wenn Sie eine bestimmte CipherSpec verwenden möchten, setzen Sie `CipherSuite` auf den Namen der CipherSpec, die für die Herstellung der SSL-Verbindung verwendet werden muss.

Der Telemetrieservice und der MQTT-Client handeln eine gemeinsame CipherSpec aus, die aus allen CipherSpecs gewählt wird, die auf jeder Seite aktiviert sind. Wenn eine bestimmte CipherSpec an einem der beiden oder beiden Enden der Verbindung angegeben ist, muss sie mit der CipherSpec am anderen Ende übereinstimmen.

Installieren Sie zusätzliche Chiffriergeräte, indem Sie JSSE zusätzliche Provider hinzufügen.

Federal Information Processing Standards (FIPS)

FIPS ist eine optionale Einstellung. Standardmäßig ist sie nicht festgelegt.

Legen Sie entweder in der Eigenschaftenanzeige des Warteschlangenmanagers oder mithilfe von **runmqsc** `SSLFIPS`fest. `SSLFIPS` gibt an, ob nur FIPS-zertifizierte Algorithmen verwendet werden sollen.

Revocation namelist

Die Widerrufsnamensliste ist eine optionale Einstellung. Standardmäßig ist sie nicht festgelegt.

Legen Sie SSLCRLNLin der Eigenschaftenanzeige des Warteschlangenmanagers oder mithilfe von **runmqsc**fest. SSLCRLNL gibt eine Namensliste mit Authentifizierungsinformationsobjekten an, die zum Angeben von Zertifikatswiderrufspositionen verwendet werden.

Es werden keine anderen Warteschlangenmanager-Parameter verwendet, die SSL-Eigenschaften festlegen.

MQTT-Java-Client

Legen Sie SSL-Eigenschaften für den Java-Client in `MqttConnectionOptions.SSLPropertiesfest`. Beispiel:

```
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);
```

Die Namen und Werte der jeweiligen Eigenschaften werden in der API-Dokumentation für `MqttConnectOptions` beschrieben.Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

`Protocol`

Das `Protokoll` ist optional.

Das Protokoll wird in den Verhandlungen mit dem Telemetrieserver ausgewählt. Wenn Sie ein bestimmtes Protokoll benötigen, können Sie eine auswählen. Wenn der Telemetrieserver das Protokoll nicht unterstützt, schlägt die Verbindung fehl.

`ContextProvider`

`ContextProvider` ist optional.

`KeyStore`

`KeyStore` ist optional. Konfigurieren Sie sie, wenn `ClientAuth` auf dem Server festgelegt ist, um die Authentifizierung des Clients zu erzwingen.

Speichern Sie das digitale Zertifikat des Clients, signiert mit seinem privaten Schlüssel, in den Schlüsselspeicher. Geben Sie den Schlüsselspeicherpfad und das Kennwort an. Der Typ und der Provider sind optional. JKS ist der Standardtyp, und IBMJCE ist der Standardprovider.

Geben Sie einen anderen Schlüsselspeicherprovider an, um auf eine Klasse zu verweisen, die einen neuen Keystore-Provider hinzufügt Übergeben Sie den Namen des Algorithmus, der vom Schlüsselspeicherprovider für die Instanziierung von `KeyManagerFactory` verwendet wird, indem Sie den Namen des Schlüsselmanagers festlegen.

`TrustStore`

`TrustStore` ist optional. Sie können alle Zertifikate, auf die Sie vertrauen, in den JRE- `cacerts` speicher stellen.

Konfigurieren Sie den Truststore, wenn Sie einen anderen Truststore für den Client verwenden möchten. Sie können den Truststore möglicherweise nicht konfigurieren, wenn der Server ein Zertifikat verwendet, das von einer bekannten Zertifizierungsinstanz ausgestellt wurde, die bereits sein Stammzertifikat in `cacerts` gespeichert hat.

Fügen Sie das öffentlich signierte Zertifikat des Servers oder des Stammzertifikats zum Truststore hinzu, und geben Sie den Truststore-Pfad und das Kennwort an. JKS ist der Standardtyp, und IBMJCE ist der Standardprovider.

Geben Sie einen anderen Truststore-Provider an, um auf eine Klasse zu verweisen, die einen neuen Truststore-Provider hinzufügt. Übergeben Sie den Namen des Algorithmus, der vom Truststore-Provider für die Instanziierung der `TrustManagerFactory` verwendet wird, indem Sie den Namen des Trust-Managers festlegen.

JRE

Sonstige Aspekte der Java-Sicherheit, die sich auf das SSL-Verhalten auf dem Client und dem Server auswirken, werden in der Java Runtime Environment (JRE) konfiguriert. Die Konfigurationsdateien unter Windows befinden sich im Verzeichnis *Java Installation Directory*\jre\lib\security. Wenn Sie die JRE verwenden, die mit IBM WebSphere MQ ausgeliefert wird, lautet der Pfad wie in der folgenden Tabelle angegeben:

Tabelle 3. Dateipfade nach Plattform für JRE-SSL-Konfigurationsdateien

Plattform	**Dateipfad**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux for System x 32 Bit	*WMQ Installation Directory*/java/jre/lib/security
Sonstige UNIX and Linux-Plattformen	*WMQ Installation Directory*/java/jre64/jre/lib/security

Well-bekannte Zertifizierungsstellen

Die Datei `cacerts` enthält die Stammzertifikate anerkanster Zertifizierungsstellen. Der `cacerts` wird standardmäßig verwendet, es sei denn, Sie geben einen Truststore an. Wenn Sie den `cacerts` -Store verwenden oder keinen Truststore bereitstellen, müssen Sie die Liste der Unterzeichner in `cacerts` überprüfen und bearbeiten, um Ihre Sicherheitsanforderungen zu erfüllen.

Sie können `cacerts` mit dem WebSphere MQ-Befehl `strmqikm` öffnen. Dieser Befehl führt das Dienstprogramm IBM Key Management aus. Öffnen Sie `cacerts` als JKS-Datei, indem Sie das Kennwort `changeit` verwenden. Ändern Sie das Kennwort, um die Datei zu sichern.

Sicherheitsklassen konfigurieren

Verwenden Sie die Datei `java.security` , um zusätzliche Sicherheitsprovider und andere Standardsicherheitseigenschaften zu registrieren.

Zulassungs-
Verwenden Sie die Datei `java.policy` , um die Berechtigungen zu ändern, die für Ressourcen erteilt wurden. `javaws.policy` erteilt Berechtigungen für `javaws.jar`.

Verschlüsselungsstärke
Einige JREs werden mit reduzierter Verschlüsselungsstärke ausgeliefert. Wenn Sie keine Schlüssel in Keystores importieren können, kann die Verschlüsselung mit verminderter Stärke die Ursache sein. Versuchen Sie entweder, **ikeyman** mit dem Befehl **strmqikm** zu starten, oder laden Sie starke, aber eingeschränkte Jurisdiction-Dateien von der Website IBM developer kits, Security information herunter.

Wichtig: Ihr Herkunftsland hat möglicherweise Einschränkungen für den Import, den Besitz, die Verwendung oder den erneuten Export in ein anderes Land der Verschlüsselungssoftware zur Verfügung. Bevor Sie die unbeschränkten Richtliniendateien herunterladen oder verwenden, müssen Sie die Gesetze Ihres Landes überprüfen. Überprüfen Sie die Bestimmungen und die Richtlinien für den Import, den Besitz, die Verwendung und den Wiederexport von Verschlüsselungssoftware, um festzustellen, ob die Software zugelassen ist.

Ändern Sie den Trust-Provider, damit der Client eine Verbindung zu einem beliebigen Server herstellen kann.

Das Beispiel veranschaulicht, wie Sie einen Trust-Provider hinzufügen und vom MQTT-Client-Code aus referenzieren können. Das Beispiel führt keine Authentifizierung für den Client oder den Server durch. Die hergestellte SSL-Verbindung ist verschlüsselt, aber nicht authentifiziert.

In dem Codeausschnitt in Abbildung 20 auf Seite 118 werden der Trust-Provider `AcceptAllProviders` und der Trust-Manager für den MQTT-Client festgelegt.

```
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);
```

Abbildung 20. Codeausschnitt des MQTT-Clients

```
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());
    }
```

Abbildung 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() };
        }
```

Abbildung 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);
      }
    }
```

Abbildung 23. AcceptAllX509TrustManager.java

JAAS-Konfiguration für Telemetriekanal

Konfigurieren Sie JAAS für die Authentifizierung des vom Client gesendeten Benutzernamens .

Der WebSphere MQ-Administrator konfiguriert unter Verwendung von JAAS, welche MQTT-Kanäle eine Clientauthentifizierung erfordern. Geben Sie den Namen einer JAAS-Konfiguration für jeden Kanal an, der die JAAS-Authentifizierung ausführen soll. Kanäle können alle dieselbe JAAS-Konfiguration verwenden, oder sie können verschiedene JAAS-Konfigurationen verwenden. Die Konfigurationen sind in *WMQData directory*\qmgrs*qMgrName*\mqxr\jaas.configdefiniert.

Die Datei `jaas.config` ist nach dem JAAS-Konfigurationsnamen organisiert. Unter jedem Konfigurationsnamen befindet sich eine Liste mit Anmeldekonfigurationen; siehe Abbildung 24 auf Seite 119.

JAAS stellt vier Standardanmeldemodule zur Verfügung. Die NT- und UNIX-Standardanmeldemodule sind begrenzt.

JndiLoginModule
Die Authentifizierung erfolgt auf Basis eines Verzeichnisservice, der in der Java Naming and Directory Interface (JNDI) konfiguriert ist.

Krb5LoginModule
Authentifiziert die Verwendung von Kerberos-Protokollen.

NTLoginModule
Authentifiziert unter Verwendung der NT-Sicherheitsinformationen für den aktuellen Benutzer.

UnixLoginModule
Die Authentifizierung erfolgt über die UNIX-Sicherheitsinformationen für den aktuellen Benutzer.

Bei `NTLoginModule` und `UnixLoginModule` besteht das Problem, dass der Telemetrieservice (MQXR) nicht mit der Identität des MQTT-Kanals, sondern mit der Identität `mqm` ausgeführt wird. `mqm` ist die Identität, die für die Authentifizierung an `NTLoginModule` oder `UnixLoginModule` übergeben wird, und nicht die Identität des Clients.

Um dieses Problem zu beheben, müssen Sie Ihr eigenes Anmeldemodul schreiben oder die anderen Standardanmeldemodule verwenden. Mit WebSphere MQ Telemetry wird das Musteranmeldemodul `JAASLoginModule.java` ausgeliefert. Es handelt sich um eine Implementierung der `javax.security.auth.spi.LoginModule` -Schnittstelle. Verwenden Sie diese Option, um Ihre eigene Authentifizierungsmethode zu entwickeln.

Alle neuen LoginModule-Klassen, die Sie bereitstellen, müssen sich auf dem Klassenpfad des Telemetrieservice (MQXR) befinden. Stellen Sie Ihre Klassen nicht in WebSphere MQ-Verzeichnisse, die sich im Klassenpfad befinden. Erstellen Sie Ihre eigenen Verzeichnisse und definieren Sie den gesamten Klassenpfad für den Telemetrieservice (MQXR).

Sie können den Klassenpfad, der vom Telemetrieservice (MQXR) verwendet wird, erweitern, indem Sie den Klassenpfad in der Datei `service.env` definieren. CLASSPATH muss aktiviert sein, und die Klassenpfadanweisung kann nur Literale enthalten. Sie können keine Variablen in CLASSPATH verwenden, z. B. CLASSPATH=%CLASSPATH% ist nicht korrekt. Der Telemetrieservice (MQXR) legt seinen eigenen Klassenpfad fest. Der in `service.env` definierte CLASSPATH wird hinzugefügt.

Der Telemetrieservice (MQXR) stellt zwei Callbacks zur Verfügung, die Werte für `Benutzername` und `Kennwort` für einen Client liefern, der mit dem MQTT-Kanal verbunden ist. Der `Benutzername` und das `Kennwort` werden im Objekt `MqttConnectOptions` festgelegt. Im Abschnitt Abbildung 25 auf Seite 120 finden Sie ein Beispiel dafür, wie auf den `Benutzernamen` und das `Kennwort` zugegriffen wird.

Beispiele

Ein Beispiel für eine JAAS-Konfigurationsdatei mit einer benannten Konfiguration, `MQXRConfig`.

```
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";
};
```

Abbildung 24. Beispieldatei `jaas.config`

Dies ist ein Beispiel eines JAAS-Anmeldemoduls, das für den Empfang der Angaben von `Benutzername` und `Kennwort`, die von einem MQTT-Client zur Verfügung gestellt werden, codiert ist.

```
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;
}
```

Abbildung 25. JAASLoginModule.Login() -Beispielmethode

Konzepte für die Clientprogrammierung

Die in diesem Abschnitt beschriebenen Konzepte verdeutlichen den Java-Client für die Version 3.1 von MQTT protocol. Die Konzepte ergänzen die API-Dokumentation, die dem Paket `com.ibm.micro.client.mqttv3` beigefügt ist.

`com.ibm.micro.client.mqttv3` enthält die Klassen, die die öffentlichen Methoden für die Java -Implementierungen des Protokolls MQTT Version 3.1 bereitstellen. Das Paket `com.ibm.micro.client.mqttv3` und die zugehörigen Pakete, die das Protokoll für Java SE und ME implementieren, werden mit der Installation von IBM WebSphere MQ Telemetrybereitgestellt.

Zum Entwickeln und Ausführen eines MQTT-Clients müssen Sie diese Pakete auf die Clienteinheit kopieren oder dort installieren. Das Installieren einer separaten Clientlaufzeit ist nicht erforderlich.

Die Lizenzbedingungen für Clients sind dem Server zugeordnet, mit dem die Clients verbunden werden.

Der Java-Client ist eine Referenzimplementierung der Version 3.1 des MQTT protocol. Sie können Ihre eigenen Clients in unterschiedlichen Sprachen implementieren, die für unterschiedliche Einheitenplattformen geeignet sind. Ausführliche Informationen hierzu finden Sie unter MQ Telemetry Transport-Format und -Protokoll.

Die Client-API-Dokumentation für das Paket `com.ibm.micro.client.mqttv3` setzt keinen bestimmten Server voraus, mit dem der Client verbunden ist. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference. Das Verhalten des Clients kann je nachdem, zu welchem Server eine Verbindung hergestellt wird, geringfügig abweichen. In den nachfolgenden Beschreibungen wird das Verhalten eines Clients erläutert, der mit dem IBM WebSphere MQ-Telemetrieservice (MQXR) verbunden ist.

MQTT-Messaging-Client für JavaScript und Web-Apps

Bis vor kurzem waren die Programmierung von Web-Apps und die Erstellung von Messaging-Apps zwei getrennte Bereiche. Egal, in welchem Bereich Ihre bisherige Erfahrung liegt - es gibt bedeutende Vorteile bei der gemeinsamen Verwendung von JavaScript und Messaging. Wenn Sie Ihre Messaging-App als Web-App codieren, kann sie in jeden aktuellen Browser übernommen und dort ausgeführt werden. Wenn Sie

die App ändern, wird die neueste Version verwendet, sobald der Browser aktualisiert wird. Der Browser berücksichtigt auch Sicherheitsaspekte sowie die zuverlässige Übertragung von Nachrichten.

Wie die Verwendung einer Web-App die Anwendungsimplementierung erleichtert

Wenn Sie über Erfahrung bei der Entwicklung und Implementierung traditioneller Messaging-Apps auf (beispielsweise) IBM WebSphere MQ verfügen, ist Ihnen unter Umständen der folgende Bereitstellungsprozess vertraut:

1. Der Systemadministrator installiert die Clientbibliothek oder bettet sie ein.
2. Der Systemadministrator sorgt dafür, dass die Messaging-App an die Endbenutzer verteilt und auf deren lokalen Systemen installiert wird.
3. Wenn sich der Code ändert, wiederholt der Systemadministrator die vorherigen Schritte (daher ist das Änderungsmanagement komplex).

Wenn Sie Ihre Messaging-App als Web-App codieren, verläuft der Bereitstellungsprozess wie folgt:

1. Der Systemadministrator stellt die Web-App und die Clientbibliothek in einer URL bereit.
2. Der Browser des Endbenutzers übernimmt die Web-App und die Clientbibliothek zusammen.
3. Wenn sich der Code ändert, wird die aktualisierte Version übernommen, wenn der Browser aktualisiert wird (daher ist das Änderungsmanagement einfach).

Vorteil der direkten Verwendung von Messaging aus dem Browser in Ihren Web-Apps

Wenn Sie über Erfahrung im Programmieren von Apps in JavaScript verfügen, sind Sie unter Umständen daran interessiert, die Vorteile kennenzulernen, die von Messaging-Systemen wie IBM WebSphere MQ bereitgestellt werden:

- Wenn Sie Nachrichten über ein Messaging-System senden und empfangen, ist dieses System dafür verantwortlich, die Zustellung der Nachrichten zu gewährleisten.
- Da sich das Messaging-System um die Zustellung kümmert, kann Ihre Web-App starten und zum nächsten Schritt übergehen. Dadurch wird die Programmierungslogik erheblich vereinfacht. Werden für Sie Nachrichten zugestellt, muss Ihr Code überprüfen, ob sie angekommen sind. Ihre App muss keine Empfangsbestätigung mehr verwalten oder die Zustellung nicht zugestellter Nachrichten später erneut versuchen.
- Messaging-Systeme bieten ereignisgesteuerte Nachrichtenübertragung. Ihre Client-App muss keine Anfrage mehr senden und anschließend eine Antwort abfragen. Stattdessen sendet der Messaging-Server eine Nachricht an Ihre Client-App, wenn ein interessantes Ereignis auftritt. Ihre Client-App wird mit Eintreten des Ereignisses benachrichtigt und muss nicht warten, bis sie den Server das nächste Mal abfragt.
- Ereignisgesteuertes Messaging reduziert außerdem erheblich die Last auf der Einheit, die Ihre Client-App hostet, den Datenaustausch im Netz zwischen Browser und Messaging-Server und die Last auf dem Messaging-Server. Dies ist zunehmend von Bedeutung, da immer mehr Systeme auf mobilen Geräten ausgeführt werden und die Verbindung über mobile Netze herstellen.

Ineinandergreifen der Einzelteile

MQTT-Messaging-Client für JavaScript beinhaltet eine Clientbibliothek und eine Beispiel-Web-App, die die Bibliothek verwendet. Sie codieren Ihre eigene Web-App, die die Bibliothek verwendet. Web-App und Clientbibliothek werden dann an Ihrer ausgewählten URL verfügbar gemacht, beispielsweise durch einen MQ-Warteschlangenmanager (wie im folgenden Diagramm) oder durch einen Anwendungsserver. Der Browser übernimmt die Web-App und die Clientbibliothek, und die Web-App verwendet anschließend den Browser, um eine Verbindung zu einem MQTT-Server wie beispielsweise IBM WebSphere MQ Telemetry oder IBM MessageSight herzustellen und Nachrichten mit ihm auszutauschen.

Dabei handelt es sich um folgende Flüsse:

1. Jede Browserinstanz aktualisiert ihre Verbindung zur URL, unter der die Web-App verfügbar ist, und eine aktuelle Version der Web-App und der Clientbibliothek wird in den Browser geladen.
2. Die Web-App verbindet sich mithilfe von MQTT über WebSocket protocol mit einem Warteschlangenmanager und subskribiert ein interessantes Thema.
3. Der Warteschlangenmanager verwendet dieselbe Verbindung, um Nachrichten, die der Subskription entsprechen, zurück zur Web-App zu senden.

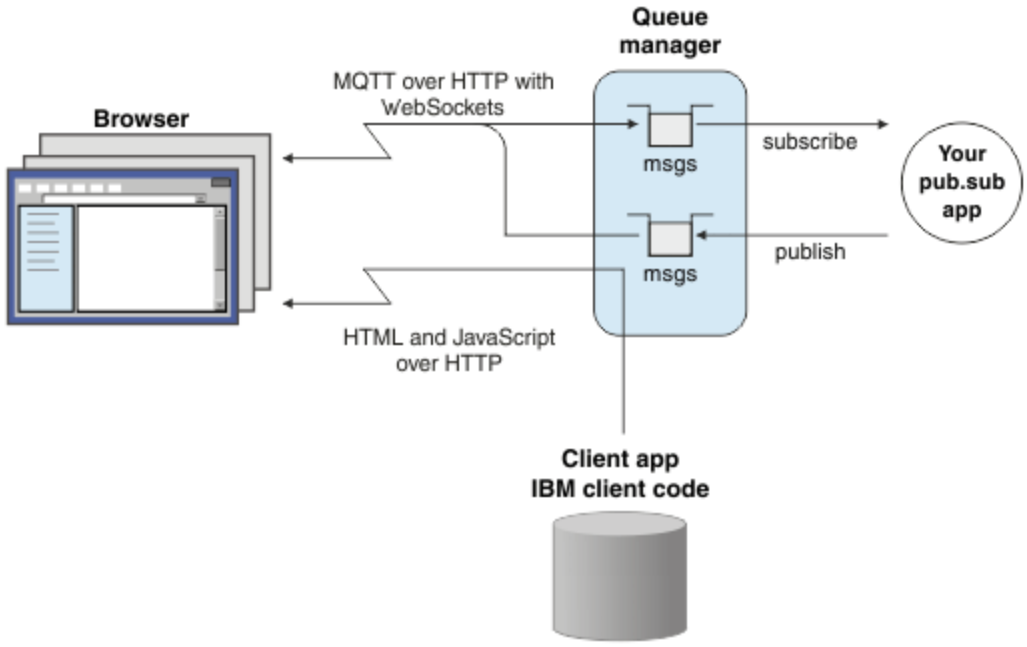


Abbildung 26. Verwendung von MQTT-Messaging-Client für JavaScript mit Publish/Subscribe-Messaging

Die Web-App enthält Anwendungslogik und die URL des MQTT-Servers. Die App verbindet sich mit dem MQTT-Server, wenn sie in einem Browser geöffnet wird, und erstellt die erforderlichen Subskriptionen. Anschließend wartet sie auf den Empfang ereignisgesteuerter Alerts und reagiert auf diese.

Die Web-App stellt eine Verbindung über WebSockets mithilfe von MQTT als Transportprotokoll her. Die meisten modernen Browser können WebSockets-Verbindungen herstellen. Durch die Verwendung von WebSockets kann die Web-App Nachrichten über Firewalls leiten, die HTTP und WebSocket protocol akzeptieren, und Datenpakete ("Frames" genannt) wie bei der Verwendung von TCP over IP senden.

Wenn eine Nachricht im MQTT-Server ankommt, die von der Web-App gesendet wurde, erscheint sie auf der serverseitigen App als Nachricht. Ihr ist nicht bekannt, dass die Nachricht von einem Browser stammt.

Verwaltung und Steuerung eines MQTT-Servers

Der MQTT-Server bearbeitet die serverseitige Komplexität des Messagings. Er gewährleistet die Zustellung der Nachrichten, die er von der Web-App erhält, und er hostet die Publish/Subscribe-Anwendung, die der Web-App antwortet. Für jeden MQTT-Server müssen Sie die folgenden Schritte abschließen:

- Erstellen Sie einen Server.
- Wählen Sie einen Port aus.
- Definieren Sie einen neuen MQTT-Kanal.
- Konfigurieren Sie Ihre Client-Web-App, um die Verbindung zu dem ausgewählten Port über den neuen MQTT-Kanal herzustellen.

Sie müssen dem Browser auch die ausführbare Web-App JavaScript bereitstellen. Wenn Sie IBM WebSphere MQ Telemetry verwenden, übernimmt dies standardmäßig der MQTT-Server für Sie. Er verwendet hierbei denselben MQTT-Kanal wie die Web-App, um die Verbindung zum MQTT-Server herzustellen.

Wenn Sie MQTT ausprobieren, hilft Ihnen dies dabei, schnell betriebsbereit zu sein. Für den Produktionseinsatz, besonders in Umgebungen mit hohem Durchsatz, könnten Sie es unter Umständen vorziehen, die ausführbare Web-App JavaScript auf einem separaten Kanal mithilfe eines dedizierten Anwendungsservers wie beispielsweise WebSphere Application Server bereitzustellen.

Anmerkung: Da er für Umgebungen mit einem hohen Durchsatz entworfen wurde, erwartet IBM MessageSight von Ihnen diese Vorgehensweise.

Wenn Sie beispielsweise IBM WebSphere MQ Telemetry verwenden, dann verwenden Sie den Assistenten IBM WebSphere MQ Explorer **New Telemetry Channel**, um die folgenden Schritte abzuschließen:

1. Erstellen Sie einen Server.
2. Wählen Sie einen Port aus (standardmäßig 1883).
3. Definieren Sie einen neuen MQTT-Kanal.
4. Konfigurieren Sie Ihre Client-Web-App, um die Verbindung zu dem ausgewählten Port über den neuen MQTT-Kanal herzustellen.

Die ausführbare Web-App JavaScript wird (optional) auch über den Warteschlangenmanager auf demselben Kanal bereitgestellt. Hierzu muss der Warteschlangenmanager sowohl MQTT als auch HTTP unterstützen. Wenn Sie bereits über einen für MQTT konfigurierten Warteschlangenmanager verfügen, können Sie das MQSC-Befehlszeilentool verwenden, um das Protokoll in der Kanaldefinition so zu ändern, dass sowohl MQTT als auch HTTP unterstützt wird. Weitere Informationen hierzu finden Sie unter KANAL ÄNDERN.

Die Web-App und die MQTT-Messaging-Client für JavaScript-Clientbibliothek werden auf einem Datenträger in einer Struktur gespeichert, die von Ihrem Anwendungsserver oder Warteschlangenmanager definiert wird. Wenn Sie IBM WebSphere MQ Telemetry verwenden, werden die Web-App und die Clientbibliothek in der folgenden Verzeichnisstruktur gespeichert:

```
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)
```

Die Beispiel-Web-App und Clientbibliothek werden im Verzeichnis *MQINSTALL*/mqxr/SDK/WebContent gespeichert. Das Material in diesem Verzeichnis wird von allen Warteschlangenmanagern bereitgestellt. Wenn Sie nicht möchten, dass Ihre Benutzer all dieses Material sehen und verwenden, dann sollten Sie Ihre eigene Version der App erstellen. Um diese App oder Ihre eigene Ersatz-App auf bestimmten Warteschlangenmanagern verfügbar zu machen, stellen Sie die App in das Verzeichnis *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent . Zur Auswahl der App und der zugehörigen JavaScript -Klassen, die über eine URL bereitgestellt werden sollen, sucht der Warteschlangenmanager zuerst in seinem eigenen Verzeichnis WebContent und anschließend im globalen Verzeichnis WebContent . In der Verzeichnisstruktur des vorherigen Beispiels stellt der Warteschlangenmanager *Ihre Client-App* und die globale Kopie der JavaScript-Klassen bereit.

Um den Warteschlangenmanager beim Bereitstellen der ausführbaren Web-App-Dateien anzuhalten oder um das Verzeichnis zu ändern, in dem der Warteschlangenmanager nach den ausführbaren Dateien sucht, konfigurieren Sie die Eigenschaft **webcontentpath** und fügen sie der Datei mqxr.properties hinzu. Weitere Informationen finden Sie im Abschnitt MQXR-Eigenschaften.

Zugehörige Konzepte

„Vorgehensweise bei der Programmierung von Messaging-Apps in JavaScript” auf Seite 124

Zugehörige Tasks

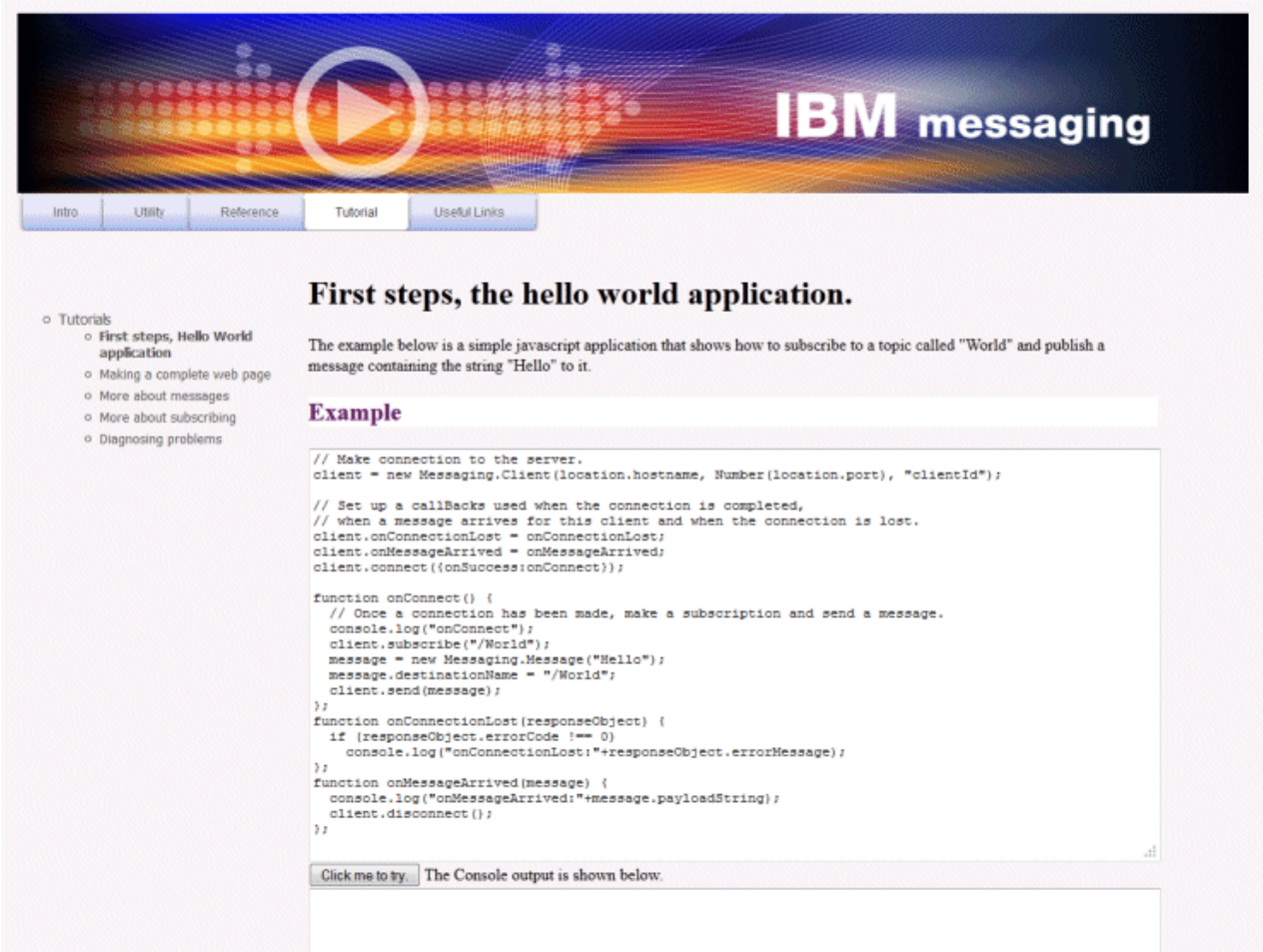
„MQTT-Messaging-Client für JavaScript über SSL und WebSockets verbinden” auf Seite 80
Verbinden Sie Ihre Web-App sicher mit IBM WebSphere MQ, indem Sie die HTML-Beispielseiten des MQTT-Messaging-Clients für MQTT-Messaging-Client für JavaScript> mit SSL und dem WebSocket protocol verwenden.

„Erste Schritte mit dem MQTT-Messaging-Client für JavaScript” auf Seite 25
Für die ersten Schritte mit dem MQTT-Messaging-Client für JavaScript können Sie die Beispielhomepage für den Messaging-Client anzeigen und die Ressourcen durchsuchen, zu denen Links vorhanden sind. Um diese Homepage anzuzeigen, konfigurieren Sie einen MQTT-Server für die Annahme von Verbindungen von den Beispielseiten des MQTT-Messaging-Clients JavaScript und geben anschließend die URL, die Sie auf dem Server konfiguriert haben, in einen Web-Browser ein. Der MQTT-Messaging-Client für JavaScript wird automatisch auf Ihrem Gerät gestartet und die Beispielhomepage für den Messaging-Client wird angezeigt. Diese Seite enthält Verknüpfungen zu Dienstprogrammen, einer Dokumentation zur Programmierschnittstelle, einem Lernprogramm und weiteren hilfreichen Informationsquellen.

Vorgehensweise bei der Programmierung von Messaging-Apps in JavaScript

Der MQTT-Messaging-Client für JavaScript enthält ein Lernprogramm, in dem die Erstellung einer einfachen Publish/Subscribe-Web-App gezeigt wird. Durch den Anwendungscode "First steps, Hello world" erhalten Sie grundlegende Kenntnisse zum Mechanismus der Programmierung von Web-Apps für das Messaging.

Wenn Sie bisher hauptsächlich konventionelle Messaging-Anwendungen entwickelt und implementiert haben, sollten Sie auch den Abschnitt „Tipps zur JavaScript-Codierung” auf Seite 125 lesen. Falls Sie ein erfahrener JavaScript-Entwickler sind, der sich noch nicht mit dem Messaging befasst hat, lesen Sie die kurze Einführung in die wichtigsten Messaging-Konzepte im Abschnitt „Grundlegende Informationen zum Messaging” auf Seite 127.



Tipps zur JavaScript-Codierung

Wenn Sie mit der Entwicklung von Messaging-Anwendungen vertraut sind, jedoch noch keine Erfahrung mit Web-Apps gesammelt haben, sind die folgenden Tipps möglicherweise für Sie hilfreich:

Wrapping des Codes für jedes Ereignis in einem onSuccess-Callback

Bei der Codierung einer Messaging-Anwendung codieren Sie die folgenden Ereignisse in der folgenden Reihenfolge:

1. Verbinden
2. Abonnieren
3. veröffentlichen
4. Nachricht empfangen

Die MQTT-Messaging-Client für JavaScript-Anwendungsprogrammierschnittstelle (API) wird gänzlich asynchron ausgeführt. Ihr Anwendungsthread wird also nicht blockiert, solange auf das Inkrafttreten von Aufrufen wie 'connect' oder 'subscribe' gewartet wird. Stattdessen signalisieren diese Aufrufe ihren Abschluss, indem sie einen `onSuccess`- oder `onFailure`-Callback aufrufen. Damit Sie sich darauf verlassen können, dass jedes Ereignis vor der Auslösung des nächsten Ereignisses abgeschlossen wurde, müssen Sie den Code für jedes Ereignis in einem `onSuccess`-Callback einschließen. Die JavaScript-Anwendung könnte beispielsweise von der Ausführung des Verbindungsaufrufs zurückkehren, bevor die Verbindung erstellt wurde. Damit Sie sich darauf verlassen können, dass die Verbindung vor Ihrer Subskription erstellt wurde, müssen Sie den Subskriptionscode in einen `onSuccess`-Callback für die Verbindung einschließen.

Der Anwendungscode "First steps, Hello world" verwendet diese Methode.

Einbettung des Anwendungscodes innerhalb einer HTML-Markup

Hier finden Sie eine JavaScript-Beispielseite:

Es folgt die Quelle für die vorherige Seite, mit der veranschaulicht wird, wie der Anwendungscode innerhalb einer HTML-Markup eingebettet wird:

```
<!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>
```

Grundlegende Informationen zum Messaging

Es folgen einige Hintergrundinformationen zum Messaging für Entwickler von Web-Apps, die noch nicht mit dem Messaging vertraut sind:

Asynchrones Messaging und Messaging ohne Empfangsbestätigung (Fire-and-Forget).
Das MQTT-Protokoll unterstützt die zuverlässige Nachrichtenübermittlung und Übertragungen ohne Empfangsbestätigungen (Fire-and-Forget). Im Protokoll verläuft die Nachrichtenübermittlung asynchron: Die App übergibt die Nachricht an die Client-API und ergreift keine weitere Maßnahme, um sicherzustellen, dass die Nachricht tatsächlich zugestellt wird. Diese Methode wird als *Fire-and-Forget* bezeichnet. Sobald eine Antwort verfügbar ist, wird sie automatisch an die App gesendet.

Die asynchrone Übermittlung bedeutet, dass die App nicht von einer Serververbindung abhängig ist und nicht auf Nachrichten warten muss. Das Interaktionsmodell ist mit dem der E-Mail vergleichbar, jedoch optimiert für die Anwendungsprogrammierung.

Weitere Informationen finden Sie im Abschnitt "MQTT-Protokoll" unter „Einführung in MQTT” auf Seite 5

Überblick über Publish-and-Subscribe-Messaging.
Der Bereitsteller von Informationen wird als *Publisher* bezeichnet. Ein Bereitsteller stellt Informationen zu einem Thema zur Verfügung und benötigt keine Angaben zu den Anwendungen, für die diese Informationen von Interesse sind. Ein Bereitsteller wählt ein *Thema*. Dabei handelt es sich um einen Container für Nachrichten zu einem bestimmten Betreff. Anschließend erstellt der Bereitsteller jede

Einzelinformation zu dem betreffenden Thema als Nachricht, die als *Veröffentlichung* bezeichnet wird, und stellt diese im zugehörigen Thema bereit.

Der Nutzer der Informationen wird als *Subskribent* bezeichnet. Ein Subskribent erstellt eine *Subskription* für ein Thema, an dem er interessiert ist. Sobald eine neue Nachricht im Thema bereitgestellt wird, wird sie an alle Subskribenten des Themas weitergeleitet. Subskribenten können mehrere Subskriptionen erstellen und Informationen von vielen verschiedenen Publishern erhalten.

Siehe auch Einführung in das IBM WebSphere MQ-Publish/Subscribe-Messaging

Abgleich der Subskriptionen mit den Themen

Wenn Sie IBM WebSphere MQ als MQTT-Server verwenden, müssen Sie verstehen, wie Themen von IBM WebSphere MQ angegeben werden. In IBM WebSphere MQ erstellt ein Bereitsteller eine Nachricht und veröffentlicht sie mit einer Themenzeichenfolge, die dem Betreff der Veröffentlichung am besten entspricht. Um Veröffentlichungen zu empfangen, erstellt eine Subskribent eine Subskription mit einer Zeichenfolge mit Mustererkennung, um Veröffentlichungsthemen auszuwählen. Der Warteschlangenmanager liefert Veröffentlichungen an Subskribenten, deren Subskriptionen dem Veröffentlichungsthema entsprechen und für den Empfang von Veröffentlichungen autorisiert sind.

Normalerweise werden Betrefftexte hierarchisch in Themenstrukturen organisiert. Mit dem Zeichen '/' werden in der Themazeichenfolge Unterthemen erstellt. Themen sind Knoten in einer Themenstruktur. Themen können Blattknoten ohne weitere Unterthemen oder Zwischenknoten mit Unterthemen sein. Platzhalter ermöglichen es Subskribenten, gleichzeitig mehrere Themen zu subskribieren. Mit einer für `/sport/tennis` eingerichteten Subskription werden beispielsweise nur Nachrichten abgerufen, die im Unterthema 'tennis' veröffentlicht werden, während bei einer Subskription von `/sport/#` Nachrichten abgerufen werden, die in sämtlichen Unterthemen von `/sport` bereitgestellt werden.

Siehe auch Themen, Themenstrukturen und Platzhalterschemas.

Zugehörige Konzepte

„MQTT-Messaging-Client für JavaScript und Web-Apps” auf Seite 120

Zugehörige Tasks

„MQTT-Messaging-Client für JavaScript über SSL und WebSockets verbinden” auf Seite 80
Verbinden Sie Ihre Web-App sicher mit IBM WebSphere MQ, indem Sie die HTML-Beispielseiten des MQTT-Messaging-Clients für MQTT-Messaging-Client für JavaScript> mit SSL und dem WebSocket protocol verwenden.

„Erste Schritte mit dem MQTT-Messaging-Client für JavaScript” auf Seite 25
Für die ersten Schritte mit dem MQTT-Messaging-Client für JavaScript können Sie die Beispielhomepage für den Messaging-Client anzeigen und die Ressourcen durchsuchen, zu denen Links vorhanden sind. Um diese Homepage anzuzeigen, konfigurieren Sie einen MQTT-Server für die Annahme von Verbindungen von den Beispielseiten des MQTT-Messaging-Clients JavaScript und geben anschließend die URL, die Sie auf dem Server konfiguriert haben, in einen Web-Browser ein. Der MQTT-Messaging-Client für JavaScript wird automatisch auf Ihrem Gerät gestartet und die Beispielhomepage für den Messaging-Client wird angezeigt. Diese Seite enthält Verknüpfungen zu Dienstprogrammen, einer Dokumentation zur Programmierschnittstelle, einem Lernprogramm und weiteren hilfreichen Informationsquellen.

Callbacks und Synchronisation in MQTT -Client-Apps

Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. Die Threads entkoppeln eine MQTT -Client-App so weit wie möglich von Verzögerungen bei der Übertragung von Nachrichten zum und vom Server. Veröffentlichungen, Zustellungstoken und Verbindungsverlustereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Callbacks

Die `MqttCallback`-Schnittstelle verfügt über drei Callback-Methoden; eine Beispielimplementierung finden Sie unter Callback.java.

`connectionLost(java.lang.Throwable cause)`

`connectionLost` wird aufgerufen, wenn ein Übertragungsfehler zum Löschen der Verbindung führt. Diese Methode wird auch aufgerufen, wenn der Server die Verbindung als Ergebnis eines

Fehlers auf dem Server löscht, nachdem die Verbindung hergestellt wurde. Serverfehler werden im Fehlerprotokoll des Warteschlangenmanagers protokolliert. Der Server löscht die Verbindung zum Client und der Client ruft `MqttCallback.connectionLost` auf.

Die einzigen Fernzugriffsfehler, die als Ausnahmen in demselben Thread wie die Client-App ausgelöst werden, sind Ausnahmen von `MqttClient.connect`. Fehler, die vom Server nach der Verbindungsherstellung ermittelt werden, werden der Callback-Methode `MqttCallback.connectionLost` als `throwables` gemeldet.

Typische Serverfehler, die zu `connectionLost` führen, sind Berechtigungsfehler. Diese treten beispielsweise auf, wenn der Telemetrieserver ein Thema auf Anweisung eines Clients veröffentlichen möchte, der nicht zur Veröffentlichung dieses Themas berechtigt ist. Alles, was dazu führt, dass der Bedingungscode `MQCC_FAIL` an den Telemetrieserver zurückgegeben wird, kann zur Folge haben, dass die Verbindung unterbrochen wird.

`deliveryComplete(MqttDeliveryToken token)`

`deliveryComplete` wird vom MQTT -Client aufgerufen, um ein Zustellungstoken zurück an die Client-App zu übergeben (siehe „Zustellungstoken" auf Seite 133). Mithilfe des Zustellungstokens kann der Callback mit der Methode `token.getMessage` auf die veröffentlichte Nachricht zugreifen.

Wenn der Anwendungscallback die Steuerung an den MQTT -Client zurückgibt, nachdem er von der Methode `deliveryComplete` aufgerufen wurde, ist die Zustellung abgeschlossen. Bis zum Abschluss der Übergabe werden Nachrichten mit der Servicequalität `QoS` von 1 oder 2 von der Persistenzklasse beibehalten.

Der Anruf von `deliveryComplete` stellt einen Synchronisationspunkt zwischen der Anwendung und der Persistenzklasse dar. Die Methode `deliveryComplete` wird für dieselbe Nachricht nie zweimal aufgerufen.

Wenn der Anwendungscallback die Steuerung nach seinem Aufruf durch die Methode `deliveryComplete` an den MQTT-Client zurückgibt, ruft der Client `MqttClientPersistence.remove` für Nachrichten mit der Servicequalität (QoS) 1 oder 2 auf. `MqttClientPersistence.remove` löscht die lokal gespeicherte Kopie der veröffentlichten Nachricht.

Aus Sicht der Transaktionsverarbeitung ist der Aufruf von `deliveryComplete` eine einphasige Transaktion, mit der die Übergabe festgeschrieben wird. Falls während des Callbacks ein Verarbeitungsfehler auftritt, wird `MqttClientPersistence.remove` beim Neustart des Clients erneut aufgerufen, um die lokale Kopie der veröffentlichten Nachricht zu löschen. Der Callback wird nicht erneut aufgerufen. Wenn Sie den Callback zum Speichern eines Protokolls der zugestellten Nachrichten verwenden, können Sie das Protokoll nicht mit dem MQTT-Client synchronisieren. Um ein Protokoll auf zuverlässige Weise zu speichern, müssen Sie das Protokoll in der Klasse `MqttClientPersistence` aktualisieren.

Das Zustellungstoken und die Nachricht werden vom Hauptanwendungsthread und vom MQTT-Client referenziert. Der MQTT-Client nimmt die Referenz zum `MqttMessage`-Objekt zurück, sobald die Übergabe abgeschlossen ist, und das Zustellungstokenobjekt, wenn der Client die Verbindung trennt. Das `MqttMessage`-Objekt kann fehlerhafte Daten, die nach Abschluss der Zustellung erfasst werden, enthalten, wenn die Client-App die Referenz löscht. Das Zustellungstoken kann fehlerhafte Daten enthalten, die nach der Unterbrechung der Sitzung erfasst werden.

Nach dem Veröffentlichen einer Nachricht können Sie die Attribute `MqttDeliveryToken` und `MqttMessage` abrufen. Wenn Sie versuchen, beliebige `MqttMessage`-Attribute nach dem Veröffentlichen einer Nachricht festzulegen, führt dies zu einem nicht definierten Ergebnis.

Der MQTT-Client fährt mit der Verarbeitung von Empfangsbestätigungen fort, wenn sich der Client mit derselben `Client-ID` wieder mit der vorherigen Sitzung verbindet; siehe „Sitzungen bereinigen" auf Seite 131. Die MQTT -Client-App muss `MqttClient.CleanSession` für die vorherige Sitzung auf `false` und in der neuen Sitzung auf `false` setzen. Der MQTT-Client erstellt neue Zustellungstokens und Nachrichtenobjekte in der neuen Sitzung für anstehende Übergaben. Er stellt die Objekte mithilfe der Klasse `MqttClientPersistence` wieder her. Falls der Anwendungsclient noch Referenzen auf die alten Zustellungstokens und Nachrichten enthält, löschen Sie diese Referenzen. Der Anwendungscallback wird in der neuen Sitzung für alle Übergaben aufgerufen, die in der früheren Sitzung eingeleitet wurden und in dieser Sitzung abgeschlossen werden.

Der Anwendungscallback wird nach Herstellung der Anwendungsclientverbindung aufgerufen, wenn eine anstehende Übergabe abgeschlossen ist. Bevor der Anwendungsclient die Verbindung herstellt, kann er anstehende Übergaben mit der Methode `MqttClient.getPendingDeliveryTokens` abrufen.

Beachten Sie, dass das Nachrichtenobjekt, das veröffentlicht wird, und dessen Bytefeldgruppe mit Nutzdaten ursprünglich von der Client-App erstellt wurden. Der MQTT-Client verweist auf diese Objekte. Das Nachrichtenobjekt, das vom Zustellungstoken in der Methode `token.getMessage` zurückgegeben wird, ist nicht notwendigerweise dasselbe Nachrichtenobjekt, das vom Client erstellt wurde. Wenn eine neue MQTT-Clientinstanz das Zustellungstoken erneut erstellt, erstellt die Klasse `MqttClientPersistence` das Objekt `MqttMessage` erneut. Aus Konsistenzgründen gibt `token.getMessage` den Wert `null` zurück, wenn `token.isCompleted` auf `true` gesetzt ist, unabhängig davon, ob das Nachrichtenobjekt vom Anwendungsclient oder von der Klasse `MqttClientPersistence` erstellt wurde.

`messageArrived(MqttTopic topic, MqttMessage message)`

`messageArrived` wird aufgerufen, wenn eine Veröffentlichung für den Client eintrifft, die mit einem Subskriptionsthema übereinstimmt. `topic` gibt das Veröffentlichungsthema an, nicht den Subskriptionsfilter. Die beiden können unterschiedlich sein, wenn der Filter Platzhalterzeichen enthält.

Wenn das Thema mit mehreren vom Client erstellten Subskriptionen übereinstimmt, empfängt der Client mehrere Kopien der Veröffentlichung. Wenn ein Client eine Veröffentlichung für ein Thema durchführt, das er selbst subskribiert hat, empfängt er eine Kopie seiner eigenen Veröffentlichung.

Wenn eine Nachricht mit der Servicequalität QoS von 1 oder 2 gesendet wird, wird sie von der Klasse `MqttClientPersistence` gespeichert, bevor der MQTT-Client `messageArrived` aufruft. `messageArrived` verhält sich wie `deliveryComplete`: Dieses Element wird nur einmal für eine Veröffentlichung aufgerufen und die lokale Kopie der Veröffentlichung wird von `MqttClientPersistence.remove` entfernt, wenn `messageArrived` an den MQTT-Client zurückgegeben wird. Der MQTT-Client löscht seine Verweise auf das Thema und die Nachricht, wenn `messageArrived` an den MQTT-Client zurückgegeben wird. Die Themen- und Nachrichtenobjekte enthalten fehlerhafte Daten, wenn der Anwendungsclient keine Referenzen auf die Objekte beibehalten hat.

Synchronisation von Callbacks, Threading und Client-App

Der MQTT-Client ruft eine Callback-Methode nicht im Hauptanwendungsthread, sondern in einem separaten Thread auf. Die Client-App erstellt keinen Thread für den Callback, sondern wird vom MQTT -Client erstellt.

Der MQTT-Client synchronisiert Callback-Methoden. Es wird immer nur eine Instanz der Callback-Methode ausgeführt. Dank der Synchronisation ist es einfach, ein Objekt zu aktualisieren, das mitzählt, welche Veröffentlichungen zugestellt wurden. Da immer nur eine Instanz von `MqttCallback.deliveryComplete` ausgeführt wird, ist die Aktualisierung des Veröffentlichungszählers auch ohne weitere Synchronisation ein zuverlässiger Vorgang. Es trifft außerdem immer nur eine Veröffentlichung ein. Ihr Code in der Methode `messageArrived` kann ein Objekt aktualisieren, ohne es zu synchronisieren. Wenn Sie sich in einem anderen Thread auf den Zähler oder das Objekt, das aktualisiert wird, beziehen, synchronisieren Sie den Zähler oder das Objekt.

Das Zustellungstoken stellt einen Synchronisationsmechanismus zwischen dem Hauptanwendungsthread und der Übergabe einer Veröffentlichung bereit. Die Methode `token.waitForCompletion` wartet, bis die Übergabe einer bestimmten Veröffentlichung abgeschlossen ist oder bis ein optionales Zeitlimit überschritten wird. Sie können `token.waitForCompletion` auf verschiedene Weise verwenden, um eine Veröffentlichung nach der anderen zu verarbeiten:

1. Halten Sie den Anwendungsclient an, bis die Zustellung der Veröffentlichung abgeschlossen ist (siehe PubSync.java).
2. Führen Sie eine Synchronisation mit der Methode `MqttCallback.deliveryComplete` durch. `token.waitForCompletion` wird erst fortgesetzt, wenn `MqttCallback.deliveryComplete` an den MQTT-Client zurückgegeben wird. Mithilfe dieses Mechanismus können Sie aktiven Code in

`MqttCallback.deliveryComplete` synchronisieren, bevor Code im Hauptanwendungsthread ausgeführt wird.

Was ist, wenn Sie Veröffentlichungen durchführen wollen, ohne auf die Übergabe der einzelnen Veröffentlichungen zu warten, sondern erst dann eine Bestätigung möchten, wenn alle Veröffentlichungen übergeben wurden? Wenn Sie die Veröffentlichungen in einem Einzelthread durchführen, ist die letzte Veröffentlichung, die gesendet wird, auch die letzte, die übergeben wird.

Synchronisation von an den Server gesendete Anforderungen

Tabelle 4. Synchronisationsverhalten von Methoden, die zu Anforderungen an den Server führen.

Diese Tabelle listet die Methoden im MQTT -Java-Client auf, die eine Anfrage an den Server senden. Die Tabelle beschreibt für jede Methode die Bedingungen, unter denen die Methode entweder wartet oder zurückgibt, und wie lange die Methode wartet.

Methode	Synchronisation	Zeitlimitintervall
`MqttClient.Connect`	Wartet auf die Herstellung einer Verbindung mit dem Server.	30 Sekunden (Standardwert) oder ein mit einem Parameter festgelegter Wert.
`MqttClient.Disconnect`	Wartet, bis der MQTT-Client eventuell erforderliche Arbeitsschritte abgeschlossen hat und bis die TCP/IP-Sitzung getrennt wurde.	30 Sekunden (Standardwert) oder ein mit einem Parameter festgelegter Wert.
`MqttClient.Subscribe`	Wartet, bis die Subskriptionsanforderung abgeschlossen ist.	30 Sekunden (Standardwert) oder ein mit einem Parameter festgelegter Wert.
`MqttClient.UnSubscribe`	Wartet, bis die Anforderung zur Aufhebung der Subskription abgeschlossen ist.	30 Sekunden (Standardwert) oder ein mit einem Parameter festgelegter Wert.
`MqttClient.Publish`	Kehrt nach der Übergabe der Anforderung an den MQTT-Client unverzüglich zum Anwendungsthread zurück.	Keine.
`MqttDeliveryToken.waitForCompletion`	Wartet auf die Rückgabe des Zustellungstokens.	Unendlich (Standardwert) oder ein mit einem Parameter festgelegter Wert.

Sitzungen bereinigen

Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Wenn Sie mithilfe der Methode `MqttClient.connect` eine MQTT-Client-App verbinden, identifiziert der Client die Verbindung über die Client-ID und die Adresse des Servers. Der Server prüft, ob Sitzungsdaten aus einer vorherigen Verbindung zum Server gespeichert wurden. Wenn noch Informationen zu

einer früheren Sitzung vorhanden sind und `cleanSession=true` festgelegt ist, werden die vorherigen Sitzungsdaten im Client und Server gelöscht. Wenn `cleanSession=false` festgelegt ist, wird die vorherige Sitzung fortgesetzt. Wenn keine Informationen zu einer vorherigen Sitzung vorhanden sind, wird eine neue Sitzung gestartet.

Anmerkung: Der WebSphere MQ Administrator kann eine offene Sitzung zwangsweise schließen und alle Sitzungsdaten löschen. Wenn der Client eine Sitzung mit dem Wert `cleanSession=false` erneut öffnet, wird eine neue Sitzung gestartet.

Veröffentlichungen

Wenn Sie vor der Herstellung einer Verbindung zum Client den Standardwert `MqttConnectOptions` verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, werden alle Übergaben von anstehenden Veröffentlichungen für den Client entfernt, wenn der Client eine Verbindung herstellt.

Die Einstellung zum Bereinigen einer Sitzung hat keine Auswirkung auf Veröffentlichungen, die mit `QoS=0` gesendet werden. Die Verwendung von `cleanSession=true` kann für `QoS=1` und `QoS=2` zum Verlust einer Veröffentlichung führen.

Subskriptionen

Wenn Sie den Standardwert `MqttConnectOptions`verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor der Client verbunden wird, werden alle alten Subskriptionen für den Client entfernt, wenn der Client eine Verbindung herstellt. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung auf `false` setzen, werden alle Subskriptionen, die der Client erstellt, zu allen Subskriptionen hinzugefügt, die für den Client vor dem Herstellen der Verbindung vorhanden waren. Alle Subskriptionen bleiben aktiv, wenn die Clientverbindung getrennt wird.

Eine andere Möglichkeit, um zu verstehen, wie sich das Attribut `cleanSession` auf Subskriptionen auswirkt, besteht darin, es sich als modales Attribut vorzustellen. Der Standardmodus (`cleanSession=true`) bedeutet, dass der Client nur im Rahmen der Sitzung Subskriptionen erstellt und Veröffentlichungen empfängt. Im alternativen Modus (`cleanSession=false`) sind Subskriptionen permanent. Der Client kann Verbindungen herstellen und trennen, seine Subskriptionen bleiben aktiv. Bei der Wiederherstellung einer Verbindung empfängt der Client alle nicht zugestellten Veröffentlichungen. Solange die Verbindung besteht, kann er die Gruppe der Subskriptionen, die in seinem Auftrag aktiv sind, ändern.

Sie müssen den Modus `cleanSession` festlegen, bevor Sie eine Verbindung herstellen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie die Modi von `cleanSession=false` in `cleanSession=true`ändern, werden alle vorherigen Subskriptionen für den Client und alle Veröffentlichungen, die nicht empfangen wurden, verworfen.

Client-ID

Die Client-ID ist eine Zeichenfolge mit 23 Byte, die einen MQTT -Client identifiziert. Die Client-ID muss unter allen Clients, die sich mit dem Server verbinden, eindeutig sein, und sie darf nicht identisch mit dem Namen des Warteschlangenmanagers auf dem Server sein. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine bestimmte Vorgehensweise zur Zuordnung von Client-IDs zu nutzen. Dies ermöglicht die Konfiguration eines Clients mit der zugehörigen gewählten ID.

Die Client-ID wird bei der Verwaltung eines MQTT-Systems verwendet. Da unter Umständen hunderttausende Clients verwaltet werden, muss es möglich sein, einen bestimmten Client schnell zu erkennen. Nehmen wir beispielsweise an, bei einem Gerät liegt eine Störung vor und Sie werden von einem Kunden, der bei einem Help-Desk anruft, davon benachrichtigt. Wie wird das Gerät vom Kunden identifiziert und wie korrelieren Sie diese Identifikation mit dem Server, der für gewöhnlich mit dem Client verbunden ist? Müssen Sie eine Datenbank zu Rate ziehen, die jedes Gerät einer Client-ID und einem Server zuordnet? Gibt der Name des Geräts Aufschluss darüber, mit welchem Server es verbunden ist? In der Anzeige

der MQTT-Clientverbindungen sehen Sie zu jeder Verbindung die Client-ID. Müssen Sie in einer Tabelle nachschlagen, um eine Client-ID einer physische Einheit zuzuordnen?

Gibt die Client-ID ein bestimmtes Gerät, einen Benutzer oder eine Anwendung an, die auf dem Client ausgeführt wird? Wenn ein Kunde ein fehlerhaftes Gerät durch ein neues ersetzt, hat das neue Gerät dieselbe ID wie das alte Gerät? Wird von Ihnen eine neue ID angelegt? Wenn Sie ein physisches Gerät ändern, aber dieselbe ID beibehalten, werden ausstehende Veröffentlichungen und aktive Subskriptionen automatisch auf das neue Gerät übertragen.

Wie stellen Sie sicher, dass die Client-IDs eindeutig sind? Neben einem System für die Generierung eindeutiger IDs müssen Sie über einen zuverlässigen Prozess zur Festlegung der ID auf dem Client verfügen. Möglicherweise ist der Client eine Funktionseinheit ohne Benutzerschnittstelle. Fertigen Sie das Gerät mit einer Client-ID - beispielsweise unter Verwendung seiner MAC-Adresse? Oder verfügen Sie über einen Softwareinstallations- und Konfigurationsprozess, mit dem das Gerät vor seiner Aktivierung konfiguriert wird?

Sie können beispielsweise eine Client-ID auf Basis der aus 48 Bit bestehenden MAC-Adresse des Geräts erstellen, damit die ID kurz und eindeutig ist. Falls die Übertragungsgröße eine untergeordnete Rolle spielt, können Sie die verbleibenden 17 Bytes verwenden, um die Verwaltung der Adresse zu vereinfachen.

Zustellungstoken

Wenn ein Client ein Thema veröffentlicht, wird ein neues Zustellungstoken erstellt. Verwenden Sie das Zustellungstoken, um die Zustellung einer Veröffentlichung zu überwachen oder die Client-App zu blockieren, bis die Zustellung abgeschlossen ist.

Beim Token handelt es sich um ein `MqttDeliveryToken`-Objekt. Es wird durch den Aufruf der Methode 'MqttTopic.publish()' erstellt und im MQTT-Client aufbewahrt, bis die Clientsitzung getrennt wird und die Übergabe abgeschlossen ist.

Die normale Verwendung des Tokens ist die Überprüfung, ob die Übergabe abgeschlossen wurde. Sie können die Client-App blockieren, bis die Zustellung abgeschlossen ist, indem Sie mit dem zurückgegebenen Token die Funktion `token.waitForCompletion` aufrufen. Als Alternative stellen Sie einen `MqttCallBack`-Handler bereit. Wenn der MQTT-Client alle Bestätigung empfangen hat, die er als Teil der Übergabe der Veröffentlichung erwartet, wird `MqttCallBack.deliveryComplete` aufgerufen und das Zustellungstoken wird als Parameter übergeben.

Bis zum Abschluss der Übergabe können Sie die Veröffentlichung mithilfe des zurückgegebenen Zustellungstokens überprüfen, indem Sie `token.getMessage` aufrufen.

Abgeschlossene Übergaben

Der Abschluss von Übergaben ist asynchron und hängt von der Qualität des Service ab, der der Veröffentlichung zugeordnet ist.

At most once (Höchstens einmal)

`QoS=0`
Die Übergabe ist sofort nach der Rückgabe von `MqttTopic.publish` abgeschlossen. `MqttCallback.deliveryComplete` wird unverzüglich aufgerufen.

At least once (Mindestens einmal)

`QoS=1`
Die Übergabe ist abgeschlossen, wenn eine Bestätigung der Veröffentlichung vom Warteschlangenmanager empfangen wurde. `MqttCallback.deliveryComplete` wird aufgerufen, wenn die Bestätigung empfangen wurde. Die Nachricht wird möglicherweise mehrfach vor dem Aufrufen von `MqttCallback.deliveryComplete` übergeben, wenn die Datenübertragung langsam oder störanfällig ist.

Exactly once (Exakt einmal)

`QoS=2`

Die Übergabe ist abgeschlossen, wenn der Client eine Beendigungsnachricht darüber empfängt, dass die Veröffentlichung für Subskribenten veröffentlicht wurde. `MqttCallback.deliveryComplete` wird aufgerufen, sobald die Veröffentlichungsnachricht empfangen wurde. Es wird nicht auf die Beendigungsnachricht gewartet.

In seltenen Fällen kehrt Ihre Client-App möglicherweise nicht normal von `MqttCallback.deliveryComplete` zum MQTT -Client zurück. Sie wissen, dass die Übergabe abgeschlossen ist, da `MqttCallback.deliveryComplete` aufgerufen wurde. Wenn der Client die gleiche Sitzung erneut startet, wird `MqttCallback.deliveryComplete` nicht erneut aufgerufen.

Unvollständige Übergaben

Wenn die Übergabe nach dem Trennen der Clientsitzung nicht abgeschlossen ist, können Sie den Client erneut verbinden und die Übergabe abschließen. Sie können die Übergabe einer Nachricht nur abschließen, wenn die Nachricht in einer Sitzung veröffentlicht wurde, in der das Attribut `MqttConnectionOptions` auf `false` gesetzt ist.

Erstellen Sie den Client mit der gleichen Client-ID und Serveradresse und stellen Sie dann die Verbindung her, wobei das `cleanSession`-Attribut `MqttConnectionOptions` erneut auf `false` gesetzt ist. Wenn Sie `cleanSession` auf `true` setzen, werden anstehende Zustellungstoken verworfen.

Durch das Aufrufen von `MqttClient.getPendingDeliveryTokens` können Sie prüfen, ob anstehende Übergaben vorhanden sind. Sie können `MqttClient.getPendingDeliveryTokens` vor dem Herstellen einer Verbindung zum Client aufrufen.

Veröffentlichung "Last Will and Testament"

Wenn eine MQTT -Clientverbindung unerwartet beendet wird, dann können Sie WebSphere MQ Telemetry so konfigurieren, dass eine "Last Will and Testament" -Veröffentlichung gesendet wird. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Erstellen Sie ein Thema für "Last Will and Testament". Sie können ein Thema wie `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Losterstellen`.

Richten Sie mit der Methode `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` ein "Last Will and Testament" ein.

Sie können in der Nachricht `lastWillPayload` auch eine Zeitmarke erstellen. Integrieren Sie weitere Clientinformationen, die das Ermitteln des Clients und die Bedingungen der Verbindung unterstützen. Übergeben Sie das Objekt `MqttConnectionOptions` an den `MqttClient`-Konstruktor.

Setzen Sie `lastWillQos` auf 1 oder 2, damit die Nachricht in IBM WebSphere MQ persistent ist und die Zustellung sichergestellt wird. Um die Informationen der letzten Verbindung aufzubewahren, setzen Sie `lastWillRetained` auf `true`.

Die Veröffentlichung "Last Will and Testament" wird an Subskribenten gesendet, wenn die Verbindung unerwartet beendet wird. Sie wird gesendet, wenn der Client beim Beenden der Verbindung nicht die Methode `MqttClient.disconnect` aufruft.

Um Verbindungen zu überwachen, ergänzen Sie die Veröffentlichung "Last Will and Testament" mit anderen Veröffentlichungen, damit Verbindungen und programmierte Unterbrechungen aufgezeichnet werden.

Nachrichtenpersistenz in MQTT-Clients

Veröffentlichungsnachrichten werden zu persistenten Nachrichten, wenn sie mit der Servicequalität "Mindestens einmal" oder "Genau einmal" gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

In MQTT sind bei der Nachrichtenpersistenz zwei Aspekte relevant: wie die Nachricht übertragen wird, und ob sie als persistente Nachricht im MQTT-Server eingereiht wird.

1. Der MQTT-Client verbindet die Nachrichtenpersistenz mit der Servicequalität. Abhängig von der von Ihnen für eine Nachricht ausgewählten Servicequalität wird die Nachricht zu einer persistenten Nachricht. Die Nachrichtenpersistenz ist für die Implementierung der erforderlichen Servicequalität notwendig.

   Wenn Sie "höchstens einmal" (QoS=0) angeben, löscht der Client die Nachricht, sobald sie veröffentlicht wurde. Wenn es bei der vorgelagerten Verarbeitung der Nachricht zu Fehlern kommt, wird die Nachricht nicht erneut gesendet. Selbst wenn der Client weiterhin aktiv ist, wird die Nachricht nicht erneut gesendet. Das Verhalten von QoS=0-Nachrichten entspricht dem Verhalten von schnellen, nicht persistenten IBM WebSphere MQ-Nachrichten.

   Wenn eine Nachricht von einem Client veröffentlicht wird, für den QoS auf 1 oder 2 gesetzt ist, wird die Nachricht zu einer persistenten Nachricht. Die Nachricht wird lokal gespeichert und nur vom Client gelöscht, wenn sie nicht mehr benötigt wird, um die Zustellung des Typs "mindestens einmal" (QoS=1) oder "genau einmal" (QoS=2) sicherzustellen.
2. Ist eine Nachricht mit dem Wert 1 oder 2 für QoS markiert, wird sie als persistente Nachricht eingereiht. Ist sie als QoS=0 markiert, wird sie als nicht persistente Nachricht eingereiht. In IBM WebSphere MQ werden nicht persistente Nachrichten "genau einmal" zwischen Warteschlangenmanagern übertragen, es sei denn, für den Nachrichtenkanal ist das Attribut NPMSPEED auf FASTgesetzt.

Eine persistente Veröffentlichung wird auf dem Client gespeichert, bis sie von einer Client-App empfangen wird. Wenn QoS=2 festgelegt ist, wird die Veröffentlichung aus dem Client gelöscht, wenn der Callback der Anwendung die Steuerung zurückgibt. Wenn QoS=1 festgelegt ist, empfängt die Anwendung die Veröffentlichung im Falle eines Fehlers möglicherweise erneut. Wenn QoS=0 festgelegt ist, empfängt der Callback die Veröffentlichung höchstens einmal. Die Veröffentlichung wird möglicherweise nicht empfangen, wenn ein Fehler auftritt oder die Verbindung zum Client zum Zeitpunkt der Veröffentlichung getrennt wird.

Wenn Sie ein Thema subskribieren, können Sie die Servicequalität QoS, mit der der Subskribent Nachrichten empfängt, reduzieren, damit sie mit den Funktionen der zugehörigen Persistenz übereinstimmt. Veröffentlichungen, die mit einer höheren QoS-Einstellung erstellt werden, werden mit der höchsten QoS-Einstellung gesendet, die vom Subskribenten angefordert wird.

Nachrichten speichern

Bei der Implementierung von Datenspeichern auf kleinen Einheiten gibt es große Unterschiede. Das Modell zum temporären Speichern persistenter Nachrichten in einem vom MQTT-Client verwalteten Speicher ist möglicherweise zu langsam oder beansprucht einen zu großen Speicherbereich. Das Betriebssystem für mobile Geräte stellt möglicherweise einen Speicherservice bereit, der für MQTT-Nachrichten optimal geeignet ist.

Um eine Flexibilität bei der Einhaltung der Einschränkungen von kompakten Endgeräten bereitzustellen, verfügt der MQTT-Client über zwei Schnittstellen für die Persistenz. Die Schnittstellen definieren die Operationen, die beim Speichern persistenter Nachrichten beteiligt sind. Die Schnittstellen werden in der API-Dokumentation für den MQTT-Client für Java beschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference. Sie können die Schnittstellen auf ein Gerät angepasst implementieren. Der MQTT -Client, der in Java SE ausgeführt wird, verfügt über eine Standardimplementierung der Schnittstellen, die persistente Nachrichten im Dateisystem speichern. Dabei wird das Paket `java.io` verwendet. Der Client hat auch eine Standardimplementierung für Java ME, `MqttDefaultMIDPPersistence`.

Persistenzklassen

`MqttClientPersistence`
: Übergibt eine Instanz Ihrer Implementierung von `MqttClientPersistence` an den MQTT-Client als Parameter des `MqttClient`-Konstruktors. Wenn Sie den Parameter `MqttClientPersistence` vom `MqttClient`-Konstruktor übergehen, speichert der MQTT-Client persistente Nachrichten mithilfe der Klasse `MqttDefaultFilePersistence` oder `MqttDefaultMIDPPersistence`.

MqttPersistable
MqttClientPersistence ruft MqttPersistable-Objekte mithilfe eines Speicherschlüssels ab und reiht sie ein. Sie müssen eine Implementierung von MqttPersistable sowie die Implementierung von MqttClientPersistence bereitstellen, wenn Sie die Klassen MqttDefaultFilePersistence oder MqttDefaultMIDPPersistence nicht verwenden.

MqttDefaultFilePersistence

Der MQTT-Client stellt die Klasse MqttDefaultFilePersistence bereit. Wenn Sie MqttDefaultFilePersistence in Ihrer Client-App instanziieren, können Sie das Verzeichnis, in dem persistente Nachrichten gespeichert werden sollen, als Parameter des Konstruktors MqttDefaultFilePersistence zur Verfügung stellen.

Alternativ dazu kann der MQTT-Client eine Instanz von MqttDefaultFilePersistence erstellen und Dateien in einem Standardverzeichnis speichern. Der Name des Verzeichnisses lautet *client identifier*-tcp *hostname portnumber*. Die Zeichen "\", "\\", "/", ":" und " " werden aus der Zeichenfolge des Verzeichnisnamens entfernt.

Der Pfad zum Verzeichnis entspricht dem Wert der Systemeigenschaft rcp.data. Wenn rcp.data nicht festgelegt ist, entspricht der Pfad dem Wert der Systemeigenschaft usr.data.

rcp.data ist eine Eigenschaft, die der Installation einer OSGi- oder Eclipse-Rich-Client-Plattform (RCP) zugeordnet ist.

usr.data ist das Verzeichnis, in dem der Java-Befehl, der die Anwendung gestartet hat, gestartet wurde.

MqttDefaultMIDPPersistence
MqttDefaultMIDPPersistence enthält einen Standardkonstruktor und keine Parameter. Zum Speichern von Nachrichten wird das Paket javax.microedition.rms.RecordStore verwendet.

Veröffentlichungen

Veröffentlichungen sind Instanzen von MqttMessage , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Die Nutzdaten von MqttMessage umfassen eine Bytefeldgruppe. Nachrichten sollten so klein wie möglich sein. Das MQTT-Protokoll ermöglicht eine maximale Nachrichtenlänge von 250 MB.

Ein MQTT-Clientprogramm verwendet normalerweise java.lang.String oder java.lang.StringBuffer zum Bearbeiten von Nachrichteninhalten. Für eine einfachere Verarbeitung enthält die Klasse MqttMessage eine toString-Methode, mit der die zugehörigen Nutzdaten in eine Zeichenfolge umgewandelt werden können. Verwenden Sie die Methode getBytes, um die Nutzdaten aus der Bytefeldgruppe aus java.lang.String oder java.lang.StringBuffer zu erstellen.

Die Methode getBytes wandelt eine Zeichenfolge in den Standardzeichensatz für die Plattform um. Als Standardzeichensatz wird im Allgemeinen UTF-8 verwendet. MQTT-Veröffentlichungen, die nur Text enthalten, werden normalerweise in UTF-8 codiert. Überschreiben Sie den Standardzeichensatz mit der Methode getBytes("UTF8").

In IBM WebSphere MQ wird eine MQTT-Veröffentlichung als jms-bytes-Nachricht empfangen. Die Nachricht enthält einen Ordner MQRFH2, in den die Ordner <mqtt> und <mqps> integriert sind. Der Ordner <mqtt> enthält die clientId und qos, aber dieser Inhalt kann sich in Zukunft ändern.

MqttMessage verfügt über drei zusätzliche Attribute: Servicequalität, Angabe darüber, ob die Methode aufbewahrt wird und ob es sich um eine Kopie handelt. Das Flag für eine Kopie wird nur festgelegt, wenn die Servicequalität auf "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesetzt ist. Wenn die Nachricht zuvor gesendet und nicht schnell genug vom MQTT-Client bestätigt wurde, wird die Nachricht erneut gesendet, wobei das doppelte Attribut auf true gesetzt wird.

Veröffentlichung

Erstellen Sie eine `MqttMessage`, um eine Veröffentlichung in einer MQTT -Client-App zu erstellen. Legen Sie die Nutzdaten und die Servicequalität fest, geben Sie an, ob das Objekt gespeichert werden soll, und rufen Sie die Methode `MqttTopic.publish(MqttMessage message)` auf;`MqttDeliveryToken` wird zurückgegeben und der Abschluss der Veröffentlichung ist asynchron.

Alternativ kann der MQTT -Client ein temporäres Nachrichtenobjekt aus den Parametern der Methode `MqttTopic.publish(byte [] payload, int qos, boolean retained)` erstellen, wenn er eine Veröffentlichung erstellt.

Wenn für die Veröffentlichung die Servicequalität "at least once" oder "exactly once" festgelegt ist (`QoS=1` oder `QoS=2`), ruft der MQTT-Client die Schnittstelle `MqttClientPersistence` auf. Die Nachricht wird in der Schnittstelle `MqttClientPersistence` gespeichert, bevor ein Zustellungstoken an die Anwendung zurückgegeben wird.

Die Anwendung kann mithilfe der Methode `MqttDeliveryToken.waitForCompletion` gesperrt werden, bis die Nachricht an den Server übergeben wird. Sie kann aber auch ohne Sperren fortgesetzt werden. Wenn Sie überprüfen möchten, ob Veröffentlichungen ohne Sperren übergeben wurden, registrieren Sie eine Instanz einer Callback-Klasse, die `MqttCallback` im MQTT-Client implementiert. Der MQTT-Client ruft die Methode `MqttCallback.deliveryComplete` auf, sobald die Veröffentlichung übergeben wurde. Je nach Servicequalität findet die Übergabe bei `QoS=0` direkt statt oder beansprucht bei `QoS=2` einige Zeit.

Fragen Sie mit der Methode `MqttDeliveryToken.isComplete` ab, ob die Übergabe abgeschlossen ist. Wenn der Wert von `MqttDeliveryToken.isComplete` auf `false` gesetzt ist, können Sie die Methode `MqttDeliveryToken.getMessage` zum Abrufen der Nachrichteninhalte aufrufen. Wenn nach dem Aufrufen von `MqttDeliveryToken.isComplete` das Ergebnis `true` lautet, wurde die Nachricht gelöscht und beim Aufrufen von `MqttDeliveryToken.getMessage` würde eine Nullzeigerausnahme ausgelöst. Es gibt keine integrierte Synchronisierung zwischen `MqttDeliveryToken.getMessage` und `MqttDeliveryToken.isComplete`.

Wenn die Verbindung zum Client vor dem Empfang aller anstehender Zustellungstoken getrennt wird, kann eine neue Instanz des Clients anstehende Zustellungstoken vor dem Verbinden abfragen. Bis zum Herstellen einer Verbindung zum Client werden keine neuen Übergaben abgeschlossen und die Methode `MqttDeliveryToken.getMessage` kann aufgerufen werden. Finden Sie mithilfe der Methode `MqttDeliveryToken.getMessage` heraus, welche Veröffentlichungen nicht übergeben wurden. Anstehende Zustellungstoken werden gelöscht, wenn `MqttConnectOptions.cleanSession` beim Herstellen einer Verbindung auf den Standardwert `true` gesetzt ist.

Subskribieren

Ein Warteschlangenmanager oder IBM MessageSight ist für das Erstellen von Veröffentlichungen verantwortlich, die an einen MQTT-Subskribenten gesendet werden. Der Warteschlangenmanager überprüft, ob der von einem MQTT-Client erstellte Themenfilter in einer Subskription mit der Themenzeichenfolge in einer Veröffentlichung übereinstimmt. Es kann sich dabei um eine exakte Übereinstimmung oder um eine Übereinstimmung mit Platzhalterzeichen handeln. Vor dem Weiterleiten der Veröffentlichung an den Subskribenten durch den Warteschlangenmanager überprüft der Warteschlangenmanager die Themenattribute, die der Veröffentlichung zugeordnet sind. Er folgt dabei der im Abschnitt Subskription mithilfe einer Themenzeichenfolge mit Platzhalterzeichen beschriebenen Suchprozedur, um zu ermitteln, ob ein administratives Themenobjekt dem Benutzer die Berechtigung zur Subskription erteilt.

Wenn der MQTT-Client eine Veröffentlichung mit der Servicequalität "at least once" empfängt, ruft er zur Verarbeitung der Veröffentlichung die Methode `MqttCallback.messageArrived` auf. Wenn die Servicequalität der Veröffentlichung "exactly once" (`QoS=2`) ist, ruft der MQTT-Client die Schnittstelle `MqttClientPersistence` auf, um die Nachricht nach dem Empfang zu speichern. Anschließend wird die Methode `MqttCallback.messageArrived` aufgerufen.

Von einem MQTT-Client bereitgestellte Servicequalität

Ein MQTT -Client bietet drei Servicequalitäten für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT -Client: "höchstens einmal", "mindestens einmal" und "genau einmal". Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Bei der Servicequalität einer Veröffentlichung handelt es sich um ein Attribut von `MqttMessage`. Es wird von der Methode `MqttMessage.setQos` festgelegt.

Die Methode `MqttClient.subscribe` kann die für Veröffentlichungen angewendete Servicequalität verringern, die einem Client zu einem Thema gesendet werden. Die Servicequalität einer Veröffentlichung, die an einen Subskribenten weitergeleitet wird, kann von der Servicequalität der Veröffentlichung abweichen. Zur Weiterleitung einer Veröffentlichung wird der niedrigere der beiden Werte verwendet.

At most once (Höchstens einmal)
`QoS=0`

Die Nachricht wird höchstens einmal oder gar nicht übermittelt. Die Übergabe im Netz wird nicht bestätigt.

Die Nachricht wird nicht gespeichert. Die Nachricht kann verloren gehen, wenn die Verbindung zum Client getrennt wird oder der Server fehlschlägt.

`QoS=0` ist der schnellste Übertragungsmodus. Er wird auch als "Fire-and-Forget" (Abfeuern und vergessen) bezeichnet.

Das MQTT-Protokoll verlangt nicht von Servern, Veröffentlichungen mit der Servicequalität `QoS=0` an einen Client weiterzuleiten. Wenn die Verbindung zum Client zu dem Zeitpunkt getrennt wird, an dem der Server die Veröffentlichung empfängt, wird die Veröffentlichung je nach Server möglicherweise gelöscht. Der Telemetrieservice (MQXR) löscht keine Nachrichten, die mit der Servicequalität `QoS=0` gesendet wurden. Sie werden als nicht persistente Nachrichten gespeichert und nur gelöscht, wenn der Warteschlangenmanager angehalten wird.

At least once (Mindestens einmal)
`QoS=1`

`QoS=1` ist der Standardmodus für die Übertragung.

Die Nachricht wird immer mindestens einmal übertragen. Wenn der Absender keine Bestätigung empfängt, wird die Nachricht erneut gesendet und das Flag DUP wird festgelegt, bis eine Bestätigung empfangen wird. Dadurch kann die gleiche Nachricht mehrfach an einen Empfänger gesendet und möglicherweise mehrfach verarbeitet werden.

Die Nachricht muss bis zur Verarbeitung lokal im Absender und im Empfänger gespeichert sein.

Die Nachricht wird aus dem Empfänger gelöscht, nachdem dieser die Nachricht verarbeitet hat. Wenn es sich beim Empfänger um einen Broker handelt, wird die Nachricht auf den zugehörigen Subskribenten veröffentlicht. Wenn es sich beim Empfänger um einen Client handelt, wird die Nachricht an die Subskribentenanwendung übergeben. Nach dem Löschen der Nachricht sendet der Empfänger eine Bestätigung an den Absender.

Die Nachricht wird nach dem Empfang der Bestätigung vom Empfänger aus dem Absender gelöscht.

Exactly once (Exakt einmal)
`QoS=2`

Die Nachricht wird immer exakt einmal übergeben.

Die Nachricht muss bis zur Verarbeitung lokal im Absender und im Empfänger gespeichert sein.

Bei `QoS=2` handelt es sich um die sicherste Servicequalität, aber auch um den langsamsten Übertragungsmodus. Es müssen mindestens zwei Übertragungen zwischen dem Absender und dem Empfänger stattfinden, bevor die Nachricht aus dem Absender gelöscht wird. Die Nachricht kann nach der ersten Übertragung im Empfänger verarbeitet werden.

Bei der ersten Übertragung überträgt der Absender die Nachricht und erhält eine Bestätigung vom Empfänger, dass dieser die Nachricht gespeichert hat. Wenn der Absender keine Bestätigung emp-

fängt, wird die Nachricht erneut gesendet und das Flag DUP wird festgelegt, bis eine Bestätigung empfangen wird.

Bei der zweiten Übertragung teilt der Absender dem Empfänger mit, dass dieser die Verarbeitung der Nachricht durchführen kann ("PUBREL"). Wenn der Absender keine Bestätigung der Nachricht "PUBREL" empfängt, wird die Nachricht "PUBREL" solange erneut gesendet, bis eine Bestätigung empfangen wird. Der Absender löscht die bei ihm gespeicherte Nachricht, wenn er die Bestätigung für die Nachricht "PUBREL" empfängt.

Der Empfänger kann die Nachricht in der ersten oder zweiten Phase verarbeiten, unter der Voraussetzung, dass die Nachricht nicht erneut verarbeitet wird. Wenn es sich beim Empfänger um einen Broker handelt, wird die Nachricht für Subskribenten veröffentlicht. Wenn es sich beim Empfänger um einen Client handelt, wird die Nachricht für die Subskribentenanwendung übergeben. Der Empfänger sendet dem Absender eine Beendigungsnachricht mit der Information, dass die Verarbeitung der Nachricht abgeschlossen wurde.

Ständige Veröffentlichungen und MQTT-Clients

Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema sofort an Sie weitergeleitet.

Geben Sie mithilfe der Methode `MqttMessage.setRetained` an, ob eine Veröffentlichung für ein Thema aufbewahrt wird oder nicht.

Zum Löschen einer ständigen Veröffentlichung in IBM WebSphere MQ führen Sie den WebSphere MQ-Scriptbefehl CLEAR TOPICSTR**CLEAR TOPICSTR** aus.

Wenn Sie eine Veröffentlichung ohne Nutzdaten erstellen, wird die leere Veröffentlichung an Subskribenten weitergeleitet. Andere MQTT-Broker leiten eine leere Veröffentlichung möglicherweise nicht an Subskribenten weiter.

Wenn Sie zu einem Thema, dem eine ständige Veröffentlichung zugeordnet ist, eine nicht ständige Veröffentlichung erstellen, wirkt sich dies nicht auf die ständige Veröffentlichung aus. Bereits bestehende Subskribenten erhalten die neue Veröffentlichung. Neue Subskribenten erhalten zunächst die ständige Veröffentlichung und dann die neue Veröffentlichung.

Wenn Sie eine ständige Veröffentlichung erstellen oder aktualisieren, senden Sie die Veröffentlichung mit QoS oder 1 oder 2. Wenn Sie es mit der QoS 0 senden, erstellt IBM WebSphere MQ eine nicht persistente ständige Veröffentlichung. Die Veröffentlichung wird nicht beibehalten, wenn der Warteschlangenmanager gestoppt wird.

Verwenden Sie ständige Veröffentlichungen, um den neuesten Wert einer Messung aufzuzeichnen. Neue Subskribenten des Themas mit der ständigen Veröffentlichung empfangen sofort den neuesten Wert der Messung. Falls keine neuen Messungen durchgeführt wurden, seitdem der Subskribent das Veröffentlichungsthema zuletzt subskribiert hat, erhält der Subskribent bei einer weiteren Subskription dieses Themas erneut die aktuellste ständige Veröffentlichung zu diesem Thema.

Subskriptionen

Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Erstellen Sie mithilfe der `MqttClient.subscribe`-Methoden Subskriptionen und übergeben Sie dabei mindestens einen Themenfilter und Parameter für die Servicequalität. Der Parameter für die Servicequalität legt die maximale Servicequalität fest, die der Subskribent zum Empfang einer Nachricht verwenden möchte. Nachrichten, die an diesen Client gesendet werden, können nicht mit einer höheren Servicequalität zugestellt werden. Die Servicequalität wird auf den kleineren der ursprünglichen Werte gesetzt, die galten, als die Nachricht veröffentlicht und die Stufe für die Subskription angegeben wurde. Die standardmäßige Servicequalität für den Nachrichtenempfang lautet QoS=1 (mindestens einmal).

Die Subskriptionsanforderung selbst wird mit der Einstellung QoS=1 gesendet.

Veröffentlichungen werden von einem Subskribenten empfangen, wenn der MQTT-Client die Methode `MqttCallback.messageArrived` aufruft. Die Methode `messageArrived` übergibt außerdem die Themenzeichenfolge, mit der die Nachricht an den Subskribenten übergeben wurde.

Sie können eine Subskription oder Subskriptionsgruppe mithilfe der `MqttClient.unsubscribe`-Methoden entfernen.

Ein WebSphere MQ-Befehl kann eine Subskription entfernen.Listen Sie Subskriptionen mit WebSphere MQ Explorer oder mit **runmqsc** -oder PCF-Befehlen auf. Alle MQTT-Clientsubskriptionen haben einen Namen. Sie erhalten einen Namen im folgenden Format: *ClientIdentifier*:*Topic name*

Wenn Sie den Standardwert `MqttConnectOptions`verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor der Client verbunden wird, werden alle alten Subskriptionen für den Client entfernt, wenn der Client eine Verbindung herstellt. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung auf `false` setzen, werden alle Subskriptionen, die der Client erstellt, zu allen Subskriptionen hinzugefügt, die für den Client vor dem Herstellen der Verbindung vorhanden waren. Alle Subskriptionen bleiben aktiv, wenn die Clientverbindung getrennt wird.

Eine andere Möglichkeit, um zu verstehen, wie sich das Attribut `cleanSession` auf Subskriptionen auswirkt, besteht darin, es sich als modales Attribut vorzustellen. Der Standardmodus (`cleanSession=true`) bedeutet, dass der Client nur im Rahmen der Sitzung Subskriptionen erstellt und Veröffentlichungen empfängt. Im alternativen Modus (`cleanSession=false`) sind Subskriptionen permanent. Der Client kann Verbindungen herstellen und trennen, seine Subskriptionen bleiben aktiv. Bei der Wiederherstellung einer Verbindung empfängt der Client alle nicht zugestellten Veröffentlichungen. Solange die Verbindung besteht, kann er die Gruppe der Subskriptionen, die in seinem Auftrag aktiv sind, ändern.

Sie müssen den Modus `cleanSession` festlegen, bevor Sie eine Verbindung herstellen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie die Modi von `cleanSession=false` in `cleanSession=true`ändern, werden alle vorherigen Subskriptionen für den Client und alle Veröffentlichungen, die nicht empfangen wurden, verworfen.

Veröffentlichungen, die aktiven Subskriptionen entsprechen, werden unmittelbar bei ihrer Veröffentlichung an den Client gesendet. Veröffentlichungen, die aktiven Subskriptionen entsprechen, werden unmittelbar bei ihrer Veröffentlichung an den Client gesendet. Wenn der Client nicht verbunden ist, werden sie an den Client gesendet, sobald dieser die Verbindung zu demselben Server wiederherstellt. Dabei muss die Client-ID identisch sein und `MqttConnectOptions.cleanSession` muss auf `false` gesetzt sein.

Subskriptionen für einen bestimmten Client werden über die Client-ID ermittelt. Sie können den Client von einem anderen Clientgerät aus erneut mit demselben Server verbinden und weiterhin dieselben Subskriptionen nutzen und nicht zugestellte Veröffentlichungen empfangen.

Themenzeichenfolgen und Themenfilter in MQTT-Clients

Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM WebSphere MQ identisch.

Mit Themenzeichenfolgen werden Veröffentlichungen an Subskribenten gesendet. Erstellen Sie mit der Methode `MqttClient.getTopic(java.lang.String topicString)`eine Themenzeichenfolge.

Mit Themenfiltern werden Themen subskribiert und Veröffentlichungen empfangen. Themenfilter können Platzhalterzeichen enthalten. Mithilfe von Platzhalterzeichen können mehrere Themen subskribiert werden. Erstellen Sie einen Topicfilter mit einer Subskriptionsmethode, z. B. `MqttClient.subscribe(java.lang.String topicFilter)`.

Themenzeichenfolgen

Die Syntax einer IBM WebSphere MQ-Themenzeichenfolge ist im Abschnitt Themenzeichenfolgen beschrieben. Die Syntax von MQTT -Themenzeichenfolgen wird in der Klasse `MqttClient` in der API-Dokumentation für MQTT-Client für Javabeschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

Die Syntax ist bei allen Themenzeichenfolgen nahezu identisch. Es gibt vier geringfügige Unterschiede:

1. Themenzeichenfolgen, die von MQTT -Clients an IBM WebSphere MQ gesendet werden, müssen der Konvention für Warteschlangenmanagernamen entsprechen. Vor allem dürfen Themenzeichenfolgen keine Silbentrennungsstriche enthalten.
2. Die maximale Länge ist unterschiedlich. IBM WebSphere MQ-Themenzeichenfolgen sind auf 10.240 Zeichen begrenzt. Ein MQTT-Client kann Themenzeichenfolgen mit bis zu 65.535 Bytes erstellen.
3. Eine Themenzeichenfolge, die von einem MQTT-Client erstellt wurde, kann kein Nullzeichen enthalten.
4. In WebSphere Message Broker war ein Thema der Ebene null (`'...//...'`) ungültig. IBM WebSphere MQ unterstützt Themen der Ebene null.

Anders als die Publish/Subscribe-Funktion von IBM WebSphere MQ kennt das Protokoll `mqttv3` nicht das Konzept eines verwalteten Themenobjekts. Es kann keine Themenzeichenfolge aus einem Themenobjekt und einer Themenzeichenfolge erstellt werden. Eine Themenzeichenfolge wird jedoch einem Verwaltungsthema in WebSphere MQ zugeordnet. Die Zugriffssteuerung, die dem Verwaltungsthema zugeordnet ist, bestimmt, ob eine Veröffentlichung für ein Thema veröffentlicht oder verworfen wird. Die Attribute, die auf eine Veröffentlichung angewendet werden, wenn sie an Subskribenten weitergeleitet wird, werden durch die Attribute des Verwaltungsthemas beeinflusst.

Themenfilter

Die Syntax eines IBM WebSphere MQ-Themenfilters ist im Abschnitt Themenbasiertes Platzhalterschema beschrieben. Die Syntax der Themenfilter, die Sie mit einem MQTT-Client erstellen können, ist in der Klasse `MqttClient` in der API-Dokumentation für den MQTT-Client für Java beschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

Die Syntax ist bei allen Themenfiltern nahezu identisch. Lediglich die Art und Weise, wie verschiedene MQTT-Broker einen Themenfilter interpretieren, ist unterschiedlich. In WebSphere Message Broker Version 6 kann ein Platzhalterzeichen für mehrere Ebenen nur am Ende eines Themenfilters verwendet werden. In WebSphere MQ kann ein Platzhalterzeichen für mehrere Ebenen auf jeder Ebene der Baumstruktur verwendet werden (Beispiel: `USA/#/Dutchess County`).

Programmierreferenz für MQTT-Clients

Hier finden Sie Links zum Mobile Messaging und M2M Client-Pack und zur zugehörigen Client-API-Dokumentation.

Im Mobile Messaging und M2M Client-Pack sind die MQTT-Clientbibliotheken mit ihrer generierten API-Dokumentation enthalten. Sie können das Clientsoftwarepaket von der Website Downloads für IBM Messaging-Community herunterladen.

Unter den folgenden Links zum Eclipse Paho-Projekt finden Sie Onlinekopien der aktuellsten API-Dokumentation:

- MQTT-Client für Java-Klassen
- MQTT-Clientbibliothek für C
- Asynchrone MQTT-Clientbibliothek für C

Anmerkung:

1. Verknüpfen Sie MQTT Java -Anwendungen mit dem Paket `org.eclipse.paho.client.mqttv3` und nicht mit `com.ibm.micro.client.mqttv3`. ersetzt. Das Paket `com.ibm.micro.client.mqttv3` wird zur Unterstützung vorhandener MQTT Java -Anwendungen bereitgestellt.

2. V 7.5.0.1 Verbinden Sie MQTT -Client-Apps für C mit der Bibliothek `MQTTAsync` und nicht mit der Bibliothek `MQTTClient` . `MQTTClient` wird zur Unterstützung vorhandener MQTT -Apps für C bereitgestellt.
3. Der MQTT-Messaging-Client für JavaScript erfordert einen MQTT-Server, der WebSockets unterstützt. Zum Beispiel IBM WebSphere MQ Version 7.5 und neuere Versionen erfüllen diese Voraussetzung.

Erste Schritte mit MQTT-Servern

Von IBM und anderen Anbietern sind Messaging-Server verfügbar, die das MQTT-Transportprotokoll unterstützen. Der grundlegendste MQTT-Server ermöglicht mobilen Apps und Geräten, die von MQTT-Clientbibliotheken unterstützt werden, den Austausch von Nachrichten. IBM WebSphere MQ und IBM MessageSight sind MQTT-Server von IBM. Neben ihrer Funktion als grundlegende MQTT-Server tauschen sie auch Nachrichten zwischen MQTT-Client-Apps und Unternehmens-Apps aus. Alle MQTT-Server von IBM unterstützen das MQTT version 3.1-Protokoll und MQTT über das WebSocket protocol.

Aktuelle MQTT-Server von IBM

IBM WebSphere MQ

- IBM WebSphere MQ bietet auf Unternehmen zugeschnittenes Messaging. Die Telemetrie-Komponente ermöglicht es IBM WebSphere MQ, auch als MQTT-Server zu fungieren.
- Dadurch werden Ihre mobilen Apps, Machine-to-Machine (M2M)-Apps und gerätebasierten Apps unterstützt und diese können Nachrichten mit Messaging-Apps der Enterprise-Klasse wie IBM WebSphere MQ- und JMS-Apps austauschen.
- Die IBM WebSphere MQ-Installation enthält eine Kopie des MQTT-SDK von IBM. Dieses SDK stellt Beispiel-Apps des MQTT-Clients und MQTT-Clientbibliotheken bereit, die diese Apps unterstützen.

  Anmerkung: Um die aktuellste Version dieses SDK zu erhalten, laden Sie das Mobile Messaging und M2M Client-Pack herunter. Weitere Informationen finden Sie im Abschnitt „Erste Schritte mit MQTT-Clients” auf Seite 11.
- Die MQTT-Unterstützung war zum ersten Mal in IBM WebSphere MQ Version 7.0.1 enthalten. Vollständige Informationen zu jedem Release von IBM WebSphere MQ finden Sie in den folgenden Produktdokumentationen:
  - WebSphere MQ Telemetry Version 7.5
  - WebSphere MQ Telemetry Version 7.1

  Eine kurze Einführung in IBM WebSphere MQ und eine Beschreibung der ersten Schritte mit der Komponente IBM WebSphere MQ Telemetry finden Sie im Abschnitt „IBM WebSphere MQ als MQTT-Server” auf Seite 144.

IBM MessageSight

- IBM MessageSight ist ein gerätebasierter MQTT-Server, der gleichzeitig mit einer erheblichen Anzahl von MQTT-Clients eine Verbindung herstellen kann und die Leistung und Skalierbarkeit bereitstellen kann, die für die stets wachsende Anzahl an mobilen Geräten und Sensoren erforderlich sind. Er unterstützt das MQTT version 3.1-Protokoll und MQTT über das WebSocket protocol.

- Im Folgenden sind die wichtigsten Funktionen und Vorteile von IBM MessageSight als MQTT-Server aufgeführt:
  - Hohe Leistung, Zuverlässigkeit und skalierbares Messaging.
  - Speziell für M2M- (Machine-to-Machine) und IoT-Szenarios (IoT - Internet of Things) entwickelt, unterstützt große Communitys für gleichzeitig verbundene Endpunkte.
  - Einfache Installation und Verwendung. Der Server ist in weniger als 30 Minuten betriebsbereit.
  - Unterstützung für native mobile Apps, die Android und iOS enthalten.
  - Integration mit IBM WebSphere MQ als Publish/Subscribe-Broker
- Eine schnelle Einführung in IBM MessageSight, erhalten Sie in der MessageSight-Einführung auf YouTube und in der MessageSight-Ankündigung. Ausführliche technische Informationen finden Sie in der MessageSight-Produktdokumentation.

IBM WebSphere MQ Telemetry daemon for devices

- Dieser Dämon wird auch als IBM WebSphere MQ Telemetry advanced client for C bezeichnet. Dies ist ein MQTT-Server mit geringem Speicherbedarf, der normalerweise an Satellitenstandorten oder auf Geräten am Rand des Netzes ausgeführt wird, z. B. auf Set-Top-Boxen, Telemetrieeinheiten oder POS-Terminals.
- Eine typische Verwendung besteht darin, viele MQTT -Clientverbindungen zu konzentrieren, die dann über das Internet mit IBM WebSphere MQ in einer einzigen MQTT -Verbindung verbunden werden. Sie können beispielsweise eine große Zahl von Sensoren in einem Gebäude installieren, sie mit dem IBM WebSphere MQ Telemetry daemon for devices verbinden und den Dämon mit IBM WebSphere MQ verbinden.
- Das IBM WebSphere MQ Telemetry daemon for devices ist in IBM WebSphere MQ enthalten. Für eine Verbindung mit IBM WebSphere MQ ist eine separate Lizenz erforderlich. Weitere Informationen finden Sie im Abschnitt IBM United States Software Announcement 212-091.

Really Small Message Broker

- Really Small Message Broker (RSMB) ist eine Version des IBM WebSphere MQ Telemetry daemon for devices. Der Hauptunterschied liegt in der Verwendung. RSMB ist ein kleiner Testserver, der von IBM alphaWorks zur Verfügung gestellt wird und für Auswertungen von und Experimente mit MQTT-basierten Lösungen bestimmt ist. RSMB unterstützt MQTT auf einer Reihe von Linux-Plattformen, unter Windows XP, unter Apple Mac OS X Leopard und unter Unslung (Linksys NSLU12).

Vorherige MQTT-Server von IBM

WebSphere Message Broker *(jetzt bekannt als IBM Integration Bus)*

- WebSphere Message Broker Version 6 stellte einen eigenen MQTT-Server bereit. Die Unterstützung wurde in WebSphere Message Broker Version 7 durch die Telemetriekomponente von IBM WebSphere MQ ersetzt.

Andere MQTT-Server

Auf MQTT.org finden Sie auf der Seite Software eine aktuelle Liste mit MQTT-Servern und -Brokern, einschließlich Open-Source-Servern.

Zugehörige Tasks

„Erste Schritte mit MQTT-Clients" auf Seite 11
Sie können mit der Entwicklung einer mobilen App oder einer Machine-to-Machine (M2M)-App beginnen, indem Sie eine Beispiel-App für den MQTT-Client erstellen und ausführen, die eine MQTT-Clientbibliothek verwendet. Die Beispiel-Apps und zugehörigen Clientbibliotheken im Mobile Messaging und M2M Client-Pack von IBM verfügbar. Es gibt Versionen der Apps und Clientbibliotheken, die in Java, in JavaScriptund in C geschrieben wurden. Sie können diese Apps auf den meisten Plattformen und Geräten ausführen, einschließlich Android -Geräten und -Produkten von Apple.

IBM WebSphere MQ als MQTT-Server

Eine Einführung in die Verwendung des MQTT-Servers, der im Lieferumfang von IBM WebSphere MQ enthalten ist.

Führen Sie zum Einstieg die Schritte in den folgenden Abschnitten aus:

- „IBM WebSphere MQ installieren" auf Seite 144
- „MQTT-Service über die Befehlszeile konfigurieren" auf Seite 146
- „MQTT-Service über IBM WebSphere MQ Explorer konfigurieren" auf Seite 149

Anmerkung: Für einen schnellen Einstieg können Sie das Beispiel für die Befehlszeilenschnittstelle verwenden. Wenn sich Ihre Konfiguration jedoch erheblich von dem Beispiel unterscheidet, benötigen Sie weiteres Wissen und Know-how, um die Befehlszeilenschnittstelle effektiv verwenden zu können. Verwenden Sie die IBM WebSphere MQ Explorer-Schnittstelle, um sowohl erste Schritte als auch Standardkonfigurationsaufgaben ohne großen Aufwand auszuführen.

Wichtige Informationen zu den Konzepten von IBM WebSphere MQ Telemetry finden Sie in den folgenden Beiträgen der Produktdokumentation von IBM WebSphere MQ:

- Telemetriegeräte mit einem Warteschlangenmanager verbinden
- Telemetrieservice (MQXR)
- Telemetriekanäle

Zugehörige Informationen

Warteschlangenmanager für Telemetrie unter Linux und AIX konfigurieren
Warteschlangenmanager für Telemetry unter Windows konfigurieren
Verteilte Steuerung von Warteschlangen für das Senden von Nachrichten an MQTT-Clients konfigurieren
WebSphere MQ Telemetry verwalten

IBM WebSphere MQ installieren

Folgen Sie den hier beschriebenen Anweisungen, um IBM WebSphere MQ anzufordern und zu installieren und IBM WebSphere MQ Telemetry unter Windows oder Linux zu konfigurieren.

Vorbereitende Schritte

Informationen zu den Betriebssystemen, die vom MQTT-Service, der unter IBM WebSphere MQ aktiv ist, unterstützt werden, finden Sie unter IBM WebSphere MQ Telemetry-Systemvoraussetzungen.

Eine Kopie der Installationsmedien für IBM WebSphere MQ sowie die passende Lizenz erhalten Sie wie folgt:

1. Bitten Sie Ihren IBM WebSphere MQ-Administrator um das Installationsmaterial und darum, zu bestätigen, dass Sie die Lizenzvereinbarung akzeptieren können.

2. Laden Sie eine 90 Tage gültige Testversion von IBM WebSphere MQ herunter. (siehe Auswertung: IBM WebSphere MQ).
3. Kaufen Sie IBM WebSphere MQ. (siehe IBM WebSphere MQ Produktseite).

Informationen zu diesem Vorgang

Installieren Sie IBM WebSphere MQ als `root` unter Linuxund als Administrator unter Windows. Wählen Sie bei der Installation die zusätzlichen Optionen `Telemetry Service` und `Telemetry Clients` aus, um die IBM WebSphere MQ Telemetry-Komponente zu installieren. Erstellen Sie eine Benutzer-ID zur Verwaltung von IBM WebSphere MQ und überprüfen Sie, ob eine Gastbenutzer-ID definiert ist. Die Gastbenutzer-ID wird in der MQTT-Beispielservicekonfiguration verwendet, um MQTT Zugriff auf IBM WebSphere MQ zu gewähren.

Starten Sie nach der Installation von IBM WebSphere MQ den MQTT-Service, indem Sie die Schritte im Abschnitt „MQTT-Service über die Befehlszeile konfigurieren” auf Seite 146 oder „MQTT-Service über IBM WebSphere MQ Explorer konfigurieren” auf Seite 149 ausführen.

Vorgehensweise

1. Melden Sie sich als `root` unter Linuxoder als Administrator unter Windowsan.
2. Installieren Sie IBM WebSphere MQ.

   Befolgen Sie die Anweisungen im Abschnitt WebSphere MQ -Server unter Linux installieren oder WebSphere MQ -Server unter Windows installieren. Wählen Sie `Telemetry Service` und `Telemetry Clients` aus, um die IBM WebSphere MQ Telemetry-Komponente zu installieren.

   Beachten Sie unter Linux die Anweisung im Abschnitt "What to do next" (Weitere Schritte), um Ihre Installation zur primären Installation zu machen. Machen Sie diese Installation auch dann zur primären Installation, wenn sie die einzige IBM WebSphere MQ-Installation auf Ihrer Workstation ist. Informationen finden Sie unter Einzelinstallation von WebSphere MQ Version 7.1 oder höher, konfiguriert als primäre Installation.

   Damit Sie die Beispielkonfigurationsanweisungen exakt ausführen können, müssen Sie die Installation zur primären Installation machen.

   Mehrere Installationen: Wenn Sie mit einer nicht primären Installation arbeiten möchten, führen Sie den Befehl setmqenv aus. Mit ihm können Sie die IBM WebSphere MQ-Umgebung in einem Befehlsfenster auf Ihrer Workstation einrichten. Weitere Informationen finden Sie unter Mehrfachinstallationen.

   Sofern Sie die vom Installationsprogramm angebotene Standardinstallationsposition akzeptiert haben, wird IBM WebSphere MQ in folgenden Verzeichnissen installiert:

   Linux (64 Bit)

   ```
   /opt/mqm
   ```

   Windows (32 Bit)

   ```
   C:\Program Files\IBM\WebSphere MQ
   ```

   Windows (64 Bit)

   ```
   C:\Program Files (x86)\IBM\WebSphere MQ
   ```

   Das Installationsverzeichnis wird als *MQ_INSTALLATION_PATH* angezeigt.
3. Optional: Fügen Sie den Benutzer, als der Sie IBM WebSphere MQ verwalten werden, zur Gruppe `mqm` auf dieser Workstation hinzu.

   Dieser Schritt ist unter Windows optional, weil Sie IBM WebSphere MQ als Windows -Administrator verwalten können. Informationen finden Sie unter Berechtigung für die Verwaltung von WebSphere MQ auf UNIX- und Windows-Systemen.

Wenn Ihre Windows-Workstation zu einer Domäne gehört, finden Sie weitere Informationen unter Windows 2000-Domäne mit vom Standard abweichenden Sicherheitsberechtigungen oder Windows 2003- und Windows Server 2008-Domäne mit standardmäßigen Sicherheitsberechtigungen.

Unter Linuxerstellt das Installationsprogramm einen Benutzer `mqm`als Mitglied der Gruppe `mqm`. Geben Sie diesem Benutzer ein Kennwort oder erstellen Sie einen anderen Benutzer mit `mqm` als seiner Primärgruppe.

4. Optional: Melden Sie sich als der Benutzer an, den Sie zu einem Mitglied der Gruppe `mqm` gemacht haben.

   Dieser Schritt ist unter Windows optional, weil Sie IBM WebSphere MQ als Windows -Administrator verwalten können.

5. Überprüfen Sie, ob die Gastbenutzer-ID auf Ihrer Workstation definiert ist.

   Die Gastbenutzer-ID lautet `"guest"` unter Windows und `"nobody"` unter Linux. Die Gastbenutzer-ID benötigt keine Betriebssystemberechtigungen.

Ergebnisse

Sie haben IBM WebSphere MQ als primäre IBM WebSphere MQ-Installation auf Ihrer Workstation installiert und die Gruppe `mqm` erstellt. Die Installation erteilt Mitgliedern der Gruppe `mqm` die Berechtigung zur Verwaltung von IBM WebSphere MQ . Mitglieder der Administratorgruppe unter Windows sind ebenfalls berechtigt, IBM WebSphere MQ zu verwalten.

Nächste Schritte

1. Konfigurieren Sie den MQTT-Service über die Befehlszeile oder IBM WebSphere MQ Explorer (siehe „MQTT-Service über IBM WebSphere MQ Explorer konfigurieren” auf Seite 149 bzw. „MQTT-Service über die Befehlszeile konfigurieren” auf Seite 146).
2. Testen Sie Ihre Android-, iOS-, WebSockets-, Java- und "C"-MQTT-Clients.
3. Entfernen Sie nach Abschluss der Tests den Warteschlangenmanager und den MQTT -Service, indem Sie den Befehl *MQ_INSTALLATION_PATH*`\mqxr\samples\CleanupMQM.bat` unter Windows und *MQ_INSTALLATION_PATH*`/mqxr/samples/CleanupMQM.sh` unter Linuxausführen.

Zugehörige Informationen

WebSphere MQ Telemetry installieren

WebSphere MQ unter Linux installieren

WebSphere MQ-Server unter Windows installieren

MQTT-Service über die Befehlszeile konfigurieren

Folgen Sie den Anweisungen in diesem Abschnitt, um IBM WebSphere MQ über die Befehlszeile für die Ausführung der IBM WebSphere MQ Telemetry-Beispielanwendungen zu konfigurieren. Die Schritte zeigen, wie ein Script ausgeführt wird, um einen MQTT -Service auf einem neuen WS-Manager mit dem Namen `MQXR_SAMPLE_QM`zu erstellen.

Vorbereitende Schritte

Sie müssen Verwaltungszugriff auf einen IBM WebSphere MQ-Warteschlangenmanager besitzen, um den MQTT-Service konfigurieren zu können. Es gibt mehrere Möglichkeiten, Zugriff auf einen Warteschlangenmanager zu erhalten:

1. Rufen Sie eine Kopie von IBM WebSphere MQ ab und erstellen Sie einen Warteschlangenmanager auf Ihrer Linux- oder Windows-Workstation. Folgen Sie den Anweisungen im Abschnitt „IBM WebSphere MQ installieren” auf Seite 144, um IBM WebSphere MQ abzurufen und zu installieren. Denken Sie daran, dass Sie bei der Installation auch die Optionen `Telemetrieservice` und `Telemetrieclients` auswählen müssen. Sie können auch eine bestehende Installation ändern, um diese Optionen hinzuzufügen.

2. Bitten Sie einen IBM WebSphere MQ-Administrator, Ihnen Verwaltungszugriff auf einen Warteschlangenmanager auf einem Server, auf dem IBM WebSphere MQ Telemetry als eine Option installiert ist, zu erteilen. V 7.5.0.1 Zusätzlich zum Namen des Warteschlangenmanagers benötigen Sie mindestens zwei TCP/IP-Ports für MQTT und für MQTT über WebSockets. Wenn Sie planen, Verbindungen zu sicheren Clients herzustellen, sind mindestens zwei weitere Ports erforderlich.

Um diese Schritte genau wie in der Aufgabe beschrieben ausführen zu können, müssen Sie in der Lage sein, einen Warteschlangenmanager mit dem Namen `MQXR_SAMPLE_QM` zu erstellen, und TCP/IP-Port 1883 darf nicht belegt sein.

Informationen zu diesem Vorgang

In dieser Aufgabe führen Sie ein Script aus, das einen Warteschlangenmanager erstellt und dann den MQTT-Service konfiguriert, sodass er an Port 1883 für MQTT V3.1-Clientverbindungen empfangsbereit ist.Die Konfiguration erteilt jedem die Berechtigung, zu allen Themen Veröffentlichungen und Subskriptionen durchzuführen. Die Sicherheits- und Zugriffssteuerungskonfiguration ist minimal und nur für einen Warteschlangenmanager vorgesehen, der sich in einem sicheren Netz mit eingeschränktem Zugriff befindet. Um IBM WebSphere MQ und MQTT in einer nicht sicheren Umgebung ausführen zu können, ist eine weiterreichende Sicherheitskonfiguration erforderlich. Informationen zur Sicherheitskonfiguration für IBM WebSphere MQ und MQTT finden Sie unter den zugehörigen Links am Ende dieses Abschnitts.

Vorgehensweise

1. Melden Sie sich mit einer Benutzer-ID an, die über Administratorberechtigung für IBM WebSphere MQverfügt.

   Informationen zum Definieren einer Benutzer-ID mit Administratorberechtigung für IBM WebSphere MQfinden Sie in Schritt 3 in „IBM WebSphere MQ installieren” auf Seite 144.
2. Öffnen Sie ein Befehlsfenster und führen Sie das Beispielbefehlsscript aus, um den Beispielwarteschlangenmanager `MQXR_SAMPLE_QM` und den MQTT-Service zu erstellen und zu starten.

   Der Pfad zum Beispielscript lautet `%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat` unter Windows und *`MQ_INSTALLATION_PATH`*`/mqxr/samples/SampleMQM.sh` unter Linux.

   Geben Sie folgenden Befehl ein, um den Warteschlangenmanager zu erstellen und zu konfigurieren:

   - Windows

     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```

   - Linux

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

Ergebnisse

Das Beispiel erstellt einen MQTT -Kanal namens `PlainText` mit den folgenden Eigenschaften unter Windows:

```
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
```

Die Kanaleigenschaften unter Linux entsprechen denen unter Windows, mit Ausnahme von `com.ibm.mq.MQXR.UserName=nobody`.

MQTT V3.1 Clients, die eine Verbindung zu Port 1883 herstellen, greifen auf IBM WebSphere MQ mit der in der Variablen *com.ibm.mq.MQXR.UserName*festgelegten Benutzer-ID zu. Das Beispielscript berechtigt die Benutzer-ID mit folgenden IBM WebSphere MQ-Befehlen:

```
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
```

Mit dem ersten Befehl erhält der Benutzer die Berechtigung, Veröffentlichungen und Subskriptionen für Themen durchzuführen, die ihre Berechtigungen vom Basisthema übernehmen. Mit dem zweiten Befehl erhält der Benutzer die Berechtigung, Nachrichten in die Übertragungswarteschlange SYSTEM.MQTT.TRANSMIT.QUEUE einzureihen. Der MQTT-Service sendet Nachrichten in der SYSTEM.MQTT.TRANSMIT.QUEUE als Veröffentlichungen an MQTT-Subskribenten.

Das Script startet den MQTT-Service auf dem Warteschlangenmanager, um Port 1883 auf Verbindungen zu überwachen.

Nächste Schritte

Führen Sie diese Schritte aus, um die Verbindung zu testen, indem Sie die MQTT V3.1-Java-Beispielanwendung ausführen.

Die Quelle für die Beispielanwendung Java befindet sich in der Datei `MQTTV3Sample.java` .

Zur Ausführung des Beispiels sind zwei Befehlsfenster erforderlich. Führen Sie die Beispielanwendung in einem Fenster als Subskribent und im anderen Fenster als Bereitsteller von Veröffentlichungen aus.

- Windows Starten Sie den Subskribenten mit dem Befehl

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
  ```

  Führen Sie zum Veröffentlichen folgenden Befehl aus:

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
  ```

- Linux Starten Sie den Subskribenten mit dem Befehl

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
  ```

  Führen Sie zum Veröffentlichen folgenden Befehl aus:

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
  ```

Der Bereitsteller und der Subskribent von Veröffentlichungen schreiben Ausgaben in ihre Befehlsfenster:

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

Abbildung 27. Ausgabe vom Bereitsteller

```
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
```

Abbildung 28. Ausgabe vom Subskribenten

Der Server ist jetzt zum Testen Ihrer MQTT V3.1 -App bereit.

Zugehörige Tasks

MQTT-Service mit WebSphere MQ Explorer konfigurieren
Folgen Sie diesen Anweisungen, um IBM WebSphere MQ mithilfe von IBM WebSphere MQ Explorer für die Ausführung der IBM WebSphere MQ Telemetry-Beispielclients zu konfigurieren. Dabei wird gezeigt, wie ein MQTT-Service durch Ausführung des Konfigurationsassistenten `Define sample` (Beispiel definieren) erstellt wird.

Zugehörige Informationen

WebSphere MQ Telemetry
Anwendungen für WebSphere MQ Telemetry entwickeln
WebSphere MQ Telemetry verwalten
WebSphere MQ Telemetry-Sicherheit

MQTT-Service über IBM WebSphere MQ Explorer konfigurieren

Folgen Sie diesen Anweisungen, um IBM WebSphere MQ mithilfe von IBM WebSphere MQ Explorer für die Ausführung der IBM WebSphere MQ Telemetry-Beispielclients zu konfigurieren. Dabei wird gezeigt, wie ein MQTT-Service durch Ausführung des Konfigurationsassistenten `Define sample` (Beispiel definieren) erstellt wird.

Vorbereitende Schritte

Sie müssen Verwaltungszugriff auf einen IBM WebSphere MQ-Warteschlangenmanager besitzen, um den MQTT-Service konfigurieren zu können. Es gibt mehrere Möglichkeiten, Zugriff auf einen Warteschlangenmanager zu erhalten:

1. Rufen Sie eine Kopie von IBM WebSphere MQ ab und erstellen Sie einen Warteschlangenmanager auf Ihrer Linux- oder Windows-Workstation. Folgen Sie den Anweisungen im Abschnitt „IBM WebSphere MQ installieren” auf Seite 144, um IBM WebSphere MQ abzurufen und zu installieren. Denken Sie daran, dass Sie bei der Installation auch die Optionen `Telemetrieservice` und `Telemetrieclients` auswählen müssen. Sie können auch eine bestehende Installation ändern, um diese Optionen hinzuzufügen.
2. Bitten Sie einen IBM WebSphere MQ-Administrator, Ihnen Verwaltungszugriff auf einen Warteschlangenmanager auf einem Server, auf dem IBM WebSphere MQ Telemetry als eine Option installiert ist, zu erteilen. V 7.5.0.1 Zusätzlich zum Namen des Warteschlangenmanagers benötigen Sie mindestens zwei TCP/IP-Ports für MQTT und für MQTT über WebSockets. Wenn Sie planen, Verbindungen zu sicheren Clients herzustellen, sind mindestens zwei weitere Ports erforderlich.

Um diese Schritte genau wie in der Aufgabe beschrieben ausführen zu können, müssen Sie in der Lage sein, einen Warteschlangenmanager mit dem Namen `MQXR_SAMPLE_QM` zu erstellen, und TCP/IP-Port 1883 darf nicht belegt sein.

Informationen zu diesem Vorgang

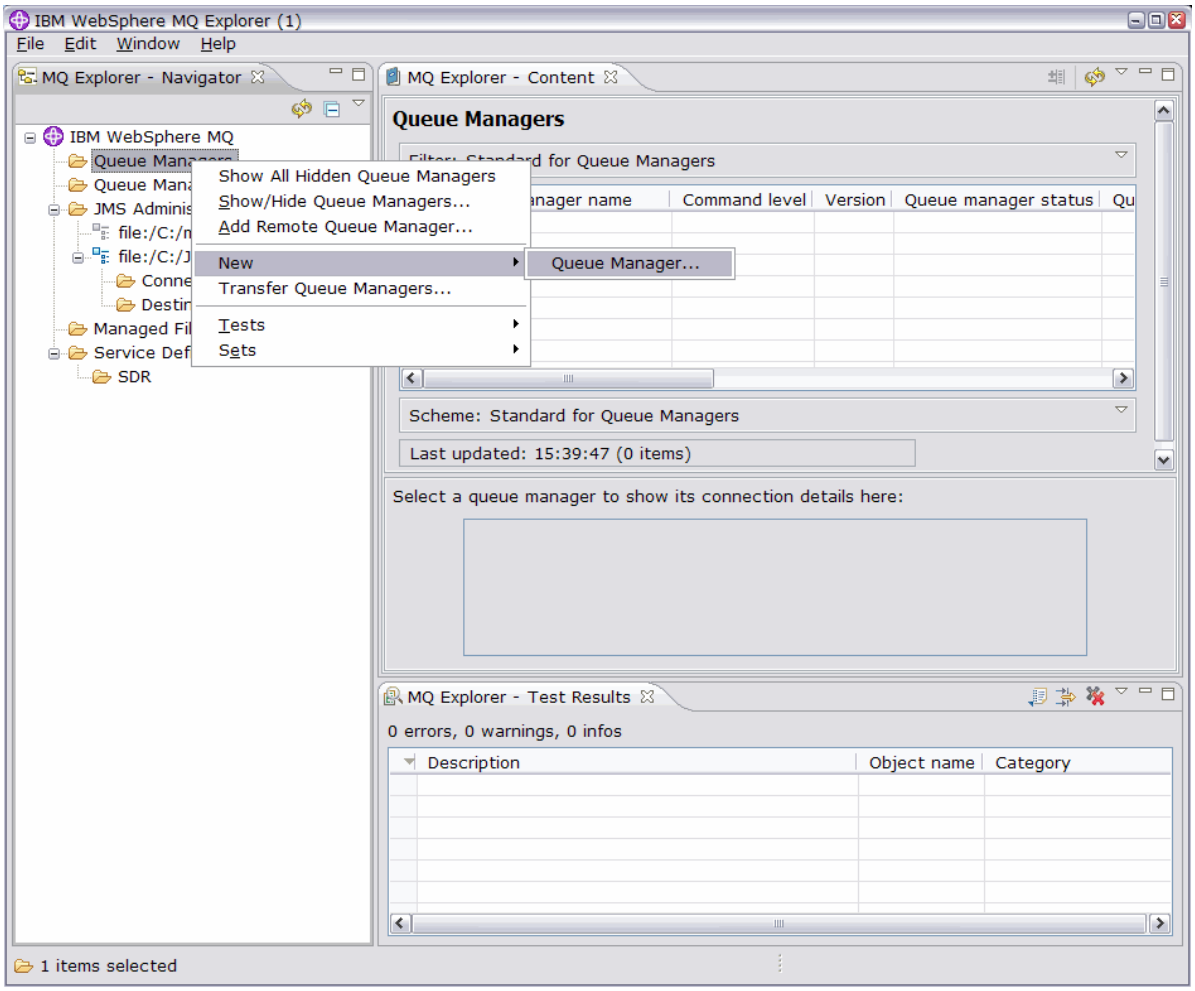
In dieser Task führen Sie den IBM WebSphere MQ Explorer `Define sample` -Konfigurationsassistenten aus, um einen MQTT -Service für den Empfang von MQTT V3.1 -Clientverbindungen an Port 1883 zu erstellen. Die Konfiguration erteilt jedem die Berechtigung, zu allen Themen Veröffentlichungen und Subskriptionen durchzuführen. Die Sicherheits- und Zugriffssteuerungskonfiguration ist minimal und nur für einen Warteschlangenmanager vorgesehen, der sich in einem sicheren Netz mit eingeschränktem Zugriff befindet. Um IBM WebSphere MQ und MQTT in einer nicht sicheren Umgebung ausführen zu können, ist eine weiterreichende Sicherheitskonfiguration erforderlich. Informationen zur Sicherheitskonfiguration für IBM WebSphere MQ und MQTT finden Sie unter den zugehörigen Links am Ende dieses Abschnitts.

Vorgehensweise

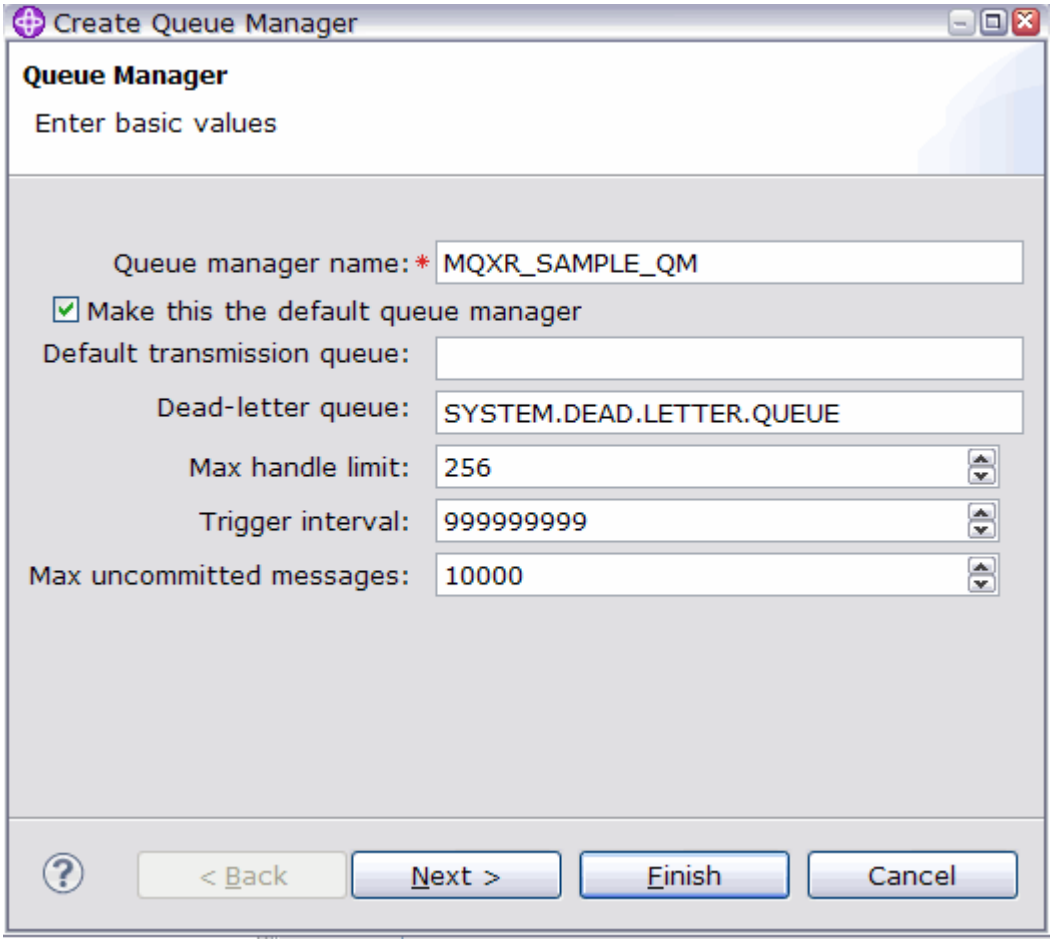
1. Melden Sie sich mit einer Benutzer-ID an, die über Administratorberechtigung für IBM WebSphere MQverfügt.

Informationen zum Definieren einer Benutzer-ID mit Administratorberechtigung für IBM WebSphere MQfinden Sie in Schritt 3 in „IBM WebSphere MQ installieren” auf Seite 144.

2. Öffnen Sie ein Befehlsfenster und führen Sie den IBM WebSphere MQ Explorer -Befehl **`strmqcfg`** aus, um IBM WebSphere MQ Explorerzu starten.
3. Einen WS-Manager erstellen
   a) Starten Sie den Assistenten **New Queue Manager** (Neuer Warteschlangenmanager).

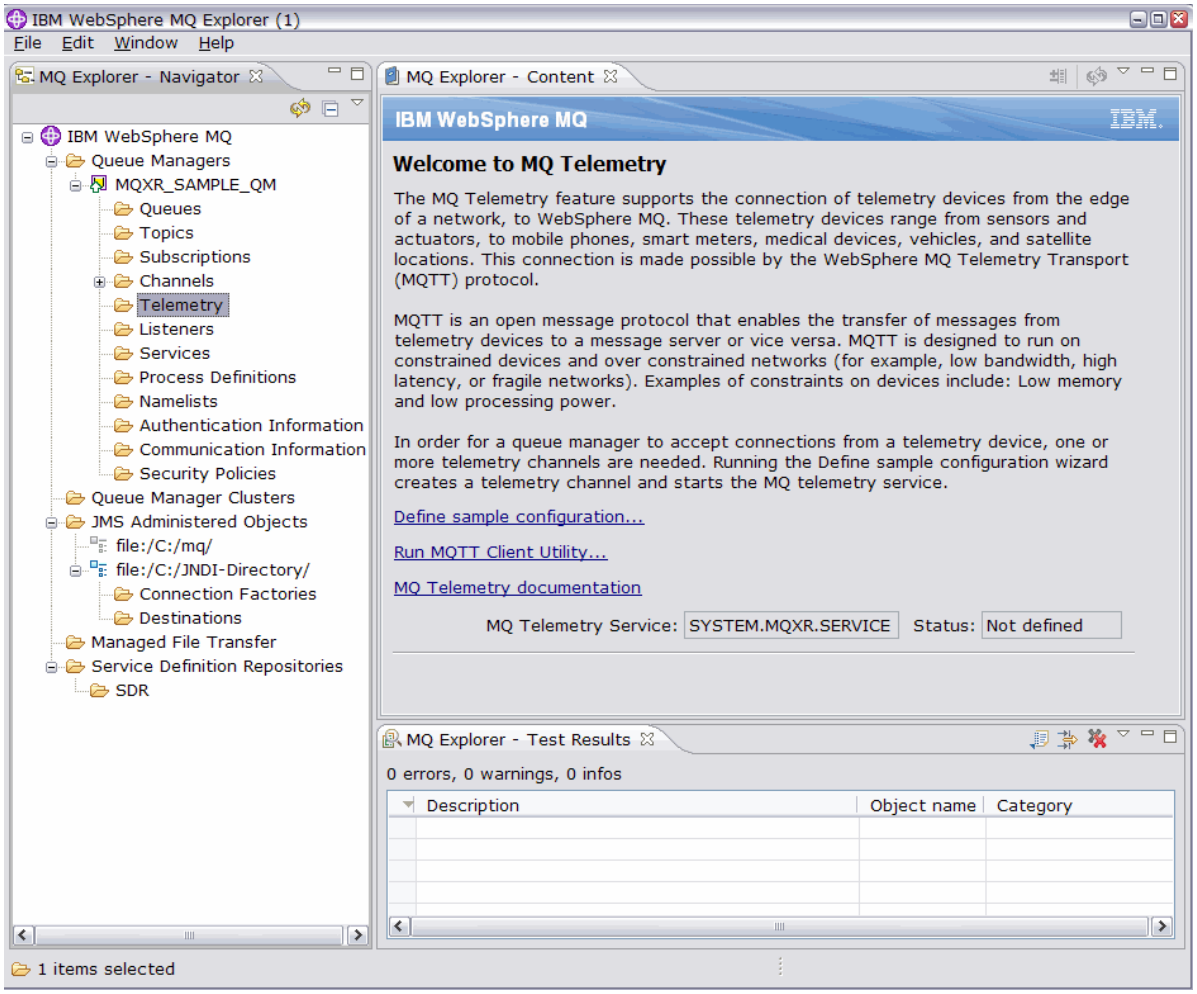


   b) Geben Sie im Feld **Queue manager name** den Namen des Warteschlangenmanagers und im Feld **Dead-letter queue** den Namen der Warteschlange für nicht zustellbare Nachrichten ein. Machen Sie den Warteschlangenmanager aus Komfortgründen zum Standard-Warteschlangenmanager. Klicken Sie auf **Fertigstellen**.



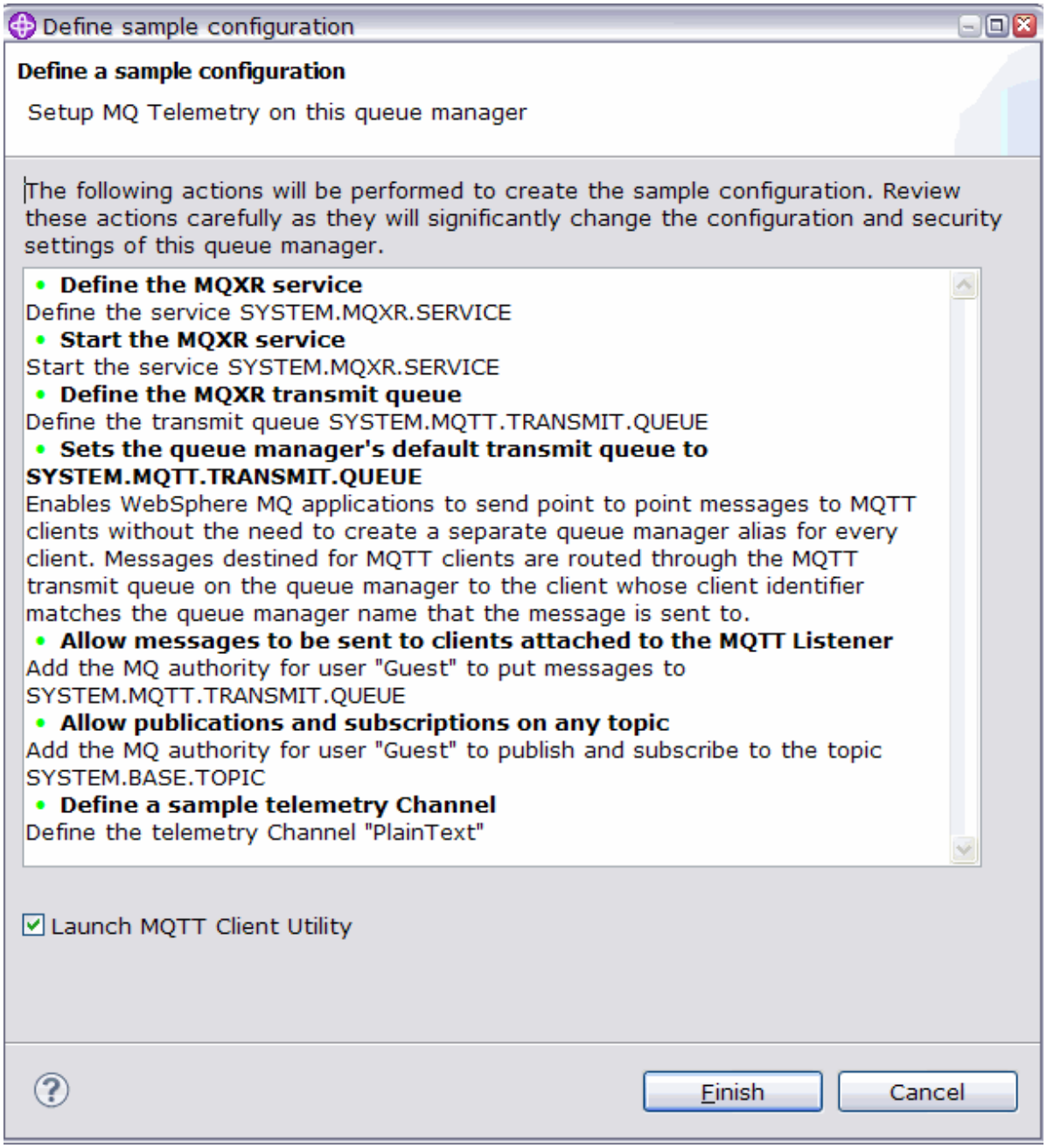
IBM WebSphere MQ Explorer erstellt den Warteschlangenmanager und startet ihn.

4. Führen Sie den Telemetry-Assistenten **Define sample configuration** (Beispielkonfiguration definieren) aus.
   a) Öffnen Sie den Telemetry-Ordner für den Warteschlangenmanager.



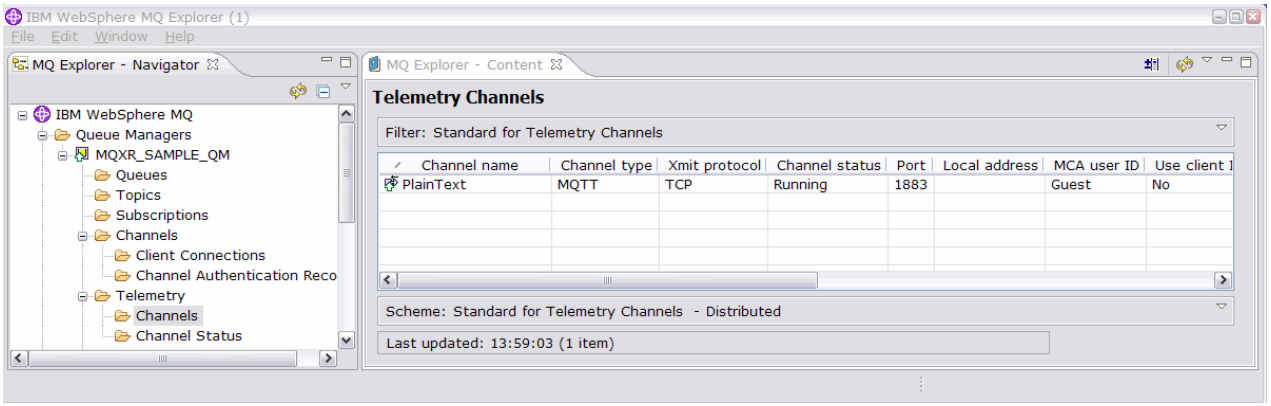
b) Klicken Sie auf **Define sample configuration**, um den Assistenten zu starten.

c) Klicken Sie auf **Finish** (Fertigstellen), um den Telemetrieservice zu erstellen und das MQTT-Client-dienstprogramm auszuführen.



Ergebnisse

Öffnen Sie den Telemetry-Ordner 'Channels' (Kanäle), um eine Liste der Beispielkanäle anzuzeigen.



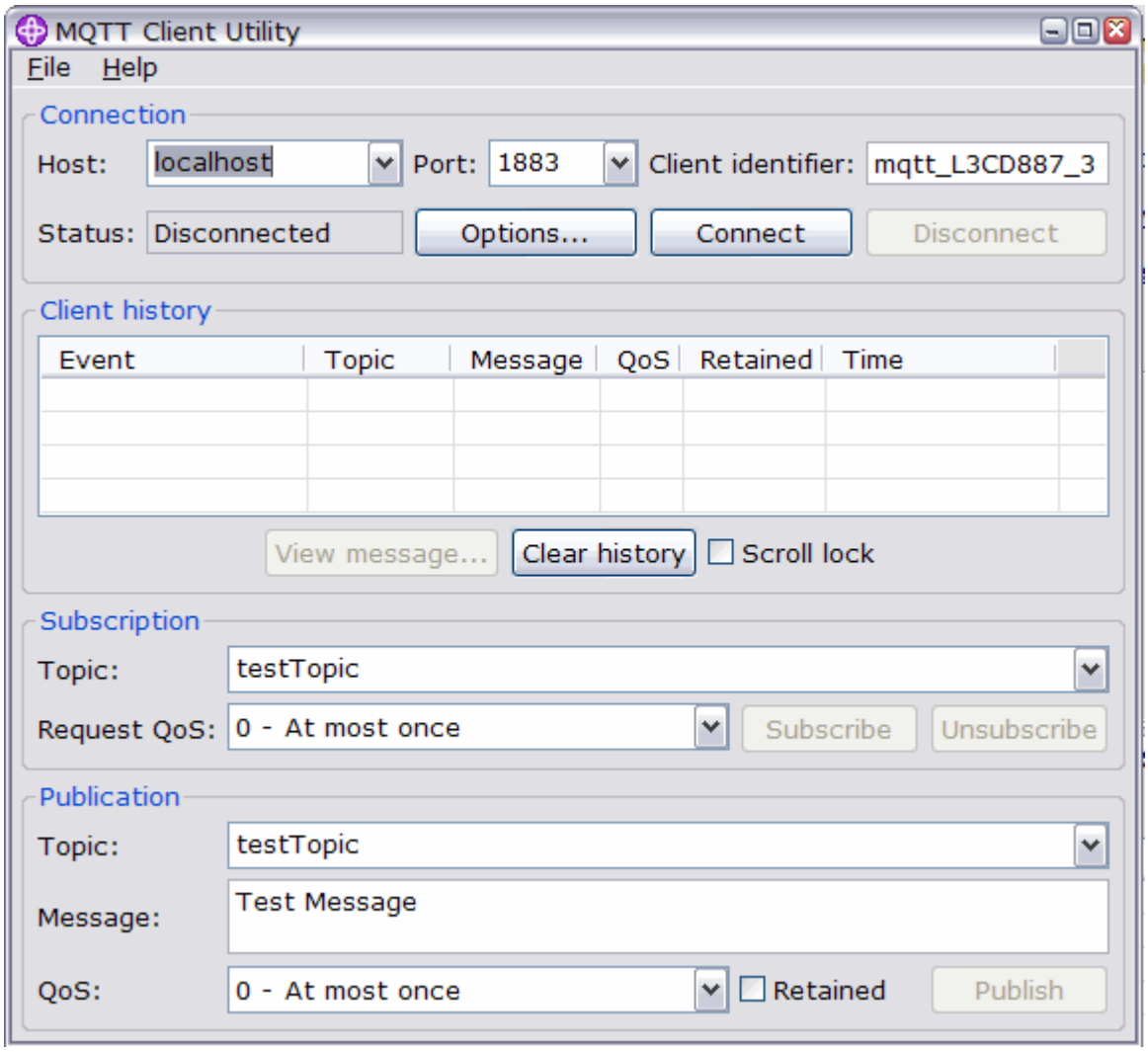
In diesem Fenster können Sie die Eigenschaften eines Kanals ändern und Kanäle hinzufügen und löschen.

Nächste Schritte

Testen Sie die Verbindung, indem Sie das MQTT-Clientdienstprogramm ausführen.

1. Öffnen Sie zum Starten des Clientdienstprogramms den Ordner **Telemetry** und klicken Sie zweimal auf **Run MQTT Client Utility** (MQTT-Clientdienstprogramm ausführen).

   Es werden zwei identische **MQTT Client Utility**-Fenster für zwei unterschiedliche Client-IDs geöffnet.



2. Klicken Sie in beiden Fenstern auf **Connect** (Verbinden).
3. Klicken Sie in beiden Fenstern auf **Subscribe** (Subskribieren).
4. Klicken Sie in jedem der beiden Fenster auf **Publish** (Veröffentlichen). Die Ergebnisse werden in Abbildung 29 auf Seite 155 gezeigt.

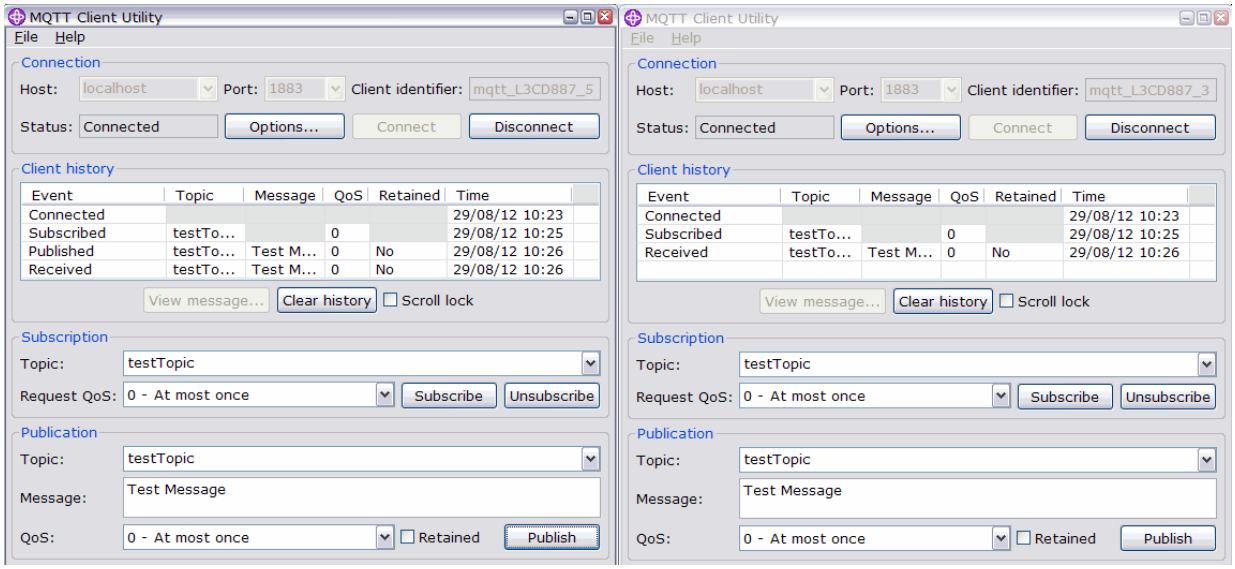


Abbildung 29. Ergebnisse

5. Klicken Sie in beiden Fenstern auf **Disconnect** (Verbindung trennen).

Der Server ist jetzt zum Testen Ihrer MQTT V3.1 -App bereit.

Zugehörige Tasks

MQTT-Service über die Befehlszeile konfigurieren

Folgen Sie den Anweisungen in diesem Abschnitt, um IBM WebSphere MQ über die Befehlszeile für die Ausführung der IBM WebSphere MQ Telemetry-Beispielanwendungen zu konfigurieren. Die Schritte zeigen, wie ein Script ausgeführt wird, um einen MQTT -Service auf einem neuen WS-Manager mit dem Namen MQXR_SAMPLE_QMzu erstellen.

WebSphere MQ Telemetry verwalten

Zugehörige Informationen

WebSphere MQ Telemetry

WebSphere MQ Telemetry mit WebSphere MQ Explorer verwalten

Anwendungen für WebSphere MQ Telemetry entwickeln

Sicherheit

WebSphere MQ Telemetry-Sicherheit

Konzepte des IBM WebSphere MQ Telemetry-Dämons für Geräte

Der IBM WebSphere MQ Telemetry-Dämon für Geräte ist eine erweiterte App für den MQTT V3-Client. Damit können Sie Nachrichten von anderen MQTT-Clients speichern und weiterleiten (Store-and-forward-Verfahren). Der Dämon stellt wie ein MQTT-Client eine Verbindung zu IBM WebSphere MQ her, Sie können jedoch auch andere MQTT-Clients mit ihm verbinden.

Der Dämon ist ein Publish/Subscribe-Broker. MQTT V3-Clients verbinden sich mit ihm, um Themen zu veröffentlichen oder zu subskribieren. Dabei werden Themenzeichenfolgen für Veröffentlichungen und Themenfilter für Subskriptionen verwendet. Die Themenzeichenfolge ist hierarchisch aufgebaut und die einzelnen Themenebenen sind durch einen Schrägstrich (/) voneinander getrennt. Themenfilter sind Themenzeichenfolgen, die Platzhalter des Typs + bei einer einzelnen Ebene und den Platzhalter # bei mehreren Ebenen als letzten Teil der Themenzeichenfolge enthalten können.

Anmerkung: Die Platzhalter im Dämon unterliegen den restriktiveren Regeln von WebSphere Message Broker Version 6. In IBM WebSphere MQ ist dies anders. Dieses Produkt unterstützt mehrere Platzhalter für mehrere Ebenen. Platzhalter können für eine beliebige Anzahl an Ebenen in der Hierarchie stehen und sich an einer beliebigen Stelle in der Themenzeichenfolge befinden.

Mehrere MQTT v3-Clients verbinden sich unter Verwendung eines Empfangsprogrammports mit dem Dämon. Der standardmäßige Empfangsprogrammport ist änderbar. Sie können mehrere Empfangsprogrammports definieren und diesen unterschiedliche Namensbereiche zuordnen. Weitere Informationen

hierzu finden Sie unter „Empfangsprogrammports für den WebSphere MQ Telemetry-Dämon für Geräte” auf Seite 164. Der Dämon ist selbst ein MQTT v3-Client. Konfigurieren Sie eine Dämonbridgeverbindung, um den Dämon mit dem Empfangsprogrammport eines anderen Dämons oder mit einem WebSphere MQ Telemetry-Service (MQXR) zu verbinden.

Sie können mehrere Bridges für den WebSphere MQ Telemetry-Dämon für Geräte konfigurieren. Mithilfe der Bridges können Sie zum Austausch von Veröffentlichungen ein Netz aus Dämonen miteinander verbinden.

Jede Bridge kann Themen bei ihrem lokalen Dämon veröffentlichen und subskribieren. Sie kann außerdem Themen bei einem anderen Dämon, einem WebSphere MQ-Publish/Subscribe-Broker oder einem beliebigen anderen MQTT v3-Broker, mit dem sie verbunden ist, veröffentlichen und subskribieren. Mithilfe eines Themenfilters können Sie die Veröffentlichungen auswählen, die von einem Broker an einen anderen weitergegeben werden sollen. Sie können Veröffentlichungen in beide Richtungen weitergeben. Sie können Veröffentlichungen aus dem lokalen Dämon an jeden seiner angehängten fernen Broker bzw. aus den angehängten Brokern an den lokalen Dämon weitergeben; siehe „Bridges des IBM WebSphere MQ Telemetry-Dämons für Geräte” auf Seite 156.

Bridges des IBM WebSphere MQ Telemetry-Dämons für Geräte

Eine Bridge des IBM WebSphere MQ Telemetry-Dämons für Geräte verbindet zwei Publish/Subscribe-Broker mithilfe des MQTT v3-Protokolls. Die Bridge gibt Veröffentlichungen von einem Broker zum anderen in beide Richtungen weiter. Am einen Ende befindet sich eine Bridgeverbindung des WebSphere MQ Telemetry-Dämons für Geräte. Am anderen Ende befindet sich ein Warteschlangenmanager oder ein anderer Dämon. Ein Warteschlangenmanager wird über einen Telemetriekanal mit der Bridgeverbindung verbunden. Ein Dämon wird über ein Dämonempfangsprogramm mit der Bridgeverbindung verbunden.

Der IBM WebSphere MQ Telemetry-Dämon für Geräte unterstützt eine oder mehrere gleichzeitige Verbindungen mit anderen Brokern. Die Verbindungen vom Dämon werden als Bridge bezeichnet und durch Verbindungseinträge in der Dämonkonfigurationsdatei definiert. Die Verbindungen zu IBM WebSphere MQ werden mithilfe von IBM WebSphere MQ Telemetry-Kanälen hergestellt, wie in der folgenden Abbildung dargestellt:

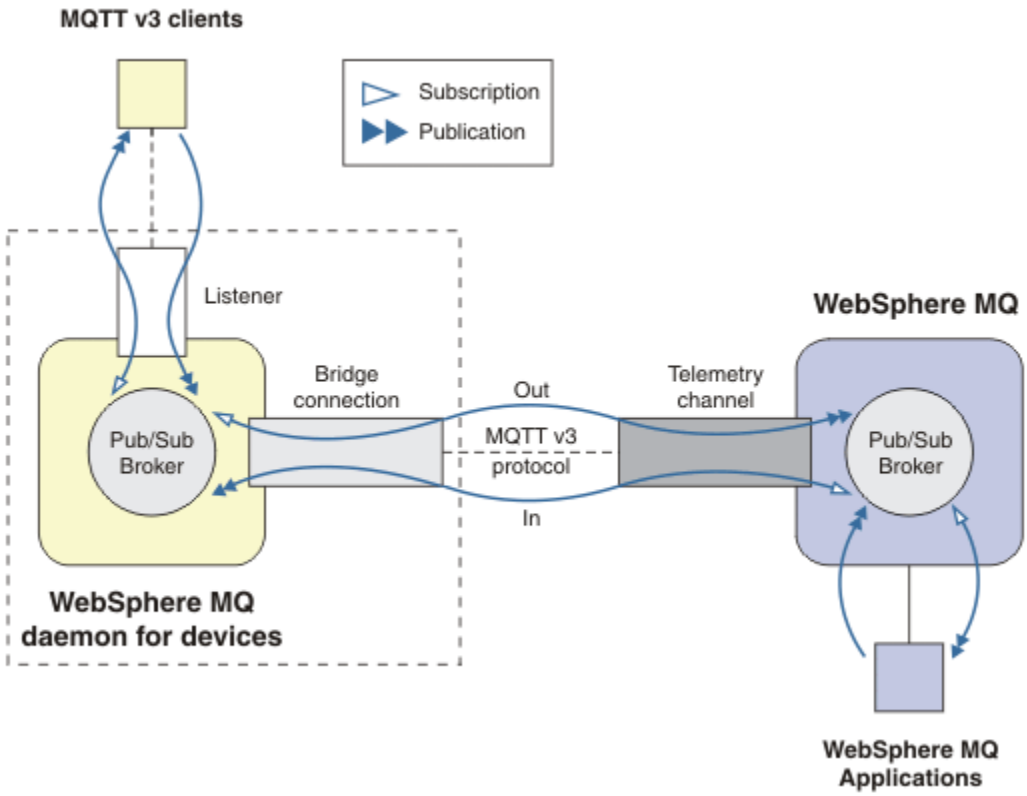


Abbildung 30. Verbindet IBM WebSphere MQ Telemetry daemon for devices mit IBM WebSphere MQ

Eine Bridge verbindet den Dämon mit einem anderen Broker in Form eines MQTT v3-Clients. Die Bridge-parameter spiegeln die Attribute eines MQTT v3-Clients.

Eine Bridge ist mehr als eine Verbindung. Sie dient als Veröffentlichungs- und Subskriptionsagent zwischen zwei Publish/Subscribe-Brokern. Der lokale Broker ist der IBM WebSphere MQ Telemetry-Dämon für Geräte, und der ferne Broker ist ein beliebiger Publish/Subscribe-Broker, der das MQTT v3-Protokoll unterstützt. Der ferne Broker ist in der Regel ein anderer Dämon oder IBM WebSphere MQ.

Die Bridge hat die Aufgabe, Veröffentlichungen zwischen den beiden Brokern weiterzugeben. Die Bridge ist bidirektional. Sie gibt Veröffentlichungen in beide Richtungen weiter. In Abbildung 30 auf Seite 157 ist dargestellt, wie die Bridge den IBM WebSphere MQ Telemetry-Dämon für Geräte mit IBM WebSphere MQ verbindet. Im Abschnitt „Beispiel für "topic"-Einstellungen für die Bridge” auf Seite 158 wird anhand von Beispielen dargestellt, wie die Bridge mithilfe des Parameters "topic" konfiguriert wird.

Die Pfeile In und Out in Abbildung 30 auf Seite 157 geben die Bidirektionalität der Bridge an. Am einen Ende des Pfeils wird eine Subskription erstellt. Die Veröffentlichungen, die der Subskription entsprechen, werden an den Broker am gegenüberliegenden Ende des Pfeils veröffentlicht. Der Pfeil wird gemäß der Flussrichtung von Veröffentlichungen gekennzeichnet. Veröffentlichungen fließen zum Dämon hin (In) und vom Dämon weg (Out). Die Kennzeichnungen sind wichtig, da sie in der Befehlssyntax verwendet werden. Mit In und Out wird die Flussrichtung von Veröffentlichungen angegeben und nicht der Ort, an den Subskriptionen gesendet werden.

Andere Clients, Anwendungen oder Broker können mit IBM WebSphere MQ oder mit dem WebSphere MQ Telemetry-Dämon für Geräte verbunden werden. Sie veröffentlichen und subskribieren Themen bei dem Broker, mit dem sie verbunden sind. Wenn es sich beim Broker um IBM WebSphere MQ handelt, können die Themen in Gruppen zusammengefasst oder verteilt sein. Zudem werden die Themen möglicherweise beim lokalen Warteschlangenmanager nicht explizit definiert.

Verwendung von Bridges

Verbinden Sie Dämonen mithilfe von Bridgeverbindungen und Empfangsprogrammen miteinander. Verbinden Sie Dämonen und Warteschlangenmanager mithilfe von Bridgeverbindungen und Telemetriekanälen. Wenn Sie mehrere Broker miteinander verbinden, entstehen möglicherweise Schleifen. Seien Sie vorsichtig: Es kann vorkommen, dass Veröffentlichungen unerkannt in einer Schleife von Brokern endlos zirkulieren.

Hier einige der Gründe für die Verwendung von Dämonen, die über eine Bridge mit IBM WebSphere MQ verbunden sind:

Geringere Anzahl von MQTT-Clientverbindungen mit WebSphere MQ
Wenn Sie eine Hierarchie aus Dämonen verwenden, können Sie viele Clients mit WebSphere MQ verbinden; mehr Clients als die Anzahl von Clients, die ein einzelner Warteschlangenmanager jeweils verbinden kann.

Speichern und Weiterleiten von Nachrichten zwischen MQTT-Clients und WebSphere MQ
Sie können das Verfahren zum Speichern und Weiterleiten verwenden, um zu vermeiden, dass zwischen Clients und IBM WebSphere MQ Dauerverbindungen bestehen, wenn die Clients keinen eigenen Speicher haben. Sie können mehrere Arten von Verbindungen zwischen dem MQTT-Client und WebSphere MQ verwenden. Informationen hierzu finden Sie unter Telemetry-Konzepte und Szenarios für die Überwachung und Steuerung.

Filtern der zwischen MQTT-Clients und WebSphere MQ ausgetauschten Veröffentlichungen
Veröffentlichungen werden üblicherweise in Nachrichten, die lokal verarbeitet werden, und Nachrichten, die andere Anwendungen einbeziehen, unterteilt. Lokale Veröffentlichungen enthalten Steuerungsabläufe zwischen Sensoren und Aktuatoren, während ferne Veröffentlichungen Anforderungen für Lesevorgänge, Statusinformationen und Konfigurationsbefehle enthalten.

Ändern des Themenbereichs von Veröffentlichungen
Verhindern Sie, dass Themenzeichenfolgen von Clients, die mit unterschiedlichen Empfangsprogrammports verbunden sind, miteinander in Konflikt geraten. Im Beispiel wird der Dämon verwendet, um Messungen von unterschiedlichen Gebäuden zu kennzeichnen. Informationen hierzu finden Sie unter Themenbereiche von unterschiedlichen Clientgruppen trennen.

Beispiel für "topic"-Einstellungen für die Bridge

Alles beim fernen Broker veröffentlichen - Standardwerte verwenden

Die Standardrichtung wird als out bezeichnet, und die Bridge veröffentlicht Themen beim fernen Broker. Der Parameter topic bestimmt mithilfe von Themenfiltern, welche Themen weitergegeben werden.

Die Bridge verwendet den Parameter topic in Abbildung 31 auf Seite 158, um all das zu subskribieren, was beim lokalen Dämon von MQTT-Clients oder von anderen Brokern veröffentlicht wird. Die Bridge veröffentlicht die Themen bei dem fernen Broker, der mit der Bridge verbunden ist.

```
connection Daemon1
topic #
```

Abbildung 31. Alles beim fernen Broker veröffentlichen

Alles beim fernen Broker veröffentlichen - explizit

Die Einstellung topic im folgenden Codefragment liefert dasselbe Ergebnis wie die Standardeinstellungen. Der einzige Unterschied besteht darin, dass der Parameter **direction** explizit ist. Verwenden Sie die Richtung `out`, um den lokalen Broker, also den Dämon, zu subskribieren und beim fernen Broker zu veröffentlichen. Auf dem lokalen Dämon erstellte Veröffentlichungen, die die Bridge subskribiert, werden beim fernen Broker veröffentlicht.

```
connection Daemon1
topic # out
```

Abbildung 32. Alles beim fernen Broker veröffentlichen - explizit

Alles beim lokalen Broker veröffentlichen

Anstelle der Richtung `out` können Sie die entgegengesetzte Richtung `in` festlegen. Im folgenden Codefragment wurde die Bridge so konfiguriert, dass sie alles subskribiert, was bei dem mit der Bridge verbundenen Broker veröffentlicht wird. Die Bridge veröffentlicht die Themen beim lokalen Broker, dem Dämon.

```
connection Daemon1
topic # in
```

Abbildung 33. Alles beim lokalen Broker veröffentlichen

Alles aus dem Exportthema beim lokalen Broker im Importthema beim fernen Broker veröffentlichen

Verwenden Sie die beiden zusätzlichen "topic"-Parameter **`local_prefix`** und **`remote_prefix`**, um den Themenfilter `#` in den vorherigen Beispielen zu ändern. Ein Parameter wird verwendet, um den in der Subskription verwendeten Themenfilter zu ändern, der andere, um das Thema, in dem die Veröffentlichung veröffentlicht wird, zu ändern. Dabei geht es darum, den Anfang der im einen Broker verwendeten Themenzeichenfolge durch eine andere Themenzeichenfolge im anderen Broker zu ersetzen.

Je nach der Richtung des Themenbefehls wird die Bedeutung von **`local_prefix`** und **`remote_prefix`** ins Gegenteil verkehrt. Wenn die Richtung `out` lautet, also die Standardeinstellung verwendet wird, wird **`local_prefix`** als Teil der Themensubskription verwendet und **`remote_prefix`** ersetzt den **`local_prefix`**-Teil der Themenzeichenfolge in der fernen Veröffentlichung. Wenn die Richtung `in` lautet, wird **`remote_prefix`** Teil der fernen Subskription und **`local_prefix`** ersetzt den **`remote_prefix`**-Teil der Themenzeichenfolge.

Der erste Teil einer Themenzeichenfolge dient häufig zur Definition eines Themenbereichs. Verwenden Sie die zusätzlichen Parameter, um den Themenbereich zu ändern, in dem ein Thema veröffentlicht wird. Sie können dies tun, um zu verhindern, dass das weitergegebene Thema mit einem anderen Thema des Zielbrokers in Konflikt gerät, oder um die Themenzeichenfolge eines Mountpunkts zu entfernen.

Beispiel: Im folgenden Codefragment werden alle Veröffentlichungen in der Themenzeichenfolge `export/#` beim Dämon in `import/#` beim fernen Broker erneut veröffentlicht.

```
topic # out export/ import/
```

Abbildung 34. Alles aus dem Exportthema beim lokalen Broker im Importthema beim fernen Broker veröffentlichen

Alles im Importthema beim lokalen Broker aus dem Exportthema beim fernen Broker veröffentlichen

Im folgenden Codefragment ist die Konfiguration umgekehrt dargestellt. Die Bridge subskribiert alles, was mit der Themenzeichenfolge `export/#` beim fernen Broker veröffentlicht wird, und veröffentlicht es in der Themenzeichenfolge `import/#` beim lokalen Broker.

```
connection Daemon1
topic # in import/ export/
```

Abbildung 35. Alles im Importthema beim lokalen Broker aus dem Exportthema beim fernen Broker veröffentlichen

Mit den ursprünglichen Themenzeichenfolgen alles aus dem Mountpunkt 1884/ beim fernen Broker veröffentlichen

Im folgenden Codefragment subskribiert die Bridge alles, was von Clients veröffentlicht wird, die mit dem Mountpunkt 1884/ am lokalen Dämon verbunden sind. Die Bridge veröffentlicht alles, was am Mountpunkt des fernen Brokers veröffentlicht wird. Die Zeichenfolge 1884/ des Mountpunkts wird aus den Themen entfernt, die am fernen Broker veröffentlicht werden. Der Parameter *local_prefix* ist mit der Zeichenfolge 1884/ des Mountpunkts identisch. Beim Parameter *remote_prefix* handelt es sich um eine leere Zeichenfolge.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

Abbildung 36. Mit den ursprünglichen Themenzeichenfolgen alles aus dem Mountpunkt 1884/ beim fernen Broker veröffentlichen

Themenbereiche von unterschiedlichen Clients trennen, die mit unterschiedlichen Dämonen verbunden sind

Eine Anwendung wird beispielsweise für Netzstromzähler geschrieben, um Zählerstände für ein Gebäude zu veröffentlichen. Die Zählerstände werden mithilfe von MQTT-Clients bei einem Dämon veröffentlicht, der in demselben Gebäude betrieben wird. Für die Veröffentlichungen wird das Thema `power` ausgewählt. Die Anwendung wird in mehreren Gebäuden in einem Komplex implementiert. Zur Standortüberwachung und Datenspeicherung werden Zählerstände von allen Gebäuden mithilfe von Bridgeverbindungen zusammengefasst. Mithilfe der Verbindungen werden alle Dämonen in demselben Gebäude an einem zentralen Standort mit WebSphere MQ verknüpft.

Eine identische Client-App wird in allen Gebäuden verwendet. Diese App veröffentlicht Daten zum Thema `power`. Bei den Daten muss jedoch eine Differenzierung nach Gebäude erfolgen. Diese Task wird durch den Dämon für jedes Gebäude ausgeführt, indem die Gebäudenummer als Präfix zum Themennamen hinzugefügt wird. Die Bridge vom ersten Gebäude im Komplex weist das Präfix `meters/building01/` auf, die vom zweiten Gebäude das Präfix `meters/building02/`. Die Zählerstände der anderen Gebäude folgen diesem Muster. WebSphere MQ empfängt daher die gemessenen Werte mit Themen wie `meters/building01/power`.

Die Konfigurationsdatei für die einzelnen Dämonen enthält eine Themenanweisung, die dem Muster im folgenden Codefragment folgt:

```
connection Daemon1
topic power out "" meters/building01/
```

Abbildung 37. Themenbereiche von Clients trennen, die mit unterschiedlichen Dämonen verbunden sind

Im vorherigen Codefragment ist die leere Zeichenfolge ein Platzhalter für den nicht verwendeten Parameter 'local_prefix'.

Anmerkung: Dieses Beispiel ist in gewisser Weise künstlich und dient nur der Illustration. In der Praxis ist der Themenbereich, in dem die Anwendung Daten veröffentlicht, in der Regel konfigurierbar.

Themenbereiche von Clients trennen, die mit einem Dämon verbunden sind

Nehmen wir an, dass nur ein Dämon verwendet wird, um alle Stromzähler zu verbinden. Vorausgesetzt, die Anwendung kann für die Verbindung mit unterschiedlichen Ports konfiguriert werden, können Sie die Gebäude unterscheiden, indem Sie die Zähler von unterschiedlichen Gebäuden mit unterschiedlichen Empfangsprogrammports verbinden. Dies wird im folgenden Codefragment dargestellt. Auch dieses Beispiel ist erfunden. Es zeigt, wie Mountpunkte verwendet werden können.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

Abbildung 38. Themenbereiche von Clients trennen, die mit einem Dämon verbunden sind

Unterschiedliche Themen für Veröffentlichungen neu zuordnen, die in beide Richtungen fließen

In der Konfiguration im folgenden Codefragment subskribiert die Bridge das einzelne Thema b beim fernen Broker und leitet Veröffentlichungen zu b an den lokalen Dämon weiter, wobei das Thema in ageändert wird. Die Bridge subskribiert auch das einzelne Thema x beim lokalen Broker und leitet Veröffentlichungen zu x an den fernen Broker weiter, wobei das Thema in ygeändert wird.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

Abbildung 39. Unterschiedliche Themen für Veröffentlichungen neu zuordnen, die in beide Richtungen fließen

Ein wichtiger Aspekt bei diesem Beispiel ist, dass unterschiedliche Themen bei beiden Brokern subskribiert und veröffentlicht werden. Die Themenbereiche der beiden Broker überschneiden sich nicht.

Dieselben Themen für Veröffentlichungen neu zuordnen, die in beide Richtungen fließen (Schleife)

Im Gegensatz zum vorherigen Beispiel ergibt die Konfiguration in Abbildung 40 auf Seite 162 generell eine Schleife. In der Themenanweisung `topic "" in a b` subskribiert die Bridge b über Fernzugriff und veröffentlicht Daten in a lokal. In der anderen Themenanweisung subskribiert die Bridge a lokal und veröffentlicht Daten in b über Fernzugriff. Diese Konfiguration kann auch wie in Abbildung 41 auf Seite 162 dargestellt geschrieben werden.

Das allgemeine Ergebnis ist, dass die Veröffentlichung als Veröffentlichung zum Thema aan den lokalen Dämon übertragen wird, wenn ein Client fern in b veröffentlicht. Bei der Veröffentlichung durch die Brücke zum lokalen Dämon für das Thema astimmt die Veröffentlichung jedoch mit der Subskription überein, die durch die Brücke zum lokalen Thema aerstellt wurde. Die Subskription lautet `topic "" out a b`. Infolgedessen wird die Veröffentlichung als Veröffentlichung zum Thema bzurück an den fernen Broker übertragen. Die Bridge ist jetzt für das ferne Thema bsubskribiert und der Zyklus beginnt erneut.

Einige Broker implementieren eine Funktion zum Auffinden von Schleifen, um Schleifen zu verhindern. Die Funktion zum Auffinden von Schleifen muss jedoch funktionieren, wenn unterschiedliche Arten von Brokern über Bridges miteinander verbunden sind. Die Funktion zum Auffinden von Schleifen funktioniert nicht, wenn WebSphere MQ über eine Bridge mit dem WebSphere MQ Telemetry-Dämon für Geräte verbunden ist. Sie funktioniert, wenn zwei IBM WebSphere MQ Telemetry-Dämonen für Geräte über eine Bridge miteinander verbunden sind. Die Funktion zum Auffinden von Schleifen ist standardmäßig aktiviert. Informationen hierzu finden Sie unter try_private.

```
connection Daemon1
topic "" in a b
topic "" out a b
```

Abbildung 40. !Ordnen Sie dieselben Themen für Veröffentlichungen zu, die in beide Richtungen fließen

```
connection Daemon1
topic "" both a b
```

Abbildung 41. !Ordnen Sie dieselben Themen für Veröffentlichungen, die in beide Richtungen fließen, mithilfe von bothneu zu.

Die Konfigurationen in Abbildung 39 auf Seite 161 und in Abbildung 40 auf Seite 162 sind identisch.

Verfügbarkeit von Bridgeverbindungen für den IBM WebSphere MQ Telemetry daemon for devices

Konfigurieren Sie mehrere Adressen von Bridgeverbindungen für den IBM WebSphere MQ Telemetry daemon for devices, um eine Verbindung mit dem ersten verfügbaren fernen Broker herzustellen. Wenn es sich beim Broker um einen Mehrinstanz-Warteschlangenmanager handelt, geben Sie beide TCP/IP-Adressen an. Konfigurieren Sie eine primäre Verbindung zum Herstellen oder Wiederherstellen einer Verbindung mit dem primären Server, sofern er verfügbar ist.

Beim Verbindungsbridgeparameter addresses handelt es sich um eine Liste mit TCP/IP-Socketadressen. Die Bridge versucht, der Reihe nach eine Verbindung mit den einzelnen Adressen herzustellen, bis mit einer Adresse eine erfolgreiche Verbindung hergestellt werden kann. Mit den Verbindungsparametern round_robin und start_type wird festgelegt, wie die Adressen nach dem erfolgreichen Herstellen einer Verbindung verwendet werden.

Wenn start_type auf auto, manual oder lazy festgelegt wird und die Verbindung fehlschlägt, versucht die Bridge, die Verbindung wiederherzustellen. Sie verwendet die einzelnen Adressen der Reihe nach mit einer Verzögerung von etwa 20 Sekunden zwischen den einzelnen Verbindungsversuchen. Wenn `start_type` auf once festgelegt wird und die Verbindung fehlschlägt, versucht die Bridge nicht automatisch, die Verbindung wiederherzustellen.

Wenn round_robin auf true gesetzt ist, versucht die Bridgeverbindung bei der ersten Adresse in der Liste zu starten und probiert die einzelnen Adressen in der Liste der Reihe nach aus. Wenn sie am Ende der Liste anlangt, beginnt sie wieder bei der ersten Adresse. Wenn die Liste nur eine Adresse enthält, wiederholt sie den Versuch alle 20 Sekunden.

Wenn round_robin auf false gesetzt ist, hat die erste Adresse in der Liste, die als "primärer Server" bezeichnet wird, Vorrang. Wenn der erste Versuch, mit dem primären Server eine Verbindung herzustellen, fehlschlägt, versucht die Bridge wiederholt im Hintergrund, zu dem Server eine Verbindung herzustellen. Gleichzeitig versucht die Bridge, eine Verbindung mithilfe der anderen Adressen in der Liste herzustellen. Wenn die Versuche im Hintergrund, zum primären Server eine Verbindung herzustellen, erfolgreich sind, trennt die Bridge die aktuelle Verbindung und wechselt zur Verbindung mit dem primären Server.

Wenn eine Verbindung beispielsweise durch Ausgabe des Befehls **connection_stop** freiwillig getrennt wird und dann versucht wird, die Verbindung wiederherzustellen, wird wieder dieselbe Adresse verwendet. Wenn die Verbindung aufgrund eines Verbindungsfehlers getrennt wird oder weil der ferne Broker die Verbindung trennt, wartet die Bridge 20 Sekunden. Dann versucht sie, mit der nächsten Adresse in der Liste oder, wenn die Liste nur eine Adresse enthält, mit derselben Adresse eine Verbindung herzustellen.

Verbindung mit einem Mehrinstanz-Warteschlangenmanager herstellen

In der Konfiguration eines Mehrinstanz-Warteschlangenmanagers wird der Warteschlangenmanager auf zwei verschiedenen Servern mit unterschiedlichen IP-Adressen ausgeführt. Telemetriekanäle werden in

der Regel ohne bestimmte IP-Adresse konfiguriert. Sie werden lediglich mit einer Portnummer konfiguriert. Der Telemetriekanal wählt beim Start standardmäßig die erste verfügbare Netzadresse auf dem lokalen Server aus.

Konfigurieren Sie den Parameter addresses der Bridgeverbindung mit den beiden vom Warteschlangenmanager verwendeten IP-Adressen. Setzen Sie round_robin auf "true".

Wenn die aktive Warteschlangenmanagerinstanz ausfällt, schaltet der Warteschlangenmanager zur Standby-Instanz um. Der Dämon erkennt, dass die Verbindung mit der aktiven Instanz getrennt wurde, und versucht, die Verbindung mit der Standby-Instanz wiederherzustellen. Er verwendet die andere IP-Adresse in der für die Bridgeverbindung konfigurierten Adressliste.

Der Warteschlangenmanager, mit dem die Bridge eine Verbindung herstellt, ist immer noch derselbe Warteschlangenmanager. Der Warteschlangenmanager stellt seinen eigenen Status wieder her. Wenn cleansession auf "false" gesetzt ist, wird die Sitzung der Bridgeverbindung im Status vor der Übernahme wiederhergestellt. Die Verbindung wird nach einer Verzögerung wieder aufgenommen. Nachrichten mit der Servicequalität "mindestens einmal" oder "höchstens einmal" gehen nicht verloren und Subskriptionen funktionieren weiterhin.

Die für die Verbindungswiederholung erforderliche Zeit hängt zum einen von der Anzahl der Kanäle und Clients ab, die beim Start der Standby-Instanz erneut starten, und zum anderen von der Anzahl der unvollständig verarbeiteten Nachrichten. Die Bridgeverbindung versucht möglicherweise, wiederholt eine Verbindung mit beiden IP-Adressen herzustellen, bevor die Verbindung wiederhergestellt werden kann.

Konfigurieren Sie den Telemetriekanal eines Mehrinstanz-Warteschlangenmanagers nicht mit einer bestimmten IP-Adresse. Die IP-Adresse ist nur auf einem Server gültig.

Wenn Sie eine andere Lösung für hohe Verfügbarkeit verwenden, mit der die IP-Adresse verwaltet wird, ist es möglicherweise in Ordnung, einen Telemetriekanal mit einer bestimmten IP-Adresse zu konfigurieren.

cleansession

Eine Bridgeverbindung ist eine MQTT v3-Clientsitzung. Sie können festlegen, ob eine Verbindung eine neue Sitzung startet oder eine bestehende Sitzung wiederherstellt. Wenn sie eine bestehende Sitzung wiederherstellt, behält die Bridgeverbindung die Subskriptionen und ständigen Veröffentlichungen aus der vorherigen Sitzung bei.

Setzen Sie cleansession nicht auf 'false', wenn addresses mehrere IP-Adressen enthält und die IP-Adressen eine Verbindung zu Telemetriekanälen, die von unterschiedlichen Warteschlangenmanagern gehostet werden, oder zu unterschiedlichen Telemetriedämonen herstellen. Der Sitzungsstatus wird zwischen Warteschlangenmanagern oder Dämonen nicht übertragen. Beim Versuch, eine bestehende Sitzung auf einem anderen Warteschlangenmanager oder Dämon erneut zu starten, wird eine neue Sitzung gestartet. Unbestätigte Nachrichten gehen verloren und Subskriptionen verhalten sich möglicherweise nicht wie erwartet.

notifications

Eine Anwendung kann mithilfe von Benachrichtigungen verfolgen, ob die Bridgeverbindung aktiv ist. Bei einer Benachrichtigung handelt es sich um eine Veröffentlichung mit dem Wert 1 für eine aktive Verbindung und dem Wert 0 für eine getrennte Verbindung. Er wird in *topicString* veröffentlicht, definiert durch den Parameter `notification_topic` . Der Standardwert von *topicString* ist $SYS/`broker`/`connection`/ *clientIdentifier*/`state`. Die Standardeinstellung von *topicString* enthält das Präfix $SYS. Subskribieren Sie Themen, die mit $SYS beginnen, indem Sie einen Themenfilter definieren, der mit $SYS beginnt. Der Themenfilter *#* (alles subskribieren) subskribiert keine Themen, die auf dem Dämon mit $SYS beginnen. $SYS definiert einen speziellen Themenbereich des Systems, der sich vom Themenbereich der Anwendung unterscheidet.

Benachrichtigungen ermöglichen es IBM WebSphere MQ Telemetry daemon for devices , MQTT-Clients zu benachrichtigen, wenn eine Bridge verbunden oder getrennt wird.

keepalive_interval

Der Parameter keepalive_interval für Bridgeverbindungen legt das Intervall zwischen TCP/IP-Ping-Befehlen von der Bridge an den fernen Server fest. Das Standardintervall beträgt 60 Sekunden. Die Überprüfung mit Ping verhindert, dass die TCP/IP-Sitzung vom fernen Server oder von einer Firewall beendet wird, die für die Verbindung einen Inaktivitätszeitraum erkennt.

clientid

Eine Bridgeverbindung ist eine MQTT v3-Clientsitzung mit einem `clientIdentifier`, der durch den Parameter clientid der Bridgeverbindung festgelegt wird. Wenn Sie angeben, dass bei Verbindungswiederholungen eine vorherige Sitzung wiederaufgenommen wird, indem Sie den Parameter cleansession auf false setzen, muss in den einzelnen Sitzungen derselbe `clientIdentifier` verwendet werden. Der Standardwert von clientid lautet *hostname.connectionName* und bleibt derselbe.

Installation, Prüfung, Konfiguration und Steuerung des WebSphere MQ Telemetry-Dämons für Geräte

Die Installation, Konfiguration und Steuerung des Dämons erfolgt dateibasiert.

Installieren Sie den Dämon, indem Sie das Software-Development-Kit auf das Gerät kopieren, auf dem Sie den Dämon ausführen werden.

Führen Sie beispielsweise das MQTT-Clientdienstprogramm aus und stellen Sie eine Verbindung zum WebSphere MQ Telemetry-Dämon für Geräte als Publish/Subscribe-Broker her; siehe WebSphere MQ Telemetry-Dämon für Geräte als Publish/Subscribe-Broker verwenden.

Konfigurieren Sie den Dämon, indem Sie eine Konfigurationsdatei erstellen; siehe Konfigurationsdatei für WebSphere MQ Telemetry-Dämon für Geräte.

Sie können einen aktiven Dämon durch die Erstellung von Befehlen in der Datei `amqtdd.upd` steuern. Alle fünf Sekunden liest der Dämon die Datei, führt die Befehle aus und löscht die Datei; siehe Befehlsdatei für WebSphere MQ Telemetry-Dämon für Geräte.

Empfangsprogrammports für den WebSphere MQ Telemetry-Dämon für Geräte

Verbinden Sie MQTT V3-Clients unter Verwendung von Empfangsprogrammports mit dem WebSphere MQ Telemetry-Dämon für Geräte. Sie können einen Empfangsprogrammport durch einen Mountpunkt und eine maximale Anzahl an Verbindungen näher bestimmen.

Ein Empfangsprogrammport muss der Portnummer entsprechen, die in der MQTT-Clientmethode `connect(serverURI)` eines Clients angegeben ist, der mit diesem Port verbunden ist. Er hat sowohl auf dem Client als auch dem Dämon den Standardwert 1883.

Sie können den Standardport für den Dämon ändern, indem Sie die globale Definition für `Port` in der Dämonkonfigurationsdatei festlegen. Sie können bestimmte Ports festlegen, indem Sie der Dämonkonfigurationsdatei eine Definition für das `Empfangsprogramm` hinzufügen.

Sie können für jeden Empfangsprogrammport außer dem Standardport einen Mountpunkt für die Eingrenzung von Clients angeben. Clients, die mit einem Port verbunden sind, der über einen Mountpunkt verfügt, werden von anderen Clients isoliert; weitere Informationen finden Sie unter „Mountpunkte für den WebSphere MQ Telemetry-Dämon für Geräte” auf Seite 165.

Sie können die Anzahl der Client begrenzen, die eine Verbindung zu einem beliebigen Port herstellen können. Legen Sie die globale Definition `max_connections` fest, um Verbindungen mit dem Standardport zu begrenzen, oder bestimmen Sie jeden Empfangsprogrammport näher mit `max_connections`.

Beispiel

Es folgt das Beispiel einer Konfigurationsdatei, die den Standardport 1883 in 1880 ändert und Verbindungen mit dem Port 1880 in 10000 ändert. Verbindungen mit dem Port 1884 sind auf 1000 begrenzt.

Clients, die an den Port 1884 angehängt sind, werden von Clients isoliert, die an andere Port angehängt sind.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

Mountpunkte für den WebSphere MQ Telemetry-Dämon für Geräte

Sie können einen Mountpunkt mit dem Port des Empfangsprogramms verbinden, der von MQTT-Clients verwendet wird, um eine Verbindung mit einem WebSphere MQ Telemetry-Dämon für Geräte herzustellen. Ein Mountpunkt grenzt die Veröffentlichungen und Subskriptionen ein, die von MQTT-Clients ausgetauscht werden, und verwendet hierzu einen Empfangsprogrammport von MQTT-Clients, die mit einem anderen Empfangsprogrammport verbunden sind.

Clients, die über einen Mountpunkt mit einem Empfangsprogrammport verbunden sind, können mit Clients, die mit einem anderen Empfangsprogrammport verbunden sind, keine Themen direkt austauschen. Clients, die ohne Mountpunkt mit einem Empfangsprogrammport verbunden sind, können auf beliebigen Clients Themen veröffentlichen oder subskribieren. Clients erkennen nicht, ob sie über einen Mountpunkt verbunden sind, und erstellen daher immer dieselben Themenzeichenfolgen.

Bei einem Mountpunkt handelt es sich um eine Textzeichenfolge, die der Themenzeichenfolge von Veröffentlichungen und Subskriptionen vorangestellt wird. Sie wird allen Themenzeichenfolgen vorangestellt, die von Clients erstellt werden, die über einen Mountpunkt mit einem Empfangsprogrammport verbunden sind. Die Textzeichenfolge wird aus allen Themenzeichenfolgen entfernt, die an Clients gesendet werden, die mit dem Empfangsprogrammport verbunden sind.

Wenn ein Empfangsprogrammport keinen Mountpunkt aufweist, werden die Themenzeichenfolgen von Veröffentlichungen und Subskriptionen nicht geändert, die von Clients erstellt und empfangen werden, die mit dem Port verbunden sind.

Erstellen Sie Mountpunktzeichenfolgen mit einem abschließenden `/`. Auf diese Weise wird der Mountpunkt zum übergeordneten Thema der Themenstruktur für den Mountpunkt.

Beispiel

Eine Konfigurationsdatei enthält folgende Empfangsprogrammports:

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

Ein mit Port 1883 verbundener Client erstellt eine Subskription für `MyTopic`. Der Dämon registriert die Subskription als `1883/MyTopic`. Ein anderer mit Port 1883 verbundener Client veröffentlicht eine Nachricht zum Thema `MyTopic`. Der Dämon ändert die Themenzeichenfolge in `1883/MyTopic` und sucht nach passenden Subskriptionen. Der Subskribent an Port 1883 empfängt die Veröffentlichung mit der ursprünglichen Themenzeichenfolge `MyTopic`. Der Dämon hat das Mountpunktpräfix aus der Themenzeichenfolge entfernt.

Ein anderer, mit Port 1884 verbundener Client veröffentlicht ebenfalls Daten zum Thema `MyTopic`. Dieses Mal registriert der Dämon das Thema als `1884/MyTopic`. Der Subskribent an Port 1883 erhält die Veröffentlichung nicht, da die unterschiedlichen Mountpunkte dazu führen, dass die Subskriptionen unterschiedliche Themenzeichenfolgen enthalten.

Ein mit Port 1885 verbundener Client veröffentlicht Daten zum Thema `1883/MyTopic`. Der Dämon ändert die Themenzeichenfolge nicht. Der Subskribent an Port 1883 empfängt die Veröffentlichung zu `MyTopic`.

Servicequalität, permanente Subskriptionen und ständige Veröffentlichungen für den WebSphere MQ Telemetry-Dämon für Geräte

Einstellungen für Servicequalität gelten nur für einen aktiven Dämon. Wenn ein Dämon kontrolliert oder durch einen Fehler beendet wird, geht der Status von unvollständigen Nachrichten verloren. Die Servicequalität "Mindestens einmal" oder "Höchstens einmal" für die Zustellung einer Nachricht kann nicht garantiert werden, wenn der Dämon beendet wird. Der WebSphere MQ Telemetry-Dämon für Geräte unterstützt eine begrenzte Persistenz. Legen Sie den Konfigurationsparameter **retained_persistence** fest, damit ständige Veröffentlichungen und Subskriptionen gespeichert werden, wenn der Dämon beendet wird.

Im Gegensatz zu WebSphere MQerfasst der WebSphere MQ Telemetry -Dämon für Geräte keine persistenten Daten. Sitzungsstatus, Nachrichtenstatus und ständige Veröffentlichungen werden nicht über Transaktionen gespeichert. Der Dämon löscht standardmäßig alle Daten, wenn er beendet wird. Sie können eine Option festlegen, um Subskriptionen und ständige Veröffentlichungen regelmäßig zu prüfen. Der Nachrichtenstatus geht immer verloren, wenn der Dämon beendet wird. Alle nicht ständigen Veröffentlichungen gehen verloren.

Setzen Sie die Konfigurationsoption `Retained_persistence` des Dämons auf `true`, um ständige Veröffentlichungen regelmäßig in einer Datei zu speichern. Beim Neustart des Dämons werden die zuletzt automatisch gespeicherten ständigen Veröffentlichungen wiederhergestellt. Von Clients erstellte ständige Nachrichten werden beim Neustart des Dämons standardmäßig nicht wiederhergestellt.

Setzen Sie die Konfigurationsoption `Retained_persistence` des Dämons auf `true`, um in einer persistenten Sitzung erstellte Subskriptionen regelmäßig in einer Datei zu speichern. Wenn `Retained_persistence` auf `true` festgelegt wird, werden Subskriptionen wiederhergestellt, die Clients in einer "persistenten Sitzung" erstellen, also in einer Sitzung, bei der `CleanSession` auf `false` festgelegt ist. Der Dämon stellt beim Neustart die Subskriptionen wieder her, die den Empfang von Veröffentlichungen starten. Der Client empfängt die Veröffentlichungen bei einem Neustart, bei dem `CleanSession` auf `false` festgelegt ist. Der Status der Clientsitzung wird beim Beenden eines Dämons standardmäßig nicht gespeichert. Somit werden Subskriptionen nicht wiederhergestellt, auch wenn der Client `CleanSession` auf `false` festlegt.

`Retained_persistence` ist eine Funktion zum automatischen Speichern. Sie speichert die letzten ständigen Veröffentlichungen oder Subskriptionen möglicherweise nicht. Sie können festlegen, wie häufig ständige Veröffentlichungen und Subskriptionen gespeichert werden. Legen Sie das Intervall zwischen Speichervorgängen oder die Anzahl der Änderungen zwischen Speichervorgängen fest. Verwenden Sie hierzu die Konfigurationsoptionen `autosave_on_changes` und `autosave_interval`.

Beispielkonfiguration zum Festlegen der Persistenz

```
# 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
```

Sicherheit des WebSphere MQ Telemetry-Dämons für Geräte

Der WebSphere MQ Telemetry-Dämon für Geräte kann Clients authentifizieren, die mit ihm verbunden sind, für die Verbindung mit anderen Broker Berechtigungsnachweise verwenden und den Zugriff auf Themen steuern. Die vom Dämon bereitgestellte Sicherheit ist begrenzt, da sie mithilfe des WebSphere MQ Telemetry C-Clients erstellt wird, der keinen SSL-Support bietet. Folglich werden Verbindungen mit dem Dämon nicht verschlüsselt und können nicht mithilfe von Zertifikaten authentifiziert werden.

Standardmäßig sind keine Sicherheitsfunktionen aktiviert.

Authentifizierung von Clients

MQTT-Clients können einen Benutzernamen und ein Kennwort mithilfe der Methoden `MqttConnectOptions.setUserName` und `MqttConnectOptions.setPassword` festlegen.

Ein Client, der eine Verbindung mit dem Dämon herstellt, wird authentifiziert, indem der von ihm angegebene Benutzername und das zugehörige Kennwort anhand von Einträgen in der Kennwortdatei überprüft werden. Um die Authentifizierung zu ermöglichen, erstellen Sie eine Kennwortdatei und legen Sie den Parameter `password_file` in der Dämonkonfigurationsdatei fest. Informationen hierzu finden Sie unter password_file.

Legen Sie den Parameter `allow_anonymous` in der Dämonkonfigurationsdatei fest, um Clients das Herstellen einer Verbindung mit einem Dämon, der die Authentifizierung überprüft, ohne Benutzername und Kennwort zu ermöglichen. Informationen hierzu finden Sie unter allow_anonymous. Wenn ein Client einen Benutzernamen oder ein Kennwort angibt, werden diese Angaben immer mit der Kennwortdatei abgeglichen, sofern der Parameter `password_file` festgelegt wurde.

Legen Sie den Parameter `clientid_prefixes` in der Dämonkonfigurationsdatei fest, um Verbindungen auf bestimmte Clients zu beschränken. Die Clients müssen `clientIdentifiers` aufweisen, die mit einem der im Parameter `clientid_prefixes` aufgeführten Präfixe beginnen. Informationen hierzu finden Sie unter clientid_prefixes.

Sicherheit für Bridgeverbindungen

Jede Bridgeverbindung des WebSphere MQ Telemetry-Dämons für Geräte entspricht einem MQTT V3-Client. Sie können den Benutzernamen und das Kennwort für jede Bridgeverbindung in Form eines Bridgeverbindungsparameters in der Dämonkonfigurationsdatei festlegen. Informationen hierzu finden Sie unter username und password. Danach kann eine Bridge sich einem Broker gegenüber selbst authentifizieren.

Zugriffssteuerung bei Themen

Für die Authentifizierung von Clients kann der Dämon auch den Zugriff auf Themen für die einzelnen Benutzer steuern. Der Dämon steuert den Zugriff, indem er das Thema, zu dem ein Client Daten veröffentlicht oder das ein Client subskribiert, mit einer Zugriffsthemenzeichenfolge in der Zugriffssteuerungsliste abgleicht. Informationen hierzu finden Sie unter acl_file.

Die Zugriffssteuerungsliste besteht aus zwei Teilen. Der erste Teil steuert den Zugriff für alle Clients, auch für anonyme Clients. Der zweite Teil enthält einen Abschnitt für alle Benutzer in der Kennwortdatei. Hier ist die Vergabe von Zugriffsrechten für die einzelnen Benutzer aufgeführt.

Beispiel

Die Sicherheitsparameter sind im folgenden Beispiel dargestellt.

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

Abbildung 42. Dämonkonfigurationsdatei

```
Fred:Fredpassword
Barney:Barneypassword
```

Abbildung 43. Kennwortdatei "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/#
```

Abbildung 44. Zugriffssteuerungsdatei "acl.txt"

Fehlerbehebung bei MQTT-Clients

Suchen Sie nach einer Task zur Fehlerbehebung, die Ihnen bei der Lösung eines Problems bei der Ausführung von MQTT-Clients helfen kann.

Zugehörige Tasks

„Traceerstellung und Debugging für den MQTT-Java-Client (Paho)" auf Seite 177
Die Standardprotokollfunktion verwendet die Java-Standardprotokollfunktion `java.util.logging` (JSR47). Sie können sie entweder mithilfe einer Konfigurationsdatei oder programmgesteuert konfigurieren.

„Traceerstellung für den MQTT-JavaScript-Client" auf Seite 179
Mit dem JavaScript-Client können Sie den Trace erfassen, indem Sie die Clientwebanwendung ändern, damit Methoden für das verbundene Clientobjekt aufgerufen werden.

„Traceerstellung für den Telemetrieservice (MQXR)" auf Seite 172
Gehen Sie wie nachfolgend beschrieben vor, um einen Trace des Telemetrieservices zu starten, die Parameter zur Steuerung des Trace festzulegen und nach der Traceausgabe zu suchen.

„Traceerstellung für den MQTT V3-Java-Client" auf Seite 174
Gehen Sie wie hier beschrieben vor, um einen Trace für den MQTT-Java-Client zu erstellen und dessen Ausgabe zu steuern.

„Traceerstellung für den MQTT-Client für C" auf Seite 175
Legen Sie die Umgebungsvariable `MQTT_C_CLIENT_TRACE` fest, um einen Trace für eine MQTT-Client-C-App zu erstellen.

„Problembehebung: MQTT-Client kann keine Verbindung herstellen" auf Seite 187
Beheben Sie das Problem, dass ein MQTT-Clientprogramm keine Verbindung zum Telemetrieservice (MQXR) herstellen kann.

„Problembehebung: MQTT-Clientverbindung aufgehoben" auf Seite 189
Stellen Sie fest, warum ein Client unerwartete Ausnahmen vom Typ `ConnectionLost` auslöst, nachdem er erfolgreich eine Verbindung herstellen konnte und der Betrieb über einen mehr oder weniger langen Zeitraum möglich war.

„Problembehebung: Verlorene Nachrichten in einer MQTT-Anwendung" auf Seite 190
Lösen Sie das Problem eines Nachrichtenverlustes. Handelt es sich um eine nicht persistente Nachricht oder wurde die Nachricht an die falsche Adresse bzw. überhaupt nicht gesendet? Bei einem falsch codierten Clientprogramm kann es unter Umständen zu Nachrichtenverlusten kommen.

„Problembehebung: Der Telemetrieservice (MQXR) wird nicht gestartet" auf Seite 192
Hier erhalten Sie Informationen zur Problembehebung, falls der Telemetrieservice (MQXR) nicht gestartet werden kann. Überprüfen Sie die Installation von WebSphere MQ Telemetry und vergewissern Sie sich, dass keine Dateien fehlen oder verschoben wurden. Vergewissern Sie sich außerdem, dass die richtigen Berechtigungen für die Dateien festgelegt sind. Überprüfen Sie die Pfade, in denen der Telemetrieservice (MQXR) nach den Programmen des Telemetrieservice (MQXR) sucht.

„Problembehebung: Das JAAS-Anmeldemodul wird vom Telemetrieservice nicht aufgerufen" auf Seite 193
Stellen Sie fest, ob Ihr JAAS-Anmeldemodul tatsächlich nicht vom Telemetrieservice (MQXR) aufgerufen wird, und konfigurieren Sie JAAS entsprechend, um das Problem zu beheben.

„Problembehebung: Dämon starten oder ausführen” auf Seite 196
Lesen Sie das Konsolenprotokoll für den IBM WebSphere MQ Telemetry-Dämon für Geräte, aktivieren Sie die Tracefunktion oder verwenden Sie die Tabelle zu den Symptomen in diesem Abschnitt, um Probleme mit dem Dämon zu beheben.

„Problembehebung: MQTT-Clients stellen keine Verbindung zum Dämon her” auf Seite 197
Clients verbinden sich nicht mit dem Dämon oder der Dämon verbindet sich nicht mit anderen Dämonen oder einem WebSphere MQ-Telemetriekanal.

Zugehörige Verweise
„Speicherposition von Telemetrieprotokollen, Fehlerprotokollen und Konfigurationsdateien” auf Seite 169
Ermitteln Sie die Speicherposition der von IBM WebSphere MQ Telemetry verwendeten Protokolle, Fehlerprotokolle und Konfigurationsdateien.

„MQTT V3 Java-Client-Ursachencodes” auf Seite 172
Suchen Sie die Ursachen für Ursachencodes in einer MQTT v3 -Java-Clientausnahme oder -Throwable.

„Systemvoraussetzungen für Verwendung von SHA-2-CipherSuites mit MQTT-Clients” auf Seite 180
Für Java 6 ab IBM SR13 können Sie SHA-2 -Cipher-Suites verwenden, um Ihre MQTT -Kanäle und Client-Apps zu schützen. SHA-2 -Cipher-Suites sind jedoch erst ab Java 7 ab IBM SR4 standardmäßig aktiviert, sodass Sie in früheren Versionen die erforderliche Suite angeben müssen. Wenn Sie MQTT-Client mit Ihrer eigenen JRE ausführen, müssen Sie sicherstellen, dass diese JRE SHA-2-Cipher-Suites unterstützt. Damit SHA-2-Cipher-Suites für Ihre Client-Apps verwendet werden können, muss der Client den SSL-Kontext auf einen Wert setzen, der TLS Version 1.2 (Transport Layer Security) unterstützt.

„Einschränkung der Browser-Unterstützung für Web-Apps für Mobile Messaging über SSL” auf Seite 181
Die verschiedenen Browser zeigen auf den einzelnen Plattformen unterschiedliches Verhalten. Diese Unterschiede sollten Sie bei der Konfiguration Ihrer Apps, Zertifizierungsstellen (CAs) und Clientzertifikate für die Verbindung mit MQTT-Messaging-Client für JavaScript über SSL und WebSockets berücksichtigen.

Speicherposition von Telemetrieprotokollen, Fehlerprotokollen und Konfigurationsdateien

Ermitteln Sie die Speicherposition der von IBM WebSphere MQ Telemetry verwendeten Protokolle, Fehlerprotokolle und Konfigurationsdateien.

Anmerkung: Die Beispiele sind für Windows codiert. Ändern Sie die Syntax für die Ausführung der Beispiele unter Linux .

Serverseitige Protokolle

Der Installationsassistent von IBM WebSphere MQ Telemetry schreibt Nachrichten in sein Installationsprotokoll:

```
WMQ program directory\mqxr
```

Der Telemetrieservice (MQXR) schreibt Nachrichten in das Fehlerprotokoll des WebSphere MQ-Warteschlangenmanagers sowie FDC-Dateien in das Fehlerverzeichnis von IBM WebSphere MQ:

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

Er schreibt zudem auch ein Protokoll für den Telemetrieservice (MQXR). Das Protokoll zeigt die Eigenschaften, mit denen der Service gestartet wurde, sowie Fehler an, die bei der Anwendung als Proxy für einen MQTT-Client aufgetreten sind. Ein möglicher Fehler könnte beispielsweise darin bestehen, dass eine Subskription zurückgenommen wird, die vom Client gar nicht erstellt wurde. Der Protokollpfad lautet:

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

Die von IBM WebSphere MQ Explorer erstellte Beispielkonfiguration von IBM WebSphere MQ Telemetry startet den Telemetrieservice über den Befehl **runMQXRService**. **runMQXRService** befindet sich in *WMQ Telemetry install directory*\bin. Er schreibt in folgendes Verzeichnis:

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

Ändern Sie den Befehl **runMQXRService** so, dass die für den Telemetrieservice (MQXR) konfigurierten Pfade angezeigt werden bzw. dass die Initialisierung vor dem Start des Telemetrieservice (MQXR) zurückgemeldet wird.

Serverseitige Konfigurationsdateien

Telemetriekanäle und Telemetrieservice (MQXR)

Einschränkung: Format, Speicherposition, Inhalt und Interpretation der Konfigurationsdatei für Telemetriekanäle können sich in zukünftigen Releases unter Umständen ändern. Zur Konfiguration der Telemetriekanäle muss der IBM WebSphere MQ Explorer verwendet werden.

IBM WebSphere MQ Explorer speichert Telemetriekonfigurationen in der Datei `mqxr_win.properties` unter Windowsund in der Datei `mqxr_unix.properties` unter Linux. Die Eigenschaftendateien werden im Telemetriekonfigurationsverzeichnis gespeichert:

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

Abbildung 45. Telemetriekonfigurationsverzeichnis unter Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Abbildung 46. Telemetriekonfigurationsverzeichnis unter Linux

JVM

Legen Sie Java-Eigenschaften fest, die als Argumente an den Telemetrieservice (MQXR) in der Datei `java.properties`übergeben werden. Die Eigenschaften in der Datei werden direkt an die JVM übergeben, auf der der Telemetrieservice (MQXR) ausgeführt wird. Sie werden als zusätzliche JVM-Eigenschaften in der Java-Befehlszeile übergeben. Eigenschaften, die in der Befehlszeile festgelegt sind, haben Vorrang vor Eigenschaften, die der Befehlszeile aus der Datei `java.properties` hinzugefügt wurden.

Suchen Sie die Datei `java.properties` in demselben Ordner wie die Telemetriekonfigurationen (siehe Abbildung 45 auf Seite 170 und Abbildung 46 auf Seite 170).

Ändern Sie die Datei `java.properties`, indem Sie jede Eigenschaft als separate Zeile angeben. Formatieren Sie jede Eigenschaft genau so, wie Sie sie bei ihrer Übergabe als Argument an die JVM formatieren würden, beispielsweise:

```
-Xmx1024m
-Xms1024m
```

JAAS

Die JAAS-Konfigurationsdatei wird im Abschnitt JAAS-Konfiguration des Telemetriekanals beschrieben. Dort finden Sie auch die JAAS-Beispielkonfigurationsdatei JAAS.config, die zum Lieferumfang von IBM WebSphere MQ Telemetry gehört.

Bei der JAAS-Konfiguration werden Sie ziemlich sicher eine Klasse zur Benutzerauthentifizierung schreiben, welche die Standardverfahren zur JAAS-Authentifizierung ersetzen soll.

Wenn Sie die Klasse `Login` in den Klassenpfad einbeziehen möchten, der vom Klassenpfad des Telemetrieservice (MQXR) verwendet wird, stellen Sie eine WebSphere MQ-Konfigurationsdatei namens `service.env` bereit.

Legen Sie den Klassenpfad für Ihr JAAS-Anmeldemodul (`LoginModule`) in `service.env` fest. In der Datei `service.env` kann die Variable `%classpath%` nicht verwendet werden. Der Klassenpfad

in der Datei `service.env` wird dem Klassenpfad hinzugefügt, der bereits in der Definition des Telemetrieservice (MQXR) festgelegt ist.

Zeigen Sie die vom Telemetrieservice (MQXR) verwendeten Klassenpfade an, indem Sie `echo set classpath` zur Datei `runMQXRService.bat` hinzufügen. Die Ausgabe wird an `mqxr.stdout` gesendet.

Die Standardposition für die Datei `service.env` ist:

```
WMQ data directory\service.env
```

Überschreiben Sie diese Einstellungen mit einer Datei `service.env` für jeden Warteschlangenmanager in folgender Position:

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

Abbildung 47 auf Seite 171 zeigt eine Beispieldatei `service.env` zur Verwendung des Beispiels `LoginModule.class`.

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

Anmerkung: `service.env` darf keine Variablen enthalten. Ersetzen Sie den tatsächlichen Wert von *`WMQ Install Directory`*.

Abbildung 47. Musterdatei `service.env` für Windows

Verfolgen

Möglicherweise werden Sie von einem IBM Servicemitarbeiter gebeten, einen Trace zu konfigurieren. Informationen hierzu finden Sie unter „Traceerstellung für den Telemetrieservice (MQXR)" auf Seite 172. Die Parameter zur Trace-Konfiguration sind in zwei Dateien gespeichert:

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

Clientseitige Protokolldateien

In: Die Standarddateipersistenzklasse im Java SE MQTT -Client, der mit IBM WebSphere MQ Telemetry bereitgestellt wird, erstellt einen Ordner mit dem Namen: *clientIdentifier*`-tcp`*hostNamePort* oder *clientIdentifier*`-ssl`*hostNamePort* im Arbeitsverzeichnis des Clients. Der Ordnername gibt also den beim Verbindungsversuch verwendeten `Hostnamen` und `Port` an.. Der Ordner enthält Nachrichten, die von der Persistenzklasse gespeichert wurden. Nach der erfolgreichen Zustellung werden die Nachrichten gelöscht.

Der Ordner wird gelöscht, wenn ein Client mit fehlerfreier Sitzung beendet wird.

Ist der Clienttrace aktiviert, wird das unformatierte Protokoll standardmäßig im Arbeitsverzeichnis des Clients gespeichert. Die Tracedatei trägt die Bezeichnung `mqtt-`*`n`*`.trc`

Clientseitige Konfigurationsdateien

Legen Sie Trace-und SSL-Eigenschaften für den MQTT-Java-Client mit Java-Eigenschaftendateien fest oder legen Sie die Eigenschaften programmgesteuert fest. Übergeben Sie die Eigenschaften mithilfe des JVM- `-D` -Switch an den MQTT-Java-Client. Beispiel:

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

Siehe „Traceerstellung für den MQTT V3-Java-Client" auf Seite 174.Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

MQTT V3 Java-Client-Ursachencodes

Suchen Sie die Ursachen für Ursachencodes in einer MQTT v3 -Java-Clientausnahme oder -Throwable.

Tabelle 5. MQTT V3 Java-Client-Ursachencodes

Ursachencode	Wert	Ursache
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	Es besteht bereits eine Clientverbindung.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	Die Clientverbindung ist bereits getrennt.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	Wird ausgelöst, wenn aus einer Methode in `MqttCallback` heraus versucht wurde, `MqttClient.disconnect` aufzurufen.
REASON_CODE_CLIENT_DISCONNECTING	32102	Die Clientverbindung wird gerade getrennt, der Client kann keine neue Arbeit annehmen.
REASON_CODE_CLIENT_EXCEPTION	0	Der Client hat eine Ausnahme festgestellt.
REASON_CODE_CLIENT_NOT_CONNECTED	32104	Der Client ist nicht mit dem Server verbunden.
REASON_CODE_CLIENT_TIMEOUT	32000	Der Client hat beim Warten auf eine Serverantwort das Zeitlimit überschritten.
REASON_CODE_FAILED_AUTHENTICATION	4	Aufgrund eines falschen Benutzernamens oder Kennworts ist die Authentifizierung beim Server fehlgeschlagen.
REASON_CODE_INVALID_CLIENT_ID	2	Der Server hat die angegebene Client-ID abgelehnt.
REASON_CODE_INVALID_PROTOCOL_VERSION	1	Die angeforderte Protokollversion wird vom Server nicht unterstützt.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	Interner Fehler. Ursache: keine neuen Nachrichten-IDs verfügbar.
REASON_CODE_NOT_AUTHORIZED	5	Es besteht keine Berechtigung, die angeforderte Operation auszuführen.
REASON_CODE_SERVER_CONNECT_ERROR	32103	Es konnte keine Verbindung zum Server hergestellt werden.
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	Die Server-URI stimmt nicht mit der angegebenen Socket-Factory überein.
REASON_CODE_SSL_CONFIG_ERROR	32106	SSL-Konfigurationsfehler.
REASON_CODE_UNEXPECTED_ERROR	6	Es ist ein unerwarteter Fehler aufgetreten.

Traceerstellung für den Telemetrieservice (MQXR)

Gehen Sie wie nachfolgend beschrieben vor, um einen Trace des Telemetrieservices zu starten, die Parameter zur Steuerung des Trace festzulegen und nach der Traceausgabe zu suchen.

Vorbereitende Schritte

Die Tracefunktion ist eine Hilfsfunktion. Gehen Sie wie im Folgenden beschrieben vor, wenn Sie von einem IBM Servicetechniker angewiesen werden, einen Trace für Ihren Telemetrieservice (MQXR) durchzuführen. Das Format der Tracedatei und die Verwendung der Datei zur Clientfehlerbehebung sind in der Produktdokumentation nicht enthalten.

Informationen zu diesem Vorgang

Sie können den IBM WebSphere MQ -Trace mit den Befehlen IBM WebSphere MQ **strmqtrc** und **endmqtrc** starten und stoppen. Mit dem Befehl **strmqtrc** wird ein Trace für den Telemetrieservice (MQXR) erstellt. Bei Verwendung von **strmqtrc** gibt es eine Verzögerung von bis zu einigen Sekunden, bevor der Telemetrieservice-Trace gestartet wird. Weitere Informationen zum IBM WebSphere MQ-Trace finden Sie im Abschnitt Trace verwenden. Alternativ können Sie einen Trace für den Telemetrieservice (MQXR) wie folgt erstellen:

Vorgehensweise

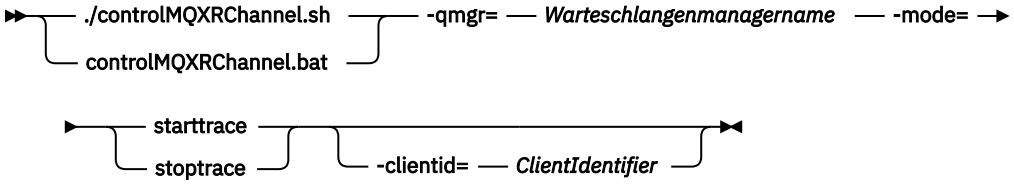
1. Legen Sie die Traceoptionen zur Steuerung der Detailmenge und Größe des Trace fest. Die Optionen werden auf einen Trace angewendet, der entweder mit dem Befehl **strmqtrc** oder dem Befehl **controlMQXRChannel** gestartet wurde.

   Legen Sie die Traceoptionen in den folgenden Dateien fest:

   ```
   mqxrtrace.properties
   trace.config
   ```

   Die Dateien befinden sich in folgendem Verzeichnis:

   - Unter Windows: *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr
   - Unter Linux: var/mqm/qmgrs/ *qMgrName*/mqxr
2. Öffnen Sie ein Befehlsfenster im folgenden Verzeichnis:
   - Auf Windows -Systemen: *WebSphere MQ installation directory*\mqxr\bin .
   - Auf Linux -Systemen /opt/mqm/mqxr/bin.
3. Führen Sie folgenden Befehl aus, um einen SYSTEM.MQXR.SERVICE-Trace zu starten:



Obligatorische Parameter

qmgr=*qmgrName*
Geben Sie für *Warteschlangenmanagername* den Namen des Warteschlangenmanagers an.

mode=starttrace| stoptrace
Legen Sie starttrace fest, um den Trace zu starten, bzw. stoptrace, um den Trace zu beenden.

Optionale Parameter

clientid=*ClientIdentifier*
Geben Sie unter *Client-ID* die Clientkennung eines Clients an. client-id filtert den Trace für einen einzelnen Client. Führen Sie den Tracebefehl mehrere Male aus, um einen Trace für mehrere Clients durchzuführen.

Beispiel:

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

Ergebnisse

Die Traceausgabe können Sie in folgendem Verzeichnis anzeigen:

- Unter Windows : *WebSphere MQ data directory*\trace
- Unter Linux: /var/mqm/trace.

Tracedateien haben die Bezeichnung mqxr_PPPPP.trc, wobei PPPPP für die Prozess-ID steht.

Zugehörige Verweise

strmqtrc

Traceerstellung für den MQTT V3-Java-Client

Gehen Sie wie hier beschrieben vor, um einen Trace für den MQTT-Java-Client zu erstellen und dessen Ausgabe zu steuern.

Vorbereitende Schritte

Dieser Abschnitt gilt nur für IBM WebSphere MQ Version 7.5.0.0. Sie finden Informationen, in denen die Traceerstellung für den Java-Client höherer Versionen beschrieben wird, unter „Traceerstellung und Debugging für den MQTT-Java-Client (Paho)" auf Seite 177.

Die Tracefunktion ist eine Hilfsfunktion. Wenn Sie von einem IBM Servicetechniker aufgefordert werden, einen Trace für Ihren MQTT-Java-Client zu erstellen, gehen Sie wie im Folgenden beschrieben vor. Das Format der Tracedatei und die Verwendung der Datei zur Clientfehlerbehebung sind in der Produktdokumentation nicht enthalten.

Die Traceerstellung funktioniert nur für den WebSphere MQ Telemetry-Java-Client.

Informationen zu diesem Vorgang

Anmerkung: Die Beispiele sind für Windows codiert. Ändern Sie die Syntax für die Ausführung der Beispiele unter Linux .[2].

Vorgehensweise

1. Erstellen Sie eine Java-Eigenschaftendatei mit der Tracekonfiguration.

   Geben Sie in der Eigenschaftendatei die folgenden optionalen Eigenschaften an. Ist der Eigenschaftsschlüssel mehrmals angegeben, wird über das letzte Vorkommen die Eigenschaft festgelegt.

   a) com.ibm.micro.client.mqttv3.trace.outputName

      Das Verzeichnis, in welches die Tracedatei geschrieben werden soll. Der Standardwert ist das Arbeitsverzeichnis des Clients. Die Tracedatei trägt die Bezeichnung mqtt-*n*.trc.

      ```
      java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
      ```

   b) com.ibm.micro.client.mqttv3.trace.count

      Die Anzahl der zu schreibenden Tracedateien. Standardmäßig handelt es sich um eine einzige Datei ohne Größenbegrenzung.

      ```
      java com.ibm.micro.client.mqttv3.trace.count=5
      ```

   c) com.ibm.micro.client.mqttv3.trace.limit

      Die maximale Dateigröße, Standardwert: 500000. Die Begrenzung gilt nur, wenn mehr als eine Tracedatei angefordert wird.

[2] Java verwendet den richtigen Pfadbegrenzer. Sie können den Begrenzer in einer Eigenschaftendatei als '/' oder '\\'codieren. '\' ist das Escapezeichen.

```
java com.ibm.micro.client.mqttv3.trace.limit=100000
```

d) `com.ibm.micro.client.mqttv3.trace.client.`*`clientIdentifier`*`.status`

Schalten Sie den Trace für jeden einzelnen Client ein bzw. aus. Bei *`clientIdentifier`*=*wird der Trace für alle Clients aktiviert oder inaktiviert. Standardmäßig ist der Trace für alle Clients ausgeschaltet.

```
java com.ibm.micro.client.mqttv3.trace.client.*.status=on
```

```
java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
```

2. Übergeben Sie die Traceeigenschaftendatei unter Verwendung einer Systemeigenschaft an die Java Virtual Machine.

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. Führen Sie den Client aus.
4. Konvertieren Sie die Tracedatei aus der Binärcodierung in Text oder `.html`. Verwenden Sie folgenden Befehl:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o outputFi
le] [-h] [-d
time]
```

Dabei gilt für die Argumente Folgendes:

-?
Zeigt Hilfe an

-i traceFile
Erforderlich. Übergibt die Eingabedatei (beispielsweise `mqtt-0.trc`).

-o outputFile
Erforderlich. Definiert die Ausgabedatei (beispielsweise `mqtt-0.trc.html` oder `mqtt-0.trc.txt`).

-h
Ausgabe als HTML-Text. Die Ausgabedateien müssen die Erweiterung `.html` haben. Wird dieses Argument nicht angegeben, erfolgt die Ausgabe als Textdatei.

-d time
Rückt eine Zeile mit dem Zeichen * ein, wenn der Zeitunterschied in Millisekunden größer-gleich (>=) der Zeitangabe ist. Für die HTML-Ausgabe nicht anwendbar.

Das folgende Beispiel gibt die Tracedatei im HTML-Format aus.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

Das zweite Beispiel gibt die Tracedatei als Klartext aus, bei dem alle aufeinanderfolgenden Zeitmarken mit einer Differenz von 50 Millisekunden oder mehr mit einem Stern (*) eingerückt sind.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

Das letzte Beispiel gibt die Tracedatei als Klartext aus:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

Traceerstellung für den MQTT-Client für C

Legen Sie die Umgebungsvariable `MQTT_C_CLIENT_TRACE` fest, um einen Trace für eine MQTT-Client-C-App zu erstellen.

Vorbereitende Schritte

Der Trace für MQTT-Client für C ist für vordefiniertes Windows als auch für Linux MQTT-Client für C-Bibliotheken und für selbst erstellte iOS-Bibliotheken.

Informationen zu diesem Vorgang

Setzen Sie die Umgebungsvariable `MQTT_C_CLIENT_TRACE` auf einen Pfad zu einer Datei, die die Traceausgabe aufnehmen soll. Die Traceausgabe wird in die Datei geschrieben.

Vorgehensweise

Legen Sie `MQTT_C_CLIENT_TRACE=mqttcclient.log` fest, bevor Sie Ihre MQTT-Client-C-App ausführen.

a) Ändern Sie beispielsweise das Beispielscript in „Erste Schritte mit dem MQTT-Client für C" auf Seite 27:

```
@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 ..\wind
ows_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) Führen Sie das Script aus dem Verzeichnis `%sdkroot%/sdk/client/c/samples` aus.

Ergebnisse

Die Traceausgabedatei beginnt mit den folgenden Zeilen:

```
============================================================
                    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
```

Zugehörige Tasks

„Erste Schritte mit dem MQTT-Client für C" auf Seite 27
Erstellen Sie den MQTT-Beispielclient für C auf einer beliebigen Plattform, auf der Sie die C-Quelle kompilieren können. Prüfen Sie, ob Sie den MQTT-Beispielclient für C mit IBM MessageSight oder IBM WebSphere MQ als MQTT -Server ausführen können.

Traceerstellung und Debugging für den MQTT-Java-Client (Paho)

Die Standardprotokollfunktion verwendet die Java-Standardprotokollfunktion `java.util.logging` (JSR47). Sie können sie entweder mithilfe einer Konfigurationsdatei oder programmgesteuert konfigurieren.

Informationen zu diesem Vorgang

Anmerkung: Der Paho-Java-Client gilt nur für Versionen von IBM WebSphere MQ Version 7.5.0.1 und höher. Informationen zur Beschreibung der Traceerstellung für den Java-Client in IBM WebSphere MQ Version 7.5.0.0finden Sie in „Traceerstellung für den MQTT V3-Java-Client" auf Seite 174.

Anmerkung: Die Tracefunktion ist eine Hilfsfunktion. Befolgen Sie diese Anweisungen, wenn ein IBM Servicetechniker Sie auffordert, einen Trace für Ihren MQTT -Java-Client zu erstellen. Das Format der Tracedatei und die Verwendung der Datei zur Clientfehlerbehebung sind in der Produktdokumentation nicht enthalten. Der Trace funktioniert nur für den IBM WebSphere MQ Telemetry-Java-Client.

Das einfachste Verfahren für die Verwendung einer Konfigurationsdatei ist die Angabe ihres Namens in der Eigenschaft `java.util.logging.config.file`.

Das Paket `org.eclipse.paho.client.mqttv3.logging` enthält eine sofort einsatzbereite Eigenschaftendatei mit dem Namen `jsr47min.properties`.

Die JSR47-Protokollierungseinrichtung kann auf verschiedene Arten verwendet werden:

- Zur Erfassung von Nachrichten aus einer ausgewählten Paketgruppe
- Zur Erfassung von Nachrichten einer bestimmten Protokollebene und darunter
- Zur Auswahl mehrerer Ziele für die Protokollnachrichten
- Durch die Bereitstellung einer integrierten Protokollfunktion, die Daten in eine Datei schreibt und die Größe sowie die Anzahl der verwendeten Dateien steuert
- Durch die Bereitstellung einer integrierten Protokollfunktion, die Daten in den Speicher schreibt und es ermöglicht, dass die Nachrichten im Speicher auf Basis eines Auslösers ausgegeben werden
- Wenn die Anwendung, die die MQTT-Clientbibliothek verwendet, ebenfalls mithilfe von JSR47 instrumentiert wurde, werden Nachrichten aus der Anwendung und aus der Clientbibliothek vermischt.

Zur Erleichterung der Erfassung von Debuginformationen wird eine Dienstprogrammklasse zur Verfügung gestellt. Diese Klasse enthält die zuvor beschriebenen Protokoll-und Tracenachrichten, kann jedoch Informationen wie Java-Systemeigenschaften und den Wert von Variablen aus dem Paho-Client erfassen.

Die Debugfunktion wird in der öffentlichen Klasse Debug bereitgestellt, die Bestandteil des Pakets `org.eclipse.paho.client.mqttv3.util` ist. Sie können eine Debuginstanz anfordern, indem Sie die Methode getDebug() sowohl für asynchrone als auch für synchrone MQTT-Clientobjekte verwenden.

Beispiel:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

Die Methode dumpClientDebug() erstellt einen Speicherauszug mit der maximalen Menge an Debuginformationen. Die Protokollierungseinrichtung muss aktiviert sein, damit die vollständigen Debuginformationen erfasst werden können, die in sie geschrieben werden. Wenn Sie vollständige Debuginformationen

erfassen möchten, rufen Sie an dem Punkt, an dem das Problem bekanntermaßen auftritt, eine Speicherauszugsmethode auf (beispielsweise nach Auftreten einer bestimmten Ausnahmebedingung).

Vorgehensweise

1. Erstellen Sie eine Konfigurationsdatei oder verwenden Sie die mitgelieferte Datei 'jsr47min.properties'.

   Wenn Sie die bereitgestellte Eigenschaftendatei verwenden, prüfen Sie, ob der Auslöser für die Push-Operation auf die richtige Fehlerebene gesetzt wurde. Er ist standardmäßig auf eine Fehlerebene vom Typ 'Schwerwiegend' gesetzt, möglicherweise müssen Sie jedoch den Trace fortlaufend in die Datei schreiben, statt ihn bis zum Auftreten eines Fehlers im Speicher aufzubewahren. Ändern Sie hierfür den Text

   ```
   java.util.logging.MemoryHandler.push=SEVERE
   ```

   zu

   ```
   java.util.logging.MemoryHandler.push=ALL
   ```

2. Übergeben Sie die Tracekonfigurationsdatei unter Verwendung einer Systemeigenschaft an die Java Virtual Machine.

   Wenn Sie die Datei 'jsr4min.properties' verwenden, ist dies:

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. Führen Sie den Client aus.

Ergebnisse

Wenn eine Ausnahmebedingung oder ein Problem auftritt, schreibt die Paho-Debugklasse den Trace im Speicher in das konfigurierte Dateiziel.

Der Trace wird bei seiner Generierung nicht automatisch in die Datei geschrieben. Dieser Vorgang wird erst dann ausgelöst, wenn entweder der Auslöser für die Push-Operation erfüllt wird oder die Debugklasse das Schreiben des Trace veranlasst. Für letztere Methode muss möglicherweise der Anwendungscode geändert werden.

Jede Zeile wird bei ihrer Erstellung in den Dateihandler geschrieben. Durch die Konfiguration eines Dateihandlers (FileHandler) können Sie das Format steuern, in dem die Nachrichten ausgegeben werden. Zusammen mit Paho wird ein angepasster Dateihandler bereitgestellt, der mehr Daten als der SimpleHandler und weniger Daten als der mit der JRE bereitgestellte XMLHandler ausgibt. Die Tracesätze, die das Paho-Formatierungsprogramm für Protokolle verwenden, haben folgendes Format:

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

Beispiel

Es wird die sofort einsatzbereite Datei `jsr47min.properties` bereitgestellt. Diese Datei enthält eine empfohlene Konfiguration für die Erfassung eines Trace, der bei der Lösung von Problemen im Zusammenhang mit dem Paho-MQTT-Client hilfreich ist. Sie legt in der Konfiguration fest, dass der Trace bei minimaler Auswirkung auf die Leistung fortlaufend im Speicher erfasst wird. Wenn der Auslöser für die Push-Operation auftritt oder eine bestimmte Anforderung bezüglich einer Push-Operation gestellt wird, wird der Speichertrace über eine Push-Operation an den konfigurierten Zielhandler übergeben. Der standardmäßige Auslöser der Push-Operation ist eine Nachricht auf der Ebene 'Schwerwiegend'. Dies entspricht einer unterbrochenen Verbindung. Der Trace, der im Speicher erfasst wird, wird standardmäßig zu diesem Zeitpunkt in die angegebene Datei geschrieben. Dabei handelt es sich standardmäßig um die Standarddatei `java.util.logging.FileHandler`. Mit der Paho-Debugklasse können Sie den Speichertrace mit einer Push-Operation an sein Ziel übertragen.

Sie finden ausführliche Details zu JSR47 in der Javadoc für das Paket 'java.util.logging'.

```
# 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.SimpleLogFormat
ter
```

Nächste Schritte

Für die programmgesteuerte Erfassung eines Trace wird eine Dienstprogrammklasse bereitgestellt, die die Erfassung von Debuginformationen erleichtert. Diese Klasse enthält die zuvor beschriebenen Protokoll-und Tracenachrichten, kann jedoch Informationen wie Java-Systemeigenschaften und den Wert von Variablen aus dem Paho-Client erfassen.

Die Debugfunktion wird in der öffentlichen Klasse Debug bereitgestellt, die Bestandteil des Pakets `org.eclipse.paho.client.mqttv3.util` ist. Sie können eine Debuginstanz anfordern, indem Sie die Methode getDebug() sowohl für asynchrone als auch für synchrone MQTT-Clientobjekte verwenden.

Beispiel:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

Die Methode dumpClientDebug() erstellt einen Speicherauszug mit der maximalen Menge an Debuginformationen. Die Protokollierungseinrichtung muss aktiviert sein, damit die vollständigen Debuginformationen erfasst werden können, die in sie geschrieben werden. Wenn Sie vollständige Debuginformationen erfassen möchten, rufen Sie an dem Punkt, an dem das Problem bekanntermaßen auftritt, eine Speicherauszugsmethode auf (beispielsweise nach Auftreten einer bestimmten Ausnahmebedingung).

Traceerstellung für den MQTT-JavaScript-Client

Mit dem JavaScript-Client können Sie den Trace erfassen, indem Sie die Clientwebanwendung ändern, damit Methoden für das verbundene Clientobjekt aufgerufen werden.

Informationen zu diesem Vorgang

Sie können die folgenden Methoden verwenden, um einen Trace zu erfassen:

- Mit `client.startTrace()` wird die Traceerstellung für den Client gestartet.
- Mit `client.stopTrace()` wird die Traceerstellung für den Client gestoppt.
- Mit `client.getTraceLog()` wird der aktuelle Tracepuffer zurückgegeben.

Sie können den Tracepuffer ausgeben, um ihn an den IBM Software Support zu senden. Hierfür haben Sie verschiedene Möglichkeiten. In dem Beispiel ist dargestellt, wie ein Trace gestartet und anschließend die Ausgabe sowohl an die Konsole als auch an eine angegebene E-Mail-Adresse gesendet wird. Zum Schluss wird der Trace schließlich gestoppt.

```
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();
 };
```

Beispielausgabe:

```
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 Systemvoraussetzungen für Verwendung von SHA-2-CipherSuites mit MQTT-Clients

Für Java 6 ab IBM SR13 können Sie SHA-2 -Cipher-Suites verwenden, um Ihre MQTT -Kanäle und Client-Apps zu schützen. SHA-2 -Cipher-Suites sind jedoch erst ab Java 7 ab IBM SR4 standardmäßig aktiviert, sodass Sie in früheren Versionen die erforderliche Suite angeben müssen. Wenn Sie MQTT-Client mit Ihrer eigenen JRE ausführen, müssen Sie sicherstellen, dass diese JRE SHA-2-Cipher-Suites unterstützt. Damit SHA-2-Cipher-Suites für Ihre Client-Apps verwendet werden können, muss der Client den SSL-Kontext auf einen Wert setzen, der TLS Version 1.2 (Transport Layer Security) unterstützt.

Für Java 7 ab IBM SR4 sind SHA-2 -Cipher-Suites standardmäßig aktiviert. Wenn Sie für Java 6 ab IBM, SR13 und höhere Service-Releases einen MQTT -Kanal definieren, ohne eine Cipher-Suite anzugeben, akzeptiert der Kanal keine Verbindungen von einem Client, der eine SHA-2 -Cipher-Suite verwendet. Zur Verwendung von SHA-2 Cipher-Suites muss die erforderliche Suite in der Kanaldefinition angegeben sein. Somit aktiviert der MQTT-Server die Suite vor der Herstellung von Verbindungen. Dies bedeutet auch, dass nur Client-Apps, die die angegebene Suite verwenden, eine Verbindung zu diesem Kanal herstellen können.

Eine ähnliche Einschränkung gilt auch für MQTT-Client für Java. Wenn der Clientcode in einer Java 1.6 JRE von IBMausgeführt wird, müssen die erforderlichen SHA-2 Cipher-Suites explizit aktiviert werden.

Zur Verwendung dieser Suites muss der Client auch den SSL-Kontext auf einen Wert setzen, der Version 1.2 des Transport Layer Security-Protokolls (TLS) unterstützt. Beispiel:

```
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);
```

Ab Juni 2013 ist Internet Explorer 10 der einzige Browser, der mit dem MQTT-Messaging-Client für JavaScript arbeitet und auch das TLS 1.2 -Protokoll unterstützt. Daher ist es der einzige Browser, den Sie verwenden können, wenn Sie SHA-2 -Verbindungen zum JavaScript -Client herstellen möchten.

Eine Liste der derzeit unterstützten Cipher-Suites finden Sie über die zugehörigen Links.

Zugehörige Konzepte

„MQTT-Clientkonfiguration für die Clientauthentifizierung mithilfe von SSL” auf Seite 108
Zur Authentifizierung des MQTT-Clients mithilfe von SSL verbindet sich der Client über SSL mit einem Telemetriekanal. Er muss einen TCP-Port angeben, der einem Telemetriekanal entspricht, welcher für die Authentifizierung von SSL-Clients konfiguriert wurde.

„MQTT-Clientkonfiguration für die Kanalauthentifizierung mithilfe von SSL” auf Seite 111
Zur Authentifizierung des Telemetriekanals mithilfe von SSL muss sich der Client über SSL mit dem Telemetriekanal verbinden. Er muss einen TCP-Port angeben, der einem Telemetriekanal entspricht, welcher für SSL konfiguriert wurde. Die Konfiguration muss einen Keystore umfassen, der durch eine Kennphrase geschützt ist und das privat signierte digitale Zertifikat des Servers enthält.

V 7.5.0.1 Einschränkung der Browser-Unterstützung für Web-Apps für Mobile Messaging über SSL

Die verschiedenen Browser zeigen auf den einzelnen Plattformen unterschiedliches Verhalten. Diese Unterschiede sollten Sie bei der Konfiguration Ihrer Apps, Zertifizierungsstellen (CAs) und Clientzertifikate für die Verbindung mit MQTT-Messaging-Client für JavaScript über SSL und WebSockets berücksichtigen.

Mobile Messaging mit JavaScript über SSL ist ein recht neues Verfahren. Daher ist es kaum verwunderlich, dass diese neue Funktion in den verschiedenen Browser- und Plattformkombinationen noch nicht ganz einheitlich und mit unterschiedlichem Umfang implementiert ist. In der folgenden Tabelle erhalten Sie einen Überblick über den derzeitigen Funktionsstand für die verschiedenen Browser (Firefox, Chrome, Internet Explorer und Safari) und Plattformen (Windows, Linux, Mac, iOS und Android).

Tabelle 6. SSL-Unterstützung nach Plattform und Browser. In dieser Tabelle ist für jede Browser- und Plattformkombination angegeben, ob anonyme oder nicht anonyme SSL-Verbindungen unterstützt werden, sowie inwieweit der Browser mit Zertifizierungsstellen (CAs) und Clientzertifikaten zusammenarbeitet.

Browser	**SSL-Unterstützung (J/N)**	**SSL funktioniert mit jeder CA (J/N)**	**Zusatzinformationen**
Firefox-Desktop.	SSL anonym - Ja SSL nicht anonym - Ja	SSL anonym - Ja SSL nicht anonym - Ja	CA und Clientzertifikat müssen dem Browser hinzugefügt werden. Firefox verwendet seinen eigenen Zertifikatsspeicher. Klicken Sie zum Importieren eines CA-Zertifikats auf **Tools** > **Optionen** > **Erweitert** > **Verschlüsselung** > **Zertifikate anzeigen** > **Zertifizierungsstellen** > **Importieren** . Um ein Clientzertifikat zu importieren, klicken Sie auf **Tools** > **Optionen** > **Erweitert** > **Verschlüsselung** > **Zertifikate anzeigen** > **Ihre Zertifikate** > **Importieren** Für eine sichere Verbindung geben Sie in der URL `https://` an. In Firefox haben Sie die Option, sich jedes Mal nach einem Zertifikat fragen oder dieses automatisch auswählen zu lassen. Außerdem haben Sie in Firefox die Wahl zwischen SSL 3.0 und TLS 1.0. Sie sollten beides wählen.

Tabelle 6. SSL-Unterstützung nach Plattform und Browser. In dieser Tabelle ist für jede Browser- und Plattformkombination angegeben, ob anonyme oder nicht anonyme SSL-Verbindungen unterstützt werden, sowie inwieweit der Browser mit Zertifizierungsstellen (CAs) und Clientzertifikaten zusammenarbeitet. *(Forts.)*

Browser	**SSL-Unterstützung (J/N)**	**SSL funktioniert mit jeder CA (J/N)**	**Zusatzinformationen**
Chrome-Desktop.	SSL anonym - Ja SSL nicht anonym - Ja	SSL anonym - Ja SSL nicht anonym - Ja	Fügen Sie CA und Clientzertifikat über den Browser zum Zertifikatsspeicher des Betriebssystems zu. Dieser wird auch von anderen Anwendungen genutzt. Klicken Sie zum Importieren eines CA-Zertifikats auf **Einstellungen** > **Erweiterte Einstellungen anzeigen** > **Zertifikate verwalten** > **Vertrauenswürdige Stammzertifizierungsstellen** > **Importieren** Klicken Sie zum Importieren eines Clientzertifikats auf **Einstellungen** > **Erweiterte Einstellungen anzeigen** > **Zertifikate verwalten** > **Persönlich** > **Import** Für eine sichere Verbindung geben Sie in der URL `https://` an. Chrome gibt Ihnen verschiedene Optionen. Wählen Sie abhängig davon, ob Sie eine anonyme oder eine nicht anonyme Verbindung konfigurieren, die passende aus.

Tabelle 6. SSL-Unterstützung nach Plattform und Browser. In dieser Tabelle ist für jede Browser- und Plattformkombination angegeben, ob anonyme oder nicht anonyme SSL-Verbindungen unterstützt werden, sowie inwieweit der Browser mit Zertifizierungsstellen (CAs) und Clientzertifikaten zusammenarbeitet. *(Forts.)*

Browser	**SSL-Unterstützung (J/N)**	**SSL funktioniert mit jeder CA (J/N)**	**Zusatzinformationen**
Internet Explorer.	SSL anonym - Ja SSL nicht anonym - Ja	SSL anonym - Ja SSL nicht anonym - Ja	Wenn Sie eine nicht anonyme SSL-Verbindung ausgewählt haben, werden Sie nach dem Clientzertifikat gefragt. Internet Explorer nutzt den Zertifikatsspeicher von Windows, der auch für andere Anwendungen freigegeben ist. Zum Importieren eines CA-Zertifikats klicken Sie auf **Tools** > **Internetoptionen** > **Inhalt** > **Zertifikate** > **Vertrauenswürdige Stammzertifizierungsstellen** > **Importieren** Um ein Clientzertifikat zu importieren, klicken Sie auf **Tools** > **Internetoptionen** > **Inhalt** > **Zertifikate** > **Persönlich** > **Import**
Safari-Desktop.	SSL anonym - Ja SSL nicht anonym - Ja	SSL anonym - Ja SSL nicht anonym - Ja	Fügen Sie CA und Clientzertifikat über den Browser zum Zertifikatsspeicher des Betriebssystems zu. Dieser wird auch von anderen Anwendungen genutzt.

Tabelle 6. SSL-Unterstützung nach Plattform und Browser. In dieser Tabelle ist für jede Browser- und Plattformkombination angegeben, ob anonyme oder nicht anonyme SSL-Verbindungen unterstützt werden, sowie inwieweit der Browser mit Zertifizierungsstellen (CAs) und Clientzertifikaten zusammenarbeitet. *(Forts.)*

Browser	**SSL-Unterstützung (J/N)**	**SSL funktioniert mit jeder CA (J/N)**	**Zusatzinformationen**
Firefox unter Android	SSL anonym - Ja SSL nicht anonym - Ja	SSL anonym - Ja SSL nicht anonym - Nein	Nicht anonym: Clientzertifikate funktionieren hier nicht, da Ihnen die Möglichkeit fehlt, Ihre Zertifizierungsstelle zur Liste in Firefox hinzuzufügen. Klicken Sie zum Importieren eines Clientzertifikats auf **Einstellungen** > **Sicherheit** > **Speicher für Berechtigungsnachweise**. Wenn Ihr Zertifikat von einer vertrauenswürdigen Zertifizierungsstelle aus der Liste signiert wurde, können Sie auch sichere Verbindungen herstellen.
Chrome unter Android	SSL anonym - Ja SSL nicht anonym - Ja	SSL anonym - Ja SSL nicht anonym - Nein	Nicht anonym: Clientzertifikate funktionieren hier nicht, da Ihnen die Möglichkeit fehlt, Ihre Zertifizierungsstelle zur Liste in Chrome hinzuzufügen. **Anmerkung:** Google plant eine Unterstützung von nicht anonymen SSL ab Chrome Version 27. Dies ist seit Version 18 ein noch nicht korrigierter Fehler. Klicken Sie zum Importieren eines Clientzertifikats auf **Einstellungen** > **Sicherheit** > **Speicher für Berechtigungsnachweise**. Wenn Ihr Zertifikat von einer vertrauenswürdigen Zertifizierungsstelle aus der Liste signiert wurde, können Sie auch sichere Verbindungen herstellen.

Tabelle 6. SSL-Unterstützung nach Plattform und Browser. In dieser Tabelle ist für jede Browser- und Plattformkombination angegeben, ob anonyme oder nicht anonyme SSL-Verbindungen unterstützt werden, sowie inwieweit der Browser mit Zertifizierungsstellen (CAs) und Clientzertifikaten zusammenarbeitet. *(Forts.)*

Browser	**SSL-Unterstützung (J/N)**	**SSL funktioniert mit jeder CA (J/N)**	**Zusatzinformationen**
Safari unter iOS	SSL anonym - Ja SSL nicht anonym - Ja	SSL anonym - Ja SSL nicht anonym - Nein	Nicht anonym: Das Gerät vertraut dem Clientzertifikat nicht, selbst wenn das CA-Zertifikat gleichzeitig installiert wird. Safari verwendet den Zertifikatsspeicher des Geräts. Klicken Sie für den Import in dieses Geschäft auf **Einstellungen** > **Allgemein** > **Profil**und stellen Sie das CA-oder Clientzertifikat von einer Webseite bereit oder senden Sie es per E-Mail an sich selbst.
Chrome unter iOS	SSL anonym - Ja SSL nicht anonym - Nein	SSL anonym - Nein SSL nicht anonym - Nein	Anonym: Nur Apple-Apps haben Zugriff auf den Stammspeicher von iOS. Chrome muss daher seine eigene CA-Liste verwenden, der Sie allerdings keine Zertifikate hinzufügen können. Nicht anonym: Clientzertifikate funktionieren hier nicht, da Ihnen die Möglichkeit fehlt, Ihre Zertifizierungsstelle zur Liste hinzuzufügen.

Zugehörige Tasks

„MQTT-Messaging-Client für JavaScript über SSL und WebSockets verbinden” auf Seite 80
Verbinden Sie Ihre Web-App sicher mit IBM WebSphere MQ, indem Sie die HTML-Beispielseiten des MQTT-Messaging-Clients für MQTT-Messaging-Client für JavaScript> mit SSL und dem WebSocket protocol verwenden.

Zugehörige Informationen

Mozilla: (SSL) Verwendet Firefox seinen eigenen CA-Speicher oder den CA-Speicher von Android?

Chromium: Problem 134418 - Implementierung der Unterstützung für Clientzertifikate

In IE10 lässt sich eine HTTPS-Site nicht ohne vertrauenswürdiges Zertifikat öffnen

Problembehebung: MQTT-Client kann keine Verbindung herstellen

Beheben Sie das Problem, dass ein MQTT-Clientprogramm keine Verbindung zum Telemetrieservice (MQXR) herstellen kann.

Vorbereitende Schritte

Liegt das Problem am Server, am Client oder an der Verbindung? Haben Sie einen eigenen MQTT v3 -Protokollverarbeitungsclient oder eine MQTT-Client-App unter Verwendung der C-oder Java WebSphere -MQTT-Clients geschrieben?

Führen Sie die mit WebSphere MQ Telemetry ausgelieferte Überprüfungsanwendung auf dem Server aus und vergewissern Sie sich, dass der Telemetriekanal und der Telemetrieservice (MQXR) ordnungsgemäß ausgeführt werden. Übertragen Sie die Überprüfungsanwendung anschließend auf den Client und führen Sie sie dort aus.

Informationen zu diesem Vorgang

Es gibt zahlreiche Gründe, warum ein MQTT-Client möglicherweise keine Verbindung zum Telemetrieserver herstellen kann bzw. Sie zu dem Schluss kommen, dass keine Verbindung hergestellt wurde.

Vorgehensweise

1. Prüfen Sie, welche Schlussfolgerungen sich aus dem vom Telemetrieservice (MQXR) an `MqttClient.Connect` zurückgegebenen Ursachencode ziehen lassen. Um welche Art von Verbindungsfehler handelt es sich?

Option	Bezeichnung
REASON_CODE_INVALID_PROTOCOL_VERSION	Vergewissern Sie sich, dass die Socketadresse einem Telemetriekanal entspricht und noch nicht für einen anderen Broker verwendet wurde.
REASON_CODE_INVALID_CLIENT_ID	Stellen Sie sicher, dass die Client-ID maximal 23 Byte umfasst und nur aus Zeichen des folgenden Bereichs besteht: `A-Z, a-z, 0-9, './_%`.
REASON_CODE_INVALID_DESTINATION	Stellen Sie sicher, dass die Client-ID nicht mit dem Warteschlangenmanagernamen identisch ist.
REASON_CODE_SERVER_CONNECT_ERROR	Prüfen Sie, ob der Telemetrieservice (MQXR) und der Warteschlangenmanager normal funktionieren. Stellen Sie mit dem Befehl **netstat** sicher, dass die Socketadresse keiner anderen Anwendung zugeordnet ist.

   Falls Sie keine der mit IBM WebSphere MQ Telemetry bereitgestellten Bibliotheken verwenden, sondern eine eigene MQTT-Clientbibliothek geschrieben haben, beachten Sie den Rückkehrcode `CONNACK`.

   Aus diesen drei Fehlern lässt sich schließen, dass der Client zwar eine Verbindung zum Telemetrieservice (MQXR) herstellen konnte, dieser jedoch einen Fehler festgestellt hat.

2. Prüfen Sie, welche Schlussfolgerungen sich aus den Ursachencodes ziehen lassen, die der Client ausgibt, wenn der Telemetrieservice (MQXR) nicht reagiert:

Option	Bezeichnung
REASON_CODE_CLIENT_EXCEPTION	Suchen Sie nach einer FDC-Datei auf dem Server. Informationen hierzu finden Sie im Thema „Serverseitige Protokolle” auf Seite 169. Wenn der Telemetrieservice (MQXR) feststellt, dass der Client das zulässige Zeitli-

Option	Bezeichnung
REASON_CODE_CLIENT_TIMEOUT	mit überschritten hat, schreibt er eine FDC-Datei (Datei zur Erfassung von Fehlerdaten beim ersten Auftreten). Eine solche Datei wird bei jeder unerwarteten Verbindungsunterbrechung geschrieben.

Möglicherweise hat der Telemetrieservice (MQXR) dem Client nicht geantwortet und das Zeitlimit beim Client läuft ab. Der Java-Client von WebSphere MQ Telemetry blockiert nur, wenn die Anwendung ein unendliches Zeitlimit festgelegt hat. Wenn nach Ablauf des für `MqttClient.Connect` festgelegten Zeitlimits ein nicht diagnostiziertes Verbindungsproblem besteht, löst der Client eine dieser Ausnahmen aus.

Sofern Sie keine FDC-Datei im Zusammenhang mit dem Verbindungsfehler finden, können Sie nicht davon ausgehen, dass der Client tatsächlich versucht hat, eine Verbindung zum Server herzustellen:

a) Vergewissern Sie sich, dass vom Client eine Verbindungsanforderung gesendet wurde.

   Prüfen Sie die TCP/IP-Anforderung mit einem Tool wie z. B. **tcpmon**, welches unter https://java.net/projects/tcpmon/ verfügbar ist.

b) Entspricht die vom Client verwendete ferne Socketadresse der für den Telemetriekanal definierten Socketadresse?

   Die Standarddateipersistenzklasse im Java SE MQTT -Client, der mit IBM WebSphere MQ Telemetry bereitgestellt wird, erstellt einen Ordner mit dem Namen: *clientIdentifier*`-tcp`*hostNamePort* oder *clientIdentifier*`-ssl`*hostNamePort* im Arbeitsverzeichnis des Clients. Der Ordnername gibt also den beim Verbindungsversuch verwendeten `Hostnamen` und `Port` an. ; siehe „Clientseitige Protokolldateien” auf Seite 171.

c) Kann die ferne Serveradresse mit Ping überprüft werden?

d) Ergibt der Befehl **netstat** auf dem Server, dass der Telemetriekanal an dem Port betrieben wird, zu dem auch der Client eine Verbindung herstellt?

3. Prüfen Sie, ob der Telemetrieservice (MQXR) ein Problem bei der Clientanforderung festgestellt hat.

   Der Telemetrieservice (MQXR) schreibt die von ihm erkannten Fehler in die Datei `mqxr.log`, während der Warteschlangenmanager Fehler in die Datei `AMQERR01.LOG` schreibt.

4. Versuchen Sie, das Problem einzugrenzen, indem Sie einen anderen Client ausführen.

   - Führen Sie die MQTT-Musteranwendung unter Verwendung desselben Telemetriekanals aus.
   - Führen Sie zur Verbindungsüberprüfung den GUI-Client **wmqttSample** aus. Rufen Sie **wmqttSample** über einen Download von SupportPac IA92 ab.

     Anmerkung: Ältere Versionen von IA92 enthalten keine MQTT v3 -Java-Clientbibliothek.

   Führen Sie die Beispielprogramme zunächst zur Überprüfung der Netzverbindung auf der Serverplattform und anschließend auf der Clientplattform aus.

5. Weitere zu überprüfende Faktoren:

   a) Versuchen Zehntausende MQTT-Clients gleichzeitig, eine Verbindung herzustellen?

      Telemetriekanäle verfügen über eine Warteschlange zum Puffern eines Rückstands eingehender Verbindungen. Mehr als 10.000 Verbindungen werden pro Sekunde verarbeitet. Die Größe des Rückstandspuffers kann über den Assistenten für Telemetriekanäle im IBM WebSphere MQ Explorer konfiguriert werden. Seine Standardgröße beträgt 4.096. Vergewissern Sie sich, dass für den Rückstand kein zu niedriger Wert konfiguriert wurde.

   b) Werden der Telemetrieservice (MQXR) und der Warteschlangenmanager immer noch ausgeführt?

   c) Hat der Client eine Verbindung zu einem Warteschlangenmanager mit hoher Verfügbarkeit hergestellt, der die TCP/IP-Adresse gewechselt hat?

   d) Werden abgehende oder zurückgegebene Datenpakete in einer Firewall selektiv gefiltert?

Problembehebung: MQTT-Clientverbindung aufgehoben

Stellen Sie fest, warum ein Client unerwartete Ausnahmen vom Typ `ConnectionLost` auslöst, nachdem er erfolgreich eine Verbindung herstellen konnte und der Betrieb über einen mehr oder weniger langen Zeitraum möglich war.

Vorbereitende Schritte

Der MQTT-Client hat erfolgreich eine Verbindung hergestellt. Er ist möglicherweise schon längere Zeit aktiv. Wenn Clients in kurzen Abständen gestartet werden, liegt möglicherweise nur ein geringer Zeitraum zwischen dem erfolgreichen Verbindungsaufbau und dem Abbau der Verbindung.

Zwischen einer abgebauten Verbindung und einer zunächst erfolgreich hergestellten Verbindung, die später abgebaut wurde, lässt sich leicht unterscheiden. Eine abgebaute Verbindung ist dadurch definiert, dass der MQTT-Client die Methode `MqttCallback.ConnectionLost` aufruft. Die Methode wird nur nach einem erfolgreichen Verbindungsaufbau aufgerufen. Die Symptome sind anders, als wenn `MqttClient.Connect` nach Erhalt einer negativen Rückmeldung oder einer Zeitlimitüberschreitung eine Ausnahme auslöst.

Falls die MQTT-Client-App nicht die von IBM WebSphere MQ bereitgestellten MQTT-Clientbibliotheken verwendet, hängt das Symptom vom Client ab. Im MQTT V3-Protokoll besteht das Symptom im Fehlen einer zeitgerechten Antwort auf eine Anforderung an den Server bzw. im Scheitern der TCP/IP-Verbindung.

Informationen zu diesem Vorgang

Der MQTT-Client ruft `MqttCallback.ConnectionLost` mit einer auslösbaren Ausnahme auf und reagiert damit auf alle festgestellten serverseitigen Probleme im Anschluss an die positive Verbindungsbestätigung. Wenn eine MQTT-Clientrückgabe von `MqttTopic.publish` und `MqttClient.subscribe` erfolgt, wird die Anforderung an einen MQTT-Client-Thread übertragen, der für das Senden und den Empfang von Nachrichten zuständig ist. Serverseitige Fehler werden asynchron gemeldet. Dabei wird eine auslösbare Ausnahme an die Rückrufmethode `ConnectionLost` übergeben.

Der Telemetrieservice (MQXR) schreibt immer eine FDC-Datei (Datei zur Erfassung von Fehlerdaten beim ersten Auftreten), wenn er die Verbindung abbaut.

Vorgehensweise

1. Wurde ein anderer Client gestartet, der denselben Wert für `ClientIdentifier` verwendet hat?

   Wird ein zweiter Client oder derselbe Client erneut unter Verwendung derselben Client-ID (`ClientIdentifier`) gestartet, wird die erste Verbindung zum ersten Client abgebaut.
2. Hat der Client auf ein Thema zugegriffen, das er weder veröffentlichen noch subskribieren darf?

   Alle Aktionen, die der Telemetrieservice für einen Client vornimmt und bei denen `MQCC_FAIL` zurückgegeben wird, führen dazu, dass der Service die Clientverbindung abbaut.

   Der Ursachencode wird nicht an den Client zurückgegeben.

   - Suchen Sie nach Protokollnachrichten in den Dateien `mqxr.log` und `AMQERR01.LOG` für den Warteschlangenmanager, mit dem der Client verbunden ist. Beachten Sie hierzu die Informationen im Thema „Serverseitige Protokolle” auf Seite 169.
3. Wurde die TCP/IP-Verbindung abgebaut?

   Möglicherweise ist in einer Firewall ein niedriges Zeitlimit festgelegt, bis eine TCP/IP-Verbindung als inaktiv markiert wird, und die Verbindung wurde daher abgebaut.

   - Verkürzen Sie die inaktive TCP/IP-Verbindungszeit über `MqttConnectOptions.setKeepAliveInterval`.

Problembehebung: Verlorene Nachrichten in einer MQTT-Anwendung

Lösen Sie das Problem eines Nachrichtenverlustes. Handelt es sich um eine nicht persistente Nachricht oder wurde die Nachricht an die falsche Adresse bzw. überhaupt nicht gesendet? Bei einem falsch codierten Clientprogramm kann es unter Umständen zu Nachrichtenverlusten kommen.

Vorbereitende Schritte

Wie sicher ist es, dass die gesendete Nachricht tatsächlich verloren wurde? Vermuten Sie, dass eine Nachricht verloren gegangen ist, nur weil sie nicht empfangen wurde? Falls es sich bei der Nachricht um eine Veröffentlichung handelt: Welche Nachricht ist verloren gegangen? Die vom Publisher gesendete Nachricht oder die an den Subskribenten gesendete Nachricht? Kann es sein, dass die Subskription nicht mehr vorhanden ist und der Broker daher keine Veröffentlichungen zu dieser Subskription mehr an den Subskribenten sendet?

Falls die Lösung dezentrales Publish/Subscribe unter Verwendung von Clustern oder Publish/Subscribe-Hierarchien umfasst, kommen zahlreiche Konfigurationsprobleme infrage, die dazu führen können, dass eine Nachricht verloren gegangen zu sein scheint.

Wenn eine Nachricht mit der Servicequalität 'Mindestens einmal' oder 'Höchstens einmal' gesendet wurde, wurde die Nachricht, die verloren zu sein scheint, vermutlich nicht in der erwarteten Weise zugestellt. Es ist unwahrscheinlich, dass die Nachricht fälschlicherweise aus dem System gelöscht wurde. Möglicherweise konnte die erwartete Veröffentlichung bzw. Subskription nicht erstellt werden.

Bei der Problembestimmung im Zusammenhang mit nicht mehr vorhandenen Nachrichten besteht der wichtigste Schritt darin zu prüfen, ob die Nachricht tatsächlich verloren gegangen ist. Reproduzieren Sie das Szenario, sodass weitere Nachrichten verschwinden. Verwenden Sie die Servicequalität 'Mindestens einmal' oder 'Höchstens einmal', um alle Fälle, in denen Nachrichten vom System gelöscht werden, auszuschließen.

Informationen zu diesem Vorgang

Die Diagnose verloren gegangener Nachrichten umfasst vier Schritte.

1. 'Fire and forget'-Nachrichten funktionieren wie vorgesehen. Bei 'Fire and forget'-Nachrichten kann es passieren, dass sie aus dem System gelöscht werden.
2. Konfiguration: Die Einrichtung eines Publish/Subscribe mit den korrekten Berechtigungen in einer verteilten Umgebung ist nicht einfach.
3. Clientprogrammierfehler: Die Verantwortung für die Nachrichtenübermittlung liegt nicht gänzlich bei dem von IBM geschriebenen Code.
4. Falls alle diese möglichen Fehlerquellen ausgeschlossen werden können, sollten Sie sich unter Umständen an den IBM Service wenden.

Vorgehensweise

1. Hatte die nicht mehr vorhandene Nachricht die Servicequalität 'Fire and forget', ist als Servicequalität 'Mindestens einmal' oder 'Höchstens einmal' festzulegen. Testen Sie, ob die Nachricht weiterhin verloren geht.
   - Nachrichten mit der Servicequalität 'Fire and forget' werden von IBM WebSphere MQ unter zahlreichen Umständen entfernt:
     - Die Übertragung wurde unterbrochen und der Kanal gestoppt.
     - Der Warteschlangenmanager wurde beendet.
     - Es sind zu viele Nachrichten vorhanden.
   - Die Zustellung von 'Fire and forget'-Nachrichten hängt von der TCP/IP-Zuverlässigkeit ab. TCP/IP sendet Datenpakete solange weiter, bis ihre Zustellung bestätigt wird. Bei einer Unterbrechung der TCP/IP-Sitzung gehen Nachrichten der Servicequalität 'Fire and forget' verloren. Grund für die

Sitzungsunterbrechung kann die Beendigung des Clients oder Servers, ein Übertragungsfehler oder eine Firewall sein, welche die Sitzung beendet.

2. Vergewissern Sie sich, dass der Client die vorherige Sitzung erneut startet, um nicht zugestellte Nachrichten mit der Servicequalität 'Mindestens einmal' oder 'Höchstens einmal' erneut zu senden.

   a) Wenn die Client-App den Java SE MQTT-Client verwendet, stellen Sie sicher, dass `MqttClient.CleanSession` auf `false` gesetzt ist.

   b) Achten Sie bei Verwendung verschiedener Clientbibliotheken darauf, dass die Sitzung ordnungsgemäß erneut gestartet wird.

3. Überprüfen Sie, ob die Client-App dieselbe Sitzung erneut startet, um auszuschließen, dass versehentlich eine andere Sitzung gestartet wird.

   Damit dieselbe Sitzung erneut gestartet wird, müssen die Angaben für `cleanSession = false`, `Mqttclient.clientIdentifier` und `MqttClient.serverURI` wie bei der vorherigen Sitzung lauten.

4. Falls eine Sitzung vorzeitig geschlossen wird, ist zu prüfen, ob die Nachricht im Persistenzspeicher auf dem Client vorhanden ist und von dort erneut gesendet werden kann.

   a) Wenn die Client-App den Java SE-MQTT-Client verwendet, überprüfen Sie, ob die Nachricht im Persistenzordner gespeichert wird (siehe „Clientseitige Protokolldateien” auf Seite 171).

   b) Falls Sie andere Clientbibliotheken verwenden oder Ihren eigenen Persistenzmechanismus implementiert haben, ist die ordnungsgemäße Funktionsweise zu prüfen.

5. Vergewissern Sie sich, dass die Nachricht vor der Zustellung von niemandem gelöscht wurde.

   Noch nicht zugestellte Nachrichten, die noch auf die Übermittlung an MQTT-Clients warten, sind in `SYSTEM.MQTT.TRANSMIT.QUEUE` gespeichert. Nachrichten, deren Übermittlung an den Telemetrieserver ansteht, werden vom Persistenzmechanismus des Clients gespeichert. Informationen hierzu finden Sie im Thema Nachrichtenpersistenz in MQTT-Clients.

6. Vergewissern Sie sich, dass der Client über eine Subskription für die zu empfangende Veröffentlichung verfügt.

   Listen Sie Subskriptionen mit WebSphere MQ Explorer oder mit **runmqsc** -oder PCF-Befehlen auf. Alle MQTT-Clientsubskriptionen haben einen Namen. Sie erhalten einen Namen im folgenden Format: *ClientIdentifier*:*Topic name*

7. Stellen Sie sicher, dass der Publisher zur Veröffentlichung und der Subskribent zur Subskription des Veröffentlichungsthemas berechtigt ist.

   ```
   dspmqaut -m qMgr -n topicName -t topic -p user ID
   ```

   In einem Publish/Subscribe-System mit Clusterbildung muss der Subskribent für das Thema auf dem Warteschlangenmanager berechtigt sein, mit dem er verbunden ist. Er muss nicht zur Subskription des Themas auf dem Warteschlangenmanager berechtigt sein, auf dem die Veröffentlichung erfolgt. Die Kanäle zwischen den Warteschlangenmanagern müssen über die korrekte Berechtigung verfügen, um die Proxy-Subskription weiterzugeben und die Veröffentlichung weiterzuleiten.

   Erstellen Sie dieselbe Subskription und veröffentlichen Sie sie mit IBM WebSphere MQ Explorer. Simulieren Sie mit dem Clientdienstprogramm die Veröffentlichung und Subskription durch den Anwendungsclient. Starten Sie das Dienstprogramm über IBM WebSphere MQ Explorer und ändern Sie seine Benutzer-ID in die ID, die von Ihrer Client-App angenommen wurde.

8. Vergewissern Sie sich, dass der Subskribent berechtigt ist, die Veröffentlichung in die Warteschlange `SYSTEM.MQTT.TRANSMIT.QUEUE` zu stellen.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

9. Überprüfen Sie, ob die IBM WebSphere MQ -Punkt-zu-Punkt-Anwendung berechtigt ist, ihre Nachricht in `SYSTEM.MQTT.TRANSMIT.QUEUE`einzureihen.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

Siehe "Nachricht direkt an einen Client senden" unter Verteilte Steuerung von Warteschlangen für das Senden von Nachrichten an MQTT-Clients konfigurieren.

Problembehebung: Der Telemetrieservice (MQXR) wird nicht gestartet

Hier erhalten Sie Informationen zur Problembehebung, falls der Telemetrieservice (MQXR) nicht gestartet werden kann. Überprüfen Sie die Installation von WebSphere MQ Telemetry und vergewissern Sie sich, dass keine Dateien fehlen oder verschoben wurden. Vergewissern Sie sich außerdem, dass die richtigen Berechtigungen für die Dateien festgelegt sind. Überprüfen Sie die Pfade, in denen der Telemetrieservice (MQXR) nach den Programmen des Telemetrieservice (MQXR) sucht.

Vorbereitende Schritte

Die Komponente 'WebSphere MQ Telemetry' muss installiert sein. Der IBM WebSphere MQ Explorer verfügt über einen Telemetrieordner in **IBM WebSphere MQ > Warteschlangenmanager > *qMgrName* > Telemetry**. Ist der Ordner nicht vorhanden, ist die Installation fehlgeschlagen.

Der Telemetrieservice (MQXR) muss erstellt worden sein, damit er gestartet werden kann. Wenn der Telemetrieservice (MQXR) nicht erstellt wurde, führen Sie die **Beispielkonfiguration definieren ...** aus. Assistent im Ordner `Telemetry` .

Wenn der Telemetrieservice (MQXR) zuvor gestartet wurde, werden unter dem Ordner `Telemetry` die zusätzlichen Ordner **Kanäle** und **Kanalstatus** erstellt. Der Telemetrieservice `SYSTEM.MQXR.SERVICE` befindet sich im Ordner **Services**. Dieser ist sichtbar, wenn das Explorer-Optionsfeld zur Anzeige der Systemobjekte angeklickt wurde.

Klicken Sie mit der rechten Maustaste auf `SYSTEM.MQXR.SERVICE` , um den Service zu starten und zu stoppen, seinen Status anzuzeigen und anzuzeigen, ob Ihre Benutzer-ID berechtigt ist, den Service zu starten.

Informationen zu diesem Vorgang

Der Telemetrieservice (MQXR) `SYSTEM.MQXR.SERVICE` kann nicht gestartet werden. Startfehler können sich auf zwei verschiedene Arten äußern:

1. Der Startbefehl schlägt sofort fehl.
2. Der Startbefehl ist erfolgreich, unmittelbar darauf wird der Service jedoch gestoppt.

Vorgehensweise

1. Starten Sie den Service.

   Ergebnis

   Der Service wird umgehend gestoppt. In einem Fenster wird eine Fehlernachricht angezeigt. Beispiel:

   ```
   WebSphere MQ cannot process the request because the
   executable specified cannot be started. (AMQ4160)
   ```

   Ursache

   Dateien der Installation fehlen oder die Berechtigungen für die installierten Dateien sind falsch gesetzt.
   Die IBM WebSphere MQ Telemetry-Komponente ist nur auf einem von zwei Warteschlangenmanagern mit hoher Verfügbarkeit installiert. Wechselt die Warteschlangenmanagerinstanz in den Standby-Betrieb, versucht sie, `SYSTEM.MQXR.SERVICE` zu starten. Der Befehl zum Starten des Service schlägt fehl, da der Telemetrieservice (MQXR) nicht im Standby-Betrieb installiert wurde.

   Untersuchung

   Prüfen Sie die Fehlerprotokolle. Informationen hierzu finden Sie im Thema „Serverseitige Protokolle” auf Seite 169.

Aktionen
Installieren Sie die WebSphere MQ Telemetry-Komponente bzw. deinstallieren Sie sie und installieren Sie sie anschließend erneut.

2. Starten Sie den Service, warten Sie 30 Sekunden, aktualisieren Sie den Explorer und prüfen Sie den Servicestatus.

Ergebnis
Der Service wird gestartet und dann gestoppt.

Ursache
Von SYSTEM.MQXR.SERVICE wurde der Befehl **runMQXRService** gestartet, dieser ist jedoch fehlgeschlagen.

Untersuchung

Prüfen Sie die Fehlerprotokolle. Informationen hierzu finden Sie im Thema „Serverseitige Protokolle" auf Seite 169.

Stellen Sie fest, ob das Problem nur beim definierten Beispielkanal auftritt. Sichern Sie den Inhalt des Verzeichnisses *WMQ data directory*\Qmgrs*qMgrName*\mqxr\ und löschen Sie ihn. Führen Sie den Assistenten für die Beispielkonfiguration aus und versuchen Sie, den Service zu starten.

Aktionen
Stellen Sie fest, ob Berechtigungs- und Pfadprobleme vorliegen.

Problembehebung: Das JAAS-Anmeldemodul wird vom Telemetrieservice nicht aufgerufen

Stellen Sie fest, ob Ihr JAAS-Anmeldemodul tatsächlich nicht vom Telemetrieservice (MQXR) aufgerufen wird, und konfigurieren Sie JAAS entsprechend, um das Problem zu beheben.

Vorbereitende Schritte

Sie haben *WMQ installation directory*\mqxr\samples\LoginModule.java geändert, um Ihre eigene Authentifizierungsklasse *WMQ installation directory*\mqxr\samples\samples\LoginModule.class zu erstellen. Möglicherweise haben Sie ja auch eigene JAAS-Authentifizierungsklassen geschrieben und in ein Verzeichnis Ihrer Wahl gestellt. Nachdem Sie einige erste Tests mit dem Telemetrieservice (MQXR) ausgeführt haben, vermuten Sie, dass Ihre Authentifizierungsklasse nicht vom Telemetrieservice (MQXR) aufgerufen wird.

Anmerkung: Sichern Sie sich gegen die Möglichkeit ab, dass Ihre Authentifizierungsklassen durch eventuelle Wartungsaufgaben in WebSphere MQ überschrieben werden. Verwenden Sie Ihren eigenen Pfad für Authentifizierungsklassen anstelle eines Pfades in der WebSphere MQ-Verzeichnisstruktur.

Informationen zu diesem Vorgang

Um die Problembehebung besser zu veranschaulichen, wird in der Task ein Szenario eingesetzt. In diesem Szenario enthält ein Paket namens security.jaas eine JAAS-Authentifizierungsklasse namens JAASLogin.class. Es ist unter dem Pfad C:\WMQTelemetryApps\security\jaas gespeichert. Lesen Sie den Abschnitt JAAS-Konfiguration für den Telemetriekanal, in dem Sie hilfreiche Informationen zur Konfiguration von JAAS (Java Authentication and Authorization Service) für IBM WebSphere MQ Telemetry finden. Bei dem Beispiel („JAAS-Beispielkonfiguration" auf Seite 194) handelt es sich um eine Beispielkonfiguration.

Vorgehensweise

1. Prüfen Sie im Protokoll mqxr.log, ob von javax.security.auth.login.LoginException eine Ausnahme ausgelöst wurde.

Überprüfen Sie mithilfe der Informationen unter „Serverseitige Protokolle” auf Seite 169 den Pfad der Datei `mqxr.log`. In Abbildung 54 auf Seite 196 finden Sie ein Beispiel der im Protokoll aufgeführten Ausnahme.

2. Korrigieren Sie die JAAS-Konfiguration und vergleichen Sie sie dazu mit dem im Thema „JAAS-Beispielkonfiguration” auf Seite 194 behandelten Beispiel.
3. Ersetzen Sie Ihre Anmeldeklasse durch das Beispielelement `JAASLoginModule`, nachdem Sie es in das Authentifizierungspaket umstrukturiert haben, und implementieren Sie es unter Verwendung desselben Pfades. Wechseln Sie für `loggedIn` zwischen den Werten `true` und `false`.

   Falls das Problem nicht mehr besteht, wenn für `loggedIn` der Wert `true` angegeben ist, und wieder auftritt, wenn `loggedIn` auf `false` gesetzt ist, liegt der Fehler bei der Anmeldeklasse.
4. Prüfen Sie, ob das Problem eher mit der Autorisierung als mit der Authentifizierung zusammenhängt.
   a) Ändern Sie die Telemetriekanaldefinition entsprechend, damit eine Berechtigungsprüfung mit einer festgelegten Benutzer-ID erfolgt. Wählen Sie eine Benutzer-ID aus der Gruppe `mqm` aus.
   b) Führen Sie die Client-App erneut aus.

      Besteht das Problem nicht mehr, liegt die Lösung in der für die Autorisierung übergebenen Benutzer-ID. Welcher Benutzername wird übergeben? Geben Sie den Namen vom Anmeldemodul aus in einer Datei aus. Überprüfen Sie die zugehörigen Zugriffsberechtigungen mit IBM WebSphere MQ Explorer oder mit dem Befehl **dspmqauth**.

JAAS-Beispielkonfiguration

Verwenden Sie den Assistenten **Neuer Telemetriekanal** in WebSphere MQ Explorer für die Konfiguration eines Telemetriekanals. Der Client stellt am Port 1884 eine Verbindung her und verbindet sich mit dem Telemetriekanal `JAASMCAUser`. In Abbildung 48 auf Seite 194 ist ein Beispiel der Eigenschaftendatei für die Telemetrie dargestellt, die vom Telemetrie-Assistenten erstellt wird. Bearbeiten Sie diese Datei nicht direkt. Die Kanalauthentifizierung erfolgt mithilfe von JAAS unter Verwendung der Konfiguration `JAASConfig`. Nach der Clientauthentifizierung verwendet der Client die Benutzer-ID `Admin` für die Zugriffsberechtigung auf IBM WebSphere MQ-Objekte.

```
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
```

Abbildung 48. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\mqxr_win.properties

Die JAAS -Konfigurationsdatei enthält eine Zeilengruppe mit dem Namen `JAASConfig` , die die Java-Klasse `security.jaas.JAASLogin`benennt, die JAAS für die Authentifizierung von Clients verwenden soll.

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

Abbildung 49. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config

Beim Start von `SYSTEM.MQTT.SERVICE` wird der Pfad in Abbildung 50 auf Seite 195 dem Klassenpfad hinzugefügt.

```
CLASSPATH=C:\WMQTelemtryApps;
```

Abbildung 50. WMQ Installation directory\data\qmgrs\qMgrName\service.env

In Abbildung 51 auf Seite 195 ist der zusätzliche Pfad in Abbildung 50 auf Seite 195 dargestellt, der dem Klassenpfad hinzugefügt wird, der für den Telemetrieservice (MQXR) eingerichtet wird.

```
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;
```

Abbildung 51. Klassenpfadausgabe von 'runMQXRService.bat'

Die Ausgabe in Abbildung 52 auf Seite 195 zeigt, dass der Telemetrieservice (MQXR) mit der Kanaldefinition gestartet wurde, die in Abbildung 48 auf Seite 194 dargestellt ist.

```
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
```

Abbildung 52. WMQ Installation directory\data\qmgrs\qMgrName\errors\mqxr.log

Wenn sich die Client-App mit dem JAAS-Kanal verbindet und `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` nicht dem Namen einer JAAS-Zeilengruppe in der Datei `jaas.config` entspricht, schlägt die Verbindung fehl und der Client löst eine Ausnahmebedingung mit dem Rückkehrcode `0` aus; siehe Abbildung 53 auf Seite 196. Die zweite Ausnahmebedingung `Client is not connected (32104)` (Keine Verbindung zu Client) wurde ausgelöst, da der Client versucht hat, die Verbindung zu unterbrechen, als gar keine Verbindung bestand.

```
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(MqttIn
putStream.java: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(Exception
Helper.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)
```

Abbildung 53. Ausgelöste Ausnahmebedingung bei der Verbindung von 'com.ibm.mq.id.PubAsyncRestartable'

In der Datei `mqxr.log` ist eine zusätzliche Ausgabe enthalten (siehe Abbildung 53 auf Seite 196).

Der Fehler wird von JAAS erkannt, der `javax.security.auth.login.LoginException` mit der Ursache `No LoginModules configured for JAAS` (Keine Anmeldemodule für JAAS konfiguriert) auslöst. Die Ursache könnte wie in Abbildung 54 auf Seite 196 ein falscher Konfigurationsname sein. Auch andere JAAS-Probleme beim Laden der JAAS-Konfiguration könnten der Grund hierfür sein.

Falls von JAAS keine Ausnahmebedingung gemeldet wird, konnte die in der Zeilengruppe `JAASConfig` benannte Klasse `security.jaas.JAASLogin` erfolgreich geladen werden.

```
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
```

Abbildung 54. `mqxr.log` - Fehler beim Laden der JAAS-Konfiguration

Problembehebung: Dämon starten oder ausführen

Lesen Sie das Konsolenprotokoll für den IBM WebSphere MQ Telemetry-Dämon für Geräte, aktivieren Sie die Tracefunktion oder verwenden Sie die Tabelle zu den Symptomen in diesem Abschnitt, um Probleme mit dem Dämon zu beheben.

Vorgehensweise

1. Überprüfen Sie das Konsolenprotokoll.

   Wenn der Dämon im Vordergrund ausgeführt wird, werden die Konsolennachrichten in das Terminalfenster geschrieben. Falls der Dämon im Hintergrund gestartet wurde, wurde die Standardausgabe (`stdout`) von Ihnen an die Konsole umgeleitet.
2. Starten Sie den Dämon erneut.

   Änderungen an der Konfigurationsdatei treten erst bei einem Neustart des Dämons in Kraft.
3. Lesen Sie die Informationen in Tabelle 7 auf Seite 197:

Tabelle 7. Tabelle mit Symptomen

Problem	Vorgeschlagene Lösung

Beim Starten des Dämons unter Windows wird die folgende Nachricht angezeigt:
`The system cannot execute the specified program`
oder
`The application has failed to start because its side-by-side configuration is incorrect.`
	Installieren Sie das Microsoft Visual C++ 2008 Redistributable Package.

Zwei oder mehr Dämonen oder MQTT-fähige Server sind durch eine Bridge oder durch Bridges vernetzt und der Prozessor zeigt eine extrem hohe Auslastung an.	Möglicherweise liegt eine Nachrichtenschleife vor, bei der eine oder mehrere Nachrichten immer wieder von einem Server an einen anderen übergeben werden. Überprüfen Sie die Themenparameter in den Konfigurationsdateien. Falls möglich, machen Sie genauere Angaben zu den Themen. Die Angabe zahlreicher Platzhalterzeichen für beide Richtungen ist die häufigste Ursache für Verbindungsschleifen.
Die Bridge kann keine Verbindung zu einem fernen MQTT-fähigen Server herstellen, zu dem andere MQTT-Clients eine Verbindung aufbauen können.	Möglicherweise ist der ferne Server nicht mit Versuchen kompatibel, bei denen festgestellt werden soll, ob der ferne Server auch ein WebSphere MQ Telemetry-Dämon für Geräte ist. Versuchen Sie, die Einstellung **`try_private`** auf `off` zu setzen, um die spezielle Verarbeitung zur Vermeidung von Nachrichtenschleifen zu inaktivieren.
Bei der Konfiguration einer Bridge wird die folgende Nachricht ausgegeben: `Warning: Connect was not first packet on socket 1888, got CONNACK.`	Möglicherweise haben Sie eine Bridge so konfiguriert, dass ein Loopback zum lokalen Dämon erfolgt. Loopback wird nicht unterstützt.

Problembehebung: MQTT-Clients stellen keine Verbindung zum Dämon her

Clients verbinden sich nicht mit dem Dämon oder der Dämon verbindet sich nicht mit anderen Dämonen oder einem WebSphere MQ-Telemetriekanal.

Informationen zu diesem Vorgang

Erstellen Sie für jedes vom Dämon gesendete und empfangene MQTT-Paket einen Trace.

Vorgehensweise

Setzen Sie den Parameter **`trace_output`** in der Konfigurationsdatei des Dämons auf `protocol` oder senden Sie unter Verwendung der Datei `amqtdd.upd` einen Befehl an den Dämon.

Unter Nachrichten zwischen dem IBM WebSphere MQ Telemetry-Dämon für Geräte und IBM WebSphere MQ übertragen finden Sie ein Beispiel für die Verwendung der Datei `amqtdd.upd`.

Der Dämon gibt unter Verwendung der Protokolleinstellung eine Nachricht an die Konsole aus, in der jedes von ihm gesendete und empfangene MQTT-Paket beschrieben wird.

Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder andere Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb von Fremdprodukten, Fremdprogrammen und Fremdservices liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieser Dokumentation ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Europe
IBM Europe, Middle East and Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Défense
U.S.A.

Bei Lizenzanforderungen zu Double-Byte-Information (DBCS) wenden Sie sich bitte an die IBM Abteilung für geistiges Eigentum in Ihrem Land oder senden Sie Anfragen schriftlich an folgende Adresse:

Lizenzierung von geistigem Eigentum

IBM Japan, Ltd.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die Angaben in dieser Veröffentlichung werden in regelmäßigen Zeitabständen aktualisiert. Die Änderungen werden in Überarbeitungen oder in Technical News Letters (TNLs) bekanntgegeben. IBM kann jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Europe, Middle East and Africa
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in diesen Informationen beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Die in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Um diese so realistisch wie möglich zu gestalten, enthalten sie auch Namen von Personen, Firmen, Marken und Produkten. Sämtliche dieser Namen sind fiktiv. Ähnlichkeiten mit Namen und Adressen tatsächlicher Unternehmen oder Personen sind zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musterprogramme, die in Quellensprache geschrieben sind. Sie dürfen diese Musterprogramme kostenlos (d. h. ohne Zahlung an IBM) kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten.

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farbabbildungen.

Informationen zu Programmierschnittstellen

Die bereitgestellten Informationen zur Programmierschnittstelle sollen Sie bei der Erstellung von Anwendungssoftware für dieses Programm unterstützen.

Dieses Handbuch enthält Informationen zu geplanten Programmierschnittstellen, die es dem Kunden ermöglichen, Programme zum Abrufen der Services von IBM WebSphere MQzu schreiben.

Diese Informationen können jedoch auch Angaben über Diagnose, Bearbeitung und Optimierung enthalten. Die Informationen zu Diagnose, Bearbeitung und Optimierung sollten Ihnen bei der Fehlerbehebung für die Anwendungssoftware helfen.

Wichtig: Verwenden Sie diese Diagnose-, Änderungs-und Optimierungsinformationen nicht als Programmierschnittstelle, da sie Änderungen unterliegen.

Marken

IBM, das IBM Logo, ibm.com, sind Marken der IBM Corporation in den USA und/oder anderen Ländern. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein.

Microsoft und Windows sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Dieses Produkt enthält Software, die von Eclipse Project (http://www.eclipse.org/) entwickelt wurde.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Teilenummer:

(1P) P/N: