

7.5

モバイル・メッセージングおよび M2M

**IBM**

## 注記

本書および本書で紹介する製品をご使用になる前に、[193 ページの『特記事項』](#)に記載されている情報をお読みください。

本書は、IBM® WebSphere® MQ バージョン 7 リリース 5、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様が IBM に情報を送信する場合、お客様は IBM に対し、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で情報を使用または配布する非独占的な権利を付与します。

© Copyright International Business Machines Corporation 2007 年, 2024.

# 目次

<b>モバイル・メッセージングおよび M2M</b> .....	<b>5</b>
MQTT の概要.....	8
MQTT クライアントの概要.....	11
Java 用の MQTT クライアントの概要.....	12
Android 上の Java 用の MQTT クライアント の概要.....	18
JavaScript 用の MQTT メッセージング・クライアントの概要.....	25
C 用の MQTT クライアントの概要.....	27
iOS における C 用 MQTT クライアントの概要.....	49
MQTT コマンド行サンプル・プログラム.....	51
MQTT セキュリティー.....	53
セキュア MQTT クライアント・サンプル Java アプリケーション のビルドと実行.....	57
SSL を介した Android での MQTT クライアント・サンプル Java アプリの接続.....	65
JAAS を使用した MQTT クライアント Java アプリの認証.....	74
SSL を介した JavaScript 用の MQTT メッセージング・クライアント と WebSockets の接続.....	79
セキュア MQTT クライアント・サンプル C アプリケーション のビルドと実行.....	87
鍵と証明書の生成.....	97
MQTT クライアントの識別、許可、および認証.....	104
SSL を使用したテレメトリー・チャンネルの認証.....	108
テレメトリー・チャンネルでのパブリケーションのプライバシー.....	111
MQTT クライアントおよびテレメトリー・チャンネルの SSL 構成.....	111
テレメトリー・チャンネルの JAAS 構成.....	116
プログラミングの概念.....	118
JavaScript 用の MQTT メッセージング・クライアント と Web アプリケーション.....	119
JavaScript でのメッセージング・アプリケーションのプログラミング方法.....	122
MQTT クライアント・アプリケーションでのコールバックと同期.....	126
クリーン・セッション.....	129
クライアント ID.....	130
送達トークン.....	131
遺言パブリケーション.....	132
MQTT クライアントでのメッセージ持続性.....	132
パブリケーション.....	134
MQTT クライアントによって提供されるサービス品質.....	135
保存パブリケーションおよび MQTT クライアント.....	136
サブスクリプション.....	137
MQTT クライアントの トピック・ストリングおよびトピック・フィルター.....	138
MQTT クライアントのプログラミング・リファレンス.....	139
MQTT サーバーの概要.....	139
MQTT サーバーとしての IBM WebSphere MQ.....	141
IBM WebSphere MQ Telemetry デーモン (デバイス用) の概念.....	152
MQTT クライアントのトラブルシューティング.....	164
テレメトリー・ログ、エラー・ログ、および構成ファイルの場所.....	165
MQTT v3 Java クライアントの理由コード.....	167
テレメトリー (MQXR) サービスのトレース.....	168
MQTT v3 Java クライアントのトレース.....	169
C 用の MQTT クライアントのトレース.....	171
MQTT (Paho) Java クライアントのトレースおよびデバッグ.....	172
MQTT JavaScript クライアントのトレース.....	175
MQTT クライアントで SHA-2 暗号スイートを使用する場合のシステム要件.....	176
SSL を介したモバイル・メッセージング Web アプリケーションのブラウザー・サポートに関する制約事項.....	177
問題の解決: MQTT クライアントが接続しない.....	181
問題の解決: MQTT クライアントの接続が切断される.....	183

問題の解決: MQTT アプリケーションで失われたメッセージ.....	184
問題の解決: テレメトリー (MQXR) サービスが開始しない.....	186
問題の解決: JAAS ログイン・モジュールがテレメトリー・サービスによって呼び出されない.....	188
問題の解決: デーモンの開始または実行.....	191
問題の解決: MQTT クライアントがデーモンに接続していない.....	191
<b>特記事項.....</b>	<b>193</b>
プログラミング・インターフェース情報.....	194
商標.....	194

## MQTT の概要

MQ テレメトリー・トランスポート (MQTT) を使用してモバイル・アプリケーション間でメッセージを送信する方法について説明します。このプロトコルは、ワイヤレス・ネットワークおよび低帯域幅ネットワークでの使用を目的としています。MQTT を使用するモバイル・アプリケーションは、MQTT ライブラリーを呼び出すことにより、メッセージの送受信を行います。メッセージは MQTT メッセージング・サーバーを介して交換されます。MQTT クライアントおよびサーバーはモバイル・アプリケーションでの高信頼性メッセージ送信の複雑な処理を行いながら、同時にネットワーク管理コストを低く抑えます。

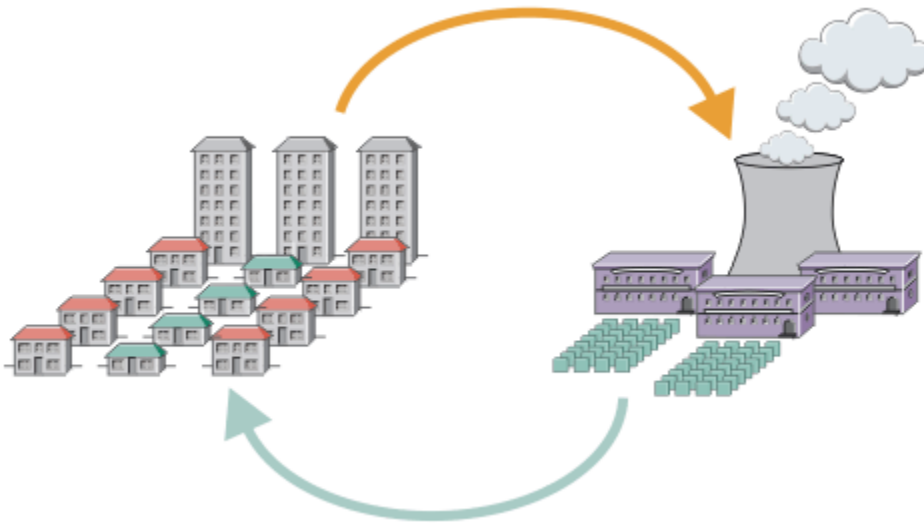
MQTT アプリケーションは、スマートフォンやタブレットなどのモバイル・デバイスで実行されます。MQTT は、遠隔測定での、センサーからのデータの受信、およびリモート側でのセンサーの制御にも使用します。MQTT は、モバイル・デバイスおよびセンサーに対して、送達保証を備えた極めてスケーラブルなパブリッシュ/サブスクライブ・プロトコルを提供します。MQTT メッセージの送受信を行うには、アプリケーションに MQTT クライアント・ライブラリーを追加します。

MQTT クライアント・ライブラリーは小さなものです。このライブラリーはメールボックスのように動作し、MQTT サーバーに接続した他の MQTT アプリケーションとメッセージを送受信します。応答を待機しているサーバーに接続し続けるのではなく、メッセージを送信することにより、MQTT アプリケーションはバッテリー駆動時間を延ばします。ライブラリーは、MQTT version 3.1 プロトコルを実行している MQTT サーバー経由でメッセージを他のデバイスに送信します。ユーザーは特定のクライアントにメッセージを送信することも、パブリッシュ/サブスクライブ・メッセージングを使用して多数のデバイスに接続することもできます。

MQTT クライアント・ライブラリーは、MQTT プロトコルを使用して、モバイル・デバイスおよびセンサー用のアプリケーションを MQTT サーバーに接続します。

IBM MessageSight および IBM WebSphere MQ は MQTT サーバーです。これらは、多数の MQTT クライアント・アプリケーションを接続したり、MQTT ネットワークや IBM WebSphere MQ ネットワークと一緒に接続したりできます。139 ページの『MQTT サーバーの概要』を参照してください。IBM WebSphere MQ および IBM MessageSight は両方とも、モバイル・デバイスやセンサー上で稼働している外部 Web アプリケーション間や、企業内で稼働しているその他のタイプのパブリッシュ/サブスクライブ・アプリケーションやメッセージング・アプリケーション間のブリッジを形成することができます。このブリッジにより、モバイル・デバイスとセンサーを組み込んだ "スマート・ソリューション" の作成が容易になります。

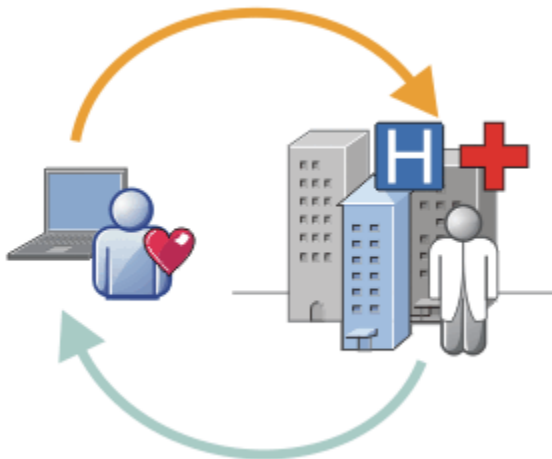
スマート・ソリューションは、モバイル・デバイスおよびセンサー・デバイス上で実行されるアプリケーションのために、インターネットで使用可能な情報の宝庫の錠を開けます。遠隔測定に基づくスマート・アプリケーションの例として、スマート電力とスマート・ヘルスケア・サービスの 2 つが挙げられます。



- エネルギー使用量データを含んだ MQTT メッセージがサービス・プロバイダーに送られます。
- 遠隔測定アプリケーションは、エネルギー使用量データの分析に基づいた制御コマンドを送信します。
- 詳しくは、[遠隔測定シナリオ: 家庭エネルギーのモニターと制御](#)を参照してください。

図 1. スマート電力計量

図 2. スマート・ヘルスケア・モニター



- 遠隔測定アプリケーションは、医療データを病院と医師に送信します。
- 医療データの分析に基づいて、MQTT 警報やフィードバックが送られます。
- 詳しくは、[遠隔測定シナリオ: 在宅患者モニター](#)を参照してください。

MQTT protocol 用の独自のアプリを作成することで、MQTT を小さなデバイスに組み込むことができます。この作業に役立つ情報は、IBM で提供されるクライアント・ライブラリー (MQTT を介して実行されるアプリケーションをサポートする) にあります。11 ページの『MQTT クライアントの概要』を参照してください。IBM は、iOS アプリケーションと Android アプリケーション用のクライアント・ライブラリー

**V7.5.0.1**、およびプラットフォーム非依存の Web アプリケーション用の JavaScript ブラウザー・クライアントを提供します。**V7.5.0.1** JavaScript クライアント・ページは、WebSockets を介して MQTT プロトコルを使用して IBM MessageSight および IBM WebSphere MQ に接続します。IBM には、Linux® および Windows 上の C および Java 用の MQTT サンプル・アプリケーションも用意されています。

C と Java のライブラリーは、iOS、Android、Windows、および多くの UNIX and Linux プラットフォームで稼働します。MQTT クライアント・ライブラリー用の C ソース・コードは、他のプラットフォームに移植できます。C および Java 用の MQTT クライアント・ライブラリーは、Eclipse Paho プロジェクトのオー

オープン・ソース・ライセンスで入手できます。[Eclipse Paho](#) を参照してください。MQTT protocol 仕様は公開されており、[MQTT.org](#) から入手できます。

## MQTT protocol

MQTT protocol は、クライアントが小型で、ネットワーク帯域幅を効率的に使用するという意味で、軽量です。MQTT プロトコルは、送達保証と応答不要送信の転送をサポートします。このプロトコルでは、メッセージ送達はアプリケーションから分離されています。アプリケーションでの分離範囲は MQTT クライアントと MQTT サーバーがどのように記述されているかに応じて異なります。送達が分離されていると、アプリケーションはサーバー接続とメッセージ待機から解放されます。対話モデルは E メールに似ていますが、アプリケーション・プログラミングに合わせて最適化されています。

MQTT V3.1 プロトコルは公開されています。[MQTT V3.1 Protocol Specification](#) を参照してください。この仕様は、このプロトコルに関する以下の際立ったフィーチャーを規定しています。

- パブリッシュ/サブスクライブ・プロトコルです。
  - 1 対多メッセージ配布を提供することに加え、パブリッシュ/サブスクライブによってアプリケーションを分離します。どちらのフィーチャーも多数のクライアントを持つアプリケーションで有用です。
- メッセージ・コンテンツにはまったく依存しません。
- 基本的なネットワーク接続を提供する TCP/IP を介して稼働します。
- メッセージ送信のサービスの品質には次の 3 種類があります。

### "最高 1 回"

メッセージは、基になるインターネット・プロトコル・ネットワークのベスト・エフォートに従って送信されます。メッセージ損失が発生する可能性があります。

このサービスの品質は、例えば環境センサー・データの通信に使用します。次の読み取りがすぐ後にパブリッシュされるのであれば、個別の読み取りが失われたかどうかは問題になりません。

### "最低 1 回"

メッセージ送信は保証されますが、重複が発生する可能性があります。

### "正確に 1 回"

メッセージは正確に 1 回着信することが保証されます。

このサービスの品質は、例えば請求システムに使用します。メッセージの重複や逸失は、不都合を生じさせたり、誤った課金を負わせることになったりする可能性があります。

- ネットワーク上のメッセージ・フローの管理の点でコストが抑えられます。例えば、固定長ヘッダーは 2 バイトのみの長さであり、プロトコル交換はネットワーク・トラフィックが削減されるように最小化されています。
- これには、MQTT サーバーからのクライアントの異常切断をサブスクライバーに通知する "「遺言」" 機能があります。[132 ページの『遺言パブリケーション』](#)を参照してください。

MQTT version 3.1 は、IBM WebSphere MQ および IBM MessageSight によってサポートされています。MQTT は TCP/IP 上に実装されています。別のバージョンのプロトコルである MQTT-S は、非 TCP/IP ネットワークで使用可能です。[MQTT-S version 1.2 仕様](#)を参照してください。

## MQTT コミュニティー

IBM は、IBM MessageSight および IBM WebSphere MQ 用のアプリケーションを作成する MQTT 開発者のために [IBM Developer メッセージング コミュニティー](#) を実行しています。

[MQTT.org](#) は MQTT プロトコルの実装と拡張機能についての学習や意見交換に適した場所です。

MQTT はオープン・ソースの Eclipse プロジェクトであり、[Eclipse Technology Project](#) の下にあります。Paho コミュニティーではオープン・ソースのクライアントとサーバーを開発しています。[Eclipse Paho](#) を参照してください。

## MQTT の概要

MQ テレメトリー・トランスポート (MQTT) を使用してモバイル・アプリケーション間でメッセージを送信する方法について説明します。このプロトコルは、ワイヤレス・ネットワークおよび低帯域幅ネットワークでの使用を目的としています。MQTT を使用するモバイル・アプリケーションは、MQTT ライブラリーを呼び出すことにより、メッセージの送受信を行います。メッセージは MQTT メッセージング・サーバーを介して交換されます。MQTT クライアントおよびサーバーはモバイル・アプリケーションでの高信頼性メッセージ送信の複雑な処理を行いながら、同時にネットワーク管理コストを低く抑えます。

MQTT アプリケーションは、スマートフォンやタブレットなどのモバイル・デバイスで実行されます。MQTT は、遠隔測定での、センサーからのデータの受信、およびリモート側でのセンサーの制御にも使用します。MQTT は、モバイル・デバイスおよびセンサーに対して、送達保証を備えた極めてスケーラブルなパブリッシュ/サブスクライブ・プロトコルを提供します。MQTT メッセージの送受信を行うには、アプリケーションに MQTT クライアント・ライブラリーを追加します。

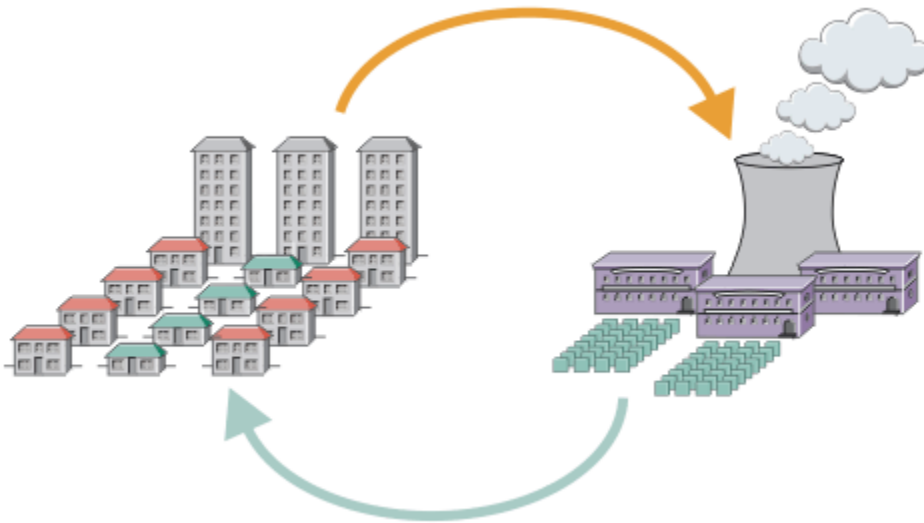
MQTT クライアント・ライブラリーは小さなものです。このライブラリーはメールボックスのように動作し、MQTT サーバーに接続した他の MQTT アプリケーションとメッセージを送受信します。応答を待機しているサーバーに接続し続けるのではなく、メッセージを送信することにより、MQTT アプリケーションはバッテリー駆動時間を延ばします。ライブラリーは、MQTT version 3.1 プロトコルを実行している MQTT サーバー経由でメッセージを他のデバイスに送信します。ユーザーは特定のクライアントにメッセージを送信することも、パブリッシュ/サブスクライブ・メッセージングを使用して多数のデバイスに接続することもできます。

MQTT クライアント・ライブラリーは、MQTT プロトコルを使用して、モバイル・デバイスおよびセンサー用のアプリケーションを MQTT サーバーに接続します。

IBM MessageSight および IBM WebSphere MQ は MQTT サーバーです。これらは、多数の MQTT クライアント・アプリケーションを接続したり、MQTT ネットワークや IBM WebSphere MQ ネットワークと一緒に接続したりできます。139 ページの『MQTT サーバーの概要』を参照してください。IBM WebSphere MQ および IBM MessageSight は両方とも、モバイル・デバイスやセンサー上で稼働している外部 Web アプリケーション間や、企業内で稼働しているその他のタイプのパブリッシュ/サブスクライブ・アプリケーションやメッセージング・アプリケーション間のブリッジを形成することができます。このブリッジにより、モバイル・デバイスとセンサーを組み込んだ "スマート・ソリューション" の作成が容易になります。

スマート・ソリューションは、モバイル・デバイスおよびセンサー・デバイス上で実行されるアプリケーションのために、インターネットで使用可能な情報の宝庫の錠を開けます。遠隔測定に基づくスマート・アプリケーションの例として、スマート電力とスマート・ヘルスケア・サービスの 2 つが挙げられます。

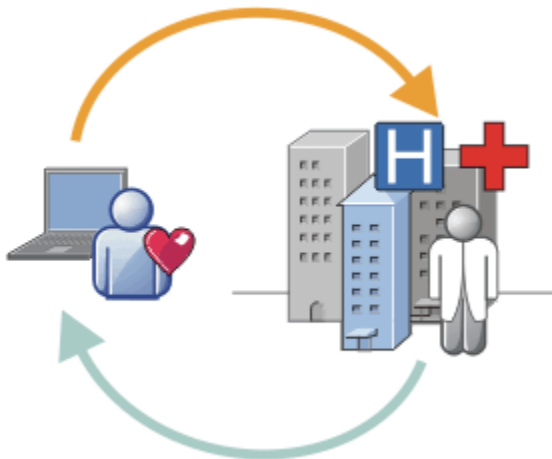




- エネルギー使用量データを含んだ MQTT メッセージがサービス・プロバイダーに送られます。
- 遠隔測定アプリケーションは、エネルギー使用量データの分析に基づいた制御コマンドを送信します。
- 詳しくは、[遠隔測定シナリオ: 家庭エネルギーのモニターと制御](#)を参照してください。

図 3. スマート電力計量

図 4. スマート・ヘルスケア・モニター



- 遠隔測定アプリケーションは、医療データを病院と医師に送信します。
- 医療データの分析に基づいて、MQTT 警報やフィードバックが送られます。
- 詳しくは、[遠隔測定シナリオ: 在宅患者モニター](#)を参照してください。

MQTT protocol 用の独自のアプリを作成することで、MQTT を小さなデバイスに組み込むことができます。この作業に役立つ情報は、IBM で提供されるクライアント・ライブラリー (MQTT を介して実行されるアプリケーションをサポートする) にあります。11 ページの『MQTT クライアントの概要』を参照してください。IBM は、iOS アプリケーションと Android アプリケーション用のクライアント・ライブラリー

**V7.5.0.1**、およびプラットフォーム非依存の Web アプリケーション用の JavaScript ブラウザー・クライアントを提供します。**V7.5.0.1** JavaScript クライアント・ページは、WebSockets を介して MQTT プロトコルを使用して IBM MessageSight および IBM WebSphere MQ に接続します。IBM には、Linux および Windows 上の C および Java 用の MQTT サンプル・アプリケーションも用意されています。

C と Java のライブラリーは、iOS、Android、Windows、および多くの UNIX and Linux プラットフォームで稼働します。MQTT クライアント・ライブラリー用の C ソース・コードは、他のプラットフォームに移植できます。C および Java 用の MQTT クライアント・ライブラリーは、Eclipse Paho プロジェクトのオー

プン・ソース・ライセンスで入手できます。[Eclipse Paho](#) を参照してください。MQTT protocol 仕様は公開されており、[MQTT.org](#) から入手できます。

## MQTT protocol

MQTT protocol は、クライアントが小型で、ネットワーク帯域幅を効率的に使用するという意味で、軽量です。MQTT プロトコルは、送達保証と応答不要送信の転送をサポートします。このプロトコルでは、メッセージ送達はアプリケーションから分離されています。アプリケーションでの分離範囲は MQTT クライアントと MQTT サーバーがどのように記述されているかに応じて異なります。送達が分離されていると、アプリケーションはサーバー接続とメッセージ待機から解放されます。対話モデルは E メールに似ていますが、アプリケーション・プログラミングに合わせて最適化されています。

MQTT V3.1 プロトコルは公開されています。[MQTT V3.1 Protocol Specification](#) を参照してください。この仕様は、このプロトコルに関する以下の際立ったフィーチャーを規定しています。

- パブリッシュ/サブスクライブ・プロトコルです。
  - 1 対多メッセージ配布を提供することに加え、パブリッシュ/サブスクライブによってアプリケーションを分離します。どちらのフィーチャーも多数のクライアントを持つアプリケーションで有用です。
- メッセージ・コンテンツにはまったく依存しません。
- 基本的なネットワーク接続を提供する TCP/IP を介して稼働します。
- メッセージ送信のサービスの品質には次の 3 種類があります。

### "最高 1 回"

メッセージは、基になるインターネット・プロトコル・ネットワークのベスト・エフォートに従って送信されます。メッセージ損失が発生する可能性があります。

このサービスの品質は、例えば環境センサー・データの通信に使用します。次の読み取りがすぐ後にパブリッシュされるのであれば、個別の読み取りが失われたかどうかは問題になりません。

### "最低 1 回"

メッセージ送信は保証されますが、重複が発生する可能性があります。

### "正確に 1 回"

メッセージは正確に 1 回着信することが保証されます。

このサービスの品質は、例えば請求システムに使用します。メッセージの重複や逸失は、不都合を生じさせたり、誤った課金を負わせることになったりする可能性があります。

- ネットワーク上のメッセージ・フローの管理の点でコストが抑えられます。例えば、固定長ヘッダーは 2 バイトのみの長さであり、プロトコル交換はネットワーク・トラフィックが削減されるように最小化されています。
- これには、MQTT サーバーからのクライアントの異常切断をサブスクライバーに通知する "「遺言」" 機能があります。[132 ページの『遺言パブリケーション』](#)を参照してください。

MQTT version 3.1 は、IBM WebSphere MQ および IBM MessageSight によってサポートされています。MQTT は TCP/IP 上に実装されています。別のバージョンのプロトコルである MQTT-S は、非 TCP/IP ネットワークで使用可能です。[MQTT-S version 1.2 仕様](#)を参照してください。

## MQTT コミュニティー

IBM は、IBM MessageSight および IBM WebSphere MQ 用のアプリケーションを作成する MQTT 開発者のために [IBM Developer メッセージング コミュニティー](#) を実行しています。

[MQTT.org](#) は MQTT プロトコルの実装と拡張機能についての学習や意見交換に適した場所です。

MQTT はオープン・ソースの Eclipse プロジェクトであり、[Eclipse Technology Project](#) の下にあります。Paho コミュニティーではオープン・ソースのクライアントとサーバーを開発しています。[Eclipse Paho](#) を参照してください。

# MQTT クライアントの概要

MQTT クライアント・ライブラリーを使用するサンプル MQTT クライアント・アプリケーションを作成して実行することにより、モバイル・アプリケーションまたは M2M (machine-to-machine) アプリケーションの開発を開始できます。サンプル・アプリケーションおよび関連するクライアント・ライブラリーは、IBM の Mobile Messaging and M2M クライアント・パックで入手できます。Java、JavaScript、および C で作成されたアプリケーションとクライアント・ライブラリーのバージョンがあります。これらのアプリケーションは、Apple の Android デバイスや製品など、ほとんどのプラットフォームおよびデバイスで実行できます。

## 始める前に

アプリケーションをビルドして実行するには、ターゲット・デバイスまたはプラットフォーム用のアプリケーションのビルドと、使用するプログラミング言語について、ある程度の経験が必要になります。選択したデバイスまたはプラットフォームでサンプル・アプリケーションを稼働するためには、少しの経験があれば通常は十分です。

IBM WebSphere MQ や IBM MessageSight などのエンタープライズ強度の MQTT サーバーを使用する場合は、サンプル・アプリからの情報を既存のエンタープライズ・アプリと交換できます。

## このタスクについて

目的は次のとおりです。

1. [クライアント・アプリケーションを接続できる MQTT サーバーを選択](#)します。
2. [Mobile Messaging and M2M クライアント・パック](#)をダウンロードします。
3. ターゲット・デバイスまたはプラットフォーム用に、クライアント・パックのサンプル・アプリケーションをビルドします。
4. サンプルを MQTT サーバーに接続して、それらが期待どおりに動作することを検証します。

デバイスまたはプラットフォーム用にサンプル・アプリケーションをビルドしてテストすることにより、独自のクライアント・アプリケーションをビルドするために使用できる作業用開発環境が作成されます。

Mobile Messaging and M2M クライアント・パックには MQTT SDK が含まれています。この SDK では、以下のリソースが提供されます。

- Java、JavaScript、および C で作成されたサンプル MQTT クライアント・アプリケーション。
- これらのクライアント・アプリケーションをサポートし、ほとんどのプラットフォームとデバイスで実行できるようにする MQTT クライアント・ライブラリー。

SDK には、C 用の MQTT クライアントのソース・コードも含まれています。このソース・コードは、他のプラットフォームで使用する C 用の MQTT クライアント・ライブラリーをビルドするために調整可能です。これを行う方法については、[31 ページの『C 用の MQTT クライアント・ライブラリーのビルド』](#)を参照してください。C 用の MQTT クライアントのソース・コードは、[Eclipse Paho](#) のオープン・ソース・ライセンスでも入手可能です。

## 手順

以下の記事は、デスクトップ・コンピューター上、または Android か Apple のモバイル・デバイス上で、サンプル MQTT アプリケーションをビルドして実行するプラットフォーム固有のステップを示しています。

- [12 ページの『Java 用の MQTT クライアントの概要』](#)
- [18 ページの『Android 上の Java 用の MQTT クライアントの概要』](#)
- **V7.5.0.1**
- [25 ページの『JavaScript 用の MQTT メッセージング・クライアントの概要』](#)
- [27 ページの『C 用の MQTT クライアントの概要』](#)
- [49 ページの『iOS における C 用 MQTT クライアントの概要』](#)

## 次のタスク

新しい MQTT アプリケーションを開発するには、以下のスキルを持っているか、または習得する必要があります。

- デバイスまたはプラットフォームで必要な言語でのプログラミング。
- ターゲットのデバイスまたはプラットフォーム用のプログラミング。
- パブリッシュ/サブスクライブのアプリケーションの設計。
- MQTT プログラミング・モデルのプログラムの設計。
- 選択したモバイル・デバイスで実行するプログラムの設計。
- プログラムを保護するための SSL および JAAS の使用。

MQTT はメッセージングおよびキューイングのシステムなので、MQTT クライアントを別のデバイスまたはアプリケーションに接続するために、ネットワーク・プログラミングのスキルは必要ありません。MQTT クライアント・ライブラリーが、アプリケーションのネットワーク接続を管理します。

MQTT クライアントを既存のエンタープライズ・アプリケーションに統合するためには、2つの選択肢があります。MQTT パブリッシュ/サブスクライブのトピックを (例えば) IBM WebSphere MQ や JMS アプリケーションと共有するか、独自の統合アダプターを別の MQTT クライアントとして記述することができます。

現在、参照できる情報のソースには、以下のものがあります。

- [WebSphere MQ Telemetry のアプリケーションの開発](#)
- [MQTT.org](#)
- [Eclipse Paho](#)

### 関連概念

[139 ページの『MQTT サーバーの概要』](#)

## Java 用の MQTT クライアントの概要

IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてを使用して、MQTT Client for Java サンプル・アプリケーションを稼働させます。サンプル・アプリケーションは、IBM の MQTT Software Development Toolkit (SDK) のクライアント・ライブラリーを使用します。SampleAsyncCallback サンプル・アプリケーションは、Android およびその他のイベント・ドリブン・オペレーティング・システム用の MQTT アプリケーションを作成するためのモデルです。

### 始める前に

- JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアントアプリを実行できます。 [. IBM Mobile Messaging and M2M クライアント・パックのシステム要件を参照](#) してください。
- クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。

### このタスクについて

このタスクの目的は、MQTT Client for Java サンプル・アプリケーションを作成して実行し、それを MQTT version 3 サーバーとして IBM WebSphere MQ または IBM MessageSight に接続して、メッセージを交換できることを確認することです。

このタスクに従って、サンプル・アプリケーションを Eclipse ワークベンチから、またはコマンド・ラインから実行します。この例に示されているステップは、Windows 用です。少しの修正をするだけで、JSE 1.5 以上をサポートする任意のプラットフォーム上でサンプル・アプリケーションを実行できます。

アプリケーションは、IBM WebSphere MQ と同じサーバーで実行できます。このサーバーでは、IBM WebSphere MQ に接続するアプリケーションを実行するための環境が構成されています。 [143 ページの『コマンド・ラインからの MQTT サービスの構成』](#) のタスクに従い、IBM WebSphere MQ に IBM WebSphere MQ Telemetry オプションを指定して、Windows または Linux にインストールして構成します。環境をイ

インストールして構成したら、サンプル・アプリケーション MQTTV3Sample を実行して、インストール済み環境を検査します。

## 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。

サーバーは MQTT version 3.1 プロトコルをサポートしている必要があります。IBM のすべての MQTT サーバー (IBM WebSphere MQ および IBM MessageSight を含む) がこれを行います。139 ページの『MQTT サーバーの概要』を参照してください。

2. オプション: MQTT サーバーを構成します。

- IBM WebSphere MQ では、以下のいずれかのタスクを実行して、キュー・マネージャーをセットアップし、そのテレメトリー (MQXR) サービスを構成する必要があります。
  - 143 ページの『コマンド・ラインからの MQTT サービスの構成』
  - 146 ページの『IBM WebSphere MQ Explorer を使用した MQTT サービスの構成』
- 他のサーバーの場合は、そのサーバーの文書を参照してください。Really Small Message Broker には構成ステップは必要ありません。Really Small Message Broker を参照してください。

3. Mobile Messaging and M2M クライアント・パックをダウンロードし、MQTT SDK をインストールします。

インストール・プログラムは用意されておらず、ダウンロードしたファイルを展開するだけです。

- a. Mobile Messaging and M2M クライアント・パックをダウンロードします。

- b. SDK をインストールするフォルダーを作成します。

フォルダーの名前を MQTT にすることもできます。ここでは、このフォルダーへのパスを *sdkroot* として示します。

- c. 圧縮された Mobile Messaging and M2M クライアント・パック ファイルの内容を *sdkroot* に展開します。展開すると、*sdkroot*\SDK から始まるディレクトリー・ツリーが作成されます。

4. Java Development Kit (JDK) バージョン 6 以降をインストールします。

Java アプリ for Android を開発しているため、JDK は Oracle からのものでなければなりません。JDK は [Java SE ダウンロード](#) から入手できます。

5. 1 つ以上の MQTT Client for Java サンプル・アプリケーションをコンパイルして実行します。

- 14 ページの『コマンド・ラインから、Paho サンプル・プログラムをコンパイルして実行する』
- 16 ページの『Eclipse から、すべての MQTT クライアント・サンプル Java アプリケーションをコンパイルして実行する』
- 18 ページの『Android 上の Java 用の MQTT クライアントの概要』

以下の MQTT クライアント・サンプル Java アプリケーションが SDK に含まれています。

### MQTTV3Sample

サンプルには、IBM WebSphere MQ、および `com.ibm.micro.client.mqttv3.jar` パッケージへのリンクも含まれています。

### Sample

Sample は Paho パッケージ内にあり、`org.eclipse.paho.client.mqttv3` パッケージへのリンクです。これは MQTTV3Sample と似ており、各 MQTT アクションが完了するまで待機します。

### SampleAsyncWait

SampleAsyncWait は、`org.eclipse.paho.client.mqttv3` パッケージ内にあります。これは非同期 MQTT API を使用し、アクションが完了するまで異なるスレッドで待機します。メインスレッドは、MQTT アクションの完了を待機しているスレッドと同期するまでの間、他の作業を行うことができます。

## SampleAsyncCallback

SampleAsyncCallback は、org.eclipse.paho.client.mqttv3 パッケージ内にあります。これは非同期 MQTT API を呼び出します。非同期 API は、MQTT が呼び出しの処理を完了するまで待機しないで、アプリケーションに戻ります。アプリケーションは他のタスクを実行してから、処理の必要な次のイベントの到着を待機します。MQTT は、処理を完了したとき、イベント通知をアプリケーションに返送します。イベント・ドリブン MQTT インターフェースは、Android および他のイベント・ドリブン・オペレーティング・システムのサービスとアクティビティのプログラミング・モデルに適しています。

例として、mqttExerciser サンプルが、サービスとアクティビティのプログラミング・モデルを使用して MQTT を Android に組み込む方法に注目してください。

## mqttExerciser

mqttExerciser は、Android のサンプル・プログラムです。これは別個にビルドされて実行されるので、別の場所で説明されています。[18 ページの『Android 上の Java 用の MQTT クライアントの概要』](#)を参照してください。

## タスクの結果

IBM WebSphere MQ または IBM MessageSight に MQTT サーバーとして接続された MQTT Java サンプル・アプリケーションを、コンパイルして実行しました。

## 次のタスク

Javadoc の参照情報を確認します。[16 ページの『Eclipse から、すべての MQTT クライアント・サンプル Java アプリケーションをコンパイルして実行する』](#)のステップ [17 ページの『3』](#)を参照してください。あるいは、Mobile Messaging and M2M クライアント・パックの SDK\clients\java\doc\javadoc ディレクトリーにある Javadoc html ファイルを開きます。

## コマンド・ラインから、Paho サンプル・プログラムをコンパイルして実行する

コマンド行から Paho サンプル・アプリケーション Sample.java をコンパイルして実行します。サンプルは、MQTT SDK 内にあります。このサンプルは、org.eclipse.paho.client.mqttv3 パッケージ内の MQTT Paho クライアント・ライブラリーを使用して作成されています。これは MQTT のパブリッシャーおよびサブスクライバーを示しています。同じディレクトリー内の他の 2 つの Paho サンプルも、同じ方法でビルドして実行できます。それらは、MQTT ライブラリーを非同期で呼び出す点が異なります。

## 始める前に

メインタスクのステップ [13 ページの『1』](#) から [13 ページの『4』](#) を実行して、MQTT サーバーを構成し、Mobile Messaging and M2M クライアント・パックをダウンロードします。

## このタスクについて

SDK クライアントのサンプル・サブディレクトリー SDK\clients\java\samples から Sample.java をコンパイルして実行します。Java コードは、ディレクトリー SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app にあります。

## 手順

1. 選択したプラットフォームで Sample をコンパイルして実行するためのスクリプトを、クライアントの samples ディレクトリーに作成します。

次のスクリプトは、Windows 上でサンプルをコンパイルして実行します。

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

図 5. *Sample.java* のコンパイルおよび実行

2. スクリプトを実行します。

結果:

```
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
```

図 6. *MQTTV3Sample* サブスクライバー

```
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
```

図 7. *MQTTV3Sample* パブリッシャー

サブスクライバー・アプリケーションを実行したままにしていると、その同じサブスクライバーを再び実行することはできません。新しいサブスクライバーのクライアント ID は、以前のサブスクライバーのものと同じです。同じクライアント ID を持つ 2 つの MQTT クライアントを同時に実行することはできません。-i オプションを設定することにより、クライアント ID を設定して、複数の異なるサブスクライバーを同時に実行可能にすることができます。

同じクライアントを再び実行する場合には、開始するための -c オプションを利用し、cleansession を false に設定してクライアントを開始することができます。そのオプションにより、割り込みの生じたクライアント・セッションの動作を調べることができます。

3. Enter キーを押すか、ウィンドウを閉じることにより、サブスクライバーを終了します。

## 次のタスク

SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app サブディレクトリー内の他のサンプルをコンパイルして実行するためのスクリプトを作成します。15 ページの図 5 のスクリプトをコピーして、Sample を SampleAsyncWait または SampleAsyncCallBack に置き換えます。その他のサンプルは、同期 Sample プログラムの非同期バージョンです。

### SampleAsyncWait

SampleAsyncWait は、org.eclipse.paho.client.mqttv3 パッケージ内にあります。これは非同期 MQTT API を使用し、アクションが完了するまで異なるスレッドで待機します。メインスレッドは、MQTT アクションの完了を待機しているスレッドと同期するまでの間、他の作業を行うことができます。

### SampleAsyncCallBack

SampleAsyncCallBack は、org.eclipse.paho.client.mqttv3 パッケージ内にあります。これは非同期 MQTT API を呼び出します。非同期 API は、MQTT が呼び出しの処理を完了するまで待機しないで、アプリケーションに戻ります。アプリケーションは他のタスクを実行してから、処理の必要な次のイベントの到着を待機します。MQTT は、処理を完了したとき、イベント通知をアプリケーションに返送します。イベント・ドリブン MQTT インターフェースは、Android および他のイベント・ドリ

ブン・オペレーティング・システムのサービスとアクティビティのプログラミング・モデルに適しています。

例として、mqttExerciser サンプルが、サービスとアクティビティのプログラミング・モデルを使用して MQTT を Android に組み込む方法に注目してください。

非同期サンプルは、MQTT クライアントの待機中に、MQTT アプリケーションでブロックが生じる時間を短縮する方法を示します。応答性とモバイル環境でのバッテリー寿命とを向上させるためには、メインスレッドでのブロック呼び出しをなくすることが重要です。

サンプルは、MQTT クライアントで非同期インターフェースを呼び出すための 2 つのパターンを示します。

1. `SampleAsyncWait` は、MQTT がネットワーク対話を待機している間はブロックしません。
2. `SampleAsyncCallBack` は、MQTT クライアントによるいずれかのアクションの完了を待機する際にブロックしません。JavaScript ページが MQTT クライアントをブラウザから呼び出すときには、このことが必要になります。JavaScript ページがブロックしないようにする必要があります。アクションに対する応答は、メインのブラウザ・スレッドに送り返す必要があります。そのスレッドは、通知を処理するために記述した MQTT イベント・ハンドラーを呼び出します。

## Eclipse から、すべての MQTT クライアント・サンプル Java アプリケーションをコンパイルして実行する

Mobile Messaging and M2M クライアント・パックにある MQTT クライアント・サンプル Java ・アプリケーションをコンパイルして実行します。それらは MQTT のパブリッシャーおよびサブスクライバーを示しています。

### このタスクについて

Eclipse で MQTT Java サンプル、`Sample`、および `MQTTV3Sample` をコンパイルして実行します。`Sample` は `sdkroot\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` SDK クライアントのサブディレクトリにあり、`MQTTV3Sample.java` は `sdkroot\SDK\clients\java\samples` にあります。

### 手順

1. [Eclipse IDE for Java Developers](#) をダウンロードします。
2. MQTT Samples という名前の Java プロジェクトを Eclipse に作成します。
  - a) 「ファイル」 > **新規** > **Java プロジェクト**。MQTT Samples と入力します。「次へ」をクリックします。

JRE が正しいバージョンまたはそれ以降のバージョンであることを確認してください。JSE は、バージョン 1.5 以降でなければなりません。
  - b) 「**Java 設定**」ウィンドウで、「**追加のソース・フォルダーをリンクする**」をクリックします。
  - c) MQTT Java SDK フォルダーをインストールしたディレクトリーを参照します。`sdkroot\SDK\clients\java\samples` フォルダーを選択し、**OK** > 「次へ」 > 「終了」をクリックします。
  - d) 「**Java 設定**」ウィンドウで、「**ライブラリー**」 > 「**外部 JAR の追加**」をクリックします。
  - e) MQTT Java SDK フォルダーをインストールしたディレクトリーを参照します。`sdkroot\SDK\clients\java` フォルダーを見つけて、`org.eclipse.paho.client.mqttv3.jar` ファイルと `com.ibm.micro.client.mqttv3.jar` ファイルを選択し、「開く」 > 「終了」をクリックします。

`MQTTV3Sample.java` サンプルは `com.ibm.micro.client.mqttv3.jar` にリンクし、`paho` ディレクトリー・ツリー内のサンプルは `org.eclipse.paho.client.mqttv3.jar` にリンクします。`com.ibm.micro.client.mqttv3.jar` は保持されるため、既存の MQTT アプリケーションは変更せずにビルドと実行を続行できます。



MQTT Samples プロジェクトをビルドする際にいくつかの警告が発生しますが、エラーは発生しません。

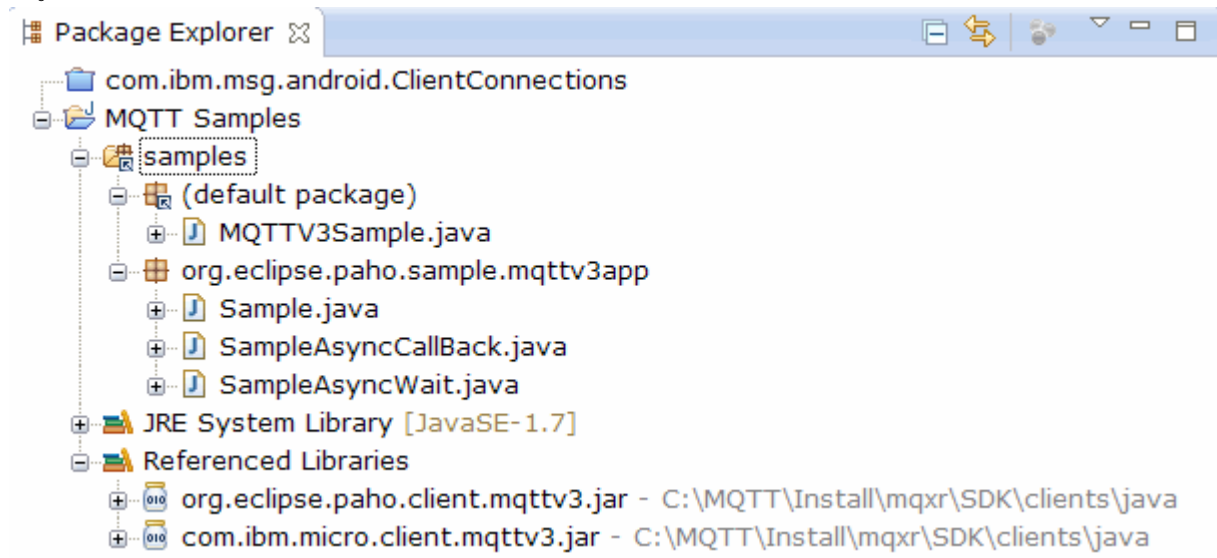


図 8. Java プロジェクト用の MQTT クライアント

3. オプション: MQTT クライアント Javadoc をインストールします。

MQTT クライアント Javadoc がインストールされていると、Java エディターの吹き出しヘルプに MQTT クラスの説明が表示されます。

- a) Java プロジェクトで「パッケージ・エクスプローラー」 > 「参照されるライブラリー」を開きます。 org.eclipse.paho.client.mqttv3.jar > 「プロパティ」を右クリックします。
- b) プロパティ・ナビゲーターで、「**Javadoc ロケーション**」をクリックします。
- c) 「**Javadoc ロケーション**」 ページで **Javadoc URL** > 「参照」をクリックし、 SDK\clients\java\doc\javadoc フォルダ > 「**OK**」を見つけます。
- d) 「**検証**」 > 「**OK**」をクリックします。

ドキュメンテーションを表示するためにブラウザを開くよう促すプロンプトが出されます。

- e) com.ibm.micro.client.mqttv3.jar ファイルについて、この手順を繰り返します。

4. mqttv3app.Sample アプリケーションを実行するための、パブリッシャーおよびサブスクリバのランタイム構成を作成します。

- a) **Sample** クラスを右クリックし、**Run as > Run configurations** をクリックします。
- b) 「**Java アプリケーション**」 > 「**新規**」を右クリックし、名前 **SampleSubscriber** を入力します。
- c) 引数タブをクリックして、プログラム実引数を入力してから、「**適用**」をクリックします。

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

- d) 最後のステップを -a subscribe パラメーターを除去して繰り返し、**SamplePublisher** 構成を作成します。

5. mqttv3app.Sample のサブスクリバ、そして次にパブリッシャーを実行します。

- a) 「**実行**」 > 「**構成の実行**」をクリックします。
- b) **SampleSubscriber** > 「**実行**」をクリックします。

「**コンソール**」ビューを開きます。サブスクリバはパブリケーションを待機中です。


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

- c) **SamplePublisher** > 「**実行**」をクリックします。

「**コンソール**」ビューを開きます。パブリッシャーによって作成されたパブリケーションが表示されます。

```
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) **コンソール**・ビューをサブスクライバー・**コンソール**に切り替えます。

**コンソール**切り替えのアイコンは、です。

サブスクライバーはパブリケーションを受け取りました。

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

6. オプション: MQTTV3Sample のための、パブリッシャーおよびサブスクライバーのランタイム構成を作成します。

- a) **MQTTV3Sample** クラスを右クリックし、「**実行**」>**実行構成**をクリックします。  
b) 「**Java アプリケーション**」>「**新規**」を右クリックし、名前 MQTTV3SampleSubscriber を入力します。  
c) 引数タブをクリックして、プログラム実引数を入力してから、「**適用**」をクリックします。

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

- d) 最後のステップを -a subscribe パラメーターを除去して繰り返し、MQTTV3SamplePublisher 構成を作成します。

7. オプション: MQTTV3Sample のサブスクライバー、そして次にパブリッシャーを実行します。

- a) 「**実行**」>「**構成の実行**」をクリックします。  
b) **MQTTV3SampleSubscriber** > **実行**をクリックします。

「**コンソール**」ビューを開きます。サブスクライバーはパブリケーションを待機中です。


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

- c) **MQTTV3SamplePublisher** > 「**実行**」をクリックします。

「**コンソール**」ビューを開きます。パブリッシャーによって作成されたパブリケーションが表示されます。

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

- d) **コンソール**・ビューをサブスクライバー・**コンソール**に切り替えます。

**コンソール**切り替えのアイコンは、です。

サブスクライバーはパブリケーションを受け取りました。

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

## Android 上の Java 用の MQTT クライアントの概要

MQTT サーバーとメッセージを交換する Android 用の MQTT クライアント・サンプル Java アプリケーションをインストールできます。アプリケーションは、IBM の MQTT SDK のクライアント・ライブラリーを

使用します。アプリケーションを自分でビルドすることも、事前にビルドされたサンプル・アプリケーションをダウンロードすることもできます。

## 始める前に

- MQTT クライアントのサポートされるプラットフォームと参照プラットフォームについて詳しくは、[IBM Mobile Messaging and M2M クライアント・パックのシステム要件](#)を参照してください。
- クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。
- MQTT クライアント・サンプル・アプリケーションは Ice Cream Sandwich (Android 4.0) 以上で稼働します。このバージョンの Android では、タブレットのディスプレイ解像度も向上します。

## このタスクについて

Android 用の MQTT クライアント・サンプル Java アプリケーションは、"mqttExerciser"と呼ばれます。このアプリケーションでは、MQTT SDK のクライアント・ライブラリーを使用して、MQTT サーバーとメッセージを交換します。

サンプル・アプリは、自分で作成して Eclipse から mqttExerciser.apk としてエクスポートすることも、Mobile Messaging and M2M クライアント・パックの `sdkroot\SDK\clients\android\samples\apks` フォルダー内のファイル mqttExerciser.apk として提供されている事前作成されたサンプル・アプリを使用することもできます。自分でアプリケーションをビルドすることを選択した場合は、構築する開発環境は、モバイル・メッセージングを for Android のアプリケーションに組み込むように調整されます。この調整は、独自のアプリケーションへのモバイル・メッセージングの組み込みを始める際に役立ちます。

## 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。

サーバーは MQTT version 3.1 プロトコルをサポートする必要があります。IBM のすべての MQTT サーバー (IBM WebSphere MQ および IBM MessageSight を含む) がこれを行います。[139 ページの『MQTT サーバーの概要』](#)を参照してください。

2. 適切なツールを取得します。

Java Development Kit (JDK) バージョン 6 以降をインストールします。Java アプリ for Android を開発しているため、JDK は Oracle からのものでなければなりません。JDK は [Java SE ダウンロード](#) から入手できます。

Eclipse 開発環境も必要です。Eclipse 3.6.2 (Helios) 以上でなければなりません。ご使用の JDK に対応させるには、Eclipse にある Java コンパイラー・レベルが 6 以上でなければなりません。これらはすべて、[Eclipse Foundation](#) から入手できます。

最後に、Android SDK が必要です。これは、[Get the Android SDK](#) から取得できます。

3. Mobile Messaging and M2M クライアント・パック をダウンロードし、MQTT SDK をインストールします。

インストール・プログラムは用意されておらず、ダウンロードしたファイルを展開するだけです。

- a. [Mobile Messaging and M2M クライアント・パック](#) をダウンロードします。

- b. SDK をインストールするフォルダーを作成します。

フォルダーの名前を MQTT にすることもできます。ここでは、このフォルダーへのパスを `sdkroot` として示します。

- c. 圧縮された Mobile Messaging and M2M クライアント・パック ファイルの内容を `sdkroot` に展開します。展開すると、`sdkroot\SDK` から始まるディレクトリー・ツリーが作成されます。

4. オプション: for Android サンプル・アプリケーション mqttExerciser をビルドします。

Eclipse ツールと Android ツールを構成し、MQTT SDK から mqttExerciser プロジェクトをインポートしてビルドします。

注: これをすぐに実行しない場合は、MQTT SDK の `sdkroot\SDK\clients\android\samples\apks` フォルダ内のファイル `mqttExerciser.apk` として提供されている、事前作成されたサンプル・アプリを使用できます。

a) JDK の JRE によって、Eclipse 開発環境を開始します。

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

b) Android SDK から、パッケージとプラットフォームのセットを選択してインストールします。

Google が推奨するプラットフォームとパッケージのリストについては、「[プラットフォームとパッケージの追加](#)」を参照してください。


注: SDK プラットフォームは Android API レベル 16 以上でなければなりません。これより前の API レベルでは、プロジェクトを正常にコンパイルできません。

c) [Android Development Tools \(ADT\) plug-in](#) を Eclipse に追加します。

d) サンプル `mqttExerciser` アプリケーション・プロジェクトを Eclipse にインポートして、エラーを修正します。

i) MQTT SDK から、パス `sdkroot\SDK\clients\android\samples\mqttExerciser` にサンプル・アプリケーション・プロジェクトをインポートします。

「問題」ビューに、多数のビルド・エラーがリストされます。次のいくつかのステップで、ビルド・エラーを解決します。

ii) `org.eclipse.paho.client.mqttv3.jar` ライブラリーを Android プロジェクトの `libs` フォルダにコピーします。Windows 例:  例: Windows の場合、これは `sdkroot\SDK\clients\java` フォルダの下にあります。「ファイル操作」ウィンドウが表示されます。「ファイルのコピー」選択を受け入れてから、「OK」をクリックします。

iii) プロジェクト・フォルダ `com.ibm.msg.android` を右クリックし、**Android ツール ... > サポート・ライブラリーの追加 ...**。ライセンス条項を読み、受諾して、「インストール」をクリックします。

iv) プロジェクト・フォルダ `com.ibm.msg.android` を右クリックし、**Android ツール ... > 「プロジェクト・プロパティーの修正」**。

v) ワークスペースに、スーパークラス・メソッドのオーバーライドに言及する約 84 件のエラーが依然としてある場合、コンパイラー・コンプライアンス・レベルが 1.5 以下に設定されている可能性があります。Android SDK バージョン 16 は、コンパイラー・コンプライアンス・レベルが 1.5 以下であることを想定しています。残りのエラーを修正するには、以下のステップを実行します。

a) Android SDK および対応する Eclipse プラグインが Android SDK バージョン 17 であるか確認し、(必要に応じて) 更新します。

b) `com.ibm.msg.android` プロジェクト・フォルダを右クリックして、「プロパティー」 > 「Java コンパイラー」を選択します。コンパイラー・コンプライアンス・レベルを確認し、1.6 以上に設定してから、ワークスペースを再ビルドします。

プロジェクトをビルドする際にいくつかの警告が発生しますが、エラーは発生しません。

5. MQTT クライアント・サンプル Java アプリケーションを Android デバイスにインストールして開始します。

[developer.android.com](#) ページの、[Running your app](#) を参照してください。

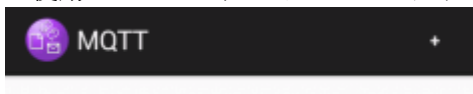
アプリケーションを Eclipse プロジェクトとして自分でビルドする場合は、Eclipse からアプリケーションを開始できます。

アプリケーション・パッケージ (APK) ファイル `mqttExerciser.apk` がある場合は、[Android Debug Bridge \(ADB\)](#) インストール・コマンドを使用して、Eclipse の外部にインストールできます。このコマンドでは APK ファイルの場所を引数として使用します。事前作成されたサンプル・アプリケーションを使用している場合、場所は `sdkroot\SDK\clients\android\samples\apks\mqttExerciser.apk` です。

6. for Android サンプル・アプリケーション mqttExerciser を使用して、トピックに接続し、サブスクライブし、パブリッシュします。

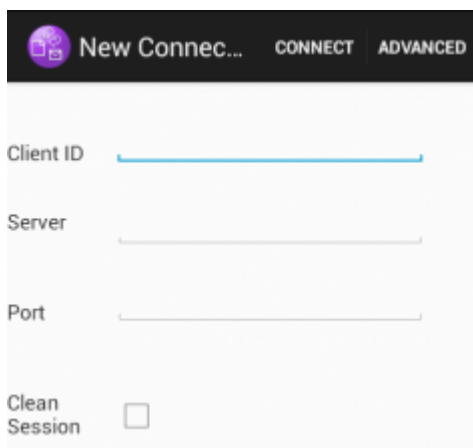
a) Android 用の MQTT クライアント・サンプル Java アプリケーションを開きます。

ご使用の Android デバイスでこのウィンドウが開きます。



b) MQTT サーバーに接続します。

i) + 記号をクリックして、新しい MQTT 接続を開きます。



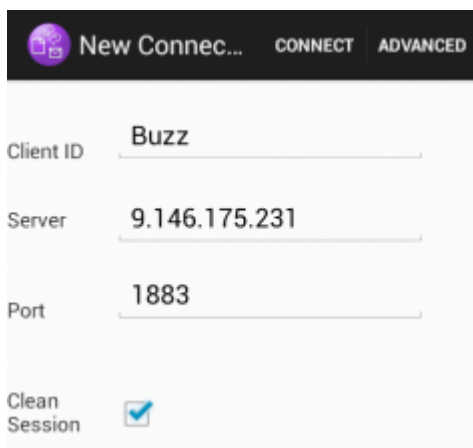
ii) 「クライアント ID」フィールドに、任意の固有 ID を入力します。キー・ストロークは遅いことがあるので、急がないでください。

iii) 「サーバー」フィールドに、ご使用の MQTT サーバーの IP アドレスを入力します。

これは、最初のメインステップで選択したサーバーです。IP アドレスを 127.0.0.1 にすることはできません。

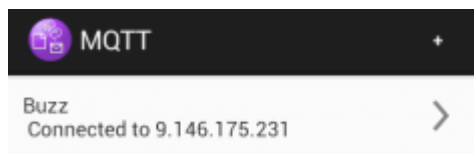
iv) MQTT 接続のポート番号を入力します。

通常の MQTT 接続のデフォルト・ポート番号は 1883 です。



v) 「接続」をクリックします。

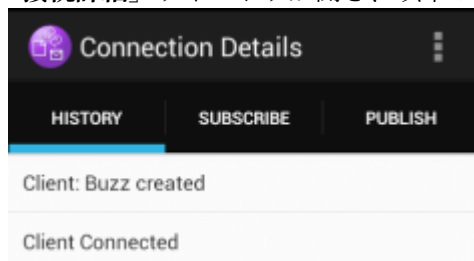
接続が成功すると、「接続中 (Connecting)」メッセージが表示され、その後に次のウィンドウが表示されます。



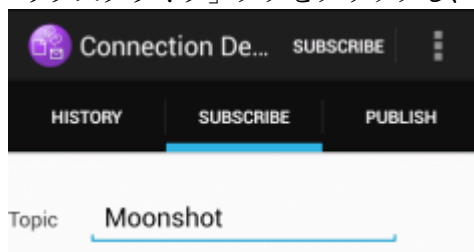
c) トピックにサブスクライブします。

i) 「接続」メッセージをクリックします。

「接続詳細」ウィンドウが開き、以下のように履歴がリストされます。



ii) 「サブスクライブ」タブをクリックし、トピック・ストリングを入力します。

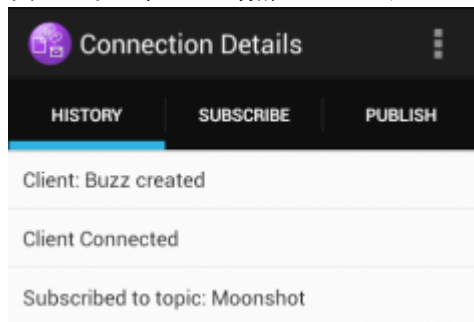


iii) 「サブスクライブ」アクションをクリックします。

"サブスクライブ済み" というメッセージが短時間表示されます。

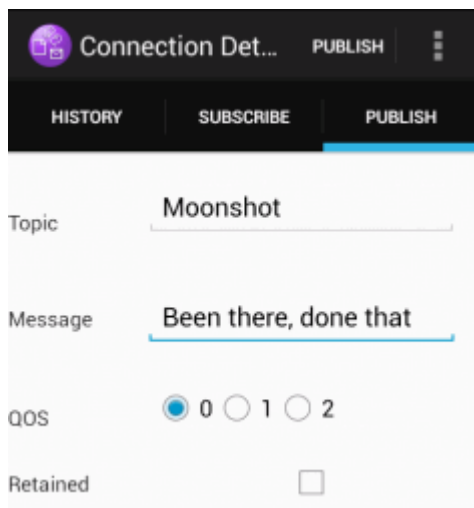
iv) 「履歴」タブをクリックします。

次のように、この時点で履歴にはサブスクリプションが含まれます。



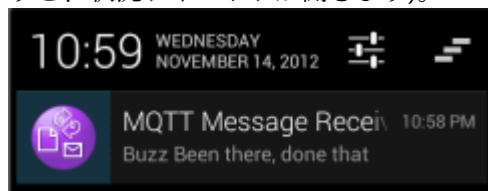
d) 次に、同じトピックに対してパブリッシュします。

i) 「パブリッシュ」タブをクリックして、サブスクライブの場合と同じトピック・ストリングを入力します。メッセージを入力します。

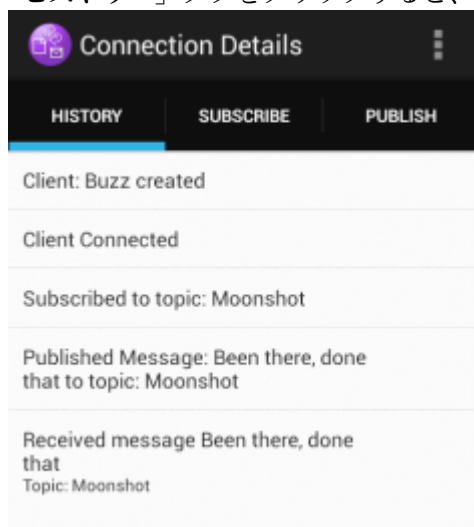


ii) 「パブリッシュ」アクションをクリックします。

短時間に2つのメッセージが表示されます。「パブリッシュ済み」の後に「サブスクライブ済み」が続きます。パブリケーションが状況域に表示されます(セパレーター・バーを引き下ろすと、状況ウィンドウが開きます)。



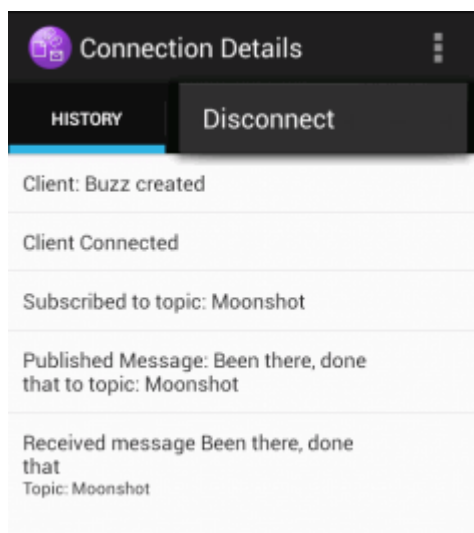
iii) 「ヒストリー」タブをクリックすると、詳細なヒストリーが表示されます。



e) クライアント・インスタンスを切断します。

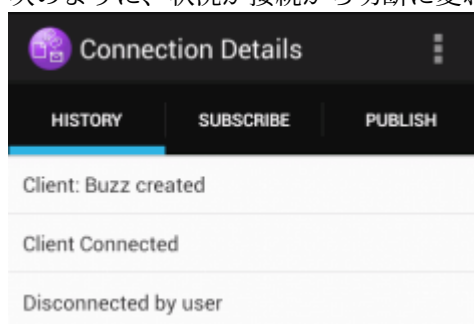
i) アクション・バーのメニュー・アイコンをクリックします。

Android用のMQTTクライアント・サンプルJavaアプリケーションは、「切断」ボタンをMQTTの「接続の詳細」ウィンドウに追加します。

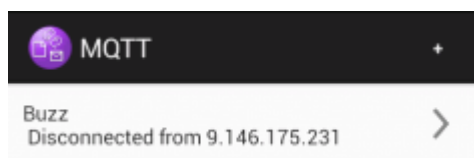


ii) 「切断」をクリックします。

次のように、状況が接続から切断に変わります。



f) 「戻る」をクリックして、MQTT クライアント・サンプル Java アプリケーション・セッションのリストに戻ります。



- 正符号をクリックして、新しい MQTT クライアント・サンプル Java アプリケーション・セッションを開始します。
  - 切断されたクライアントをクリックして、再接続します。
  - 「戻る」をクリックして、ランチパッドに戻ります。
- g) タスク・ボタンをクリックして、実行中のアプリをリストします。MQTT クライアント・サンプル Java アプリケーションを見つけます。アイコンを画面の外に移動して、その画面を閉じます。

## 次のタスク

自分でサンプル・アプリをビルドした場合、MQTT ライブラリーを呼び出してメッセージを交換する独自の Android アプリを開発する準備ができています。mqttExerciser のクラスで Android アプリケーションをモデル化できます。サンプルを学習するには、mqttExerciser プロジェクトの com.ibm.msg.android および com.ibm.msg.android.service でクラスの Javadoc を生成します。

## 関連情報

[ADT を使用する Eclipse からのプロジェクトの管理 \(Managing Projects from Eclipse with ADT\)](#)



## JavaScript 用の MQTT メッセージング・クライアントの概要

JavaScript 用の MQTT メッセージング・クライアントは、メッセージング・クライアント・サンプル・ホーム・ページを表示し、リンク先のリソースを参照することで開始できます。このホーム・ページを表示するには、MQTT メッセージング・クライアント・サンプル JavaScript ページからの接続を受け入れるように MQTT サーバーを構成してから、サーバーで構成した URL を Web ブラウザーに入力します。デバイス上の JavaScript 用の MQTT メッセージング・クライアントは自動的に始動し、メッセージング・クライアント・サンプル・ホーム・ページが表示されます。このページには、ユーティリティー、プログラミング・インターフェース文書、チュートリアル、およびその他の役立つ情報へのリンクが含まれています。

### 始める前に

高度に使用する場合や、実動で使用する場合は、メッセージング・クライアント・サンプル・ホーム・ページの形状変更または削除が必要になります。サンプル・コードを使用して生成されるユーザー・インターフェースは、アクセシビリティ標準やアクセシビリティ要件に対する準拠が保証されないことに注意してください。

JavaScript 用の MQTT メッセージング・クライアントをサポートするには、MQTT サーバーが必要です。このサーバーは、WebSockets を介した MQTT V3.1 プロトコルをサポートする必要があります。IBM MessageSight および IBM WebSphere MQ Version 7.5.0, Fix Pack 1 以降のバージョンでは、MQTT protocol over WebSockets がサポートされます。139 ページの『MQTT サーバーの概要』を参照してください。無料で 90 日間評価できる IBM WebSphere MQ をインストールする場合は、141 ページの『のインストール IBM WebSphere MQ』を参照してください。

WebSocket protocol は最近確立されたものです。クライアントとサーバーの間にファイアウォールがある場合、それが WebSockets のトラフィックをブロックしないことを確認してください。同様に、ご使用のブラウザーが WebSocket protocol をまだサポートしていない場合も、<sup>1</sup>メッセージング・クライアントのサンプル・ホーム・ページから入手できるクライアント・ユーティリティーやチュートリアルを使用することはできません。メッセージング・クライアントを操作するための最新バージョンがテスト済みのブラウザーは、以下の 25 ページの表 1 にリストされています。

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

### このタスクについて

このタスクでのステップの大半は、MQTT サーバーを構成するためのものです。JavaScript 用のメッセージング・クライアントにアクセスするために必要なのは、WebSocket protocol をサポートするブラウザーを実行することです。

IBM WebSphere MQ で、ステップに従ってサンプル・チャンネルを作成し、IBM WebSphere MQ Telemetry を有効にします。ポート 1883 で、サンプルのデフォルト MQTT WebSockets チャンネルに接続します。メッセージング・クライアントのサンプル・ホーム・ページ URL は、IBM WebSphere MQ 上の `http://hostname:1883` です。

IBM MessageSight で、アプライアンスをインストールしてセットアップし、接続を受け入れるようにメッセージング・ハブを構成し、MQTT WebSockets エンドポイントを作成します。

<sup>1</sup> 具体的には、RFC 6455 (WebSocket) 標準をサポートしていない場合です。

## 手順

1. [Mobile Messaging and M2M クライアント・パック](#)をダウンロードし、クライアント・アプリケーションを接続できる MQTT サーバーを選択します。

11 ページの『[MQTT クライアントの概要](#)』を参照してください。

2. JavaScript 用の MQTT メッセージング・クライアント サンプル HTML ページからの接続を受け入れるように MQTT サーバーを構成します。

• IBM WebSphere MQ の場合:

– MQTT 用に構成された IBM WebSphere MQ キュー・マネージャーが既にある場合は、MQTT と HTTP の両方をサポートするようにチャンネル定義のプロトコルを変更します。[ALTER CHANNEL](#) を参照してください。

– IBM WebSphere MQ キュー・マネージャーを作成して、サンプル MQTT WebSockets エンドポイントを構成するには、以下のタスクのいずれかを実行します。

– [143 ページの『コマンド・ラインからの MQTT サービスの構成』](#)

– [146 ページの『IBM WebSphere MQ Explorer を使用した MQTT サービスの構成』](#)

3. デバイス上で Web ブラウザーを開きます。

4. メッセージング・クライアント・サンプル・ホーム・ページの URL を入力します。

• IBM WebSphere MQ の場合、これは `http://hostname:1883` です。

• IBM MessageSight では、これは `http://hostname:port` となります。

ここで、*hostname* はクライアントの接続先エンドポイントとして IBM MessageSight アプライアンスに構成したイーサネット・ソケットの DNS 名または IP アドレスで、*port* はクライアントのためにエンドポイントに割り当てた TCP/IP ポート番号です。

メッセージング・クライアント・サンプル・ホーム・ページが表示されます。

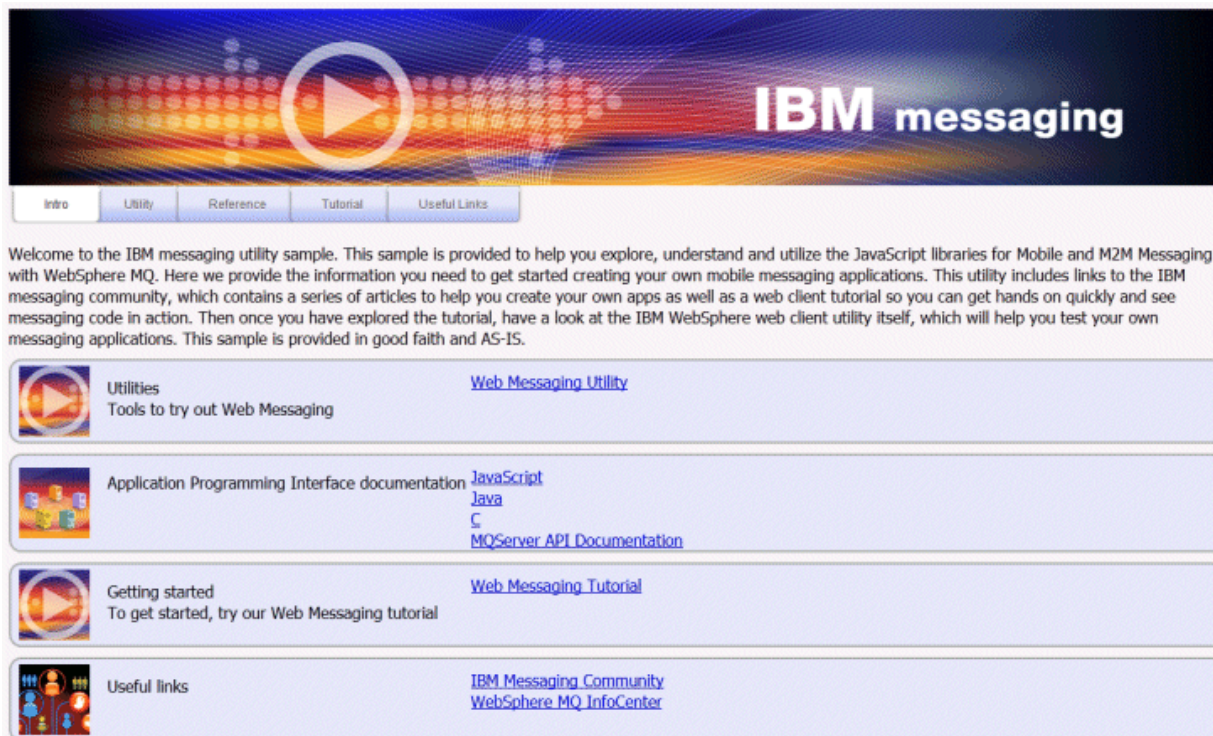


図 9. JavaScript 用の MQTT メッセージング・クライアントのサンプル・ホーム・ページ

## タスクの結果

WebSockets 用の MQTT チャネルが構成されました。

JavaScript 用の MQTT メッセージング・クライアントのサンプル・ホーム・ページで、「**Web メッセージング・ユーティリティー**」をクリックして、メッセージング・クライアント API のさまざまな機能を試行します。例えば、キュー・マネージャーに接続し、メッセージをサブスクライブしてからいくつかのメッセージをパブリッシュすることができます。また、「**Web メッセージング・チュートリアル**」をクリックして、JavaScript 用の MQTT メッセージング・クライアント API を呼び出す Web ページの作成方法を学習することもできます。

### 関連概念

[119 ページの『JavaScript 用の MQTT メッセージング・クライアントと Web アプリケーション』](#)

[122 ページの『JavaScript でのメッセージング・アプリケーションのプログラミング方法』](#)

### 関連タスク

[79 ページの『SSL を介した JavaScript 用の MQTT メッセージング・クライアントと WebSockets の接続』](#)

SSL を使用する JavaScript 用の MQTT メッセージング・クライアント サンプル HTML ページと WebSocket protocol を使用して、Web アプリを IBM WebSphere MQ に安全に接続します。

## C 用の MQTT クライアントの概要

C ソースをコンパイルできる任意のプラットフォームで C 用のサンプル MQTT クライアントを起動して稼働状態にします。C 用のサンプル MQTT クライアントを IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとして実行できることを確認します。

### 始める前に

- クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。
- C 用の MQTT クライアントのサポートされるプラットフォームと参照プラットフォームについては、[IBM Mobile Messaging and M2M クライアント・パックのシステム要件](#)を参照してください。

### このタスクについて

このタスクに従って、Windows 上でコマンド行または Microsoft Visual Studio 2010 から、C 用のサンプル MQTT クライアントをコンパイルして実行します。Microsoft Visual Studio 2010 は、コマンド行の例でクライアントをコンパイルするためにも使用されます。コマンド行スクリプトを変更して、サンプルをコンパイルし、他のプラットフォームで実行します。

### 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。

サーバーは MQTT version 3.1 プロトコルをサポートしている必要があります。IBM のすべての MQTT サーバー (IBM WebSphere MQ および IBM MessageSight を含む) がこれを行います。[139 ページの『MQTT サーバーの概要』](#)を参照してください。

2. ビルド場所のプラットフォームに C 開発環境をインストールします。

このトピックの例にある Make ファイルは、以下のツールをターゲットとしています。

- **iOS** iOS の場合、OS X 10.8.2 の Apple Mac で、[Xcode](#) の iOS 開発ツールを使用。
- **Linux** Linux の場合、Red Hat® Enterprise Linux バージョン 6.2 の gcc バージョン 4.4.6。サポートされる最小レベルは、glibc C ライブラリーが 2.12、Linux カーネルが 2.6.32 です。
- **Windows** Microsoft Windows の場合、Visual Studio バージョン 10.0。

3. Mobile Messaging and M2M クライアント・パック をダウンロードし、MQTT SDK をインストールします。

インストール・プログラムは用意されておらず、ダウンロードしたファイルを展開するだけです。

- a. [Mobile Messaging and M2M クライアント・パック](#)をダウンロードします。
  - b. SDK をインストールするフォルダーを作成します。  
フォルダーの名前を MQTT にすることもできます。ここでは、このフォルダーへのパスを *sdkroot* として示します。
  - c. 圧縮された Mobile Messaging and M2M クライアント・パック ファイルの内容を *sdkroot* に展開します。展開すると、*sdkroot\SDK* から始まるディレクトリー・ツリーが作成されます。
4. オプション: [31 ページの『C 用の MQTT クライアント・ライブラリーのビルド』](#)のステップに従います。
- このステップは、Mobile Messaging and M2M クライアント・パックに、ターゲット・プラットフォーム用の C クライアント・ライブラリーが含まれていない場合にのみ実行してください。
5. MQTT クライアント・サンプル C アプリケーション *MQTTV3Sample.c* をコンパイルして実行します。
- コマンド・ラインから、[28 ページの『コマンド行からの MQTT クライアント・サンプル C アプリケーションのコンパイルおよび実行』](#)の手順に従います。
  - IDE から、[29 ページの『Microsoft Visual Studio からの MQTT クライアント・サンプル C アプリケーションのコンパイルおよび実行』](#)の手順に従います。

## コマンド行からの MQTT クライアント・サンプル C アプリケーションのコンパイルおよび実行

コマンド行から MQTT クライアント・サンプル C アプリケーションをコンパイルして実行します。サンプルは、MQTT SDK 内にあります。これは MQTT のパブリッシャーおよびサブスクリイバーを示しています。

### 始める前に

C 開発環境 (例で使用されている Microsoft Visual Studio 2010 など) をインストールします。

### このタスクについて

SDK クライアントのサブディレクトリー *sdkroot\SDK\clients\c\samples* にある C サンプル *MQTTV3Sample* をコンパイルして実行します。

### 手順

選択したプラットフォームで *Sample* をコンパイルして実行するためのスクリプトを、クライアントの *samples* ディレクトリーに作成します。

次のスクリプトは、Windows 32 ビット・プラットフォーム上で、Microsoft Visual Studio 2010 によってビルドされたサンプルをコンパイルして実行します。 *sdkroot\SDK\clients\c\samples* サブディレクトリーからスクリプトを実行します。

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

### タスクの結果

パブリッシャーとサブスクリイバーは、出力をそれぞれのコマンド・ウィンドウに書き込みます。

```
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 . . .
```

図 10. パブリッシャーからの出力

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

図 11. サブスクライバーからの出力

## Microsoft Visual Studio からの MQTT クライアント・サンプル C アプリケーションのコンパイルおよび実行

Microsoft Visual Studio から MQTT クライアント・サンプル C アプリケーションをコンパイルして実行します。サンプルは Mobile Messaging and M2M クライアント・パック 内にあります。これは MQTT のパブリッシャーおよびサブスクライバーを示しています。

### 始める前に

この例では、Microsoft Visual Studio 2010 を使用します。Windows や他のプラットフォーム上で、他の C 開発環境 ([Eclipse IDE for C/C++ Developers](#) など) を使用することもできます。

### このタスクについて

Microsoft Visual Studio 2010 で C サンプル MQTTV3Sample をコンパイルして実行します。MQTTV3Sample.c は、SDK クライアントのサブディレクトリー `sdkroot\SDK\clients\c\samples` にあります。

### 手順

1. Microsoft Visual Studio を開始します。
2. 既存のコードから新しいプロジェクトを作成します。
  - a) 「ファイル」 > 「新規」 > 「既存のコードからのプロジェクト」をクリックします。
  - b) 作成するプロジェクトのタイプとして、「**Visual C++**」を選択します。
  - c) 「次へ」をクリックします。
3. 「プロジェクトの場所とソース・ファイル」ウィンドウにパラメーターを指定します。
  - a) 「参照」をクリックして、`sdkroot\SDK\clients\c\samples` ディレクトリーを見つけます。
  - b) プロジェクトの名前を MQTTV3Sample とします。
  - c) 「次へ」をクリックします。
4. 「プロジェクト・タイプ」リストから、「**コンソール・アプリケーション・プロジェクト**」を選択します。「完了」をクリックします。
5. デバッグ構成だけを構成します。

デフォルトで、Microsoft Visual Studio はリリースとデバッグ構成の両方を作成します。チュートリアルでは、デバッグ構成を構成します。ビルド・エラーを抑止するために、リリース構成の「**ビルド**」オプションをクリアします。

- a) 「プロジェクト > MQTTV3Sample プロパティ > 構成マネージャー」をクリックします。「アクティブ・ソリューション構成」に「リリース」を選択して、「ビルド」をクリアします。
- b) アクティブ・ソリューション構成 > 「閉じる」として「デバッグ」を選択します。

以下のすべてのステップで、デバッグ構成を変更していることを確認します。

6. 「MQTTV3Sample プロパティ・ページ」で、「C/C++」設定値を変更します。
  - a) MQTTV3Sample 「プロパティ・ページ」ウィンドウで、構成プロパティ > C/C++ > 一般を開きます。
  - b) 一般プロパティのリストで、「追加のインクルード・ディレクトリ」をクリックし、ディレクトリ・パスを `sdkroot\SDK\clients\c\include` に追加して、「適用」をクリックします。
7. 「リンカー」設定を変更します。
  - a) 構成プロパティ > 「リンカー」 > 「一般」を開きます。
  - b) 一般プロパティのリストで、「追加ライブラリ・ディレクトリ」をクリックし、ディレクトリ・パスを `sdkroot\SDK\clients\c\windows_ia32` に追加します。
  - c) リンカー・プロパティのリストで、「コマンド行」をクリックします。「追加オプション」データ入力域に `mqttv3c.lib` と入力して、「適用」をクリックします。
8. MQTTV3SSample.c ソース・ファイルをプロジェクトから除去します。
  - a) 「ソリューションエクスプローラ」ウィンドウで MQTTV3sample > 「ソース・ファイル」フォルダーを開きます。
  - b) MQTTV3SSample.c > 「プロジェクトから除外」を右クリックします。
9. MQTTV3Sample プロジェクトをビルドします。
  - a) Solution Explorer で「MQTTV3sample」プロジェクトを右クリックして、「ビルド」をクリックします。  
ビルドはエラーなしで完了します。
10. 2つの新しいプロジェクトを追加して、MQTTV3Sample をサブスクリバースとパブリッシャーの両方の実行時インスタンスとして実行します。

パブリッシャーおよびサブスクリバースのプロジェクトは、MQTTV3Sample を実行するためのコマンドを入れるためのものです。それらはビルドされず、コードを含んでいません。

- a) 「ソリューションエクスプローラ」で、「ソリューション」`MQTTV3Sample` > 「追加」 > 「新規プロジェクト」を右クリックします。
- b) 「名前」フィールドに Subscriber と入力します。「Win32 コンソール・アプリケーション」は選択されたままにします。「OK」をクリックします。

Win32 Application Wizard が開始します。

- c) Win32 Application Wizard で、「次へ」をクリックします。「空のプロジェクト」 > 「完了」にチェック・マークを付けます。
- d) これらのステップを繰り返して、パブリッシャー・プロジェクトを追加します。
- e) サブスクリバース・プロジェクトを右クリックして、「スタートアップ・プロジェクトとして設定」をクリックします。

「Solution Explorer」ウィンドウが [30 ページの図 12](#) に示されています。

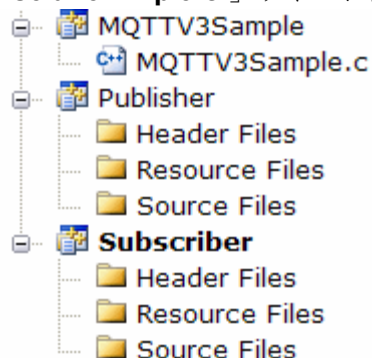


図 12. MQTTV3Sample ソリューション

11. サブスクリバースのプロパティ・ページを構成します。

- a) Solution Explorer 「プロパティ」 > 「構成プロパティ」 > 「プロパティ」 > 「デバッグ」 で「サブスクリイバー」を右クリックします。  
ウィンドウ・タイトルが「サブスクリイバー・プロパティ・ページ」であることを確認します。
  - b) 「環境」をクリックします。 `path=%path%;sdkroot\SDK\clients\c\windows_ia32` と入力し、「適用」をクリックします。  
ご使用の環境に合わせて `sdkroot` を変更します。
  - c) 「コマンド」をクリックして、\$(TargetPath) を MQTTV3Sample モジュールへのパスに置き換えます。  
例: `sdkroot\SDK\clients\c\samples\debug\MQTTV3Sample`  
ご使用の環境に合わせて `sdkroot` を変更します。
  - d) 「コマンド引数」をクリックし、`-a subscribe -b localhost -p 1883` と入力して、「適用」をクリックします。
12. パブリッシャーのプロパティ・ページを構成します。
- ヒント:** 「Solution Explorer」ウィンドウ内のプロジェクトをクリックして、プロパティ・ページ・プロジェクトを切り替えることができます。
- a) サブスクリイバー用のステップを、パブリッシャーについても繰り返します。  
コマンド引数は `-b localhost -p 1883` です。
13. パブリッシャーおよびサブスクリイバーのプロジェクトをビルドするビルド・プロセスを停止します。
- a) いずれかのプロジェクトのプロパティ・ページで、「構成マネージャー」をクリックして、リリース構成とデバッグ構成の両方でパブリッシャーおよびサブスクリイバーの「ビルド」をクリアします。「クローズ」をクリックします。
14. サンプルを実行します。
- a) 「F5」をクリックして、サブスクリイバーを開始します。
  - b) Solution Explorer で「パブリッシャー」を右クリックして、「デバッグ」 > 「新しいインスタンスの開始」を選択します。

## タスクの結果

パブリッシャーおよびサブスクリイバーは、コマンド・ウィンドウに出力します。Visual Studio がパブリッシャー・ウィンドウを閉じます。以下の図に示されているサブスクリイバー・ウィンドウを参照してから、このサブスクリイバー・ウィンドウを閉じます。

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

図 13. サブスクリイバーからの出力

## 次のタスク

非同期のパブリッシャーおよびサブスクリイバーをビルドして実行します。例は、`sdkroot\SDK\clients\c\samples` の `MQTTV3ASample.c` および `MQTTV3ASSample.c` です。

## C 用の MQTT クライアント・ライブラリーのビルド

以下のステップに従って、C 用の MQTT クライアント・ライブラリーをビルドします。このトピックには、複数のプラットフォーム用のコンパイルおよびリンク・スイッチ、および iOS と Windows でライブラリーをビルドする例が含まれています。

## 始める前に

1. C クライアント・ライブラリーは、必要とときにのみビルドしてください。ターゲット・プラットフォームに一致するものがある場合は、`SDK\clients\c` サブディレクトリーにある (Software Development Kit) SDK 内の事前作成されたクライアント・ライブラリーをリンクします。
2. MQTT クライアント・サンプル C アプリケーションを使用してビルドするライブラリーをテストするように MQTT サーバーを構成します。139 ページの『MQTT サーバーの概要』を参照してください。MQTT クライアント・サンプル・アプリケーションのいずれかを実行して、サーバー構成を検証します。
3. SSL (Secure Sockets Layer) をサポートする、C ライブラリーのセキュア・バージョンをビルドする場合は、OpenSSL ライブラリーもビルドする必要があります。46 ページの『OpenSSL パッケージのビルド』を参照してください。

**重要:** OpenSSL パッケージのダウンロードと再配布には、厳密なインポートとエクスポートの規制、およびオープン・ソースのライセンス条件が適用されます。パッケージをダウンロードするかどうかを決定する前に、制限と警告に注意を向けてください。

## このタスクについて

46 ページの『OpenSSL パッケージのビルド』の指示に従って、OpenSSL ライブラリーをダウンロードして作成します。OpenSSL をビルドして、C 用の MQTT クライアント・ライブラリーのセキュア・バージョンを構築する必要があります。MQTT ライブラリーの非セキュアなバージョンを構築するためには、OpenSSL は必要ありません。この手順には、iOS および Windows 用のライブラリーを構築するための例が含まれています。

C 開発ライブラリー・ツールおよび MQTT ソフトウェア開発ツールキット (SDK) をビルド・プラットフォームにダウンロードして、C 用の MQTT クライアント・ライブラリーをビルドします。ターゲット・プラットフォーム用のライブラリーを構築するために、33 ページの『さまざまなプラットフォーム用の MQTT ビルド・オプション』に記載されているオプションを取り入れた Make ファイルを記述します。Make ファイルをビルドして実行するプラットフォーム固有のステップは以下のとおりです。

- **iOS** 35 ページの『iOS デバイスで使用する C 用の MQTT クライアント・ライブラリーを Apple Mac でビルドする』
- **Windows** 40 ページの『Windows での MQTT ライブラリーのビルド』

## 手順

1. ビルド場所のプラットフォームに C 開発環境をインストールします。  
このトピックの例にある Make ファイルは、以下のツールをターゲットとしています。
  - **iOS** iOS の場合、OS X 10.8.2 の Apple Mac で、`Xcode` の iOS 開発ツールを使用。
  - **Linux** Linux の場合、Red Hat Enterprise Linux バージョン 6.2 の `gcc` バージョン 4.4.6。サポートされる最小レベルは、`glibc` C ライブラリーが 2.12、Linux カーネルが 2.6.32 です。
  - **Windows** Microsoft Windows の場合、Visual Studio バージョン 10.0。
2. Mobile Messaging and M2M クライアント・パックをダウンロードし、MQTT SDK をインストールします。  
インストール・プログラムは用意されておらず、ダウンロードしたファイルを展開するだけです。
  - a. Mobile Messaging and M2M クライアント・パックをダウンロードします。
  - b. SDK をインストールするフォルダーを作成します。  
フォルダーの名前を MQTT にすることもできます。ここでは、このフォルダーへのパスを `sdkroot` として示します。
  - c. 圧縮された Mobile Messaging and M2M クライアント・パック ファイルの内容を `sdkroot` に展開します。展開すると、`sdkroot\SDK` から始まるディレクトリー・ツリーが作成されます。



3. C用のMQTTクライアント・ライブラリーのソース・コードを展開します。

ソース・コード圧縮ファイルは `sdkroot\SDK\clients\c\source.zip` です。

4. オプション: OpenSSL をビルドします。

46 ページの『OpenSSL パッケージのビルド』を参照してください。

5. C用のMQTTクライアント・ライブラリーをビルドします。

ライブラリーをビルドするためのコマンドとオプションは、33 ページの『さまざまなプラットフォーム用のMQTTビルド・オプション』にリストされています。

次の例に示されているステップに従って、ターゲット・プラットフォームで使用するC用のMQTTクライアント・ライブラリーをビルドするためのMakeファイルを記述します。

- 35 ページの『iOS デバイスで使用するC用のMQTTクライアント・ライブラリーをApple Macでビルドする』
- 40 ページの『Windows でのMQTTライブラリーのビルド』

### さまざまなプラットフォーム用のMQTTビルド・オプション

以下の表には、さまざまなプラットフォームでC用のMQTTクライアント・ライブラリーをビルドするためのコンパイラおよびビルド・オプションがリストされています。

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

表 2. さまざまなプラットフォーム用の MQTT ビルド・オプション (続き)

プラットフォーム	Compiler	Compiler Options	Linker Options	Extra Options
Windows 32ビット	cl	<pre> /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS" / nologo /c /O2 /W3 / Fd /MD /TC                     </pre>	<pre> /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)                     </pre>	
iOS ARMv7	gcc -arch armv7	<pre> -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isyroot / Applications/ Xcode.app/\ Contents/Developer/ Platforms/\ iPhoneOS.platform/ Developer/SDKs/\ iPhoneOS6.0.sdk                     </pre>	<pre> -L/Applications/Xcode.app/ Contents/Developer/ Platforms/ iPhoneOS.platform/ Developer/SDKs/ iPhoneOS6.0.sdk/usr/lib/ system                     </pre>	
iOS ARMv7s	gcc -arch armv7s			
iOS ARMi386	gcc -arch i386	<pre> -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isyroot / Applications/ Xcode.app/\ Contents/Developer/ Platforms/\ iPhoneSimulator.platform/ Developer/SDKs/ iPhoneSimulator6.0.sdk                     </pre>	<pre> -L/Applications/Xcode.app/ Contents/Developer/ Platforms/ iPhoneSimulator.platform/ Developer/SDKs/ iPhoneSimulator6.0.sdk/usr /lib/system                     </pre>	

## iOS デバイスで使用する C 用の MQTT クライアント・ライブラリーを Apple Mac でビルドする

以下のステップに従って、後で iOS デバイスで使用する C 用の MQTT クライアント・ライブラリーを Apple Mac でビルドするための Make ファイルを記述します。

### 始める前に

1. Apple Mac (OS X 10.8.2 以降) で、ビルド・ツールをインストールし、Make ファイルを開発および実行します。
2. **make** プログラムを含む Xcode 用のコマンド・ライン・ツールをインストールします。コマンド行ツールを [Xcode](#) からダウンロードします。

### このタスクについて

ARMv7 または ARMv7s プロセッサを実行する iPhone または iPad、および i386-64 ビット・プロセッサで稼働する iPhone シミュレーターのための、C 用の MQTT クライアント・ライブラリーをビルドする Make ファイルを作成します。iOS デバイスのリストを参照してください。

**ヒント:** 39 ページの『[Make ファイル MQTTios.mak のリスト](#)』には Make ファイルの全体がリストされています。

1. リストされている内容をコピーしてファイルに貼り付けます。
2. ターゲットに続く各行の先行文字をタブに変換します。ステップ 37 ページの『[8](#)』を参照してください。
3. それを、手順のステップ 38 ページの『[9](#)』にリストされているコマンドを使用して実行します。

### 手順

1. iOS 開発ツールをダウンロードしてインストールします。
  - a. 管理特権を持つユーザー ID を使ってログオンします。
  - b. Apple Mac がバージョン 10.8.2 以降であることを確認します。
  - c. Web サイト [Xcode](#) に進んで、Xcode を Mac App Store からダウンロードします。
  - d. Xcode、コマンド行環境、およびシミュレーターをインストールします。

Mac App Store が複数のバージョンのシミュレーターを提供する場合、アプリのためにターゲットとしている iOS のレベルと互換性のあるバージョンを選択します。

2. Make ファイル MQTTios.mak を作成します。

次のプロローグを追加します。

```
# ビルド出力は現行ディレクトリーに作成されます。
# MQTTCLIENT_DIR は、MQTT クライアント・ソース・コードを含む基本ディレクトリーを指す必要があります。
# デフォルトの MQTTCLIENT_DIR は現行ディレクトリーです
# デフォルトの TOOL_DIR は /Applications/Xcode.app/Contents/Developer/Platforms です。
# デフォルト OPENSSL_DIR は sdkroot/openssl であり、sdkroot/sdk/clients/c/mqttv3c/src を基準とします。
# OPENSSL_DIR は、OpenSSL ビルドを含むベース・ディレクトリーを指す必要があります。
# 例: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. MQTT ソース・コードの場所を設定します。

MQTT ソース・ファイルと同じディレクトリーで Make ファイルを実行するか、MQTTCLIENT\_DIR コマンド行パラメーターを設定します。

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

以下の行を Make ファイルに追加します。

```
ifndef MQTTCLIENT_DIR (ifndef MQTTCLIENT_DIR)
MQTTCLIENT_DIR = ${CURDIR}
```

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

この例では、VPATH を、**make** が明示的に識別されていないソース・ファイル (例えば、ビルドに必要なすべてのヘッダー・ファイル) を検索するディレクトリーに設定します。

#### 4. オプション: OpenSSL ライブラリーの場所を設定します。

このステップは、C ライブラリー用の MQTT クライアントの SSL バージョンをビルドするために必要です。

OpenSSL ライブラリーへのデフォルト・パスを、MQTT SDK を展開したディレクトリーと同じディレクトリーに設定します。あるいは、OPENSSL\_DIR をコマンド行パラメーターに設定します。

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

**ヒント:** *OpenSSL* は、すべての OpenSSL サブディレクトリーを含む OpenSSL ディレクトリーです。ディレクトリー・ツリーは、不必要な空の親ディレクトリーを含んでいるために、その展開場所から移動しなければならないことがあります。

#### 5. 開発ツール・ディレクトリーを設定します。

Xcode を異なる場所にインストールした場合、コマンド・ラインに TOOL\_DIR を設定してください。

```
ifndef TOOL_DIR (IFDEF 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. 各 MQTT ライブラリーのビルドに必要なすべてのソース・ファイルを選択します。また、ビルドする MQTT ライブラリーの名前と場所を設定します。

以下の行を Make ファイルに追加して、すべての MQTT ソース・ファイルをリストします。

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

ソース・ファイルは、同期または非同期のどちらのライブラリーをビルドするか、およびライブラリーに SSL が含まれるかどうかによって決まります。

ビルドするターゲットに応じて、以下の行の 1 つ以上を追加します。共有ライブラリーが darwin\_x86\_64 ディレクトリーに作成されます。

##### • 同期、非セキュア:

```
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
```

##### • 同期、セキュア:

```
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
```

##### • 非同期、非セキュア:

```
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
```

##### • 非同期、セキュア:

```
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. コンパイラーおよびコンパイラー・オプションを定義します。

各種プラットフォームの MQTT ビルド・オプションに示されている各種プラットフォームのオプションを参照してください。

### a) Gnu プロジェクト C および C++ (**gcc**) をコンパイラーとして設定します。

さまざまなデバイスと iPhone シミュレーター用のライブラリーをビルドするために、3つのクロスコンパイラーを選択します。

```
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) コンパイラー・オプションを追加します。

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

### c) インクルード・パスを追加します。

```
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. ビルド・ターゲットを定義します。

**ヒント**：ターゲットの実装を定義する後続の各行は、タブ文字で開始しなければなりません。

### a) **all** ターゲットを定義します。

"all" ターゲットは、すべてのライブラリーをビルドします。

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

最初にリストすることにより、これがデフォルトのターゲットとなります。

### a) 同期する非セキュアなライブラリー、**libmqttv3c.a** をビルドします。

```
${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 ${@}
```

ステートメント `rm *.o` は、各ライブラリー用に作成されたすべてのオブジェクト・ファイルを削除します。 `lipo` は、3つのライブラリーすべてを1つのファイルに連結します。

### b) 非同期の非セキュアなライブラリー、**libmqttv3a.a** をビルドします。

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

```
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
libtool -static -syslibroot ${SDK_i386} -o ${@.i386 *.o
rm *.o
lipo -create ${@.armv7} ${@.armv7s} ${@.i386} -output ${@
```

ステートメント `rm *.o` は、各ライブラリー用に作成されたすべてのオブジェクト・ファイルを削除します。 `lipo` は、3つのライブラリーすべてを1つのファイルに連結します。

- c) 同期するセキュアなライブラリー、 `libmqttv3cs.a` をビルドします。

```

${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 ${@

```

ステートメント `rm *.o` は、各ライブラリー用に作成されたすべてのオブジェクト・ファイルを削除します。 `lipo` は、3つのライブラリーすべてを1つのファイルに連結します。

- d) 非同期のセキュアなライブラリー、 `libmqttv3as.a` をビルドします。

```

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

```

ステートメント `rm *.o` は、各ライブラリー用に作成されたすべてのオブジェクト・ファイルを削除します。 `lipo` は、3つのライブラリーすべてを1つのファイルに連結します。

- e) `clean` ターゲットを定義します。

"clean" ターゲットは、`makefile` によって生成されたすべてのファイルおよびディレクトリーを除去します。

```
Symphony: クリーン
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

9. Make ファイルを実行します。

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

## タスクの結果

以下のファイルが `sdkroot/sdk/clients/c/mqttv3c/src/darwin_x86_64` ディレクトリーに作成されます。

```
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
```

## Make ファイル MQTTios.mak のリスト

```
# ビルド出力は現行ディレクトリーに作成されます。
# MQTTCLIENT_DIR は、MQTT クライアント・ソース・コードを含む基本ディレクトリーを指す必要があります。
# デフォルトの MQTTCLIENT_DIR は現行ディレクトリーです
# デフォルトの TOOL_DIR は /Applications/Xcode.app/Contents/Developer/Platforms です。
# デフォルト OPENSSL_DIR は sdkroot/openssl であり、 sdkroot/sdk/clients/c/mqttv3c/src を基準としま
す。
# OPENSSL_DIR は、OpenSSL ビルドを含むベース・ディレクトリーを指す必要があります。
# 例: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

ifndef MQTTCLIENT_DIR (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 (IFDEF 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
-DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7 *.o
rm *.o
${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
libtool -static -syslibroot ${SDK_ARM} -o ${@.armv7s *.o
rm *.o
${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
libtool -static -syslibroot ${SDK_i386} -o ${@.i386 *.o
rm *.o
lipo -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
-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 ${@

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

Symphony: クリーン
clean:
-rm -f *.obj
-rm -f -r darwin_x86_64

```

## Windows Windows での MQTT ライブラリーのビルド

以下のステップに従って、Windows 上に C 用の MQTT クライアント・ライブラリーをビルドするための Make ファイルを記述します。

### 始める前に

1. 必要に応じて、Gnu make 用に作成された Make ファイルと互換性のあるバージョンの **Make** をビルド・ワークステーションにインストールします。それ以外の場合は、Gnu Make をダウンロードしてビ



ルドします。 [Gnu Make](#) を参照してください。 Web サイト [Make for Windows](#) には、 **Make for Windows** のインストール可能バージョンが用意されています。

2. Make ファイルの例で clean ターゲットを使用するには、Windows 用の Linux コマンドも必要です。Windows 用の Linux コマンドは、 [Cygwin](#) などの Web サイトから入手できます。

## このタスクについて

Windows 32 ビットの C 用の MQTT クライアント・ライブラリーをビルドする Make ファイルを作成します。

**ヒント:** 45 ページの『[Make ファイル MQTTwin.mak のリスト](#)』には Make ファイルの全体がリストされています。

1. リストされている内容をコピーしてファイルに貼り付けます。
2. ターゲットに続く各行の先行文字をタブに変換します。ステップ [43 ページの『7』](#) を参照してください。
3. それを、手順のステップ [44 ページの『9』](#) にリストされているコマンドを使用して実行します。

## 手順

1. Make ファイル MQTTwin.mak を作成します。

次のプロローグを追加します。

```
# ビルド出力は現行ディレクトリーに作成されます。
# MQTTCLIENT_DIR は、MQTT クライアント・ソース・コードを含む基本ディレクトリーを指す必要があります。
# デフォルトの MQTTCLIENT_DIR は現行ディレクトリーです
# Default OPENSSL_DIR is sdkroot\openssl, relative to sdkroot\ sdk\ clients\ c\ mqttv3c\ src
# OPENSSL_DIR は、OpenSSL ビルドを含むベース・ディレクトリーを指す必要があります。
# 例: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# ビルド環境を設定します。以下に例を示します。
# %comspec% /k "" C: ¥ Program Files¥ Microsoft Visual Studio 9.0\VC\vcvarsall.bat" x86
# set path= %path%; C: ¥ Program Files\GnuWin32\bin;C:\cygwin\bin
```

2. MQTT ソース・コードの場所を設定します。

MQTT ソース・ファイルと同じディレクトリーで Make ファイルを実行するか、MQTTCLIENT\_DIR コマンド行パラメーターを設定します。

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

以下の行を Make ファイルに追加します。

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

この例では、VPATH を、**make** が明示的に識別されていないソース・ファイル (例えば、ビルドに必要なすべてのヘッダー・ファイル) を検索するディレクトリーに設定します。

3. オプション: OpenSSL ライブラリーの場所を設定します。

このステップは、C ライブラリー用の MQTT クライアントの SSL バージョンをビルドするために必要です。

OpenSSL ライブラリーへのデフォルト・パスを、MQTT SDK を展開したディレクトリーと同じディレクトリーに設定します。あるいは、OPENSSL\_DIR をコマンド行パラメーターに設定します。

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

**ヒント:** *OpenSSL* は、すべての *OpenSSL* サブディレクトリーを含む *OpenSSL* ディレクトリーです。ディレクトリー・ツリーは、不必要な空の親ディレクトリーを含んでいるために、その展開場所から移動しなければならないことがあります。

4. 各 MQTT ライブラリーのビルドに必要なすべてのソース・ファイルを選択します。また、ビルドする MQTT ライブラリーの名前と場所を設定します。

以下の行を Make ファイルに追加して、すべての MQTT ソース・ファイルをリストします。

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

ソース・ファイルは、同期または非同期のどちらのライブラリーをビルドするか、およびライブラリーに SSL が含まれるかどうかによって決まります。

ビルドするターゲットに応じて、以下の行の 1 つ以上を追加します。共有ライブラリーおよびマニフェストは、*windows\_ia32* ディレクトリーに作成されます。

- 同期、非セキュア:

```
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
```

- 同期、セキュア:

```
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
```

- 非同期、非セキュア:

```
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
```

- 非同期、セキュア:

```
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. コンパイラーおよびコンパイラー・オプションを定義します。

[各種プラットフォームの MQTT ビルド・オプション](#)に示されている各種プラットフォームのオプションを参照してください。

- a) Microsoft Visual C++ をコンパイラーとして設定します。

```
CC = cl
```

- b) プリプロセッサ・オプションを追加します。

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

- c) コンパイラー・オプションを追加します。

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

- d) インクルード・パスを追加します。

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

- e) オプション: セキュアなライブラリーをビルドする場合、プリプロセッサ・オプションを追加します。

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

- f) オプション: セキュアなライブラリーをビルドする場合、OpenSSL ヘッダー・ファイルを追加します。

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

**ヒント:** ヘッダー・ファイルは \${OPENSSL\_DIR}/inc32/openssl 内にありますが、ssl.h ファイルは "openssl/ssl.h" に含まれています。

6. リンカーおよびリンカー・オプションを設定します。

- a) Microsoft Visual C++ をリンカーとして設定します。

```
LD = link
```

- b) リンカー・オプションを追加します。

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

- c) ライブラリー・パスを追加します。

```
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) 中間出力ファイルを追加します。

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

- e) オプション: セキュアなライブラリーをビルドする場合、OpenSSL ライブラリーを追加します。

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

- f) オプション: セキュアなライブラリーをビルドする場合、OpenSSL ライブラリー・パスを追加します。

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

7. 4つのビルド・ターゲットを定義します。

- a) **all** ターゲットを定義します。

**ヒント:** ターゲットの実装を定義する後続の各行は、タブ文字で開始しなければなりません。

"all" ターゲットは、すべてのライブラリーをビルドします。

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

- b) 同期する非セキュアなライブラリー、mqttv3c.dll をビルドします。

```
${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}
```

ステートメント `-rm ${CURDIR}/MQTTAsync.obj` は、以前のターゲット用に作成されたすべての `MQTTAsync.obj` を削除します。 `MQTTAsync.obj` と `MQTTClient.obj` とは相互に排他的です。

- c) 非同期の非セキュアなライブラリー、mqttv3a.dll をビルドします。

```

${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}

```

ステートメント `-rm ${CURDIR}/MQTTClient.obj` は、以前のターゲット用に作成されたすべての `MQTTClient.obj` を削除します。 `MQTTAsync.obj` と `MQTTClient.obj` とは相互に排他的です。

- d) 同期するセキュアなライブラリー、`mqttv3cs.dll` をビルドします。

```

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

```

ステートメント `-rm ${CURDIR}/MQTTAsync.obj` は、以前のターゲット用に作成されたすべての `MQTTAsync.obj` を削除します。 `MQTTAsync.obj` と `MQTTClient.obj` とは相互に排他的です。

- e) 非同期のセキュアなライブラリー、`mqttv3as.dll` をビルドします。

```

${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}

```

ステートメント `-rm $(CURDIR}/MQTTClient.obj` は、以前のターゲット用に作成されたすべての `MQTTClient.obj` を削除します。 `MQTTAsync.obj` と `MQTTClient.obj` とは相互に排他的です。

- f) **clean** ターゲットを定義します。

"clean" ターゲットは、`makefile` によって生成されたすべてのファイルおよびディレクトリーを除去します。

```

Symphony: クリーン
clean:
-rm -f *.obj
-rm -f -r windows_ia32

```

8. Make ファイルを実行するための Windows パスを設定します。

イタリックの部分は、インストール済み環境と一致するように設定します。

- a) Microsoft Visual Studio 環境を設定します。

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

- b) Path 変数を設定して、Make プログラムと Linux コマンド環境を含めます。

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

9. Make ファイルを実行します。

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

**ヒント:** ファイル区切り文字は、円記号ではなくスラッシュにしなければなりません。

## タスクの結果

以下のファイルが `sdkroot\SDK\clients\c\mqttv3c\src\windows_ia32` ディレクトリーに作成されます。

```
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
```

## Make ファイル MQTTwin.mak のリスト

```
# ビルド出力は現行ディレクトリーに作成されます。
# MQTTCLIENT_DIR は、MQTT クライアント・ソース・コードを含む基本ディレクトリーを指す必要があります。
# デフォルトの MQTTCLIENT_DIR は現行ディレクトリーです
# Default OPENSSL_DIR is sdkroot\openssl, relative to sdkroot\clients\c\mqttv3c\src
# OPENSSL_DIR は、OpenSSL ビルドを含むベース・ディレクトリーを指す必要があります。
# 例: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# ビルド環境を設定します。以下に例を示します。
# %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 (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} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out: $
{MQTTDLL_S}
${MANIFEST_S}

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

Symphony: クリーン
clean:
-rm -f *.obj
-rm -f -r windows_ia32

```

## OpenSSL パッケージのビルド

C、mqttv3cs、および mqttv3as 用のセキュアな MQTT クライアント・ライブラリーをビルドする前に、OpenSSL パッケージをビルドします。このビルドにより、C 用の MQTT クライアント・ライブラリーのセキュア・バージョンをビルドするために必要なライブラリー、および OpenSSL 証明書管理ツールが作成されます。

### 始める前に

1. **iOS** iOS Make ファイル・カスタマイズは、iOS6 を実行するターゲット・デバイス用です。このカスタマイズは、iOS のバージョンが新しいか古いかにより異なることがあります。
2. **Windows** Windows Make ファイル・カスタマイズは、32 ビット Windows 用です。

### このタスクについて

OpenSSL パッケージおよび前提ソフトウェアをダウンロードしてインストールします。OpenSSL Make ファイルをカスタマイズし、ターゲット・プラットフォーム用の OpenSSL ライブラリーをビルドします。Windows および Linux では、Make によって OpenSSL 鍵作成と管理ツールもビルドされます。

### 手順

1. OpenSSL パッケージをインストールします。
  - a) OpenSSL パッケージを [OpenSSL](#) からダウンロードします。

**重要:** OpenSSL パッケージのダウンロードと再配布には、厳密なインポートとエクスポートの規制、およびオープン・ソースのライセンス条件が適用されます。パッケージをダウンロードするかどうかを決定する前に、制限と警告に注意を向けてください。

b) 圧縮されたファイルの内容を `sdkroot` に展開します。

OpenSSL サイトで「**News**」タブの下を参照して、最新のパッケージのダウンロード場所を見つけます。パッケージは圧縮されて、拡張子 `tar.gz` のある `tar` ファイルになっています。パッケージを展開すると、最上位フォルダー `opensslversion` が作成されます。例えば、`openssl-1.0.1c` となります。この例では、フォルダーへのパスを Windows では `%openssl%`、iOS では `$openssl` として示しています。例えば、Windows では `%openssl%` は `sdkroot\openssl-1.0.1c` です。

**ヒント:** OpenSSL パッケージを解凍して作成されたディレクトリー・パスを確認します。いくつかのパッケージには、重複したレベルの `opensslversion` フォルダーがあります。

## 2. Windows

オプション: Windows で、Perl をダウンロードしてインストールします。 [perl.org](http://perl.org) を参照してください。この例で、Perl は [ActivePerl Downloads](#) からダウンロードされています。

## 3. iOS

オプション: iOS で、さらに 3 つのディレクトリーを作成します。

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

iOS では、3 つの異なるハードウェア・プラットフォーム用に OpenSSL パッケージを作成する必要があります。

4. 使用するハードウェアとオペレーティング・システム用に OpenSSL パッケージをビルドするための、OpenSSL Make ファイルを生成します。

a) `%openssl%` または `$openssl` ディレクトリーにコマンド・ウィンドウを開きます。

b) **Configure** Perl コマンドに適切なパラメーターを指定して実行します。

### 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

iOS で、生成された OpenSSL Make ファイルを、さまざまな Apple デバイス用にカスタマイズします。

a) 生成された Make ファイル `$openssl/Makefile` のコピーを 3 つ作成します。

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

b) 各 Make ファイルで、`"CC=gcc"` ステートメントを変更します。

`CC=gcc` ステートメントは 62 行目かその付近にあります。それを次のコマンドに変更します。

**\$openssl/Makefile\_armv7**

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

## \$openssl/Makefile\_armv7s

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

## \$openssl/Makefile\_i386

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

- c) 各 Make ファイルで、"CFLAG=..." ステートメントを変更します。

このステートメントは 63 行目かその付近にあります (読みやすくするために 3 行に分けて示します)。

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

示される SDK の場所は、Xcode インストール済み環境の選択によって異なります。SDK のバージョンは、Make ファイルをビルドするオペレーティング・システムのレベルによって異なります。

### iPhone シミュレーター

iPhone シミュレーターの Make ファイルは、\$openssl/Makefile\_i386 です。

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

### iOS

iOS の Make ファイルは、\$openssl/Makefile\_arm7 および \$openssl/Makefile\_arm7s です。

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

6. 生成された Make ファイルを実行します。

- Windows

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

- iOS の場合:

```
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
```

## タスクの結果

このビルドによって、セキュアなバージョンの C 用の MQTT クライアント・ライブラリーをビルドするために必要な共有ライブラリー・ファイル、lib ファイル、およびヘッダー・ファイルが生成されます。



## iOS における C 用 MQTT クライアントの概要

iOS アプリケーションが MQTT サーバーとメッセージを交換するようにする方法を学習します。iOS デバイス (つまり、iPhone および iPad) で使用する場合は、MQTT Software Development Kit の一部として提供されるソース・コードから C 用の MQTT クライアント・ライブラリーをビルドする必要があります。

### 始める前に

1. [iOS Dev Center](#) にリンクし、iOS 用のアプリケーションを開発する方法を理解します。
2. Apple Mac (OS X 10.8.2 以降) を取得して、Xcode 統合開発環境 (IDE) を実行します。
3. (オプション) Windows または Linux での C 開発環境を構成します。MQTT iOS アプリケーションを開発する前に、Windows または Linux で MQTT クライアント・サンプル C アプリケーションをビルドして実行しておくとお便利です。代わりに、サンプルをビルドしないでサンプル・ソース・コードを検討することもできます。
4. C 用の MQTT クライアントのサポートされるプラットフォームと参照プラットフォームについては、[IBM Mobile Messaging and M2M クライアント・パックのシステム要件](#)を参照してください。
5. クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。

### このタスクについて

この手順では、以下のステップをガイドします。

1. MQTT クライアント・サンプル・アプリケーションと C 用の MQTT クライアント・ライブラリーを検出し、ビルドして実行することにより、MQTT のプログラミングについて学習します。
2. iOS 用の Xcode 開発環境を、Apple Mac にインストールします。
3. [31 ページの『C 用の MQTT クライアント・ライブラリーのビルド』](#) タスクを実行して、iOS 装置用の C ライブラリー用の MQTT クライアントをビルドします。

### 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。  
サーバーは MQTT version 3.1 プロトコルをサポートしている必要があります。IBM のすべての MQTT サーバー (IBM WebSphere MQ および IBM MessageSight を含む) がこれを行います。[139 ページの『MQTT サーバーの概要』](#)を参照してください。
2. Mobile Messaging and M2M クライアント・パックをダウンロードし、MQTT SDK をインストールします。  
インストール・プログラムは用意されておらず、ダウンロードしたファイルを展開するだけです。
  - a. [Mobile Messaging and M2M クライアント・パック](#)をダウンロードします。
  - b. SDK をインストールするフォルダーを作成します。  
フォルダーの名前を MQTT にすることもできます。ここでは、このフォルダーへのパスを *sdkroot* として示します。
  - c. 圧縮された Mobile Messaging and M2M クライアント・パック ファイルの内容を *sdkroot* に展開します。展開すると、*sdkroot*\SDK から始まるディレクトリー・ツリーが作成されます。
3. オプション: MQTT クライアント・サンプル C アプリケーションを学習して、MQTT API についてよく理解します。
  - a) Windows または Linux 用の同期 MQTT クライアント・サンプル C アプリケーション MQTTV3sample.c をビルドします。[27 ページの『C 用の MQTT クライアントの概要』](#)を参照してください。
  - b) MQTT version 3 サーバーに接続して、サーバー上のトピックにパブリッシュおよびサブスクライブします。

- c) ソース・コードと MQTT API の資料を確認してください。MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、[MQTT クライアント・プログラミング・リファレンス](#)を参照してください。

同期サンプルを検討することにより、MQTT クライアントを作成して再開する方法、および MQTT トピックにパブリッシュしてサブスクライブする方法を学びます。同期サンプルは非同期サンプルよりも単純です。以前に MQTT をプログラミングしたことがない場合は、同期 MQTT プログラムを記述して、MQTT プログラミング・モデルと API に関する理解を深めてください。

- d) C 用の非同期 MQTT クライアント・ライブラリーを Windows または Linux 上にビルドします。31 ページの『[C 用の MQTT クライアント・ライブラリーのビルド](#)』を参照してください。
- e) 非同期 MQTT クライアント・サンプルのパブリッシュおよびサブスクライブ C アプリケーションをビルドして実行します。
- f) MQTT クライアント・サンプル非同期 C アプリケーションのソース・コード、および MQTT 参照文書を検討します。

モバイル・デバイス用の MQTT アプリケーションを記述するためには、非同期インターフェースを使用する必要があります。上手に記述されたアプリで非同期インターフェースを呼び出すものは、同期インターフェース用に記述されたアプリと比較して、即応性が良く、バッテリー寿命が長くなります。

非同期インターフェースには、次の 2 つの度合いの非同期性があります。

- i) 1 番目の度合いは、MQTT クライアント・ライブラリーがサーバーからのパブリケーションを待機している間、アプリケーションをブロック解除します。
- ii) 2 番目の度合いはクライアント・ライブラリーがサーバーに接続し、サブスクリプションを作成し、パブリケーションを送信する間、アプリケーションをブロック解除します。

#### 4. iOS 開発ツールをダウンロードしてインストールします。

- a. 管理特権を持つユーザー ID を使ってログオンします。
- b. Apple Mac がバージョン 10.8.2 以降であることを確認します。
- c. Web サイト [Xcode](#) に進んで、Xcode を Mac App Store からダウンロードします。
- d. Xcode、コマンド行環境、およびシミュレーターをインストールします。

Mac App Store が複数のバージョンのシミュレーターを提供する場合、アプリのためにターゲットとしている iOS のレベルと互換性のあるバージョンを選択します。

#### 5. iOS で C 用の MQTT クライアント・ライブラリーをビルドします。31 ページの『[C 用の MQTT クライアント・ライブラリーのビルド](#)』を参照してください。

## 次のタスク

1. ビルドした C 用の MQTT クライアント・ライブラリーを以下のように検証します。
- a. Xcode 開発環境を使用して、非同期 MQTT クライアント・サンプル C アプリケーションをコンパイルし、非セキュアな C 用の非同期 MQTT クライアント・ライブラリーにリンクさせます。
- b. Xcode 開発環境を使用して、非同期 MQTT クライアント・サンプル C アプリケーションを iOS デバイス上で実行します。サンプルを、構成した MQTT version 3 サーバーに接続します。143 ページの『[コマンド・ラインからの MQTT サービスの構成](#)』を参照してください。
2. iOS 用の MQTT クライアント C アプリケーションを作成します。非同期 C アプリケーションのコーディング例が役立つ可能性があります。例は、`sdkroot\SDK\clients\c\samples\MQTTV3ASample.c` および `MQTTV3ASSample.c` です。演習として、MQTT パブリッシュ/サブスクライブのサンプルを実装することから開始してください。

**ヒント:** アプリの外観と動作の概要を知るために、MQTT クライアント・サンプル C アプリケーションの画面キャプチャーをご覧ください。18 ページの『[Android 上の Java 用の MQTT クライアントの概要](#)』を参照してください。

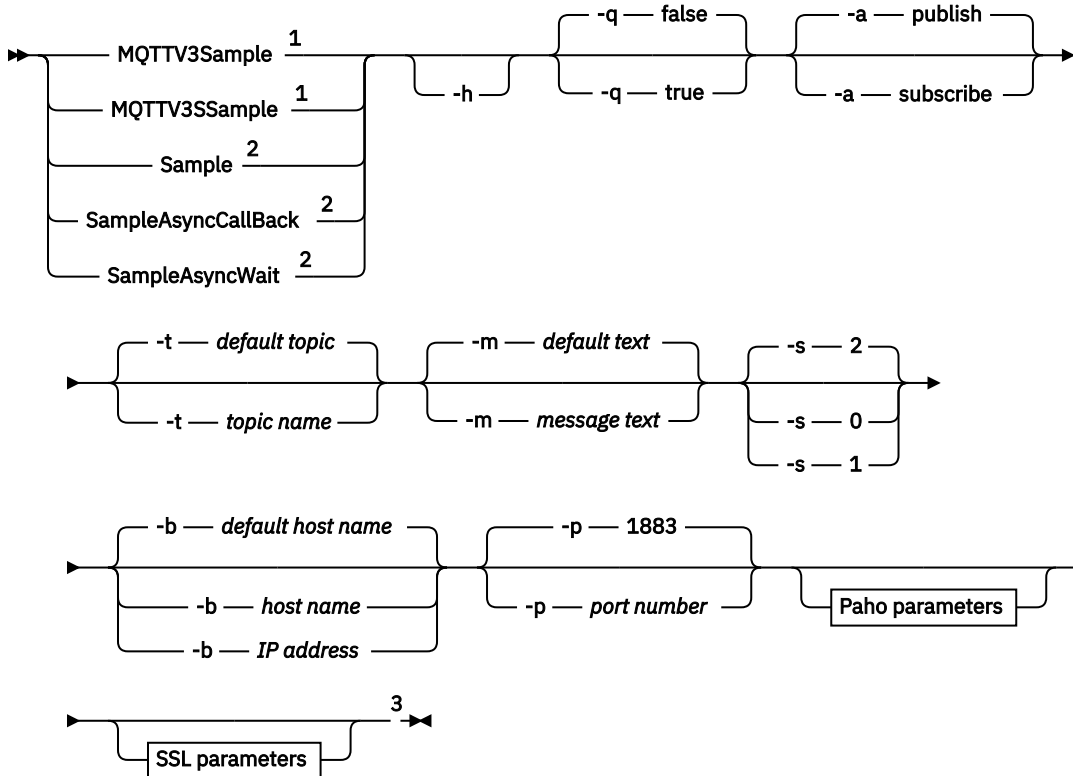
# MQTT コマンド行サンプル・プログラム

MQTT コマンド行サンプル・プログラムの構文とパラメーター

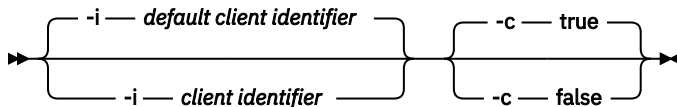
## 目的

トピックに対してパブリッシュおよびサブスクライブします。

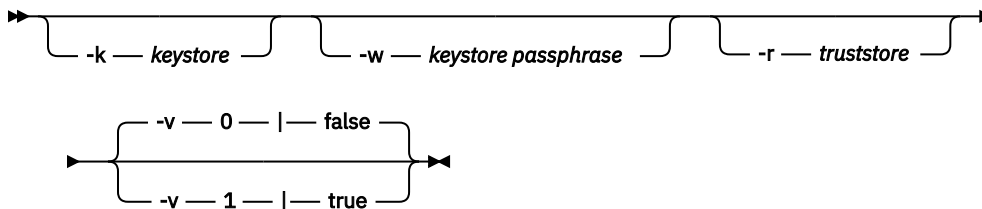
## Syntax



## Paho parameters



## SSL parameters



注:

- <sup>1</sup> IBM WebSphere MQ sample
- <sup>2</sup> Paho sample
- <sup>3</sup> Not MQTTV3Sample.

## Parameters

### -h

このヘルプ・テキストを出力して終了します。

### -q

デフォルト・モードの `false` を使用する代わりに抑止モードを設定します。

### -a **publish|subscribe**

デフォルト・アクションのパブリッシュを想定する代わりに、アクションを `publish` または `subscribe` に設定します。

### -t **topic name**

デフォルトのトピックにパブリッシュまたはサブスクライブする代わりに、`topic name` にパブリッシュまたはサブスクライブします。デフォルトのトピックは以下のとおりです。

#### Paho サンプル

パブリッシュ

Sample/Java/v3

サブスクライブ

Sample/#

#### IBM WebSphere MQ のサンプル

パブリッシュ

MQTTV3Sample/Java/v3 または MQTTV3Sample/C/v3

サブスクライブ

MQTTV3Sample/#

### -m **message text**

デフォルト・テキストを送信する代わりに、`message text` をパブリッシュします。デフォルトのテキストは、"Message from MQTTv3 C client" または "Message from MQTTv3 Java client" のいずれかです。

### -s **0|1|2**

サービスの品質 (QoS) を、デフォルトの QoS の 2 を使用する代わりに設定します。

### -b **host name**

デフォルトのホスト名に接続する代わりに、`host name` または IP アドレスに接続します。Paho サンプルのデフォルトのホスト名は、`m2m.eclipse.org` です。IBM WebSphere MQ サンプルの場合、それは `localhost` です。

### -p **port number**

デフォルトのポート 1883 を使用する代わりに、ポート `port number` を使用します。

## Paho パラメーター

### -i **client identifier**

クライアント ID を `client identifier` に設定します。デフォルトのクライアント ID は `SampleJavaV3_"+action` です。ここで、`action` は `publish` または `subscribe` です。

### -c **true|false**

クリーン・セッション・フラグを設定します。デフォルトは `true` で、サブスクリプションは永続的ではありません。

## SSL

### -k **keystore**

クライアントを識別する秘密鍵を含んでいる鍵ストアへのパスを、`keystore` に設定します。C サンプルでは、ストアは Privacy-Enhanced Mail (PEM) ファイルです。Java サンプルでは、それは Java 鍵ストア (JKS) です。

### **-w keystore passphrase**

クライアントに鍵ストアへのアクセスを許可するパスフレーズを、*keystore passphrase* に設定します。

### **-r truststore**

クライアントが信頼する MQTT サーバーの公開鍵を含む鍵ストアのパスを *truststore* に設定します。鍵ストアは、Privacy-Enhanced Mail (PEM) ファイルです。C サンプルでは、ストアは Privacy-Enhanced Mail (PEM) ファイルです。Java サンプルでは、それは Java 鍵ストア (JKS) です。

### **-v 0|false|1>true**

検査オプションを *1|true* に設定して、サーバー証明書を必要にします。デフォルトは *0|false* です。サーバー証明書は検査されません。SSL チャネルは常に暗号化されます。

C プログラムの場合はオプションを *0|1* に設定し、Java プログラムの場合は *true|false* に設定します。

## 関連タスク

### 12 ページの『Java 用の MQTT クライアントの概要』

IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてを使用して、MQTT Client for Java サンプル・アプリケーションを稼働させます。サンプル・アプリケーションは、IBM の MQTT Software Development Toolkit (SDK) のクライアント・ライブラリーを使用します。SampleAsyncCallback サンプル・アプリケーションは、Android およびその他のイベント・ドリブン・オペレーティング・システム用の MQTT アプリケーションを作成するためのモデルです。

### 27 ページの『C 用の MQTT クライアントの概要』

C ソースをコンパイルできる任意のプラットフォームで C 用のサンプル MQTT クライアントを起動して稼働状態にします。C 用のサンプル MQTT クライアントを IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてで実行できることを確認します。

### 31 ページの『C 用の MQTT クライアント・ライブラリーのビルド』

以下のステップに従って、C 用の MQTT クライアント・ライブラリーをビルドします。このトピックには、複数のプラットフォーム用のコンパイルおよびリンク・スイッチ、および iOS と Windows でライブラリーをビルドする例が含まれています。

## MQTT セキュリティー

MQTT セキュリティーの基本は 3 つの概念、すなわち、ID、認証、および許可です。ID とは許可を受けて権限を受けようとするクライアントのネーミングです。認証とはクライアントの ID を証明すること、許可とはクライアントに付与される権利の管理のことです。

### セキュリティ・サンプルの試行

- 57 ページの『セキュア MQTT クライアント・サンプル Java アプリケーションのビルドと実行』
- 65 ページの『SSL を介した Android での MQTT クライアント・サンプル Java アプリの接続』
- 74 ページの『JAAS を使用した MQTT クライアント Java アプリの認証』
- **V7.5.0.1** 79 ページの『SSL を介した JavaScript 用の MQTT メッセージング・クライアントと WebSockets の接続』
- 87 ページの『セキュア MQTT クライアント・サンプル C アプリケーションのビルドと実行』

### ID

クライアント ID、ユーザー ID、または公開デジタル証明書を使用して MQTT クライアントを識別します。これらの属性のいずれかにより、クライアント ID を定義します。MQTT サーバーはクライアントから SSL プロトコルで送信された証明書、またはクライアントが設定したパスワードを伴うクライアント ID を認証します。サーバーは、クライアント ID に基づいて、クライアントがどのリソースにアクセスできるかを制御します。

MQTT サーバーは、クライアントに IP アドレスとデジタル証明書を使用して自身を示します。MQTT クライアントは、サーバーが送信した証明書を SSL プロトコルを使用して認証します。場合によっては、サー

バーの DNS 名を使用して、証明書を送信したサーバーが証明書所有者として登録されていることを検証します。

クライアントの ID を以下のいずれかの方法で設定します。

#### クライアント ID

MqttClient クラス (C では MQTTClient\_create または MQTTAsync\_create) によりクライアント ID を設定します。パラメーターとしてクライアント ID を設定するか、ランダムに生成したクライアント ID を返すため、クラス・コンストラクターを呼び出します。クライアント ID は、サーバーに接続するすべてのクライアントで固有である必要があります、サーバー上のキュー・マネージャー名と同じであってはなりません。すべてのクライアントは、ID 検査に使用されない場合でも、クライアント ID を持っている必要があります。130 ページの『クライアント ID』を参照してください。

#### ユーザー ID

MqttClient クラス (C では MQTTClient\_create または MQTTAsync\_create) により、MqttConnectOptions (C では MqttClient\_ConnectOptions) の属性としてクライアント・ユーザー ID を設定します。ユーザー ID はクライアント固有のものである必要はありません。

#### クライアント・デジタル証明書

クライアント・デジタル証明書はクライアント鍵ストアに保管されます。鍵ストアの場所はクライアントによって異なります。

##### • Java

クライアント鍵ストアの場所とプロパティーを、MqttConnectOptions の setSSLProperties メソッドを呼び出して鍵ストアのプロパティーを渡すことにより、設定します。[Example.java](#) に対する [SSL 変更](#) を参照してください。**keytool** ツールは、Java 鍵および鍵ストアを管理します。

##### • C

MQTTClient\_create または MQTTAsync\_create は、鍵ストア・プロパティーを MQTTClient\_SSLOptions ssl\_opts の属性として設定します。**openssl** ツールは、C 用の MQTT クライアントによってアクセスされる鍵および鍵ストアを作成および管理します。

##### • Android

Android デバイスの鍵ストアは、「設定」 > 「セキュリティ」メニューから管理します。新しい証明書は SD カードからロードします。

以下のように、サーバーの ID を、秘密鍵をサーバーの鍵ストアに保管することにより設定します。

#### IBM WebSphere MQ

MQTT サーバーの鍵ストアは、クライアントが接続される遠隔測定チャネルの属性です。

IBM WebSphere MQ Explorer または **DEFINE CHANNEL** コマンドを使用して、鍵ストアの場所と属性を設定します。[DEFINE CHANNEL \(MQTT\)](#) を参照してください。複数のチャネルで 1 つの鍵ストアを共有できます。

## 認証

MQTT クライアントは、接続先 MQTT サーバーを認証でき、サーバーは接続元クライアントを認証できます。

クライアントは SSL プロトコルを使用してサーバーを認証します。MQTT サーバーは SSL プロトコル、パスワード、またはその両方を使用してクライアントを認証します。

クライアントがサーバーを認証するものの、サーバーはクライアントを認証しない場合、そのクライアントは多くの場合、匿名クライアントと呼ばれます。匿名クライアント接続を SSL を介して確立し、その後 SSL セッションで暗号化されたパスワードを使用してクライアントを認証するのが一般的です。また、クライアントの認証にクライアント証明書を使用すると証明書の配布と管理の問題が生じるので、パスワードを使用するほうがより一般的です。クライアント証明書の使用は、ATM や Chip-and-Pin カード対応機などの高価値デバイスや、スマート電力計量などのカスタム・デバイスでよく見られます。

## クライアントによるサーバー認証

MQTT クライアントは、SSL プロトコルを使用してサーバー証明書を認証することにより、正しいサーバーに接続していることを検証します。この検証形式は、HTTPS プロトコルを介して Web サイトをブラウズするときによく見られます。

サーバーは、認証局の署名を受けた自身の公開証明書をクライアントに送信します。クライアントは認証局の公開鍵を使用して、サーバー証明書上の認証局の署名を検証します。また、証明書が現行のものであることも検査します。このような検査により、証明書が有効であることが確認されます。

認証局証明書(ルート証明書とも呼ばれる)は、クライアントのトラストストアに保管されています。

### • Java

クライアント・トラストストアの場所とプロパティを、`MqttConnectOptions` の `setSSLProperties` メソッドを呼び出してトラストストアのプロパティを渡すことにより、設定します。[Example.java](#) に対する [SSL 変更](#) を参照してください。証明書とトラストストアは **keytool** ツールで管理します。

### • C

`MQTTClient_create` または `MQTTAsync_create` は、トラストストア・プロパティを `MQTTClient_SSLOptions ssl_opts` の属性として設定します。証明書とトラストストアは **openSSL** ツールで管理します。

### • Android

Android デバイスのトラストストアは、「設定」 > 「セキュリティ」メニューから管理します。新しいルート証明書は SD カードからロードします。

## サーバーによるクライアント認証

MQTT サーバーは、SSL プロトコルを使用してクライアント証明書を認証するか、パスワードを伴うクライアント ID を認証することにより、正しいクライアントに接続していることを検証します。

サーバーは、クライアントからサーバーに MQTT protocol・ヘッダーで送信されたパスワードでクライアントを認証します。サーバーは、クライアント ID、ユーザー ID、または証明書をパスワードで認証することを選択する場合があります。これはサーバーによって異なります。通常は、サーバーはユーザー ID を認証します。サーバーを検証することによって保護された SSL 接続を介してパスワードを確認します。平文でパスワードを送信することは避けてください。

### • IBM WebSphere MQ

IBM WebSphere MQ は SSL プロトコルを使用してクライアント証明書を認証します。ルート証明書は IBM WebSphere MQ Telemetry 鍵ストアに保管します。クライアント証明書は相互 SSL 認証の一部としてのみ認証できます。つまり、サーバーにクライアント証明書を提供するだけではなく、クライアントにサーバー公開証明書を提供しなければなりません。

IBM WebSphere MQ Telemetry は、自身のプライベート証明書と公開証明書、および認証局が提供するルート証明書などの他の公開証明書の両方に、同じストアを使用します。

IBM WebSphere MQ Explorer または **DEFINE CHANNEL** コマンドを使用して、鍵ストアの場所と属性を設定します。[DEFINE CHANNEL \(MQTT\)](#) を参照してください。複数のチャンネルで 1 つの鍵ストアを共有できます。

IBM WebSphere MQ は、クライアント・ユーザー ID またはクライアント ID を、Java 認証・承認サービス (JAAS) を呼び出すことにより認証します。

`jaas.config` ファイルに保管された `MQXRConfig` 構成スタンプで JAAS を構成します。このファイルは、IBM WebSphere MQ データ・パスの `qmgrs\QmgrName\mqxr` ディレクトリーに保管されます。

クライアントの認証性は、`JAASLoginModule` の `login` メソッドを記述することによって検査します。[116 ページの『テレメトリー・チャンネルの JAAS 構成』](#) を参照してください。

IBM WebSphere MQ Telemetry は、`JAASLoginModule.login` メソッドに以下のパラメーターを渡します。

- ユーザー ID
- パスワード
- クライアント ID
- ネットワーク ID
- チャンネル名
- ValidPrompts

## 許可

許可は MQTT protocol の一部ではありません。これは MQTT サーバーによって提供されます。何が許可されるかは、サーバーが何を行うかに依存します。MQTT サーバーはパブリッシュ/サブスクライブ・ブローカーであり、どのクライアントがサーバーに接続できるか、また、クライアントがどのトピックをパブリッシュまたはサブスクライブできるかは、有用な MQTT 許可規則によって制御されます。MQTT クライアントがサーバーを管理できる場合、より多くの許可規則を使用して、どのクライアントがサーバーのそれぞれ異なる面を管理できるかを制御します。

クライアント数は非常に多数になる可能性があるため、各クライアントに対して別個に許可を行うのは実現可能とは言えません。MQTT サーバーにはプロファイルまたはグループによってクライアントをグループ化する手段があります。

クライアントの ID は、アクセス権限と許可の観点からは、MQTT クライアント固有のものではありません。クライアントの ID (identity of a client) とクライアント ID (client identifier) を同一視しないでください。これらは同じ可能性があります。例えば、多くのサービスに共通するユーザー名があり、これらのサービスの一部が "シングル・サインオン" で連携しているとします。エンタープライズ・スケールの MQTT サーバーでは、複数の異なるアプリケーションに ID と権限を提供する共通の許可サービスを呼び出す可能性が高くなります。

## IBM WebSphere MQ

IBM WebSphere MQ にはプラグ可能な許可サービスがあります。Windows と Linux で提供されるデフォルトの許可サービスは、オブジェクト権限マネージャー (OAM) です。[UNIX、Linux および Windows システムでの OAM によるオブジェクトへのアクセス制御を参照してください。](#)これは、オペレーティング・システムのユーザー ID とグループを、トピックやキューなどの IBM WebSphere MQ オブジェクトへの操作と関連付けます。

固定ユーザー ID で IBM WebSphere MQ にアクセスするように遠隔測定チャンネルを構成することができます。これがサンプル・チャンネルのセットアップ方法です。あるいは、MQTT クライアントによって設定されたユーザー ID を使用して IBM WebSphere MQ にアクセスすることもできます。[MQTT クライアントに WebSphere MQ オブジェクトへのアクセス権限を付与するに、クライアント・アクセス制御を大ざっぱに、またはきめ細かく、あるいはその中間で実現するように IBM WebSphere MQ Telemetry をセットアップする方法の説明があります。](#)

## 関連タスク

### [87 ページの『セキュア MQTT クライアント・サンプル C アプリケーションのビルドと実行』](#)

Windows の例に基づいて、C ソースをコンパイルできる任意のオペレーティング・システムでセキュア・サンプル C アプリケーションを起動して稼働状態にすることができます。IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとして、サンプル C アプリケーションを実行できることを確認します。

### [57 ページの『セキュア MQTT クライアント・サンプル Java アプリケーションのビルドと実行』](#)

Windows の例に基づいて、IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてセキュア・サンプル Java アプリを稼働させることができます。JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアント アプリを実行できます。

### [79 ページの『SSL を介した JavaScript 用の MQTT メッセージング・クライアントと WebSockets の接続』](#)

SSL を使用する JavaScript 用の MQTT メッセージング・クライアント サンプル HTML ページと WebSocket protocol を使用して、Web アプリを IBM WebSphere MQ に安全に接続します。

### [65 ページの『SSL を介した Android での MQTT クライアント・サンプル Java アプリの接続』](#)



SSL を介して IBM WebSphere MQ に接続されたサンプル Android MQTT クライアントを稼働させます。

74 ページの『[JAAS を使用した MQTT クライアント Java アプリの認証](#)』

JAAS を使用してクライアントを認証する方法を学びます。このタスクのステップを実行して、サンプル・プログラム JAASLoginModule.java を変更し、JAAS で MQTT クライアント Java アプリケーションを認証するように IBM WebSphere MQ を構成します。

## セキュア MQTT クライアント・サンプル Java アプリケーションのビルドと実行

Windows の例に基づいて、IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてセキュア・サンプル Java アプリを稼働させることができます。JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアント アプリを実行できます。

### 始める前に

1. MQTT protocol over SSL をサポートする MQTT version 3.1 サーバーにアクセスできる必要があります。
2. クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。
3. JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアント アプリを実行できます。 [. IBM Mobile Messaging and M2M クライアント・パックのシステム要件](#) を参照してください。
4. SSL チャンネルを開始する必要があります。

### このタスクについて

例として、この記事では、コマンド行からセキュア MQTT クライアント・サンプル Java アプリケーション on Windows をコンパイルして実行する方法を示します。

SSL チャンネルを、認証局署名鍵か自己署名鍵のどちらかを使用して保護します。

### 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。  
サーバーは、SSL を介した MQTT version 3.1 プロトコルをサポートする必要があります。IBM のすべての MQTT サーバー (IBM WebSphere MQ および IBM MessageSight を含む) がこれを行います。 [139 ページの『MQTT サーバーの概要』](#) を参照してください。
2. オプション: Java Development Kit (JDK) バージョン 7 以降をインストールします。  
**keytool** コマンドを実行して証明書を認証するには、バージョン 7 が必要です。証明書の認証を行わない場合、バージョン 7 JDK は必要ありません。
3. Mobile Messaging and M2M クライアント・パック をダウンロードし、MQTT SDK をインストールします。  
インストール・プログラムは用意されておらず、ダウンロードしたファイルを展開するだけです。
  - a. [Mobile Messaging and M2M クライアント・パック](#) をダウンロードします。
  - b. SDK をインストールするフォルダーを作成します。  
フォルダーの名前を MQTT にすることもできます。ここでは、このフォルダーへのパスを `sdkroot` として示します。
  - c. 圧縮された Mobile Messaging and M2M クライアント・パック ファイルの内容を `sdkroot` に展開します。展開すると、`sdkroot\SDK` から始まるディレクトリー・ツリーが作成されます。
4. 鍵ペアと証明書を生成するスクリプトを作成して実行し、IBM WebSphere MQ を MQTT サーバーとして構成します。

[97 ページの『鍵と証明書の生成』](#) のステップに従って、スクリプトを作成して実行します。これらのスクリプトのリストも [59 ページの『Windows の SSL 証明書を構成するスクリプトの例』](#) にあります。

5. SSL チャンネルが実行されていて、期待どおりにセットアップされていることを確認します。

IBM WebSphere MQ のコマンド・ウィンドウに以下のコマンドを入力します。

• **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. スクリプトを作成し、セキュア MQTT クライアント・サンプル Java アプリケーションをビルドして実行します。

- ssjavaclient.bat を作成して実行し、自己署名証明書で保護された SSL チャンネルをテストします。
- cajavaclient.bat を作成して実行し、認証局署名証明書で保護された SSL チャンネルをテストします。

### MQTT セキュア Java クライアントを実行するスクリプト

ここで示すスクリプトを実行する前に、[59 ページの『Windows の SSL 証明書を構成するスクリプトの例』](#)にあるスクリプトを実行します。

#### 自己署名証明書を使用する MQTT セキュア Java クライアント。

このスクリプトは、sscerts.bat スクリプトを実行して作成した自己署名証明書を使用して実行します。

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

図 14. ssjavaclient.bat

#### 認証局署名証明書を使用した MQTT セキュア Java クライアントの実行。

このスクリプトは、cacerts.bat スクリプトを実行して作成した認証局署名証明書を使用して実行します。

```

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

```

図 15. *cajavaclient.bat*

## 関連概念

[53 ページの『MQTT セキュリティー』](#)

MQTT セキュリティーの基本は3つの概念、すなわち、ID、認証、および許可です。IDとは許可を受けて権限を受けようとするクライアントのネーミングです。認証とはクライアントのIDを証明すること、許可とはクライアントに付与される権利の管理のことです。

## 関連タスク

[97 ページの『鍵と証明書の生成』](#)

以下の手順に従って、Java および C クライアント (Android と iOS アプリケーション、および IBM WebSphere MQ と IBM MessageSight サーバーを含む) 用の鍵と証明書を生成します。

[65 ページの『SSL を介した Android での MQTT クライアント・サンプル Java アプリの接続』](#)

SSL を介して IBM WebSphere MQ に接続されたサンプル Android MQTT クライアントを稼働させます。

[74 ページの『JAAS を使用した MQTT クライアント Java アプリの認証』](#)

JAAS を使用してクライアントを認証する方法を学びます。このタスクのステップを実行して、サンプル・プログラム JAASLoginModule.java を変更し、JAAS で MQTT クライアント Java アプリケーションを認証するように IBM WebSphere MQ を構成します。

## Windows の SSL 証明書を構成するスクリプトの例

### &nbsp;

タスクの手順に示す説明に従って、サンプル・コマンド・ファイルで証明書と証明書ストアを作成します。さらに、例では、サーバー証明書ストアを使用するように MQTT クライアント・キュー・マネージャーをセットアップします。この例では、IBM WebSphere MQ で提供されている SampleMQM.bat スクリプトを呼び出すことによって、キュー・マネージャーを削除して再作成します。

### initcert.bat

initcert.bat は、**keytool** および **openssl** コマンドが必要とする、証明書の名前とパス、およびその他のパラメーターを設定します。スクリプト内のコメントに、設定の説明があります。

```

@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT

```

```
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

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

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

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

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

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

```
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

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

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

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

## cleancert.bat

cleancert.bat スクリプト内のコマンドは、MQTT クライアント・キュー・マネージャーを削除することにより、サーバー証明書ストアがロックされないようにします。その後、サンプル・セキュリティー・スクリプトによって作成されたすべての鍵ストアと証明書を削除します。

```
@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 %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
```

```

erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

## genkeys.bat

genkeys.bat スクリプト内のコマンドは、プライベート認証局、サーバー、およびクライアント用の鍵ペアを作成します。

```

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

```

```

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

```

```

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

```

## sscerts.bat

sscerts.bat スクリプト内のコマンドは、クライアントとサーバーの各鍵ストアにあるそれぞれの自己署名証明書をエクスポートし、サーバー証明書をクライアント・トラストストアにインポートし、クライアント証明書をサーバーの鍵ストアにインポートします。サーバーにはトラストストアはありません。このコマンドは、クライアント JKS トラストストアから PEM 形式のクライアント・トラストストアを作成します。

```

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

```

```

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

```

```

@rem
@echo -----
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%

```

```

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

```

```

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

```

## cacerts.bat

このスクリプトは、認証局ルート証明書をプライベート鍵ストアにインポートします。CA ルート証明書は、ルート証明書と署名済み証明書との鍵チェーンを作成するために必要になります。cacerts.bat スクリプトがクライアントとサーバーの各鍵ストアからそれぞれの証明書要求をエクスポートします。スクリプトは、cajkskeystore.jks 鍵ストアにあるプライベート認証局の鍵を使用して、証明書要求に署名します。そして、要求が来た元の同じ鍵ストアに、署名済み証明書をインポートします。このインポートによって、CA ルート証明書との証明書チェーンが作成されます。スクリプトはクライアント JKS トラストストアから、PEM 形式のクライアント・トラストストアを作成します。

```

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

```

```

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

```

```

@rem
@echo -----
@echo Create certificate signing requests: %svrcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %svrcertreq% -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: %srvcertassigned% and %cltcertassigned%
@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 %srvcertassigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertassigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

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

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

```
@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
%clt12keystore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass% -deststorepass %clt12keystorepass%
%openssl%\bin\openssl pkcs12 -in %clt12keystore% -out %cltpepemkeystore% -passin
pass:%clt12keystorepass% -passout pass:%cltpepemkeystorepass%
```

## mqcerts.bat

スクリプトは、証明書ディレクトリーにある鍵ストアと証明書をリストします。そして、MQTT サンプル・キュー・マネージャーを作成し、セキュア・テレメトリー・チャンネルを構成します。

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

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



```
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
```

```
V7.5.0.1
```

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

## SSL を介した Android での MQTT クライアント・サンプル Java アプリの接続

SSL を介して IBM WebSphere MQ に接続されたサンプル Android MQTT クライアントを稼働させます。

### 始める前に

この記事では、少なくとも Android API レベル 14 (ICS 4.0) で実行することを前提とします。以前のレベルには鍵ストアがありましたが、これにアクセスできるのはシステム・アプリケーションだけでした。

1. MQTT protocol over SSL をサポートする MQTT version 3.1 サーバーにアクセスできる必要があります。
2. クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。
3. 以前の Android デバイスで接続をテストする場合は、そのデバイスに証明書を転送するために SD カードが必要になる可能性があります。
4. 仮想 Android デバイスで接続をテストする場合は、仮想デバイス用に SD カードを構成します。
5. SSL チャネルを開始する必要があります。

### このタスクについて

SSL を介して Android 用の MQTT クライアント・サンプル Java アプリケーションを実行するには、このタスクを完了します。SSL 接続が正常に行われると、Android デバイスと MQTT サーバーとの間のセキュアな暗号化チャネルが確立されます。サーバーの ID は認証されます。

Android では、SSL を使用してサーバーを認証できます。デバイスも認証できますが、サンプル・アプリケーションではサポートされません。デバイスを認証するには、[KeyChain API](#) を使用するか、JAAS を使用して、クライアントの ID、クライアントの IP アドレス、または MQTT Android アプリケーションが提供するユーザー名とパスワードを認証します。

Android トラストストアにインストールするすべての X.509 証明書には、認証局による署名が必要です。例では、Android デバイスにインストールする証明書に署名する認証局を作成します。Android デバイスには多数のルート証明書が事前にインストールされています。

信頼できる証明書をインストールする前に、Android デバイスでロックを作成する必要があります。ロックは、知らないうちに誰かがデバイスに証明書をインストールできないようにします。

### 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。  
サーバーは、SSL を介した MQTT version 3.1 プロトコルをサポートする必要があります。IBM のすべての MQTT サーバー (IBM WebSphere MQ および IBM MessageSight を含む) がこれを行います。[139 ページの『MQTT サーバーの概要』](#)を参照してください。
2. 保護されていない MQTT チャネルで MQTT クライアント・サンプル・アプリ for Android "MQTTExerciser" を実行します。[18 ページの『Android 上の Java 用の MQTT クライアントの概要』](#)を参照してください。  
再度アプリケーションを使用して、セキュア・チャネルをテストします。  
Android 仮想デバイスを開始した場合は、稼働させたままにします。
3. オプション: Java Development Kit (JDK) バージョン 7 以降をインストールします。

**keytool** コマンドを実行して証明書を認証するには、バージョン7が必要です。証明書の認証を行わない場合、バージョン7 JDK は必要ありません。

4. 鍵ペアと証明書を生成するスクリプトを作成して実行し、IBM WebSphere MQ を MQTT サーバーとして構成します。

97 ページの『[鍵と証明書の生成](#)』のステップに従って、スクリプトを作成して実行します。これらのスクリプトのリストも 69 ページの『[Windows の SSL 証明書を構成するスクリプトの例](#)』にあります。

認証局の公開証明書とサーバー鍵ストアが必要になります。クライアント証明書や、.pem または .p12 形式の証明書は必要ありません。

5. SSL チャンネルが実行されていて、期待どおりにセットアップされていることを確認します。

IBM WebSphere MQ のコマンド・ウィンドウに以下のコマンドを入力します。

• **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. 認証局証明書を Android トラストストアにインストールします。

例の認証局ファイルは `cacert.cer` です。

- a) 証明書の名前を `cacert.crt` に変更します。
- b) 証明書をルート内部ストレージ、または SD カードにコピーします。

稼働中の仮想デバイスでは、以下のように、Eclipse を開くか、Android Debug Bridge (ADB) を実行して、証明書を仮想デバイスにコピーします。

### Eclipse

- i) Eclipse を実行して、DDMS パースペクティブを開きます。
- ii) メイン・ビューで、「**ファイル・エクスプローラー**」ウィンドウを開きます。
- iii) `mnt/sdcard` ディレクトリーを開きます。
- iv) `cacert.crt` ファイルを `mnt/sdcard` ディレクトリーにドラッグします。

### ADB

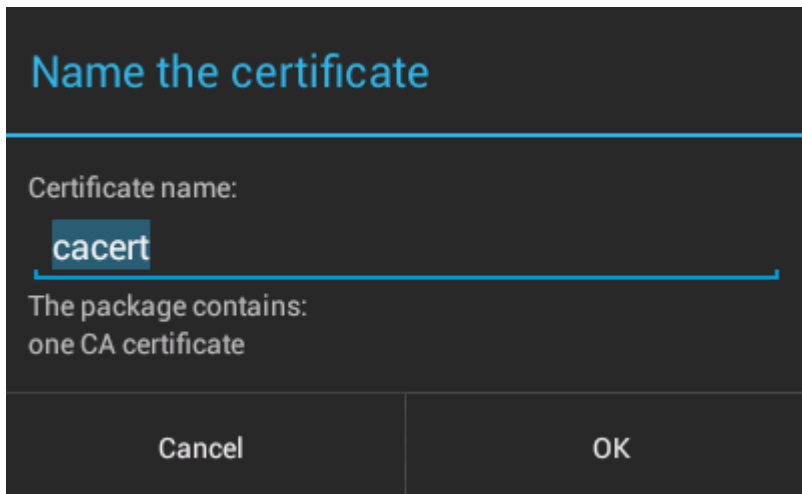
- i) コマンド・ウィンドウを開き、その現行ディレクトリーを `android` インストール・ディレクトリーの `android-sdk\platform-tools` に設定します (例: `C:\Program Files\Android\android-sdk\platform-tools`)。
- ii) 証明書を以下に示すように `mnt/sdcard` ディレクトリーにコピーします。

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

7. 証明書を Android デバイスの証明書トラストストアにインストールします。

証明書には、値が `Subject Type=CA` の基本制約節が含まれている必要があります。

- a) デバイスをアンロックして、「**ウィジェット**」ボタンをクリックします。
- b) **設定** > 「**セキュリティー**」 > 「**資格情報ストレージ**」 > **SD カードからのインストール**をクリックします。

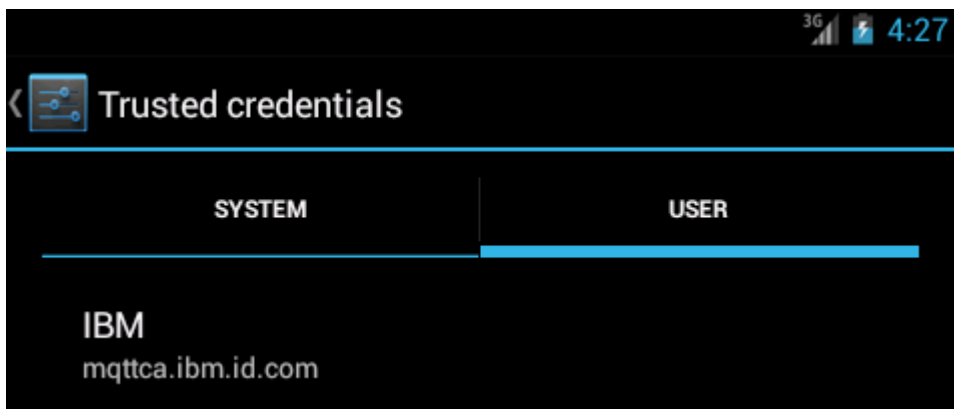


c) 証明書ファイル名が正しいことを確認して、「OK」をクリックします。

注：デバイスのロックを定義していない場合は、ここでロックを設定するように Android によってプロンプトが出されます。

8. 証明書がデバイスにインストールされていることを確認します。

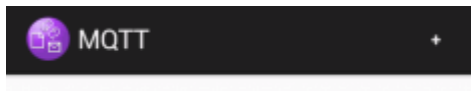
a) 「信頼された資格情報」 > 「ユーザー」 をクリックし、ユーザー証明書のリストに証明書が表示されるまで数分程度待ちます。



9. MQTTExerciser アプリケーションを再実行し、匿名 SSL クライアント用に構成した MQTT チャネルに接続します。

a) Android 用の MQTT クライアント・サンプル Java アプリケーションを開きます。

ご使用の Android デバイスでこのウィンドウが開きます。



b) MQTT サーバーに接続します。

i) + 記号をクリックして、新しい MQTT 接続を開きます。

- ii) 「クライアント ID」フィールドに、任意の固有 ID を入力します。キー・ストロークは遅いことがあるので、急がないでください。
- iii) 「サーバー」フィールドに、ご使用の MQTT サーバーの IP アドレスを入力します。  
これは、最初のメインステップで選択したサーバーです。IP アドレスを 127.0.0.1 にすることはできません。
- iv) MQTT 接続のポート番号を入力します。

スクリプト例の変数 `%sslportopt%` で設定されたポート番号を 8884 に設定します。このポート番号は、ステップ 66 ページの『4』のサンプル・スクリプトを実行して匿名 SSL クライアント用に構成した MQTT チャンネルのポート番号です。

- v) 「詳細設定」タブをクリックして、「SSL」オプションを選択します。「保存」をクリックします。
- vi) 「接続」をクリックします。

接続が成功すると、「接続中 (Connecting)」メッセージが表示され、その後に次のウィンドウが表示されます。

## タスクの結果

MQTTExciser アプリケーションが接続されて、メッセージが交換されるまでの時間が少し長くなりますが、それ以外の動作は非セキュア接続による接続と変わりません。

### 関連概念

53 ページの『MQTT セキュリティー』

MQTT セキュリティーの基本は3つの概念、すなわち、ID、認証、および許可です。IDとは許可を受けて権限を受けようとするクライアントのネーミングです。認証とはクライアントのIDを証明すること、許可とはクライアントに付与される権利の管理のことです。

## 関連タスク

### 97 ページの『鍵と証明書の生成』

以下の手順に従って、Java および C クライアント (Android と iOS アプリケーション、および IBM WebSphere MQ と IBM MessageSight サーバーを含む) 用の鍵と証明書を生成します。

### 57 ページの『セキュア MQTT クライアント・サンプル Java アプリケーションのビルドと実行』

Windows の例に基づいて、IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてセキュア・サンプル Java アプリを稼働させることができます。JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアント アプリを実行できます。

### 74 ページの『JAAS を使用した MQTT クライアント Java アプリの認証』

JAAS を使用してクライアントを認証する方法を学びます。このタスクのステップを実行して、サンプル・プログラム JAASLoginModule.java を変更し、JAAS で MQTT クライアント Java アプリケーションを認証するように IBM WebSphere MQ を構成します。

## Windows の SSL 証明書を構成するスクリプトの例

### &nbsp;

タスクの手順に示す説明に従って、サンプル・コマンド・ファイルで証明書と証明書ストアを作成します。さらに、例では、サーバー証明書ストアを使用するように MQTT クライアント・キュー・マネージャーをセットアップします。この例では、IBM WebSphere MQ で提供されている SampleMQM.bat スクリプトを呼び出すことによって、キュー・マネージャーを削除して再作成します。

### initcert.bat

initcert.bat は、**keytool** および **openSSL** コマンドが必要とする、証明書の名前とパス、およびその他のパラメーターを設定します。スクリプト内のコメントに、設定の説明があります。

```
@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 srvidname="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 chllopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

### cleancert.bat

cleancert.bat スクリプト内のコマンドは、MQTT クライアント・キュー・マネージャーを削除することにより、サーバー証明書ストアがロックされないようにします。その後、サンプル・セキュリティー・スクリプトによって作成されたすべての鍵ストアと証明書を削除します。

```
@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 %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

### genkeys.bat

genkeys.bat スクリプト内のコマンドは、プライベート認証局、サーバー、およびクライアント用の鍵ペアを作成します。

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

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

```

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

```

## sscerts.bat

sscerts.bat スクリプト内のコマンドは、クライアントとサーバーの各鍵ストアにあるそれぞれの自己署名証明書をエクスポートし、サーバー証明書をクライアント・トラストストアにインポートし、クライアント証明書をサーバーの鍵ストアにインポートします。サーバーにはトラストストアはありません。このコマンドは、クライアント JKS トラストストアから PEM 形式のクライアント・トラストストアを作成します。

```

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

```

```

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

```

```

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

```

```

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

```

## cacerts.bat

このスクリプトは、認証局ルート証明書をプライベート鍵ストアにインポートします。CA ルート証明書は、ルート証明書と署名済み証明書のための鍵チェーンを作成するために必要になります。cacerts.bat スクリプトがクライアントとサーバーの各鍵ストアからそれぞれの証明書要求をエクスポートします。スクリプトは、cajkskeystore.jks 鍵ストアにあるプライベート認証局の鍵を使



用して、証明書要求に署名します。そして、要求が来た元の同じ鍵ストアに、署名済み証明書をインポートします。このインポートによって、CA ルート証明書との証明書チェーンが作成されます。スクリプトはクライアント JKS トラストストアから、PEM 形式のクライアント・トラストストアを作成します。

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

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

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

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

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

```
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass  
%cltjkskeystorepass%
```

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

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

## mqcerts.bat

スクリプトは、証明書ディレクトリーにある鍵ストアと証明書をリストします。そして、MQTT サンプル・キュー・マネージャーを作成し、セキュア・テレメトリー・チャンネルを構成します。

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

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

## JAAS を使用した MQTT クライアント Java アプリの認証

JAAS を使用してクライアントを認証する方法を学びます。このタスクのステップを実行して、サンプル・プログラム JAASLoginModule.java を変更し、JAAS で MQTT クライアント Java アプリケーションを認証するように IBM WebSphere MQ を構成します。

### 始める前に

1. JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアント アプリを実行できます。 [IBM Mobile Messaging and M2M クライアント・パックのシステム要件](#) を参照してください。
2. クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。

3. IBM WebSphere MQ インストール済み環境で MQXR JAASLoginModule および JAASPrincipal Java のサンプルにアクセスできる必要があります。サンプルは、パス %MQ\_FILE\_PATH%\mqxr\samples. にあります。
4. Windows または Linux でステップを実行してください。例は、Windows に基づいています。
5. ステップ 75 ページの『1』を実行するには、IBM WebSphere MQ で MQXR\_SAMPLE\_QM キュー・マネージャーを作成する権限が必要です。

## このタスクについて

このタスクでは、ご使用のバージョンの JAASLoginModule から MQTT Sample クライアント識別パラメーターを出力します。クライアント・パラメーターを書き出すには、サンプル JAASLoginModule プログラムを変更し、ご使用のバージョンの JAASLoginModule をロードするように IBM WebSphere MQ を構成する必要があります。

## 手順

1. 16 ページの『Eclipse から、すべての MQTT クライアント・サンプル Java アプリケーションをコンパイルして実行する』のステップを実行して、Paho MQTT Sample クライアントを実行します。

JAAS 認証を開発してテストするための開発環境を準備することが目的です。JAAS 認証モジュールを作り替えるには、Java 開発環境が必要です。例では、Java 用のサンプル Paho クライアントを実行して JAAS 構成をテストします。簡単にするために、サンプル・クライアントとサンプル JAAS ログイン・モジュールの両方の変更には同じ開発環境を使用します。また、JAAS ログイン・モジュールは、C 用の MQTT クライアントやその他の MQTT クライアントでもテストします。

2. オプション: MQTT Paho サンプルにユーザー名およびパスワードのパラメーターを追加します。

**注:** Java 用の Paho クライアントにユーザー名およびパスワードのパラメーターが含まれている場合、このステップは不要です。ダウンロード・サイトで、アップデートがないか確認します。[IBM メッセージング・コミュニティのダウンロード](#)を参照するか、Sample.java コピーを変更します。

- a) Paho サンプル・プロジェクトの org.eclipse.paho.sample.mqttv3app パッケージでパッケージ・エクスプローラーを開きます。
- b) Sample.java を右クリックして、「コピー」 > 「貼り付け」をクリックします。「名前の競合」ウィンドウで、名前 SampleForJAAS を入力します。
- c) main メソッドに以下のコード行を追加します。

- i) "boolean ssl = false;"という行の後に、userName 変数と password 変数を宣言します。

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

古い MQTT サーバーとの互換性のため、デフォルトでパスワードおよびユーザー名のパラメーターは設定されていません。

- ii) 行 "case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;" の後で、以下の 2 つの新しい入力パラメーターを構文解析します。

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

- iii) 行 "if (action.equals("publish")) {" の前に、userName および password を Sample コンストラクター引数に追加します。

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

- d) userName および password を Sample のコンストラクターに追加します。

変更点:

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

終了:

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

e) Sample コンストラクターに以下のコード行を追加します。

"conOpt.setCleanSession(clean);"という行の後に、Sample メソッドの conOpt オブジェクトで userName 変数と password 変数を設定します。

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

f) メソッド publish と subscribe で、以下のコード行を変更します。

行 "client.connect();" を以下のように変更します。

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

3. JAAS 例のために Java プロジェクト JAASSample を作成します。

- Eclipse ワークスペースで、「**新規 Java プロジェクト**」ウィザードを開き、「**ファイル**」 > 「**新規**」 > 「**Java プロジェクト**」をクリックします。
- 「**プロジェクト名**」フィールドで、JAASSample を入力します。
- JRE のオプションで、実行環境 JRE として J2SE-1.5 を選択し、「**次へ**」をクリックします。

JRE は、IBM WebSphere MQ サーバーが実行されている JRE と一致する必要があります。IBM WebSphere MQ Version 7.5 は J2SE-1.5 を実行します。

- 「**Java 設定**」ウィンドウで、「**ライブラリー**」タブをクリックし、「**外部 JAR の追加...**」をクリックします。参照先:%MQ\_FILE\_PATH%\mqxr\lib ディレクトリーを選択し、**MQXR.jar** を選択します。「**終了**」をクリックします。
4. JAAS サンプル JAASLoginModule および JAASPrincipal クラスをインポートします。
- パッケージ・エクスプローラーで JAASSample プロジェクトを右クリックします。インポート ... > 一般 > 「**ファイル・システム**」を選択し、「**次へ**」をクリックします。
  - %MQ\_FILE\_PATH%\mqxr\samples を参照し、JAASLoginModule.java および JAASPrincipal.java にチェック・マークを付け、「**完了**」をクリックします。
  - パッケージ・エクスプローラーで両方の Java ファイルを選択して右クリックします。リファクタリング ... > **移動**。
  - 「**移動**」ウィンドウで、両方のエレメントの宛先として JAASSample が選択されていることを確認し、「**パッケージの作成...**」をクリックします。
  - 「**新規 Java パッケージ**」ウィザードの「**名前**」フィールドに samples を入力し、「**完了**」 > 「**OK**」をクリックします。

未使用の値についての多くの警告と共に、インポートされる Java クラスが Eclipse で作成されます。

5. JAASLoginModule クラスを名前変更します。

クラスの名前を変更して、IBM WebSphere MQ に付属のサンプル JAASLoginModule クラスと区別しやすくします。

- パッケージ・エクスプローラーで JAASloginModule.java を右クリックします。リファクタリング ... > **名前変更**。
- 「**コンパイル単位名の変更**」ウィンドウで、「**新規名**」フィールドを JAASloginModule から MyJAASloginModule に変更し、「**完了**」をクリックします。

6. MyJAASLoginModule クラスを変更して、コールバック・フィールドの内容を出力します。

a) 以下のコード行を MyJAASLoginModule.java に追加します。

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

ステートメントの直前に行を配置します。"if (true) loggedIn = true;"

b) CTRL+Shift+O を押して、インポートを再編成し、ファイルを保存します。

7. JAASPrincipal を MyJAASPrincipal に名前変更します。

クラスを名前変更して、サンプル JAASPrincipal クラスとの混同を避けます。例では、MyJAASPrincipal クラスの内容は変更しません。

8. キュー・マネージャー・プロセス read および execute を実行しているユーザー ID に、JAAS クラスに対する許可を付与します。

a) Windows エクスプローラーで、Eclipse ワークスペース・ディレクトリーを開きます。この例では、Eclipse ワークスペースの場所は、Eclipse 変数 *workspace\_loc* で表されています。

b) クラス MyJAASLoginModule と MyJAASPrincipal が含まれているディレクトリーを参照します。

ディレクトリー・パスは *workspace\_loc\JAASSample\bin\samples* です。

c) 両方のクラスを選択して右クリックし、「プロパティー」をクリックして、「プロパティー」ウィンドウの「セキュリティ」タブをクリックします。

d) 「追加 ...」をクリックします。オブジェクト名 *mqm* を入力し、「名前の確認」をクリックして確認します。「OK」をクリックします。

e) 「グループ名またはユーザー名」のリストで *mqm* を選択して、*mqm* の許可リストで「読み取りと実行」および「読み取り」にチェック・マークを付け、「OK」をクリックします。

9. IBM WebSphere MQ を構成して、MyJAASLoginModule クラスを実行します。

a) *service.env* ファイルを IBM WebSphere MQ 構成に追加して、MyJAASLoginModule クラスをロードするクラスパスを定義します。

以下のクラスパス・ステートメントを持つ *service.env* ファイルを *WMQ\_DATA\_PATH* ディレクトリーに作成します。

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

ここで、*user.dir* は、Eclipse ワークスペースでコンパイルされるクラス・ファイルのディレクトリー・ルートです。*WMQ\_DATA\_PATH* ディレクトリーには、*qmgrs* ディレクトリーが含まれています。[追加の環境変数を参照してください](#)。

**ヒント:** *CLASSPATH=user.dir\JAASSample\bin* は、*service.env* ファイルの唯一のステートメントになる場合があります。

b) スタンザ MyJAASStanza を *jaas.config* ファイルに追加して、MyJAASLoginModule クラスが、*service.env* ファイルのクラスパスの相対として識別されるようにします。

*jaas.config* は、キュー・マネージャー *mqxr* ディレクトリー、*WMQ\_DATA\_PATH\Qmgrs\MQXR\_SAMPLE\_QM\mqxr* にあります。

スタンザは以下のとおりです。

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

c) JAAS 構成スタンザの名前で IBM WebSphere MQ Telemetry チャネルを構成します。

コマンド・ウィンドウで以下のコマンドを実行します。

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

このコマンドによって、MyJAAS チャンネルが `jaas.config` ファイルの MyJAASStanza に結び付けられています。チャンネル定義で、MCAUSER オプションを指定しない、または USECLTID オプションを指定することによって、チャンネルは MQTT クライアント・プログラムで提供されるユーザー名でのキュー・マネージャー・リソースへのアクセスを許可します。この例では、クライアントによって提供されるユーザー名は "Guest" に設定されます。SampleMQM コマンド・ファイルで設定される Guest の既存の許可はこの例でも使用されます。

10. IBM WebSphere MQ Telemetry サービスを再始動し、新規構成データを読み取ります。  
IBM WebSphere MQ Telemetry サービスを再始動するには、キュー・マネージャーを開始するか、IBM WebSphere MQ Explorer からサービスを開始するか、またはサンプル構成用に以下のコマンドを実行します。

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

11. Sample プログラムを実行します。

SampleForJAAS の実行構成を行うには、ステップ 75 ページの『1』と同じ手順に従い、以下の点を変更します。

- a) ポート番号を 1890 に設定して MQTT チャンネル構成に一致するようにします。
- b) Sample プログラム用に作成したサブスクライバー構成とパブリッシャー構成の両方のために「(x)= 引数」タブでパスワードにパラメーター `-u Guest -z password` を追加します。

サンプル・プログラムが実行され、ポート番号が 1883 ではなく 1890 になる以外は出力に変更はありません。

WMQ\_DATA\_PATH\Qmgrs\MQXR\_SAMPLE\_QM ディレクトリーで、mqxr.stdout ファイルを開きます。MyJAASLoginModule の出力が以下のように mqxr.stdout に書き込まれています。

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

## 次のタスク

例が正常に実行されない場合は、JAAS のトラブルシューティング・トピック、188 ページの『[問題の解決: JAAS ログイン・モジュールがテレメトリー・サービスによって呼び出されない](#)』を参照して、デバッグのヒントを試行してください。

1. WMQ\_DATA\_PATH\Qmgrs\MQXR\_SAMPLE\_QM\mqxr\java.properties のパラメーターに `-verbose` を追加します。このログから、クラスが正常にロードされたかどうかわかります。  
出力は WMQ\_DATA\_PATH\Qmgrs\MQXR\_SAMPLE\_QM\mqxr.stderr に書き込まれます。
2. MyJAASLoginModule でスローされた例外がないか  
WMQ\_DATA\_PATH\Qmgrs\MQXR\_SAMPLE\_QM\errors\mqxr.log を調べます。例えば、ヌルの password を出力しようとしていた場合、これはストリングではなく文字配列であるため、例外がスローされます。
3. WMQ\_DATA\_PATH\Qmgrs\MQXR\_SAMPLE\_QM\errors\AMQERR01.log を調べます。userName のユーザー名がキュー・マネージャー・リソースへのアクセスを許可されておらず、チャンネルが MCAUSER オプションなしまたは USECLTID オプションありで構成されている場合、エラーはこのレポートに書き込まれます。

4. `WMQ_DATA_PATH\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config` 内のスタanzasの名前が、Sample クライアントが接続しようとしているポート用に構成されている MQTT チャンネル内の名前と同じであることを確認してください。
5. Stanzasのパスが Eclipse の `MyJAASLoginModule` クラスへのパスと一致していることを確認します。例えば、以下のとおりです。

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

6. `WMQ_DATA_PATH` の `service.env` ファイルのクラスパスに障害があるかどうかを除去するには、`%MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT` の `"set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%"` 行を変更して、クラスパスを含めます。クラスパスをエコー出力することもできます。ただし、クラスパスには `service.env` に設定されているクラスパスは含まれないので、これが機能するのは `controlMQXR.BAT` ファイルを変更する場合のみです。

## 関連概念

### [53 ページの『MQTT セキュリティー』](#)

MQTT セキュリティーの基本は3つの概念、すなわち、ID、認証、および許可です。IDとは許可を受けて権限を受けようとするクライアントのネーミングです。認証とはクライアントのIDを証明すること、許可とはクライアントに付与される権利の管理のことです。

### [116 ページの『テレメトリー・チャンネルの JAAS 構成』](#)

クライアントから送られた Username を認証するように JAAS を構成します。

## 関連タスク

### [188 ページの『問題の解決: JAAS ログイン・モジュールがテレメトリー・サービスによって呼び出されない』](#)

JAAS ログイン・モジュールがテレメトリー (MQXR) サービスによって呼び出されていないかどうかを突き止め、問題が修正されるように JAAS を構成します。

### [57 ページの『セキュア MQTT クライアント・サンプル Java アプリケーションのビルドと実行』](#)

Windows の例に基づいて、IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてセキュア・サンプル Java アプリを稼働させることができます。JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアント アプリを実行できます。

### [65 ページの『SSL を介した Android での MQTT クライアント・サンプル Java アプリの接続』](#)

SSL を介して IBM WebSphere MQ に接続されたサンプル Android MQTT クライアントを稼働させます。

## 関連情報

### [追加の環境変数](#)

# SSL を介した JavaScript 用の MQTT メッセージング・クライアントと WebSockets の接続

SSL を使用する JavaScript 用の MQTT メッセージング・クライアント サンプル HTML ページと WebSocket protocol を使用して、Web アプリを IBM WebSphere MQ に安全に接続します。

## 始める前に

1. WebSockets を介した MQTT protocol をサポートする MQTT version 3 サーバーへのアクセス権限が必要です。
2. ブラウザーは SSL と WebSocket protocol をサポートする必要があります。[177 ページの『SSL を介したモバイル・メッセージング Web アプリケーションのブラウザー・サポートに関する制約事項』](#) を参照してください。
3. SSL チャンネルを開始する必要があります。

## このタスクについて

SSL を介して JavaScript サンプル・ページの MQTT メッセージング・クライアントを実行するには、このタスクを完了します。このタスクでは、[97 ページの『鍵と証明書の生成』](#)に指示して証明書を作成し、IBM WebSphere MQ を構成します。

SSL チャンネルを、認証局署名鍵か自己署名鍵のどちらかを使用して保護します。

## 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。  
サーバーは、セキュアな WebSockets を介した MQTT protocol をサポートする必要があります。
  - IBM MessageSight、および IBM WebSphere MQ バージョン 7.5.0.1 以降のリリースでこれを行います。
2. オプション: バージョン 7 以降の Java Development Kit (JDK) をインストールします。

テスト・システムをセットアップし、自己署名証明書を使用する場合には、JDK バージョン 7 の **keytool** コマンドを使用して、証明書を認証する必要があります。実動システムをセットアップし、外部認証局に証明書署名要求 (CSR) を送信する場合は、バージョン 7 JDK は必要ありません。

3. 鍵ペアと証明書を生成するスクリプトを作成して実行し、IBM WebSphere MQ を MQTT サーバーとして構成します。

[97 ページの『鍵と証明書の生成』](#)のステップに従って、スクリプトを作成して実行します。これらのスクリプトのリストも [82 ページの『Windows の SSL 証明書を構成するスクリプトの例』](#)にあります。

サーバー証明書の共通名はサーバー・チャンネルの DNS 名と一致していなければなりません。一部のブラウザは共通名のリストを含む証明書を受け入れます。次に例を示します。

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

その他のブラウザで受け入れる共通名は 1 つだけです。例えば、Firefox の場合、バージョン 18 まで 1 つの共通名のみを受け入れます。後のバージョンでは、異なることがあります。

4. SSL チャンネルが実行されていて、期待どおりにセットアップされていることを確認します。

IBM WebSphere MQ のコマンド・ウィンドウに以下のコマンドを入力します。

### 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. 証明書をブラウザの証明書ストアにインストールします。  
例えば、以下のいずれかのサーバー証明書を選択します。
  - a. 自己署名サーバー証明書の場合、証明書は `svrcertselfsigned.cer` です。
  - b. プライベート認証局が署名したサーバー証明書の場合、証明書は `cacert.cer` です。
  - c. 外部認証局が署名したサーバー証明書の場合、その認証局のルート証明書が証明書ストアに既にインストール済みであることを確認します。

ブラウザで利用可能なサポートに応じて、`cacert.cer` を信頼されたルート証明機関のリストにインストールします。[177 ページの『SSL を介したモバイル・メッセージング Web アプリケーションのブラウザ・サポートに関する制約事項』](#)を参照してください。

6. オプション: クライアントを認証します。



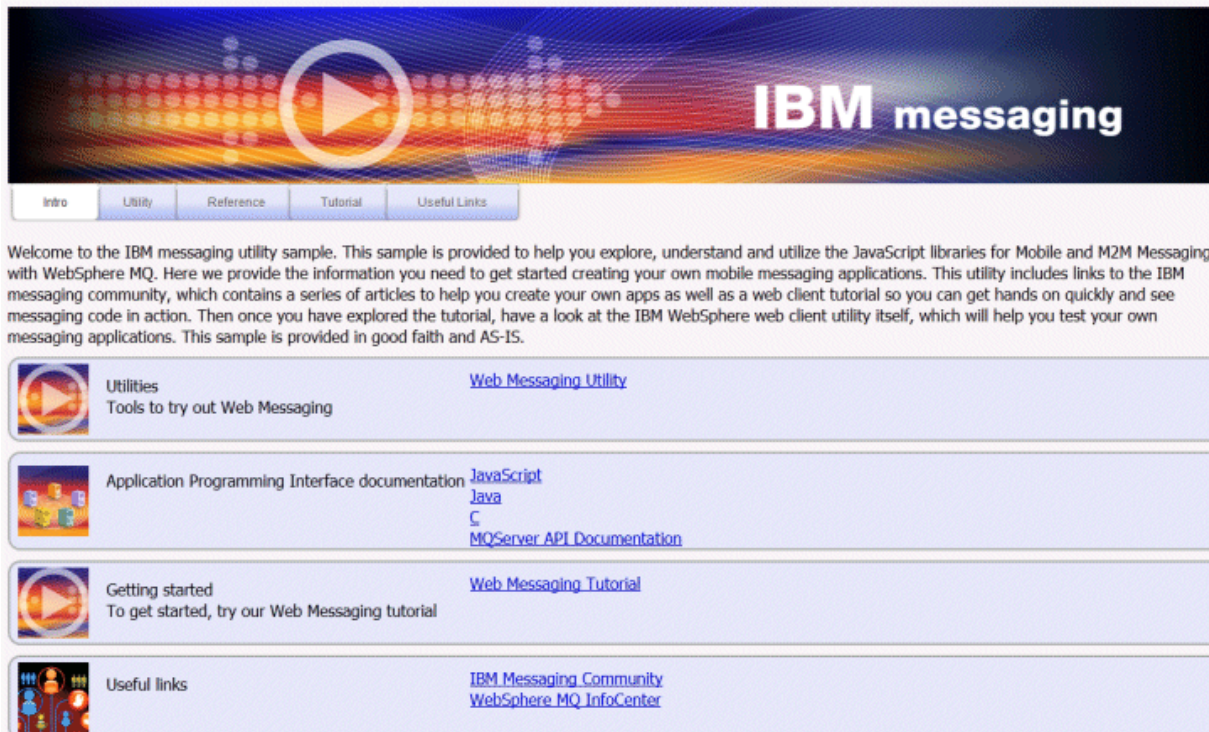
例では、cacert.cer をインストールすることにより、サーバーを認証してチャンネルを暗号化しますが、クライアントは認証しません。クライアントを認証するには、クライアント鍵ストア cltkeystore.p12 をブラウザにインストールする必要があります。すべてのブラウザがクライアントの認証をサポートするわけではありません。177 ページの『SSL を介したモバイル・メッセージング Web アプリケーションのブラウザ・サポートに関する制約事項』を参照してください。

#### 7. セキュアな WebSockets チャンネルに接続します。

ブラウザを開いて、アドレス・バーに WebSockets チャンネルの URL を入力します。以下に例を示します。

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

IBM WebSphere MQ は MQTT メッセージング・クライアント・サンプル JavaScript ・ ページの最初のページで応答します。



接続に失敗しても、スクリプト例を実行してサンプル MQTT キュー・マネージャーをセットアップしてある場合は、ポート 1886 で通常の WebSockets チャンネルへの接続を試行します。ポート 1886 で成功した場合には、SSL 接続の障害が特定されます。

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

#### 関連概念

119 ページの『JavaScript 用の MQTT メッセージング・クライアントと Web アプリケーション』

122 ページの『JavaScript でのメッセージング・アプリケーションのプログラミング方法』

#### 関連タスク

97 ページの『鍵と証明書の生成』

以下の手順に従って、Java および C クライアント (Android と iOS アプリケーション、および IBM WebSphere MQ と IBM MessageSight サーバーを含む) 用の鍵と証明書を生成します。

25 ページの『JavaScript 用の MQTT メッセージング・クライアントの概要』

JavaScript 用の MQTT メッセージング・クライアントは、メッセージング・クライアント・サンプル・ホーム・ページを表示し、リンク先のリソースを参照することで開始できます。このホーム・ページを表示するには、MQTT メッセージング・クライアント・サンプル JavaScript ・ ページからの接続を受け入れるように MQTT サーバーを構成してから、サーバーで構成した URL を Web ブラウザーに入力します。デバイス上の JavaScript 用の MQTT メッセージング・クライアントは自動的に始動し、メッセージング・クラ

イアント・サンプル・ホーム・ページが表示されます。このページには、ユーティリティー、プログラミング・インターフェース文書、チュートリアル、およびその他の役立つ情報へのリンクが含まれています。

## 関連資料

[177 ページの『SSL を介したモバイル・メッセージング Web アプリケーションのブラウザー・サポートに関する制約事項』](#)

ブラウザーの違いや、ブラウザーが稼働するプラットフォームの違いにより、機能に違いが生じます。これらの違いを理解することで、JavaScript 用の MQTT メッセージング・クライアント over SSL および WebSockets を使用して接続するようにアプリケーション、認証局 (CA)、およびクライアント証明書を構成することができます。

## Windows の SSL 証明書を構成するスクリプトの例

### &nbsp;

タスクの手順に示す説明に従って、サンプル・コマンド・ファイルで証明書と証明書ストアを作成します。さらに、例では、サーバー証明書ストアを使用するように MQTT クライアント・キュー・マネージャーをセットアップします。この例では、IBM WebSphere MQ で提供されている SampleMQM.bat スクリプトを呼び出すことによって、キュー・マネージャーを削除して再作成します。

### initcert.bat

initcert.bat は、**keytool** および **openSSL** コマンドが必要とする、証明書の名前とパス、およびその他のパラメーターを設定します。スクリプト内のコメントに、設定の説明があります。

```
@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 srvidname="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 srvcertassigned=%certpath%\srvcertassigned.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 cltcertassigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer

```

```

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

```

```

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

```

```

@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'

```

```

set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log

```

## cleancert.bat

cleancert.bat スクリプト内のコマンドは、MQTT クライアント・キュー・マネージャーを削除することにより、サーバー証明書ストアがロックされないようにします。その後、サンプル・セキュリティー・スクリプトによって作成されたすべての鍵ストアと証明書を削除します。

```

@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 %svrcertreq%
erase %svrcertcasigned%
erase %svrcertsselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertsselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

## genkeys.bat

genkeys.bat スクリプト内のコマンドは、プライベート認証局、サーバー、およびクライアント用の鍵ペアを作成します。

```

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

```

```

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

```

```

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

```

## sscerts.bat

sscerts.bat スクリプト内のコマンドは、クライアントとサーバーの各鍵ストアにあるそれぞれの自己署名証明書をエクスポートし、サーバー証明書をクライアント・トラストストアにインポートし、クライアント証明書をサーバーの鍵ストアにインポートします。サーバーにはトラストストアはありません。このコマンドは、クライアント JKS トラストストアから PEM 形式のクライアント・トラストストアを作成します。

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

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

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

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

## cacerts.bat

このスクリプトは、認証局ルート証明書をプライベート鍵ストアにインポートします。CA ルート証明書は、ルート証明書と署名済み証明書のための鍵チェーンを作成するために必要になります。

cacerts.bat スクリプトがクライアントとサーバーの各鍵ストアからそれぞれの証明書要求をエクスポートします。スクリプトは、cajkskeystore.jks 鍵ストアにあるプライベート認証局の鍵を使用して、証明書要求に署名します。そして、要求が来た元の同じ鍵ストアに、署名済み証明書をインポートします。このインポートによって、CA ルート証明書との証明書チェーンが作成されます。スク

リプトはクライアント JKS トラストストアから、PEM 形式のクライアント・トラストストアを作成します。

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

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

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

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

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

```

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

```

```

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

```

## mqcerts.bat

スクリプトは、証明書ディレクトリーにある鍵ストアと証明書をリストします。そして、MQTT サンプル・キュー・マネージャーを作成し、セキュア・テレメトリー・チャンネルを構成します。

```

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

```

```

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

```

## セキュア MQTT クライアント・サンプル C アプリケーションのビルドと実行

Windows の例に基づいて、C ソースをコンパイルできる任意のオペレーティング・システムでセキュア・サンプル C アプリケーションを起動して稼働状態にすることができます。IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてで、サンプル C アプリケーションを実行できることを確認します。

### 始める前に

1. MQTT protocol over SSL をサポートする MQTT version 3.1 サーバーにアクセスできる必要があります。
2. クライアントとサーバーの間にファイアウォールがある場合は、MQTT トラフィックをブロックしていないことを確認してください。
3. C クライアント・ライブラリーのバイナリー・バージョンが、複数のオペレーティング・システム用に提供されています。これらのオペレーティング・システムの一部では、クライアントのセキュア・バージョンはバイナリー・ファイルとしては提供されません。そのようなオペレーティング・システムでは、31 ページの『C 用の MQTT クライアント・ライブラリーのビルド』にある指示に従ってください。

4. 問題を解決するために、IBM サポートはユーザーに対して、参照プラットフォームでの C 用 MQTT クライアントの実行を求める場合があります。
5. SSL チャンネルを開始する必要があります。

サポートされるプラットフォームと参照プラットフォームの概要については、[IBM Mobile Messaging and M2M クライアント・パックのシステム要件](#)を参照してください。C クライアントでのサポート内容について詳しくは、[System Requirements for WebSphere MQ V7.5 Telemetry](#) の関連セクションを参照してください。

## このタスクについて

例として、この記事では、コマンド行からセキュア MQTT クライアント・サンプル C アプリケーション on Windows をコンパイルして実行する方法を示します。この図では、Microsoft Visual Studio 2010 がクライアントのコンパイルに使用されています。Linux や iOS など、他のオペレーティング・システムでサンプル・アプリケーションをコンパイルして実行する場合は、コマンド行スクリプトを変更できます。

### 注:

この記事に示されている Windows スクリプトでは、ソースから OpenSSL パッケージ全体を作成することを前提としています。IBM から提供されるプリコンパイル・ライブラリーを使用するように選択した場合は、OpenSSL のプリコンパイル・バイナリー・リリースも取得できます。プリコンパイル・ライブラリーを iOS で使用することはできません。

SSL チャンネルを、認証局署名鍵か自己署名鍵のどちらかを使用して保護します。

## 手順

1. クライアント・アプリケーションを接続できる MQTT サーバーを選択します。

サーバーは、SSL を介した MQTT version 3.1 プロトコルをサポートする必要があります。IBM のすべての MQTT サーバー (IBM WebSphere MQ および IBM MessageSight を含む) がこれを行います。139 ページの『[MQTT サーバーの概要](#)』を参照してください。
2. オプション: Java Development Kit (JDK) バージョン 7 以降をインストールします。

**keytool** コマンドを実行して証明書を認証するには、バージョン 7 が必要です。証明書の認証を行わない場合、バージョン 7 JDK は必要ありません。
3. ビルド場所のプラットフォームに C 開発環境をインストールします。

このトピックの例にある Make ファイルは、以下のツールをターゲットとしています。

  - **iOS** iOS の場合、OS X 10.8.2 の Apple Mac で、[Xcode](#) の iOS 開発ツールを使用。
  - **Linux** Linux の場合、Red Hat Enterprise Linux バージョン 6.2 の gcc バージョン 4.4.6。サポートされる最小レベルは、glibc C ライブラリーが 2.12、Linux カーネルが 2.6.32 です。
  - **Windows** Microsoft Windows の場合、Visual Studio バージョン 10.0。
4. Mobile Messaging and M2M クライアント・パック をダウンロードし、MQTT SDK をインストールします。

インストール・プログラムは用意されておらず、ダウンロードしたファイルを展開するだけです。

  - a. [Mobile Messaging and M2M クライアント・パック](#) をダウンロードします。
  - b. SDK をインストールするフォルダーを作成します。

フォルダーの名前を MQTT にすることもできます。ここでは、このフォルダーへのパスを *sdkroot* として示します。
  - c. 圧縮された Mobile Messaging and M2M クライアント・パック ファイルの内容を *sdkroot* に展開します。展開すると、*sdkroot\SDK* から始まるディレクトリー・ツリーが作成されます。
5. オプション: [31 ページの『C 用の MQTT クライアント・ライブラリーのビルド』](#) のステップに従います。



このステップは、MQTT SDK に、ターゲットのオペレーティング・システム用セキュア C クライアント・ライブラリーが含まれていない場合にのみ実行してください。

- **Windows** ライブラリーは、コンパイル用は `mqttv3cs.lib`、実行用は `mqttv3cs.dll` となります。
- **Linux** ライブラリーは `libmqttv3cs.so` です。
- **iOS** ライブラリーは `libmqttv3cs.a` です。

6. 鍵ペアと証明書を生成するスクリプトを作成して実行し、IBM WebSphere MQ を MQTT サーバーとして構成します。

97 ページの『鍵と証明書の生成』のステップに従って、スクリプトを作成して実行します。これらのスクリプトのリストも 91 ページの『Windows の SSL 証明書を構成するスクリプトの例』にあります。

7. SSL チャンネルが実行されていて、期待どおりにセットアップされていることを確認します。

IBM WebSphere MQ のコマンド・ウィンドウに以下のコマンドを入力します。

- **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. スクリプトを作成し、セキュア MQTT クライアント・サンプル C アプリケーションをビルドして実行します。
  - a) `sscclient.bat` を作成して実行し、自己署名証明書で保護された SSL チャンネルをテストします。
  - b) `cacclient.bat` を作成して実行し、認証局署名証明書で保護された SSL チャンネルをテストします。

## タスクの結果

結果は非セキュア・クライアントを実行した場合と同様です。

```
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
```

図 16. セキュア・サブスクライバー

```

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 . . .

```

図 17. セキュア・パブリッシャー

### セキュア MQTT クライアント・サンプル C アプリケーションを実行するスクリプト

ここで示すスクリプトを実行する前に、[91 ページの『Windows の SSL 証明書を構成するスクリプトの例』](#)にあるスクリプトを実行します。

#### 自己署名証明書を使用するセキュア MQTT クライアント・サンプル C アプリケーション。

このスクリプトは、[sscerts.bat](#) スクリプトを実行して作成した自己署名証明書を使用して実行します。

```

@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl /nologo /D "WIN32" /I ".\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal

```

図 18. *ssclient.bat*

#### 認証局署名証明書を使用した MQTT セキュア・クライアント・サンプル C アプリケーションの実行。

このスクリプトは、[cacerts.bat](#) スクリプトを実行して作成した認証局署名証明書を使用して実行します。

```

@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl /nologo /D "WIN32" /I "..\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal

```

図 19. cacclient.bat

## 関連概念

53 ページの『MQTT セキュリティー』

MQTT セキュリティーの基本は3つの概念、すなわち、ID、認証、および許可です。IDとは許可を受けて権限を受けようとするクライアントのネーミングです。認証とはクライアントのIDを証明すること、許可とはクライアントに付与される権利の管理のことです。

## 関連タスク

97 ページの『鍵と証明書の生成』

以下の手順に従って、Java および C クライアント (Android と iOS アプリケーション、および IBM WebSphere MQ と IBM MessageSight サーバーを含む) 用の鍵と証明書を生成します。

## Windows の SSL 証明書を構成するスクリプトの例

### 例

タスクの手順に示す説明に従って、サンプル・コマンド・ファイルで証明書と証明書ストアを作成します。さらに、例では、サーバー証明書ストアを使用するように MQTT クライアント・キュー・マネージャーをセットアップします。この例では、IBM WebSphere MQ で提供されている SampleMQM.bat スクリプトを呼び出すことによって、キュー・マネージャーを削除して再作成します。

### initcert.bat

initcert.bat は、**keytool** および **openssl** コマンドが必要とする、証明書の名前とパス、およびその他のパラメーターを設定します。スクリプト内のコメントに、設定の説明があります。

```

@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=mqtta.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=mqttsrver.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 svcertreq=%certpath%\svcertreq.csr
set svcertcasigned=%certpath%\svcertcasigned.cer
set svcertselfsigned=%certpath%\svcertselfsigned.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=mqtclient.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%\cltcertcasigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```

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

```

```

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

```

```

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

```

## cleancert.bat

cleancert.bat スクリプト内のコマンドは、MQTT クライアント・キュー・マネージャーを削除することにより、サーバー証明書ストアがロックされないようにします。その後、サンプル・セキュリティー・スクリプトによって作成されたすべての鍵ストアと証明書を削除します。

```

@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 %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%

```

```

erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

## genkeys.bat

genkeys.bat スクリプト内のコマンドは、プライベート認証局、サーバー、およびクライアント用の鍵ペアを作成します。

```

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

```

```

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

```

```

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

```

## sscerts.bat

sscerts.bat スクリプト内のコマンドは、クライアントとサーバーの各鍵ストアにあるそれぞれの自己署名証明書をエクスポートし、サーバー証明書をクライアント・トラストストアにインポートし、クライアント証明書をサーバーの鍵ストアにインポートします。サーバーにはトラストストアはありません。このコマンドは、クライアント JKS トラストストアから PEM 形式のクライアント・トラストストアを作成します。

```

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

```

```

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

```

```

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

```

```
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

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

## cacerts.bat

このスクリプトは、認証局ルート証明書をプライベート鍵ストアにインポートします。CA ルート証明書は、ルート証明書と署名済み証明書のための鍵チェーンを作成するために必要になります。

**cacerts.bat** スクリプトがクライアントとサーバーの各鍵ストアからそれぞれの証明書要求をエクスポートします。スクリプトは、**cajkskeystore.jks** 鍵ストアにあるプライベート認証局の鍵を使用して、証明書要求に署名します。そして、要求が来た元の同じ鍵ストアに、署名済み証明書をインポートします。このインポートによって、CA ルート証明書との証明書チェーンが作成されます。スクリプトはクライアント JKS トラストストアから、PEM 形式のクライアント・トラストストアを作成します。

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

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

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

```

@rem
@echo -----
@echo Sign certificate requests: %srcertcasigned% 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 %srcertreq% -outfile %srcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%

```

```

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

```

```

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

```

```

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

```

## mqcerts.bat

スクリプトは、証明書ディレクトリーにある鍵ストアと証明書をリストします。そして、MQTT サンプル・キュー・マネージャーを作成し、セキュア・テレメトリー・チャンネルを構成します。

```

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

```

```

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

```

**V7.5.0.1**



```

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

```

## 鍵と証明書の生成

以下の手順に従って、Java および C クライアント (Android と iOS アプリケーション、および IBM WebSphere MQ と IBM MessageSight サーバーを含む) 用の鍵と証明書を生成します。

### 始める前に

1. **keytool** コマンドのコピーを持っている必要があります。 **keytool** のすべてのバージョンが、Java 鍵ストア (JKS) から Public Key Cryptographic System (PKCS) への鍵ストアの変換、または証明書要求の署名をサポートしているわけではありません。 サンプルでは JDK バージョン 7.0 の **keytool** コマンドを使用します。これは、これら両方の機能をサポートしています。
2. C クライアント用の鍵と証明書 (Privacy-Enhanced Mail (PEM) 形式) を生成する予定の場合は、**openssl** コマンドのコピーが必要です。 31 ページの『C 用の MQTT クライアント・ライブラリーのビルド』に示した手順に従って openssl パッケージをビルドしてください。
3. 要件に合わせて **initcert.bat** スクリプト内のパラメーター値を変更します。特に、パスワードを書き込むことを避けるため、パスワード・パラメーターを省略することを選択するかもしれません。 **keytool** コマンドから、欠落しているパスワードを入力するよう、プロンプトが出されます。

### このタスクについて

MQTT クライアントとサーバーの間のセキュア SSL 接続を作成するには、鍵と証明書が必要です。このタスクでは、必要な鍵と証明書を作成する 2 つの異なる方法、すなわち自己署名による方法と、独自の認証局の署名による方法を示します。方式の選択は、鍵ストアと証明書をどのように管理することを計画しているかに依存します。

外部の認証局によって署名された証明書を使用するには、**cacerts.bat** の署名手順を、外部認証局への証明書要求送信に置き換えます。認証局は、署名済み証明書に加え、中間証明書やルート証明書を返す場合があります。返された証明書をどこにインストールするかに関して、外部 CA から提供される指示に従ってください。

IBM WebSphere MQ サーバーは、遠隔測定チャネル構成パラメーターに指定した証明書ストアのみから証明書を検索します。JSE **cacerts** ストアでの追加検索は行いません。Java クライアントは、指定したトラストストアから証明書を検索します。トラストストアを指定しない場合、JSE **jre\lib\security** ディレクトリー内の **cacerts** 鍵ストアで検索されます。Android クライアントは、Android デバイスの事前定義証明書ストアにある証明書を検索します。C クライアント・アプリケーションと iOS アプリは、アプリケーションが指定した証明書ストアのみを検索します。

Android および Java のクライアントは、事前構成されたトラストストアから信頼証明書を検索します。CA ルート証明書は、Android トラステッド証明書ストアおよび JSE **jre\lib\security\cacerts** ストアに保管されます。サーバー証明書を認証した CA のルート証明書が既に事前構成トラストストアにインストールされている場合、クライアントのトラストストアは定義しないでください。必要な構成は、セキュア MQTT サーバー・チャネルの TCP/IP ポートを設定することだけです。

鍵と証明書の作成と、さまざまな形式すべてを管理するツールは、使用が単純ではありません。そのようなツールには管理用パラメーターが数多くあり、**openssl** では構成ファイル **openssl.cnf** とコマンド行パラメーターが必要です。C と Java の両方で実行されるアプリケーション用の鍵と証明書の管理に必要なすべての機能を提供するツールは存在しません。IBM WebSphere MQ の遠隔測定チャネルでは JKS 鍵ストアが必要なため、サンプルでは主に Java 証明書ツールである **ikeyman** と **keytool** を使用します。しかし、Java のツールでは、C クライアント・アプリケーションに必要な PEM 形式をサポートしません。PEM 形式の鍵ストアを作成するには、**openssl** ツールを実行します。**openssl** ツールは鍵ストアを

PKCS12 形式から PEM 形式に変換し、**keytool** は鍵ストアの JKS 形式と PKCS12 形式との間の変換を行います。鍵ストアの変換には `openssl.cnf` ファイルは必要ありません。C クライアント・アプリケーションまたは iOS アプリをビルドする予定の場合、**openSSL** のみが必要です。**openSSL** で作業する場合は、**keytool** を使用して証明書に署名する代わりに、これを使用して証明書に署名することができます。

## 手順

1. コマンド・ウィンドウを開いて、以下のスクリプトを実行します。
2. `initcert.bat` スクリプトを作成して実行し、MQTT セキュア・サンプル・クライアントを実行するために必要なパラメーターを設定します。
3. `cleancert.bat` スクリプトを作成して実行し、鍵ストアと証明書を新規作成する準備のできた環境をクリアします。
4. `genkeys.bat` スクリプトを作成して実行し、必要な鍵ペアを生成します。
5. 以下のオプションのいずれかを実行します。
  - `sscerts.bat` スクリプトを作成して実行し、自己署名証明書を作成します。
  - `cacerts.bat` スクリプトを作成して実行し、認証局署名証明書チェーンを作成します。
6. `mqcerts.bat` スクリプトを作成して実行し、MQXR\_SAMPLE\_QM キュー・マネージャーを作成し、その遠隔測定チャンネルを構成します。

## 関連タスク

87 ページの『セキュア MQTT クライアント・サンプル C アプリケーションのビルドと実行』  
Windows の例に基づいて、C ソースをコンパイルできる任意のオペレーティング・システムでセキュア・サンプル C アプリケーションを起動して稼働状態にすることができます。IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてで、サンプル C アプリケーションを実行できることを確認します。

57 ページの『セキュア MQTT クライアント・サンプル Java アプリケーションのビルドと実行』  
Windows の例に基づいて、IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてでセキュア・サンプル Java アプリを稼働させることができます。JSE 1.5 以上の "互換性 (Java C)" を使用して、任意のプラットフォームで Java 用の MQTT クライアント アプリを実行できます。

## Windows の SSL 証明書を構成するスクリプトの例

### 例

タスクの手順に示す説明に従って、サンプル・コマンド・ファイルで証明書と証明書ストアを作成します。さらに、例では、サーバー証明書ストアを使用するように MQTT クライアント・キュー・マネージャーをセットアップします。この例では、IBM WebSphere MQ で提供されている `SampleMQM.bat` スクリプトを呼び出すことによって、キュー・マネージャーを削除して再作成します。

### `initcert.bat`

`initcert.bat` は、**keytool** および **openSSL** コマンドが必要とする、証明書の名前とパス、およびその他のパラメーターを設定します。スクリプト内のコメントに、設定の説明があります。

```
@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%\cltcertcasigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
```

```

@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%

```

```

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

```

```

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

```

## cleancert.bat

cleancert.bat スクリプト内のコマンドは、MQTT クライアント・キュー・マネージャーを削除することにより、サーバー証明書ストアがロックされないようにします。その後、サンプル・セキュリティー・スクリプトによって作成されたすべての鍵ストアと証明書を削除します。

```

@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 %svrcertreq%
erase %svrcertcasigned%
erase %svrcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b

```

## genkeys.bat

genkeys.bat スクリプト内のコマンドは、プライベート認証局、サーバー、およびクライアント用の鍵ペアを作成します。

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

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

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

## sscerts.bat

sscerts.bat スクリプト内のコマンドは、クライアントとサーバーの各鍵ストアにあるそれぞれの自己署名証明書をエクスポートし、サーバー証明書をクライアント・トラストストアにインポートし、クライアント証明書をサーバーの鍵ストアにインポートします。サーバーにはトラストストアはありません。このコマンドは、クライアント JKS トラストストアから PEM 形式のクライアント・トラストストアを作成します。

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

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

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

```
@rem
@echo -----
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
```

```

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

```

## cacerts.bat

このスクリプトは、認証局ルート証明書をプライベート鍵ストアにインポートします。CA ルート証明書は、ルート証明書と署名済み証明書の間の鍵チェーンを作成するために必要になります。

**cacerts.bat** スクリプトがクライアントとサーバーの各鍵ストアからそれぞれの証明書要求をエクスポートします。スクリプトは、**cajkskeystore.jks** 鍵ストアにあるプライベート認証局の鍵を使用して、証明書要求に署名します。そして、要求が来た元の同じ鍵ストアに、署名済み証明書をインポートします。このインポートによって、CA ルート証明書との証明書チェーンが作成されます。スクリプトはクライアント JKS トラストストアから、PEM 形式のクライアント・トラストストアを作成します。

```

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

```

```

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

```

```

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

```

```

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

```

```
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%  
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

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

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

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

## mqcerts.bat

スクリプトは、証明書ディレクトリーにある鍵ストアと証明書をリストします。そして、MQTT サンプル・キュー・マネージャーを作成し、セキュア・テレメトリー・チャンネルを構成します。

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

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

## MQTT クライアントの識別、許可、および認証

テレメトリー (MQXR) サービスは、MQTT チャンネルを使用して MQTT クライアントの代わりに WebSphere MQ トピックをパブリッシュしたり、それにサブスクライブしたりします。WebSphere MQ 管理者は、WebSphere MQ の許可に使用される MQTT チャンネル ID を構成します。管理者はチャンネルの共通 ID を定義すること、あるいはチャンネルに接続したクライアントの Username または ClientIdentifier を使用することができます。

テレメトリー (MQXR) サービスは、クライアントが提供する Username を使用して、またはクライアント証明書を使用して、クライアントを認証できます。Username は、クライアントが提供するパスワードを使用して認証されます。

要約すると、クライアント ID はクライアントの識別要素のセレクションです。コンテキストに応じて、クライアントは ClientIdentifier、Username、管理者が作成した共通クライアント ID、またはクライアント証明書によって識別されます。認証検査に使用されるクライアント ID は、許可に使用されるものと同じ ID である必要はありません。

MQTT クライアント・プログラムは、MQTT チャンネルを使用してサーバーに送信される Username および Password を設定します。それらは接続の暗号化および認証に必要な SSL プロパティも設定できます。管理者は、MQTT チャンネルを認証するかどうか、およびチャンネルを認証する方法を決めます。

IBM WebSphere MQ オブジェクトにアクセスする MQTT クライアントを許可するには、クライアントの ClientIdentifier または Username を許可するか、または共通クライアント ID を許可します。クライアントが IBM WebSphere MQ に接続することを許可するには、Username を認証するか、クライアント証明書を使用します。Username を認証するように JAAS を構成し、クライアント証明書を認証するように SSL を構成します。

Password をクライアントで設定する場合、VPN を使用して接続を暗号化するか、または SSL を使用するように MQTT チャンネルを構成して、パスワードを秘密に保ちます。

クライアント証明書を管理することは困難です。そのため、パスワード認証に関連したリスクが許容可能であれば、パスワード認証がクライアントの認証にしばしば使用されます。

クライアント証明書を管理および保管するためのセキュアな方法があれば、証明書の認証に依存することができます。ただし、テレメトリーが使用されるタイプの環境で、証明書をセキュアに管理できることは稀です。その代わりに、クライアント証明書を使用する装置の認証はサーバーでのクライアント・パスワードの認証によって補完されます。クライアント証明書の使用はより複雑なものとなるので、機密性の高いアプリケーションに限定されます。2つの方式を使用する認証は、2因子認証と呼ばれます。一方の因子(パスワードなど)を知っていると共に、他方の因子(証明書など)を所持している必要があります。

chip-and-pin 装置などの機密性の高いアプリケーションでは、内部のハードウェアおよびソフトウェアが改ざんされないように、製造の際に装置がロックされます。信頼できる、時間の限定されたクライアント証明書が装置にコピーされます。装置は使用される場所にデプロイされます。装置が使用されるたびに、パスワードまたはスマート・カードからの別の証明書を使用して追加の認証が行われます。

### MQTT クライアントの ID および許可

ClientIdentifier、Username、または共通クライアント ID を使用して、WebSphere MQ オブジェクトにアクセスする許可を得ます。

IBM WebSphere MQ 管理者は、3つの選択肢から MQTT チャンネルの ID を選択できます。管理者はクライアントが使用する MQTT チャンネルを定義または変更するときに選択を行います。ID は、IBM WebSphere MQ トピックへのアクセスを許可するために使用されます。選択肢は以下のとおりです。

1. クライアント ID。
2. 管理者がチャンネルに提供する ID。
3. MQTT クライアントから渡される Username。

Username は、MqttConnectOptions クラスの属性です。これはクライアントがサービスに接続する前に設定する必要があります。そのデフォルト値は NULL です。



IBM WebSphere MQ の **setmqaut** コマンドを使用して、MQTT チャンネルに関連付けられた ID が使用を許可されるオブジェクトおよびアクションを選択します。例えば、キュー・マネージャー QM1 の管理者によって提供されるチャンネル ID である MQTTClient を許可するには、次のようにします。

```
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
```

## 関連情報

[MQTT クライアントに WebSphere MQ オブジェクトへのアクセス権限を付与する](#)

## パスワードを使用する MQTT クライアントの認証

クライアント・パスワードを使用して Username を認証します。トピックのパブリッシュおよびサブスクライブをクライアントに許可するために使用した ID とは別の ID を使用して、クライアントを認証できます。

テレメトリー (MQXR) サービスは、JAAS を使用してクライアント Username を認証します。JAAS は MQTT クライアントによって提供される Password を使用します。

IBM WebSphere MQ 管理者は、クライアントが接続する MQTT チャンネルを構成して、Username を認証するかまたはまったく認証しないかを決めます。クライアントを別のチャンネルに割り当てたり、各チャンネルを別の方法でクライアント認証するように構成したりできます。JAAS を使用して、クライアントを認証する必要のあるメソッド、およびオプションでクライアントを認証できるメソッドを構成できます。

認証のための ID の選択は、許可のための ID の選択に影響を与えません。管理に役立つように許可のための共通 ID をセットアップすることができますが、その場合には、その ID を使用するように各ユーザーを認証することになります。以下の手順は、共通 ID を使用するように個別のユーザーを認証する方法の概要を示しています。

1. IBM WebSphere MQ 管理者は、IBM WebSphere MQ エクスプローラーを使用して、MQTT チャンネル ID を任意の名前 (MQTTClientUser など) に設定します。
2. IBM WebSphere MQ 管理者は、MQTTClient が任意のトピックにパブリッシュおよびサブスクライブすることを許可します。

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

3. MQTT クライアント・アプリケーション開発者は、MqttConnectOptions オブジェクトを作成して、サーバーに接続する前に Username および Password を設定します。
4. セキュリティー開発者は JAAS LoginModule を作成して、Username を Password によって認証し、それを JAAS 構成ファイルに組み込みます。
5. IBM WebSphere MQ 管理者は、JAAS を使用するクライアントの UserName を認証するように MQTT チャンネルを構成します。

## SSL を使用した MQTT クライアント認証

MQTT クライアントとキュー・マネージャーとの間の接続は、常に MQTT クライアントによって開始されます。MQTT クライアントは常に SSL クライアントです。サーバーのクライアント認証および MQTT クライアントのサーバー認証は、どちらもオプションです。

クライアントに秘密署名付きデジタル証明書を提供することで、MQTT クライアントを IBM WebSphere MQ に対して認証できます。IBM WebSphere MQ 管理者は、MQTT クライアントが SSL を使用してキュー・マネージャーに対して認証するように強制できます。クライアント認証は、相互認証の一部としてのみ要求できます。

SSL を使用する代わりに、例えば IPsec など、いくつかの種類の仮想プライベート・ネットワーク (VPN) は TCP/IP 接続のエンドポイントを認証します。VPN は、ネットワーク上を流れる各 IP パケットを暗号化します。そのような VPN 接続が確立されると、トラステッド・ネットワークが確立されます。VPN ネットワーク上で TCP/IP を使用して、MQTT クライアントをテレメトリー・チャンネルに接続できます。

SSL を使用するクライアント認証は、機密事項を持つクライアントに依存します。機密事項は、クライアントの秘密鍵(自己署名証明書の場合)または認証局によって提供される鍵です。この鍵は、クライアントのデジタル証明書を署名するために使用されます。対応する公開鍵を持つユーザーであれば、デジタル証明書を検証できます。証明書は信頼できます。あるいはチェーンされた証明書の場合には、トラステッド・ルート証明書に至るまで証明書チェーンをトレースバックしてください。クライアント検査は、クライアントによって提供される証明書チェーン内のすべての証明書をサーバーに送ります。サーバーは信頼できる証明書が見つかるまで証明書チェーンを検査します。信頼できる証明書は、自己署名証明書から生成されるパブリック証明書、または通常は認証局で発行されるルート証明書のいずれかです。オプションの最終ステップとして、信頼できる証明書を「現時点で有効な」証明書失効リストと比較できます。

信頼できる証明書は、認証局によって発行されており、JRE 証明書ストアに既に含まれている場合があります。これは自己署名証明書、またはテレメトリー・チャンネルの鍵ストアに信頼できる証明書として追加されたいずれかの証明書である場合があります。

**注:** テレメトリー・チャンネルには、1つ以上のテレメトリー・チャンネルに対する秘密鍵と、クライアントの認証に必要なパブリック証明書の両方を保持する、結合された鍵ストア/トラストストアが含まれています。SSL チャンネルには鍵ストアが含まれている必要があり、鍵ストアはチャンネルのトラストストアと同じファイルであるため、JRE 証明書ストアが参照されることはありません。これは、クライアントの認証で CA ルート証明書が必要な場合、CA ルート証明書が JRE 証明書ストアに既に存在する場合でも、ルート証明書をチャンネルの鍵ストアに配置する必要があることを意味します。JRE 証明書ストアが参照されることはありません。

クライアント認証が対抗する予定の危険、およびその危険に対抗するためのクライアントとサーバーの役割について考えてください。クライアント証明書を認証するだけでは、システムに対する無許可アクセスを防止するために不十分です。他人がクライアント装置を操作するようになった場合、そのクライアント装置は証明書の所有者の権限で動作しているとは限りなくなります。望まれない攻撃に対抗するには、単一の防御だけに依存しないでください。少なくとも 2 因子認証アプローチを使用して、証明書を所有しているかどうかを秘密情報の知識があるかどうかで補足します。例えば、JAAS を使用すると共に、サーバーから発行されたパスワードを使用してクライアントを認証します。

クライアント証明書に関する第 1 の危険は、それが他人の手に渡ってしまうことです。証明書は、クライアントにあるパスワードで保護された鍵ストアに保管されています。それはどのような方法で鍵ストアに入れますか。MQTT クライアントはどのようにしてパスワードを鍵ストアに入れますか。パスワード保護はどれほどセキュアですか。テレメトリー装置は簡単に取り外せることが多く、人目につかない場所でのハッキングが可能になります。装置のハードウェアを不正な改造から保護する必要がありますか。クライアント・サイドの証明書を配布して保護することは困難であることが知られています。これは鍵管理の問題と呼ばれます。

2 次的な危険は、意図されない方法でサーバーにアクセスするために装置が誤用されることです。例えば、MQTT アプリケーションが不正に改造された場合、認証されたクライアント ID を使用してサーバー構成の弱点を利用できるようになる可能性があります。

SSL を使用して MQTT クライアントを認証するには、テレメトリー・チャンネルおよびクライアントを構成します。

- 
- 

## SSL を使用したクライアント認証のための MQTT クライアントの構成

SSL を使用する MQTT クライアントを認証するには、クライアントは SSL を使用して遠隔測定チャンネルに接続します。クライアントは SSL クライアントを認証するように構成された遠隔測定チャンネルに対応する TCP ポートを指定する必要があります。

例えば、クライアントで以下のようにします。

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

クライアント JVM は、JSSE からの標準ソケット・ファクトリーを使用する必要があります。Java ME を使用している場合、JSSE パッケージがロードされていることを確認する必要があります。Java SE を使用している場合は、Java バージョン 1.4.1 以降、JSSE が JRE に組み込まれています。

SSL 接続では、接続の前にいくつかの SSL プロパティを設定する必要があります。プロパティを設定するには、`-D` スイッチを使用してプロパティを JVM に渡すか、または `MqttConnectionOptions.setSSLProperties` メソッドを使用してプロパティを設定します。

メソッド `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)` を呼び出すことによって非標準ソケット・ファクトリーをロードする場合、SSL 設定がネットワーク・ソケットに渡される方法はアプリケーションによって定義されます。

クライアントの秘密鍵を使用して署名された、または CA により署名された、クライアントのデジタル証明書を、クライアント上のパスワードで保護された鍵ストアに追加します。証明書に鍵チェーンがある場合、その鍵チェーンからの証明書をストアに追加できます。サーバーは、クライアント証明書を検証するときに、クライアントから送信された証明書を使用して、サーバーの鍵ストア内の証明書と突き合わせます。サーバーは鍵チェーンの中で、所有する証明書を持つ最初の一致を検索します。鍵チェーンの残りの部分は無視されます。

MQTT クライアントは、鍵ストア内のすべての証明書をサーバーに送ります。クライアントが送信したいいずれかの鍵チェーンをサーバーが認証した場合、そのクライアントは認証されます。

また、SSL 暗号スイートをクライアント認証に使用することもできます。現在サポートされている SSL 暗号スイートのアルファベット順のリストを以下に示します。

- SSL\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA
- SSL\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5
- SSL\_DH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_DH\_anon\_WITH\_AES\_128\_CBC\_SHA
- SSL\_DH\_anon\_WITH\_DES\_CBC\_SHA
- SSL\_DH\_anon\_WITH\_RC4\_128\_MD5
- SSL\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA
- SSL\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_DHE\_DSS\_WITH\_AES\_128\_CBC\_SHA
- SSL\_DHE\_DSS\_WITH\_DES\_CBC\_SHA
- SSL\_DHE\_DSS\_WITH\_RC4\_128\_SHA
- SSL\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA
- SSL\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- SSL\_DHE\_RSA\_WITH\_DES\_CBC\_SHA
- SSL\_KRB5\_EXPORT\_WITH\_DES\_CBC\_40\_MD5
- SSL\_KRB5\_EXPORT\_WITH\_DES\_CBC\_40\_SHA
- SSL\_KRB5\_EXPORT\_WITH\_RC4\_40\_MD5
- SSL\_KRB5\_EXPORT\_WITH\_RC4\_40\_SHA
- SSL\_KRB5\_WITH\_3DES\_EDE\_CBC\_MD5
- SSL\_KRB5\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_KRB5\_WITH\_DES\_CBC\_MD5
- SSL\_KRB5\_WITH\_DES\_CBC\_SHA
- SSL\_KRB5\_WITH\_RC4\_128\_MD5
- SSL\_KRB5\_WITH\_RC4\_128\_SHA
- SSL\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA
- SSL\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5

- SSL\_RSA\_FIPS\_WITH\_3DES\_EDE\_CBC\_SHA
- **V7.5.0.2** SSL\_RSA\_FIPS\_WITH\_AES\_128\_CBC\_SHA256
- **V7.5.0.2** SSL\_RSA\_FIPS\_WITH\_AES\_256\_CBC\_SHA256
- SSL\_RSA\_FIPS\_WITH\_DES\_CBC\_SHA
- SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_RSA\_WITH\_AES\_128\_CBC\_SHA
- **V7.5.0.2** SSL\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- **V7.5.0.2** SSL\_RSA\_WITH\_AES\_256\_CBC\_SHA256
- SSL\_RSA\_WITH\_DES\_CBC\_SHA
- SSL\_RSA\_WITH\_NULL\_MD5
- SSL\_RSA\_WITH\_NULL\_SHA
- **V7.5.0.2** SSL\_RSA\_WITH\_NULL\_SHA256
- SSL\_RSA\_WITH\_RC4\_128\_MD5
- SSL\_RSA\_WITH\_RC4\_128\_SHA

**V7.5.0.2** SHA-2 暗号スイートの使用を計画している場合は、176 ページの『MQTT クライアントで SHA-2 暗号スイートを使用する場合のシステム要件』を参照してください。

#### 関連概念

109 ページの『SSL を使用したチャンネル認証のための MQTT クライアントの構成』

SSL を使用する遠隔測定チャンネルを認証するには、クライアントは SSL を使用して遠隔測定チャンネルに接続する必要があります。クライアントは SSL のために構成された遠隔測定チャンネルに対応するポートを指定する必要があります。この構成には、サーバーの私的に署名されたデジタル証明書を含む、パスフレーズで保護された鍵ストアが含まれている必要があります。

## SSL を使用したテレメトリー・チャンネルの認証

MQTT クライアントとキュー・マネージャーとの間の接続は、常に MQTT クライアントによって開始されます。MQTT クライアントは常に SSL クライアントです。サーバーのクライアント認証および MQTT クライアントのサーバー認証は、どちらもオプションです。

クライアントが匿名接続をサポートする CipherSpec を使用するように構成されていない限り、クライアントは常にサーバーの認証を試行します。認証が失敗すると、接続は確立されません。

SSL を使用する代わりに、例えば IPsec など、いくつかの種類の仮想プライベート・ネットワーク (VPN) は TCP/IP 接続のエンドポイントを認証します。VPN は、ネットワーク上を流れる各 IP パケットを暗号化します。そのような VPN 接続が確立されると、トラステッド・ネットワークが確立されます。VPN ネットワーク上で TCP/IP を使用して、MQTT クライアントをテレメトリー・チャンネルに接続できます。

SSL を使用するサーバー認証は、機密情報の送信先となるサーバーを認証します。クライアントは、サーバーから送信された証明書、トラストストアに格納されている証明書、または JRE cacerts ストアに格納されている証明書と突き合わせる検査を実行します。

JRE 証明書ストアは JKS ファイル cacerts です。これは JRE InstallPath\lib\security\にあります。これはデフォルト・パスワードの changeit を使用してインストールされます。信頼するストア証明書は、JRE 証明書ストアまたはクライアントのトラストストアのいずれかに保管することができます。両方のストアを使用することはできません。クライアントが信頼するパブリック証明書を、他の Java アプリケーションが使用する証明書と分けて保管する場合は、クライアントのトラストストアを使用してください。クライアント側で実行されているすべての Java アプリケーションに、共通の証明書ストアを使用する場合は、JRE 証明書ストアを使用してください。JRE 証明書ストアを使用することにした場合は、その証明書ストアに含まれている証明書を検討して、それらの証明書が信頼できるものであることを保証してください。

別のトラスト・プロバイダーを提供して JSSE 構成を変更できます。トラスト・プロバイダーを、証明書に対して別の検査を行うようにカスタマイズできます。MQTT クライアントを使用した一部の OGSi 環境では、この環境は別のトラスト・プロバイダーを提供します。

SSL を使用してテレメトリー・チャンネルを認証するには、サーバーおよびクライアントを構成します。

## 関連概念

109 ページの『SSL を使用したチャンネル認証のための MQTT クライアントの構成』

SSL を使用する遠隔測定チャンネルを認証するには、クライアントは SSL を使用して遠隔測定チャンネルに接続する必要があります。クライアントは SSL のために構成された遠隔測定チャンネルに対応するポートを指定する必要があります。この構成には、サーバーの私的に署名されたデジタル証明書を含む、パスフレーズで保護された鍵ストアが含まれている必要があります。

## SSL を使用したチャンネル認証のための MQTT クライアントの構成

SSL を使用する遠隔測定チャンネルを認証するには、クライアントは SSL を使用して遠隔測定チャンネルに接続する必要があります。クライアントは SSL のために構成された遠隔測定チャンネルに対応するポートを指定する必要があります。この構成には、サーバーの私的に署名されたデジタル証明書を含む、パスフレーズで保護された鍵ストアが含まれている必要があります。

例えば、クライアントで以下のようにします。

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

クライアント JVM は、JSSE からの標準ソケット・ファクトリーを使用する必要があります。Java ME を使用している場合、JSSE パッケージがロードされていることを確認する必要があります。Java SE を使用している場合は、Java バージョン 1.4.1 以降、JSSE が JRE に組み込まれています。

SSL 接続では、接続の前にいくつかの SSL プロパティを設定する必要があります。プロパティを設定するには、`-D` スイッチを使用してプロパティを JVM に渡すか、または `MqttConnectionOptions.setSSLProperties` メソッドを使用してプロパティを設定します。

メソッド `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)` を呼び出すことによって非標準ソケット・ファクトリーをロードする場合、SSL 設定がネットワーク・ソケットに渡される方法はアプリケーションによって定義されます。

SSL を使用して遠隔測定チャンネルに接続するようにクライアントをコーディングして、クライアントが以下の 3 つの方法のいずれかでサーバー証明書を信頼するように構成します。

既知の認証局によって署名されたサーバー証明書を **cacerts** ストア内で使用する。

サーバーが証明書チェーン内のすべての中間キーを送る場合、追加の構成はありません。クライアント JRE の **cacerts** ストアにある証明書を検討して、**cacerts** ストアへのパスワードを変更するようにお勧めします。

## その他の証明書

信頼する証明書をクライアントのトラストストアに保管します。証明書チェーンの少なくとも 1 つの証明書をトラストストアに保管する必要があります。トラストストア・パラメーターを `MqttConnectionOptions.SSLProperty` に設定します。

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

## カスタム・トラスト・マネージャーを使用する

トラスト・プロバイダーを実装して、使用されるアルゴリズムの名前を渡します。`MqttConnectionOptions.SSLProperty` で使用するプロバイダー・クラスの名前およびアルゴリズムを設定します。

- `com.ibm.ssl.trustStoreProvider`

- `com.ibm.ssl.trustStoreManager`

また、SSL 暗号スイートをチャネル認証に使用することもできます。現在サポートされている SSL 暗号スイートのアルファベット順のリストを以下に示します。

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA`
- `SSL_KRB5_EXPORT_WITH_RC4_40_MD5`
- `SSL_KRB5_EXPORT_WITH_RC4_40_SHA`
- `SSL_KRB5_WITH_3DES_EDE_CBC_MD5`
- `SSL_KRB5_WITH_3DES_EDE_CBC_SHA`
- `SSL_KRB5_WITH_DES_CBC_MD5`
- `SSL_KRB5_WITH_DES_CBC_SHA`
- `SSL_KRB5_WITH_RC4_128_MD5`
- `SSL_KRB5_WITH_RC4_128_SHA`
- `SSL_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_RSA_EXPORT_WITH_RC4_40_MD5`
- `SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA`
- **V7.5.0.2** `SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256`
- **V7.5.0.2** `SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256`
- `SSL_RSA_FIPS_WITH_DES_CBC_SHA`
- `SSL_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_RSA_WITH_AES_128_CBC_SHA`
- **V7.5.0.2** `SSL_RSA_WITH_AES_128_CBC_SHA256`
- **V7.5.0.2** `SSL_RSA_WITH_AES_256_CBC_SHA256`
- `SSL_RSA_WITH_DES_CBC_SHA`
- `SSL_RSA_WITH_NULL_MD5`

- SSL\_RSA\_WITH\_NULL\_SHA
- **V7.5.0.2** SSL\_RSA\_WITH\_NULL\_SHA256
- SSL\_RSA\_WITH\_RC4\_128\_MD5
- SSL\_RSA\_WITH\_RC4\_128\_SHA

**V7.5.0.2** SHA-2 暗号スイートの使用を計画している場合は、176 ページの『MQTT クライアントで SHA-2 暗号スイートを使用する場合のシステム要件』を参照してください。

### 関連概念

106 ページの『SSL を使用したクライアント認証のための MQTT クライアントの構成』

SSL を使用する MQTT クライアントを認証するには、クライアントは SSL を使用して遠隔測定チャンネルに接続します。クライアントは SSL クライアントを認証するように構成された遠隔測定チャンネルに対応する TCP ポートを指定する必要があります。

## テレメトリー・チャンネルでのパブリケーションのプライバシー

テレメトリー・チャンネル間でいずれかの方向に送信される MQTT パブリケーションのプライバシーは、SSL を使用して接続上の伝送を暗号化することにより保護されます。

テレメトリー・チャンネルに接続する MQTT クライアントは、SSL を使用して、対称鍵暗号方式を使用しているチャンネル上を伝送されるパブリケーションのプライバシーを保護します。エンドポイントは認証されないため、チャンネルの暗号化を単独で信用することはできません。プライバシーの保護と、サーバー認証または相互認証とを結合します。

SSL を使用する代わりに、例えば IPsec など、いくつかの種類の仮想プライベート・ネットワーク (VPN) は TCP/IP 接続のエンドポイントを認証します。VPN は、ネットワーク上を流れる各 IP パケットを暗号化します。そのような VPN 接続が確立されると、トラステッド・ネットワークが確立されます。VPN ネットワーク上で TCP/IP を使用して、MQTT クライアントをテレメトリー・チャンネルに接続できます。

チャンネルを暗号化し、サーバーを認証する標準的な構成の場合については、108 ページの『SSL を使用したテレメトリー・チャンネルの認証』を参照してください。

サーバーを認証しないで SSL 接続を暗号化すると、接続は中間者攻撃にさらされます。交換する情報は盗聴に対しては保護されますが、その情報を交換している相手が誰であるかは分かりません。ネットワークを制御しない限り、IP 伝送を傍受し、エンドポイントになりすましている誰かにさらされます。

匿名 SSL をサポートする Diffie-Hellman 鍵交換 CipherSpec を使用することにより、サーバーを認証しなくても暗号化された SSL 接続を作成することができます。非公開署名されたサーバー証明書を交換しなくても、クライアントとサーバーの間で共有され、SSL 伝送を暗号化するために使用されるマスター・シークレットが確立されます。

匿名接続は安全ではないので、ほとんどの SSL 実装では、デフォルトでは匿名の CipherSpec は使用されません。テレメトリー・チャンネルが SSL 接続を要求するクライアント要求を受け入れる場合、そのテレメトリー・チャンネルは、鍵ストアをパスフレーズで保護する必要があります。デフォルトでは、SSL 実装は匿名の CipherSpec を使用しないので、クライアントが認証できる、非公開署名された証明書を鍵ストアに含める必要があります。

匿名の CipherSpec を使用する場合は、サーバーの鍵ストアが存在している必要がありますが、非公開署名された証明書が含まれている必要はありません。

暗号化された接続を確立するための別の方法は、クライアント側にあるトラスト・プロバイダーを独自の実装で置き換えることです。この独自の実装のトラスト・プロバイダーはサーバー証明書を認証しないでしようが、接続は暗号化されます。

## MQTT クライアントおよびテレメトリー・チャンネルの SSL 構成

MQTT クライアントおよび WebSphere MQ Telemetry (MQXR) サービスは、Java Secure Socket Extension (JSSE) を使用して、SSL を使用するテレメトリー・チャンネルに接続します。MQTT C クライアント、および WebSphere MQ Telemetry デーモン (デバイス用) は、SSL をサポートしません。

遠隔測定チャンネルおよび MQTT クライアントを認証し、クライアントと遠隔測定チャンネルとの間のメッセージ転送を暗号化するように SSL を構成します。

SSL を使用する代わりに、例えば IPsec など、いくつかの種類の仮想プライベート・ネットワーク (VPN) は TCP/IP 接続のエンドポイントを認証します。VPN は、ネットワーク上を流れる各 IP パケットを暗号化します。そのような VPN 接続が確立されると、トラステッド・ネットワークが確立されます。VPN ネットワーク上で TCP/IP を使用して、MQTT クライアントをテレメトリー・チャンネルに接続できます。

TCP/IP を介した SSL プロトコルを使用するように、Java MQTT クライアントとテレメトリー・チャンネルの間の接続を構成できます。保護対象は、JSSE を使用するために SSL がどのように構成されているのかによって異なります。最も保護の高い構成から順になっている、以下の 3 つの異なるレベルのセキュリティーを構成できます。

1. 信頼できる MQTT クライアントのみに接続を許可します。信頼できるテレメトリー・チャンネルにのみ MQTT クライアントを接続します。クライアントとキュー・マネージャーの間のメッセージを暗号化します。[105 ページの『SSL を使用した MQTT クライアント認証』](#)を参照してください。
2. 信頼できるテレメトリー・チャンネルにのみ MQTT クライアントを接続します。クライアントとキュー・マネージャーの間のメッセージを暗号化します。[108 ページの『SSL を使用したテレメトリー・チャンネルの認証』](#)を参照してください。
3. クライアントとキュー・マネージャーの間のメッセージを暗号化します。[111 ページの『テレメトリー・チャンネルでのパブリケーションのプライバシー』](#)を参照してください。

## JSSE 構成パラメーター

JSSE パラメーターを変更して、SSL 接続の構成方法を変えます。JSSE 構成パラメーターは次の 3 つのセットに配置されます。

1. [IBM WebSphere MQ テレメトリー・チャンネル](#)
2. [MQTT Java クライアント](#)
3. [JRE](#)

IBM WebSphere MQ エクスプローラーを使用して、テレメトリー・チャンネル・パラメーターを構成します。MqttConnectionOptions.SSLProperties 属性に MQTT Java クライアント・パラメーターを設定します。クライアント側およびサーバー側の両方の JRE の security ディレクトリー内のファイルを編集して、JRE セキュリティー・パラメーターを変更します。

### IBM WebSphere MQ テレメトリー・チャンネル

WebSphere MQ エクスプローラーを使用して、テレメトリー・チャンネルのすべての SSL パラメーターを設定します。

#### ChannelName

ChannelName は、すべてのチャンネルで必須のパラメーターです。

チャンネル名は、特定のポート番号に関連付けられているチャンネルを識別します。チャンネルには、MQTT クライアント・セットを管理するのに役立つ名前を付けてください。

#### PortNumber

PortNumber は、すべてのチャンネルのオプション・パラメーターです。このパラメーターのデフォルトは、TCP チャンネルの場合は 1883 で、SSL チャンネルの場合は 8883 です。

このチャンネルに関連付けられている TCP/IP ポート番号。チャンネルに対して定義されているポートを指定することにより、MQTT クライアントはそのチャンネルに接続されます。チャンネルが SSL プロパティーを持っている場合、クライアントは SSL プロトコルを使用して接続する必要があります。以下に例を示します。

```
MqttClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```



## KeyFileName

KeyFileName は、SSL チャンネルに必須のパラメーターです。TCP チャンネルの場合は、このパラメーターを省略する必要があります。

KeyFileName は、提供するデジタル証明書を含む Java 鍵ストアへのパスです。サーバー側の鍵ストアのタイプとしては、JKS、JCEKS または PKCS12 を使用します。

鍵ストア・タイプを識別するには、以下に示したいずれかのファイル拡張子を使用します。

- .jks
- .jceks
- .p12
- .pkcs12

上記以外のファイル拡張子を持つ鍵ストアは、JKS 鍵ストアと見なされます。

サーバー側のあるタイプの鍵ストアと、クライアント側のそれ以外のタイプの鍵ストアを組み合わせることができます。

サーバーのプライベート証明書は、鍵ストアに配置します。この証明書はサーバー証明書と呼ばれます。この証明書は自己署名することも、署名権限によって署名される証明書チェーンの一部にすることもできます。

証明書チェーンを使用する場合は、サーバーの鍵ストア内に関連付けられている証明書を配置します。

サーバー証明書、およびサーバーの鍵チェーン内の証明書は、サーバーの ID を認証するためにクライアントに送信されます。

ClientAuth を Required に設定していた場合、鍵ストアには、クライアントを認証するのに必要な証明書が含まれていなければなりません。クライアントは自己署名証明書、または証明書チェーンを送信し、クライアントは鍵ストア内の証明書に対するこの内容の最初の検証によって認証されます。証明書チェーンを使用すると、たとえさまざまなクライアント証明書と一緒にクライアントが発行されていたとしても、1つの証明書で複数のクライアントを検証することができます。

## PassPhrase

PassPhrase は、SSL チャンネルに必須のパラメーターです。TCP チャンネルの場合は、このパラメーターを省略する必要があります。

鍵ストアの保護には、パスフレーズが使用されます。

## ClientAuth

ClientAuth はオプションの SSL パラメーターです。このパラメーターのデフォルトは、クライアントを認証しない、です。TCP チャンネルの場合は、このパラメーターを省略する必要があります。

クライアントがテレメトリー・チャンネルに接続することを許可する前に、テレメトリー (MQXR) サービスがクライアントを認証するようにするには、ClientAuth を設定します。

ClientAuth を設定した場合、クライアントは SSL を使用しているサーバーに接続し、そのサーバーを認証しなければなりません。ClientAuth の設定に対する応答として、クライアントはそのデジタル証明書と、その鍵ストア内にあるその他の証明書をサーバーに送信します。このデジタル証明書は、クライアント証明書と呼ばれます。これらの証明書は、チャンネル鍵ストアおよび JRE の cacerts ストアに格納されている証明書と突き合わせて認証されます。

## CipherSuite

CipherSuite は、オプションの SSL パラメーターです。このパラメーターのデフォルトは、有効な CipherSpec をすべて試行する、です。TCP チャンネルの場合は、このパラメーターを省略する必要があります。

特定の CipherSpec を使用する場合は、CipherSuite を SSL 接続の確立に使用する必要のある CipherSpec の名前に設定します。

テレメトリー・サービスと MQTT クライアントは、各端点で有効になっているすべての CipherSpec から共通の CipherSpec について折衝します。特定の CipherSpec が接続の片端または両端で指定された場合、その CipherSpec はもう一方の端の CipherSpec に一致しなければなりません。

追加のプロバイダーを JSSE に追加することにより、追加の暗号をインストールします。

### 連邦情報処理標準 (FIPS)

FIPS は、オプションの設定です。デフォルトでは、これは設定されていません。

キュー・マネージャーのプロパティ・パネルで、または **runmqsc** を使用して、SSLFIPS を設定します。SSLFIPS は、FIPS によって証明されたアルゴリズムだけを使用するのかどうかを指定します。

### 取り消し名前リスト

取り消し名前リストは、オプションの設定です。デフォルトでは、これは設定されていません。

キュー・マネージャーのプロパティ・パネルで、または **runmqsc** を使用して、SSLCRLNL を設定します。SSLCRLNL は、証明書取り消し場所を提供するために使用される認証情報オブジェクトの名前リストを指定します。

SSL プロパティを設定するその他のキュー・マネージャー・パラメーターは使用されません。

### MQTT Java クライアント

Java クライアントの SSL プロパティを `MqttConnectionOptions.SSLProperties` に設定します。以下に例を示します。

```
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);
```

特定のプロパティの名前と値については、`MqttConnectOptions` に関する API 文書で説明しています。MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、[MQTT クライアント・プログラミング・リファレンス](#)を参照してください。

### プロトコル

Protocol はオプションです。

このプロトコルは、テレメトリー・サーバーとの折衝で選択されます。特定のプロトコルが必要な場合は、それを選択することができます。テレメトリー・サーバーがそのプロトコルをサポートしていない場合、接続は失敗します。

### ContextProvider

ContextProvider はオプションです。

### KeyStore

KeyStore はオプションです。クライアントの認証を強制的に行うために `ClientAuth` がサーバー側で設定されている場合は、これを構成します。

クライアントの秘密鍵を使用して署名されている、クライアントのデジタル証明書を鍵ストアに配置します。鍵ストアのパスとパスワードを指定します。タイプとプロバイダーはオプションです。デフォルトのタイプは JKS、デフォルトのプロバイダーは IBMJCE です。

新規の鍵ストア・プロバイダーを追加するクラスを参照するには、別の鍵ストア・プロバイダーを指定します。鍵ストア・プロバイダーが使用するアルゴリズムの名前を渡し、鍵マネージャー名を設定して `KeyManagerFactory` のインスタンスを生成します。

### TrustStore

TrustStore はオプションです。信頼するすべての証明書を JRE の cacerts ストアに配置できます。

クライアント用に別のトラストストアが必要な場合には、このトラストストアを構成します。既にルート証明書を cacerts に保管している既知の CA が発行した証明書をサーバーが使用している場合は、トラストストアを構成しないこともあります。

サーバーの公開署名された証明書またはルート証明書をトラストストアに追加し、トラストストアのパスとパスワードを指定します。デフォルトのタイプは JKS、デフォルトのプロバイダーは IBMJCE です。

新規のトラストストア・プロバイダーを追加するクラスを参照するには、別のトラストストア・プロバイダーを指定します。トラストストア・プロバイダーが使用するアルゴリズムの名前を渡し、トラスト・マネージャー名を設定して TrustManagerFactory のインスタンスを生成します。

## JRE

クライアント側およびサーバー側の両方での SSL の動作に影響を与える Java セキュリティーの他の側面が JRE 内に構成されます。Windows の構成ファイルは、*Java Installation Directory\jre\lib\security* にあります。IBM WebSphere MQ に同梱されている JRE を使用している場合のパスは、以下の表に示すとおりです。

プラットフォーム	ファイル・パス
Windows	<i>WMQ Installation Directory\java\jre\lib\security</i>
Linux for System x 32 ビット	<i>WMQ Installation Directory/java/jre/lib/security</i>
その他の UNIX and Linux プラットフォーム	<i>WMQ Installation Directory/java/jre64/jre/lib/security</i>

### 既知の認証局

cacerts ファイルには、既知の認証局のルート証明書が含まれています。トラストストアを指定していない限り、デフォルトでは cacerts が使用されます。cacerts ストアを使用する場合、またはトラストストアを指定していない場合は、cacerts 内の署名者のリストを検討して、セキュリティー要件を満たすようそれを編集する必要があります。

cacerts を開くには、WebSphere MQ コマンド `strmqikm` を使用します。これは、IBM 鍵管理ユーティリティーを実行します。パスワード `changeit` を使用して、cacerts を JKS ファイルとして開きます。パスワードを変更して、このファイルを保護します。

### セキュリティー・クラスの構成

java.security ファイルを使用して、追加のセキュリティー・プロバイダーとその他のデフォルトのセキュリティー・プロパティーを登録します。

### 権限

java.policy ファイルを使用して、リソースに付与された許可を変更します。javaws.policy は許可を javaws.jar に付与します。

### 暗号化の強度

一部の JRE は、暗号化の強度を下げて出荷されています。鍵を鍵ストアにインポートできない場合は、暗号化の強度が下げられているのが原因である場合があります。strmqikm コマンドを使用して `ikeyman` を始動してみるか、または、[IBM developer kits, Security information](#) から、強度はあるものの権限が制限されているファイルをダウンロードします。

**重要:** お住まいの国によっては、暗号化ソフトウェアの輸入、所持、使用、または別の国への再輸出に制限が課せられている場合があります。お住まいの国の法律を確認してから、規制されていないポリシー・ファイルをダウンロードして使用するようしてください。暗号化ソフトウェアの輸入、所持、使用、および再輸出に関する国の規制および方針を確認して、それが許可されているかどうかを決定してください。

## 任意のサーバーへの接続をクライアントに許可するようトラスト・プロバイダーを変更する

この例は、トラスト・プロバイダーを追加して、MQTT クライアント・コードからそのプロバイダーを参照する方法を示しています。この例では、クライアントまたはサーバーの認証は実行されません。その結果、SSL 接続は暗号化されますが、認証されません。

116 ページの図 20 のコード・スニペットは、MQTT クライアントに対して AcceptAllProviders トラスト・プロバイダーとトラスト・マネージャーを設定します。

```
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);
```

### 図 20. MQTT クライアントのコード・スニペット

```
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());
    }
}
```

### 図 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() };
    }
}
```

### 図 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);
    }
}
```

### 図 23. AcceptAllX509TrustManager.java

## テレメトリー・チャンネルの JAAS 構成

クライアントから送られた Username を認証するように JAAS を構成します。

WebSphere MQ 管理者は、どの MQTT チャンネルで JAAS を使用するクライアント認証が必要となるかを構成します。JAAS 認証を実行する各チャンネルの JAAS 構成の名前を指定します。すべてのチャンネルが同じ JAAS 構成を使用することもでき、それぞれが異なる JAAS 構成を使用することもできます。構成は、`WMQData directory\qmgrs\qMgrName\mqxr\jaas.config` に定義されています。

`jaas.config` ファイルは、JAAS 構成名に基づいて編成されます。それぞれの構成名の下には、ログイン構成のリストがあります。[118 ページの図 24](#) を参照してください。

JAAS は、4 つの標準ログイン・モジュールを提供します。標準の NT および UNIX ログイン・モジュールの価値は、限られたものです。

#### **JndiLoginModule**

JNDI (Java Naming and Directory Interface) の下で構成されたディレクトリー・サービスに対して認証します。

#### **Krb5LoginModule**

Kerberos プロトコルを使用して認証します。

#### **NTLoginModule**

現行ユーザーの NT セキュリティー情報を使用して認証します。

#### **UnixLoginModule**

現行ユーザーの UNIX セキュリティー情報を使用して認証します。

NTLoginModule または UnixLoginModule を使用することの問題点は、テレメトリー (MQXR) サービスが MQTT チャンネルの ID ではなく `mqm` の ID を使用して稼働することです。`mqm` は、認証のために NTLoginModule または UnixLoginModule に渡される ID であり、クライアントの ID ではありません。

この問題を解決するには、独自のログイン・モジュールを記述するか、または他の標準ログイン・モジュールを使用します。サンプルの `JAASLoginModule.java` が WebSphere MQ Telemetry で提供されています。これは `javax.security.auth.spi.LoginModule` インターフェースの実装です。これを使用して、独自の認証方式を開発します。

提供する新しい LoginModule クラスは、テレメトリー (MQXR) サービスのクラスパス上に存在しなければなりません。そのクラスを、クラスパスに含まれる WebSphere MQ ディレクトリーに入れなくてはいけません。独自のディレクトリーを作成して、テレメトリー (MQXR) サービスのためのクラスパス全体を定義します。

クラスパスを `service.env` ファイル内に設定して、テレメトリー (MQXR) サービスで使用されるクラスパスを強化することができます。CLASSPATH は大文字で記述する必要があり、クラスパス・ステートメントに含めることができるのはリテラルだけです。CLASSPATH では変数を使用できません。例えば、`CLASSPATH=%CLASSPATH%` は間違いです。テレメトリー (MQXR) サービスは、独自のクラスパスを設定します。`service.env` で定義された CLASSPATH がそれに追加されます。

テレメトリー (MQXR) サービスは、MQTT チャンネルに接続したクライアントの Username および Password を返す 2 つのコールバックを提供します。「ユーザー名」および「パスワード」は、`MqttConnectOptions` オブジェクトで設定されます。Username および Password にアクセスする方法について詳しくは、[118 ページの図 25](#) を参照してください。

#### **例**

指名された 1 つの構成 `MQXRConfig` がある JAAS 構成ファイルの例。

```

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";
};

```

図 24. `jaas.config` ファイルのサンプル

MQTT クライアントによって提供される Username および Password を受け取るようにコーディングされた、JAAS ログイン・モジュールの例。

```

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;
}

```

図 25. `JAASLoginModule.Login()` メソッドのサンプル

## クライアント・プログラミングの概念

このセクションで説明される概念は、MQTT protocol・バージョン 3.1 の Java クライアントを理解するうえで役立ちます。この概念は、パッケージ `com.ibm.micro.client.mqttv3` に付属する API 文書を補完するものです。

`com.ibm.micro.client.mqttv3` には、MQTT バージョン 3.1 プロトコルの Java 実装用の public メソッドを提供するクラスが含まれています。`com.ibm.micro.client.mqttv3` パッケージ、および Java SE と ME のプロトコルを実装する付随パッケージは、IBM WebSphere MQ Telemetry のインストールで提供されます。

MQTT クライアントを開発して実行するには、これらのパッケージをクライアント装置にコピーまたはインストールする必要があります。別個のクライアント・ランタイムをインストールする必要はありません。

クライアントのライセンス条件は、クライアントの接続先のサーバーに関連付けられます。

Java クライアントは、MQTT protocol・バージョン 3.1 の参照実装です。ユーザー独自のクライアントを、さまざまな装置プラットフォームに適したさまざまな言語で実装できます。詳細については、[MQ Telemetry Transport のフォーマットおよびプロトコル](#)を参照してください。

パッケージ `com.ibm.micro.client.mqttv3` 用のクライアント API 文書では、クライアントがどのサーバーに接続するかについて想定していません。MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、[MQTT クライアント・プログラミング・リファレンス](#)を参照してください。クライアントの動作は、異なるサーバーに接続するとき若干の違いが出る可能性があります。以下の説明は、IBM WebSphere MQ 遠隔測定 (MQXR) サービスに接続している場合のクライアントの動作について示しています。

## JavaScript 用の MQTT メッセージング・クライアント と Web アプリケーション

最近まで、Web アプリケーションのプログラミングとメッセージング・アプリケーションの作成は別々の分野でした。これまでの経験がどちらの分野であったとしても、JavaScript とメッセージングを一緒に使用することには大きなメリットがあります。メッセージング・アプリケーションを Web アプリケーションとしてコーディングすると、あらゆる最新のブラウザにそれを取り込んで実行できます。アプリケーションを変更する場合、ブラウザを最新表示すると、最新バージョンが取り込まれます。また、ブラウザはセキュリティーの実施と信頼性の高いメッセージ伝送を行います。

### Web アプリケーションを使用すると、アプリケーション・デプロイメントがどのように簡単になるか

(例えば) IBM WebSphere MQ で従来のメッセージング・アプリケーションを開発およびデプロイしたことがある読者は、恐らく次のようなデプロイメント・プロセスに従っていたことでしょう。

1. システム管理者がクライアント・ライブラリーをインストールするか、組み込みます。
2. システム管理者は、メッセージング・アプリケーションがエンド・ユーザーたちに配布され、それぞれのローカル・システムにインストールされるよう手配します。
3. コードが変更された場合、システム管理者はこれまでの手順を繰り返します (したがって変更管理は複雑です)。

メッセージング・アプリケーションを Web アプリケーションとしてコーディングした場合、デプロイメント・プロセスは次のようになります。

1. システム管理者は Web アプリケーションおよびクライアント・ライブラリーを URL として提供します。
2. エンド・ユーザーのブラウザが Web アプリケーションとクライアント・ライブラリーを一緒に取り込みます。
3. コードが変更された場合、ブラウザを最新表示するだけで更新後のバージョンが取り入れられます (したがって変更管理は単純です)。

### Web アプリケーションでブラウザからメッセージングを直接使用することができる理由

JavaScript でのアプリケーション・プログラミングの経験がある読者は、IBM WebSphere MQ のようなメッセージング・システムに備わっている以下のような利点を考慮して考えることができます。

- メッセージング・システムを介してメッセージを送受信した場合、そのシステムがメッセージの確実な送達を担当します。
- メッセージング・システムが送達を担当するため、Web アプリケーションは、いわば「発行した後は忘れる」(応答不要送信) 状態になります。これにより、プログラミング論理がかなり簡単になります。メッセージが送られた場合、コードの中で着信確認を行う必要はありません。アプリケーションで受信確認を扱ったり、未配布メッセージを保存して後で再試行したりする必要はなくなります。
- メッセージング・システムは、イベント・ドリブン型のメッセージングを提供します。クライアント・アプリケーションは、要求を送信した後で応答をポーリングし続ける必要がなくなります。代わりに、関係するイベントが発生した時点でメッセージング・サーバーがクライアント・アプリケーションにメッセージを送ります。つまり、アプリケーションがサーバーを次回ポーリングするときまで待つ代わりに、イベント発生時に直ちにクライアント・アプリケーションにアラートが送られることになります。

- また、イベント・ドリブン・メッセージングにより、クライアント・アプリケーションのホストであるデバイス上の負荷、ブラウザとメッセージング・サーバーの間のネットワーク・トラフィック、およびメッセージング・サーバー上の負荷が大幅に減ります。モバイル・デバイスで稼働し、無線ネットワークで接続されるシステムが増えていく中で、この点はますます重要になってきています。

## 各部分をどのように組み合わせるか

JavaScript 用の MQTT メッセージング・クライアントには、クライアント・ライブラリーと、ライブラリーを使用する Web アプリケーションの例が含まれています。ライブラリーを使用する独自の Web アプリケーションをコーディングする必要があります。その後、例えば(以下の図のように)MQ キュー・マネージャーまたはアプリケーション・サーバーを使用して、選択した URL において Web アプリケーションとクライアント・ライブラリーが利用可能になります。ブラウザは Web アプリケーションとクライアント・ライブラリーを取り込み、Web アプリケーションはブラウザを使って MQTT サーバー (IBM WebSphere MQ Telemetry、IBM MessageSight など) に接続してメッセージを交換します。

次のような流れになります。

1. ブラウザーの各インスタンスは、Web アプリケーションを使用可能な URL との接続をリフレッシュし、最新バージョンの Web アプリケーションとクライアント・ライブラリーがブラウザにロードされます。
2. Web アプリケーションは WebSocket protocol を介した MQTT を使用してキュー・マネージャーに接続し、関係するトピックにサブスクライブします。
3. キュー・マネージャーは同じ接続を使用して、サブスクリプションに一致するメッセージを Web アプリケーションに戻します。

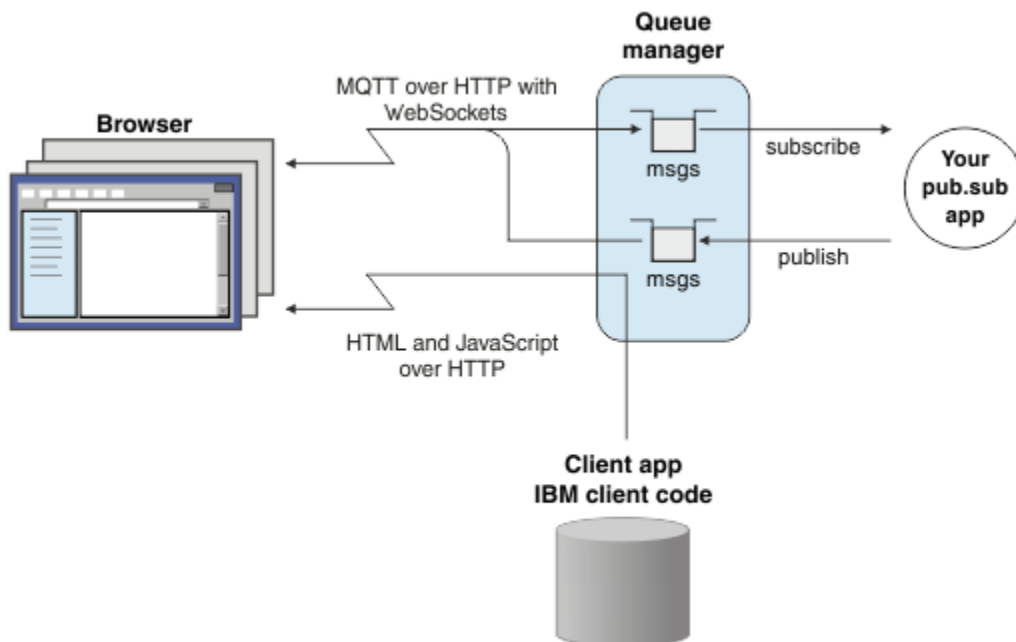


図 26. パブリッシュ/サブスクライブ・メッセージングでの JavaScript 用の MQTT メッセージング・クライアントの使用

Web アプリケーションには、アプリケーション・ロジックと MQTT サーバーの URL が含まれています。ブラウザで開かれたアプリケーションは MQTT サーバーに接続し、必要なサブスクリプションを作成し、イベント・ドリブン・アラートの受信を待機して、それらを処理します。

Web アプリケーションは、WebSockets を介して稼働するトランスポート・プロトコルとして MQTT を使って接続します。最新のほとんどのブラウザは WebSockets 接続を行うことができます。WebSockets を使用することで、Web アプリケーションは、TCP over IP を使用する場合と同様に、HTTP および



WebSocket protocol を受け入れるファイアウォールを介してメッセージを受け渡すことができ、データの packets ("フレーム"と呼ばれる) を送信することができます。

Web アプリケーションから送られたメッセージが MQTT サーバーに到着すると、サーバー・サイドのアプリケーションはそれを単なる 1 つのメッセージとして認識します。そのメッセージがブラウザから来たことは認識しません。

## MQTT サーバーの管理と制御

MQTT サーバーは、メッセージングにおけるサーバー・サイドの複雑な操作を扱います。Web アプリケーションから受け取ったメッセージを確実に送達し、Web アプリケーションに回答するパブリッシュ/サブスクライブ・アプリケーションのホストとなります。すべての MQTT サーバーに関して、以下の手順を完了する必要があります。

- サーバーを作成します。
- ポートを選びます。
- 新しい MQTT チャンネルを定義します。
- 選択されたポートに新しい MQTT チャンネルを介して接続するよう、クライアント Web アプリケーションを構成します。

また、Web アプリケーションの実行可能 JavaScript をブラウザに提供する必要もあります。IBM WebSphere MQ Telemetry を使用している場合、デフォルトでは MQTT サーバーによってこれが自動的に行われ、その際には MQTT サーバーとの接続用に Web アプリケーションで使用されるのと同じ MQTT チャンネルが使われます。MQTT を試している場合は、手早く起動して稼働状態にするうえで、この動作が役立つ可能性があります。実動 (特に高スループット環境) で使用するには、WebSphere Application Server などの専用アプリケーション・サーバーを使用して、別のチャンネル上で Web アプリケーションの実行可能 JavaScript を提供するのが適切な場合もあります。

**注:** IBM MessageSight は高スループット環境用に設計されているため、このようにすることが想定されています。

例えば IBM WebSphere MQ Telemetry を使用している場合、IBM WebSphere MQ Explorer の「**新規遠隔設定チャンネル**」ウィザードを使って以下の手順を完了します。

1. サーバーを作成します。
2. ポートを選びます (デフォルトでは 1883)。
3. 新しい MQTT チャンネルを定義します。
4. 選択されたポートに新しい MQTT チャンネルを介して接続するよう、クライアント Web アプリケーションを構成します。

また、オプションとして、同じチャンネル上のキュー・マネージャーを介して Web アプリケーション実行可能 JavaScript を提供することもできます。これを行うには、キュー・マネージャーが MQTT と HTTP の両方をサポートする必要があります。MQTT 用に構成されたキュー・マネージャーが既に存在する場合、MQSC コマンド・ライン・ツールを使用して、MQTT と HTTP の両方をサポートするようチャンネル定義の中のプロトコルを変更できます。[ALTER CHANNEL](#) を参照してください。

Web アプリケーションおよび JavaScript 用の MQTT メッセージング・クライアント ライブラリは、アプリケーション・サーバーまたはキュー・マネージャーによって定義された構造でディスク上に保管されます。IBM WebSphere MQ Telemetry を使用している場合、Web アプリケーションとクライアント ライブラリは以下のディレクトリ構造の中に保管されます。

```
MQINSTALL
|
| mqi
| |
| | SDK
| | |
| | | WebContent
| | | |
| | | | sample web app (the "Web Messaging Utility" sample HTML pages)
| | | | |
| | | | | WebSocket
| | | | | |
| | | | | | IBM client library (the messaging client JavaScript classes)
```



サンプルの Web アプリケーションとクライアント・ライブラリーが `MQINSTALL/mqxr/SDK/WebContent` ディレクトリーに保管されています。このディレクトリーの内容はすべてのキュー・マネージャーによって提供されます。このすべての内容をユーザーたちが表示および使用できないようにするには、独自のアプリケーションを作成してください。このアプリケーション (または独自の代替アプリケーション) を特定のキュー・マネージャーで使用可能にするには、`MQINSTALL/qmgrs/qmgr_name/mqxr/WebContent` ディレクトリーの中にアプリケーションを配置します。URL でサービスを提供するアプリケーションおよび関連付けられた JavaScript クラスを選択するために、キュー・マネージャーはまず独自の `WebContent` ディレクトリーを参照し、次にグローバル `WebContent` ディレクトリーを参照します。上記のディレクトリー・ツリーの例では、キュー・マネージャーは `your_client_app` および JavaScript クラスのグローバル・コピーを提供します。

Web アプリケーション実行可能ファイルを提供しているキュー・マネージャーを停止するか、キュー・マネージャーによる実行可能ファイル検索場所を変更するには、`webcontentpath` プロパティーを構成して、それを `mqxr.properties` ファイルに追加します。[MQXR プロパティー](#)を参照してください。

### 関連概念

[122 ページの『JavaScript でのメッセージング・アプリケーションのプログラミング方法』](#)

### 関連タスク

[79 ページの『SSL を介した JavaScript 用の MQTT メッセージング・クライアントと WebSockets の接続』](#)  
SSL を使用する JavaScript 用の MQTT メッセージング・クライアント サンプル HTML ページと WebSocket protocol を使用して、Web アプリを IBM WebSphere MQ に安全に接続します。

[25 ページの『JavaScript 用の MQTT メッセージング・クライアントの概要』](#)

JavaScript 用の MQTT メッセージング・クライアントは、メッセージング・クライアント・サンプル・ホーム・ページを表示し、リンク先のリソースを参照することで開始できます。このホーム・ページを表示するには、MQTT メッセージング・クライアント・サンプル JavaScript ページからの接続を受け入れるように MQTT サーバーを構成してから、サーバーで構成した URL を Web ブラウザーに入力します。デバイス上の JavaScript 用の MQTT メッセージング・クライアントは自動的に始動し、メッセージング・クライアント・サンプル・ホーム・ページが表示されます。このページには、ユーティリティー、プログラミング・インターフェース文書、チュートリアル、およびその他の役立つ情報へのリンクが含まれています。

## JavaScript でのメッセージング・アプリケーションのプログラミング方法

JavaScript 用の MQTT メッセージング・クライアントには、単純なパブリッシュ/サブスクライブ Web アプリケーションの作成方法を示すチュートリアルが含まれています。「First steps, Hello world」アプリケーション・コードを調べることで、メッセージング用 Web アプリケーションの基本的なプログラミングの仕組みを理解することができます。

これまで従来型のメッセージング・アプリケーションの開発とデプロイを主に行ってきた読者は、[123 ページの『JavaScript コーディングのヒント』](#)のセクションにある役立つ情報を参照してください。メッセージングに馴染みのない、経験豊かな JavaScript 開発者は、[125 ページの『メッセージングの基礎』](#)のセクションにある主なメッセージング概念の紹介を参照することができます。

IBM messaging

Intro Utility Reference Tutorial Useful Links

### First steps, the hello world application.

The example below is a simple javascript application that shows how to subscribe to a topic called "World" and publish a message containing the string "Hello" to it.

#### Example

```
// Make connection to the server.
client = new Messaging.Client(location.hostname, Number(location.port), "clientId");

// Set up a callBacks used when the connection is completed,
// when a message arrives for this client and when the connection is lost.
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;
client.connect({onSuccess:onConnect});

function onConnect() {
    // Once a connection has been made, make a subscription and send a message.
    console.log("onConnect");
    client.subscribe("/World");
    message = new Messaging.Message("Hello");
    message.destinationName = "/World";
    client.send(message);
};

function onConnectionLost(responseObject) {
    if (responseObject.errorCode !== 0)
        console.log("onConnectionLost:"+responseObject.errorMessage);
};

function onMessageArrived(message) {
    console.log("onMessageArrived:"+message.payloadString);
    client.disconnect();
};
```

Click me to try. The Console output is shown below.

## JavaScript コーディングのヒント

メッセージング・アプリケーションの開発に慣れているものの、Web アプリケーションには馴染みがない読者にとって、以下のヒントが役立つでしょう。

### onSuccess コールバック内でイベントごとのコードをラッピングする

メッセージング・アプリケーションをコーディングするときには、以下のイベントをこの順序でコーディングします。

1. connect
2. サブスクライブ
3. publish
4. メッセージの受信

JavaScript 用の MQTT メッセージング・クライアント API は完全に非同期的です。つまり、接続、サブスクライブなどの呼び出しが実施されるのを待つ間、アプリケーション・スレッドはブロックしません。代わりに、これらの呼び出しは onSuccess または onFailure コールバックを呼び出すことによって完了を通知します。各イベントが確実に完了した後で次のイベントがトリガーされるようにするには、onSuccess コールバックでイベントごとのコードをラップする必要があります。例えば、接続の作成が完了する前に、JavaScript アプリケーションが接続呼び出しから戻る可能性があります。必ず接続が発生した後でサブスクライブするには、接続用の onSuccess コールバックの中にサブスクライブ・コードを配置する必要があります。

「First steps, Hello world」アプリケーション・コードでは、この手法を使っています。

### HTML マークアップの中にアプリケーション・コードを組み込む

JavaScript ページの例を示します:

## Example Web Messaging web page.

Connect  
Make a connection to the server, and set up a call back used if a message arrives for this client.

Subscribe  
Make a subscription to topic "/World".

Send  
Create a Message object containing the word "Hello" and then publish it at the server.

Receive  
A copy of the published Message is received in the callback we created earlier.

Disconnect  
Now disconnect this client from the server.

上記のページのソースは次のとおりです。これは、HTML マークアップの中にアプリケーション・コードを組み込む方法を示しています。

```
<!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>
```

```

</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>

```

## メッセージングの基礎

メッセージングに馴染みがない Web アプリケーション開発者のために、背景となるメッセージング情報をいくつか示します。

### 非同期的な、応答不要送信によるメッセージング。

MQTT プロトコルは、送達保証と応答不要送信の転送をサポートします。このプロトコルでは、メッセージ送達は非同期的です。つまりアプリケーションがクライアント API に向けてメッセージを渡した後、メッセージの送達を確認するための処置を何も行いません。この手法を応答不要送信といいます。応答が使用可能になると、自動的にアプリケーションに送られます。

非同期的な送達により、アプリケーションはサーバー接続とメッセージ待機から解放されます。対話モデルは E メールに似ていますが、アプリケーション・プログラミングに合わせて最適化されています。

5 ページの『MQTT の概要』の「"MQTT プロトコル"」セクションも参照してください。

### パブリッシュ/サブスクライブ・メッセージングの概要。

情報のプロバイダーをパブリッシャーといいます。パブリッシャーは、あるサブジェクト (主題) に関する情報を提供しますが、その際、その情報を必要とするアプリケーションについて何も知る必要はありません。パブリッシャーはトピック、つまり特定のサブジェクトのメッセージを入れるコンテナを選択します。次に、パブリッシャーはそのサブジェクトに関する情報の各断片をメッセージとして生成し (これをパブリケーションという)、関連するトピックにそれを送ります。

情報の利用者のことをサブスクライバーといいます。サブスクライバーは、関心のあるトピックへのサブスクリプションを作成します。新しいメッセージがトピックに投稿されると、メッセージはそのトピックのすべてのサブスクライバーに転送されます。サブスクライバーは複数のサブスクリプションを行え、さまざまなパブリッシャーから情報を受け取ることができます。

[IBM WebSphere MQ パブリッシュ/サブスクライブ・メッセージングの概要](#) も参照してください。

### サブスクリプションとトピックのマッチング方法。

IBM WebSphere MQ を MQTT サーバーとして使用する場合、IBM WebSphere MQ でトピックが指定される方法を理解する必要があります。IBM WebSphere MQ ではパブリッシャーがメッセージを作成し、パブリケーションのサブジェクトに最適なトピック・ストリングを付けてそれをパブリッシュします。パブリケーションを受信するために、サブスクライバーは、パブリケーション・トピックを選択するためのパターン・マッチング・トピック・ストリングを指定したサブスクリプションを作成します。キュー・マネージャーは、サブスクリプションがパブリケーション・トピックに一致していて、パブリケーションを受信する権限のあるサブスクライバーにパブリケーションを配信します。

主題は通常、トピック・ツリーとして階層的に編成されます。その場合、'/' 文字を使用してトピック・ストリング内にサブトピックを作成します。トピックはトピック・ツリー内のノードです。トピックはそれ以上サブトピックのないリーフ・ノードか、サブトピックのある中間ノードのいずれかになります。サブスクライバーはワイルドカードを使用して、同時に複数のトピックにサブスクライブできます。例えば /sport/tennis というサブスクリプションの場合、tennis というサブトピックに投稿されるメッセージだけが得られますが、/sport/# というサブスクリプションの場合は /sport のすべてのサブトピックに投稿されるメッセージを取得します。

『トピック』、『トピック・ツリー』および『ワイルドカード・スキーム』も参照してください。

### 関連概念

[119 ページの『JavaScript 用の MQTT メッセージング・クライアントと Web アプリケーション』](#)

### 関連タスク

[79 ページの『SSL を介した JavaScript 用の MQTT メッセージング・クライアントと WebSockets の接続』](#)  
SSL を使用する JavaScript 用の MQTT メッセージング・クライアント サンプル HTML ページと WebSocket protocol を使用して、Web アプリを IBM WebSphere MQ に安全に接続します。

[25 ページの『JavaScript 用の MQTT メッセージング・クライアントの概要』](#)

JavaScript 用の MQTT メッセージング・クライアントは、メッセージング・クライアント・サンプル・ホーム・ページを表示し、リンク先のリソースを参照することで開始できます。このホーム・ページを表示するには、MQTT メッセージング・クライアント・サンプル JavaScript ページからの接続を受け入れるように MQTT サーバーを構成してから、サーバーで構成した URL を Web ブラウザーに入力します。デバイス上の JavaScript 用の MQTT メッセージング・クライアントは自動的に始動し、メッセージング・クライアント・サンプル・ホーム・ページが表示されます。このページには、ユーティリティ、プログラミング・インターフェース文書、チュートリアル、およびその他の役立つ情報へのリンクが含まれています。

## MQTT クライアント・アプリケーションでのコールバックと同期

MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。スレッドは、サーバーとの間でメッセージを送受信する際の遅延から、可能な限り MQTT クライアント・アプリケーションを分離します。パブリケーション、送達トークン、および接続消失イベントは、MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

### コールバック

MqttCallback インターフェースには 3 つのコールバック・メソッドがあります。実装の例については、[Callback.java](#) を参照してください。

#### connectionLost(java.lang.Throwable cause)

connectionLost は、通信エラーが原因で接続が切断されたときに呼び出されます。また、接続が確立された後にサーバー上でエラーが発生したためにサーバーが接続を切断した場合にも呼び出されます。サーバー・エラーは、キュー・マネージャーのエラー・ログに記録されます。サーバーはクライアントとの接続を切断し、クライアントは MqttCallback.connectionLost を呼び出します。

クライアント・アプリケーションと同じスレッドで例外としてスローされる唯一のリモート・エラーは、MqttClient.connect からの例外です。接続の確立後にサーバーによって検出されたエラーは、MqttCallback.connectionLost コールバック・メソッドに throwables として報告されます。

connectionLost の原因となる代表的なサーバー・エラーとして、許可エラーがあります。例えば、トピックに関してパブリッシュが許可されていないクライアントの代わりに、テレメトリー・サーバーがそのトピックに関してパブリッシュを行おうとすることがあります。その場合、MQCC\_FAIL 条件コードをテレメトリー・サーバーに戻す原因となるものはすべて、接続切断の原因となる可能性があります。

### deliveryComplete(MqttDeliveryToken token)

deliveryComplete は、送達トークンをクライアント・アプリケーションに戻すために MQTT クライアントによって呼び出されます。131 ページの『送達トークン』を参照してください。送達トークンの利用により、コールバックはメソッド token.getMessage を使ってパブリッシュ・メッセージにアクセスすることができます。

deliveryComplete メソッドによって呼び出された後、アプリケーション・コールバックが MQTT クライアントに制御を戻すと、送達が完了します。送達が完了するまで、QoS が 1 または 2 に設定されたメッセージはパーシスタンス・クラスによって保存されます。

deliveryComplete の呼び出しは、アプリケーションとパーシスタンス・クラスの間の同期点になります。deliveryComplete メソッドが同じメッセージに対して 2 度呼び出されることはありません。

アプリケーション・コールバックが deliveryComplete から MQTT クライアントに戻ると、クライアントは QoS 1 または 2 のメッセージに対して MqttClientPersistence.remove を呼び出します。MqttClientPersistence.remove は、パブリッシュされたメッセージのローカルで保管されたコピーを削除します。

トランザクション処理の観点から見ると、deliveryComplete の呼び出しは、送達をコミットする単一フェーズのトランザクションです。コールバック中に処理が失敗した場合、クライアントの再始動時に MqttClientPersistence.remove が再び呼び出され、パブリッシュ・メッセージのローカル・コピーを削除します。コールバックは再び呼び出されません。送達されたメッセージのログを保管するためにコールバックを使用する場合、ログと MQTT クライアントを同期することはできません。ログを確実に保管するには、MqttClientPersistence クラスでログを更新してください。

送達トークンとメッセージは、メイン・アプリケーション・スレッドおよび MQTT クライアントによって参照されます。MQTT クライアントは送達の完了時に MqttMessage オブジェクトを間接参照し、クライアントの切断時に送達トークン・オブジェクトを間接参照します。送達が完了した後、MqttMessage オブジェクトに対してガーベッジ・コレクションを実行することができます (クライアント・アプリケーションがそれを間接参照する場合)。セッションが切断された後、送達トークンに対してガーベッジ・コレクションを実行することができます。

メッセージがパブリッシュされた後、MqttDeliveryToken および MqttMessage 属性を取得することができます。メッセージがパブリッシュされた後で MqttMessage 属性の設定を試みた場合、結果は未定義です。

MQTT クライアントは、同じ ClientIdentifier を使用して前のセッションに再接続した場合、送達確認の処理を続行します。129 ページの『クリーン・セッション』を参照してください。MQTT クライアント・アプリケーションは、前のセッションで MqttClient.CleanSession を false に設定し、新しいセッションで false に設定する必要があります。保留中の送達に関して、MQTT クライアントは新規セッションで新しい送達トークンとメッセージ・オブジェクトを作成します。

MqttClientPersistence クラスを使用してオブジェクトをリカバーします。古い送達トークンとメッセージへの参照をアプリケーション・クライアントが依然として保持している場合は、それらを間接参照してください。前回のセッションで開始されてこのセッションで完了する送達がある場合、アプリケーション・コールバックが新規セッションで呼び出されます。

保留中の送達が完了するとき、アプリケーション・クライアントの接続後にアプリケーション・コールバックが呼び出されます。アプリケーション・クライアントは、接続前に MqttClient.getPendingDeliveryTokens メソッドを使って保留中の送達を取り出すことができます。

パブリッシュされるメッセージ・オブジェクトとそのペイロード・バイト配列を、クライアント・アプリケーションが最初に作成したことに注意してください。MQTT クライアントはそれらのオブジェクトを参照します。メソッド token.getMessage で送達トークンによって戻されるメッセージ・オブジェクトは、クライアントによって作成されたメッセージ・オブジェクトと必ずしも同

じではありません。新しい MQTT クライアント・インスタンスが送達トークンを再作成する場合、`MqttClientPersistence` クラスは `MqttMessage` オブジェクトを再作成します。一貫性を保つために、`token.isCompleted` が `true` の場合、メッセージ・オブジェクトがアプリケーション・クライアントと `MqttClientPersistence` クラスのどちらによって作成されたかに関係なく、`token.getMessage` は `null` を返します。

### **messageArrived(MqttTopic topic, MqttMessage message)**

`messageArrived` は、サブスクリプション・トピックに一致したクライアントのパブリケーションが到着したときに呼び出されます。`topic` は、サブスクリプション・フィルターではなくパブリケーション・トピックです。フィルターにワイルドカードが含まれる場合、これら 2 つは異なる可能性があります。

クライアントによって作成された複数のサブスクリプションにトピックが一致する場合、クライアントはパブリケーションの複数コピーを受信します。サブスクライブ先でもあるトピックに対してクライアントがパブリッシュを行うと、クライアントは自分のパブリケーションのコピーを受信します。

QoS が 1 または 2 に設定された状態でメッセージが送信される場合、そのメッセージは MQTT クライアントが `messageArrived` を呼び出す前に `MqttClientPersistence` クラスによって保管されます。`messageArrived` は `deliveryComplete` のように動作します。これは 1 つのパブリケーションにつき一度だけ呼び出され、`messageArrived` が MQTT クライアントに戻ったときに `MqttClientPersistence.remove` によってパブリケーションのローカル・コピーが削除されます。`messageArrived` が MQTT クライアントに戻ると、MQTT クライアントはトピックおよびメッセージへの参照を除去します。アプリケーション・クライアントがオブジェクトへの参照を保持していない場合、トピックおよびメッセージ・オブジェクトに対してガーベッジ・コレクションが実行されます。

## **コールバック、スレッド、およびクライアント・アプリケーションの同期**

MQTT クライアントはメイン・アプリケーション・スレッドとは別のスレッドでコールバック・メソッドを呼び出します。クライアント・アプリケーションはコールバックのスレッドを作成せず、MQTT クライアントによって作成されます。

MQTT クライアントはコールバック・メソッドを同期します。一度に実行されるコールバック・メソッドのインスタンスは 1 つだけです。同期により、どのパブリケーションが送達されたかを記録するオブジェクトの更新が容易になります。実行される `MqttCallback.deliveryComplete` のインスタンスは一度に 1 つなので、追加の同期を行うことなく安全に記録を更新することができます。また、パブリケーションが一度に 1 つだけ到着する場合も同様です。`messageArrived` メソッドのコードでは、同期せずにオブジェクトを更新することができます。別のスレッドで、記録(または更新されるオブジェクト)を参照している場合には、記録またはオブジェクトを同期します。

送達トークンは、メイン・アプリケーション・スレッドとパブリケーションの送達のための同期メカニズムを提供します。メソッド `token.waitForCompletion` は、特定のパブリケーションの送達が完了するか、オプションのタイムアウトが満了するまで待機します。次のように、いくつかの簡単な方法で `token.waitForCompletion` を使用してパブリケーションを一度に 1 つずつ処理できます。

1. パブリケーションの送達が完了するまでアプリケーション・クライアントを一時停止する。  
`PubSync.java` を参照してください。
2. `MqttCallback.deliveryComplete` メソッドと同期する。`MqttCallback.deliveryComplete` が MQTT クライアントに戻るときにのみ、`token.waitForCompletion` が再開します。このメカニズムを使用することで、メイン・アプリケーション・スレッドでコードが実行される前に、`MqttCallback.deliveryComplete` で実行されるコードを同期できます。

それぞれのパブリケーションが送達されるのを待たずにパブリッシュしながら、すべてのパブリケーションが送達されたときに確認するにはどうしたらよいでしょうか? 単一スレッドでパブリッシュを行う場合には、最後に送信されるパブリケーションが最後の送達になります。



## サーバーに送信される要求の同期

メソッド	同期	タイムアウト間隔
<code>MqttClient.Connect</code>	サーバーとの接続が確立されるのを待ちます。	デフォルトの 30 秒、またはパラメーターによる設定値。
<code>MqttClient.Disconnect</code>	MQTT クライアントが行わなければならない処理を完了して TCP/IP セッションが切断されるのを待ちます。	デフォルトの 30 秒、またはパラメーターによる設定値。
<code>MqttClient.Subscribe</code>	サブスクライブ要求が完了するのを待ちます。	デフォルトの 30 秒、またはパラメーターによる設定値。
<code>MqttClient.UnSubscribe</code>	アンサブスクライブ要求が完了するのを待ちます。	デフォルトの 30 秒、またはパラメーターによる設定値。
<code>MqttClient.Publish</code>	MQTT クライアントに要求を渡した後、直ちにアプリケーション・スレッドに戻ります。	なし。
<code>MqttDeliveryToken.waitForCompletion</code>	送達トークンが戻されるのを待ちます。	デフォルトの無期限、またはパラメーターによる設定値。

## クリーン・セッション

MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実にものとするために使用されます。セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

`MqttClient.connect` メソッドを使用して MQTT クライアント・アプリケーションに接続する際、クライアントはクライアント ID とサーバーのアドレスを使って接続を識別します。サーバーは、前回のサーバーへの接続のセッション情報が保存されているかどうかを調べます。前回のセッションがまだ存在し、`cleanSession=true` の場合、クライアント側とサーバー側の前回のセッション情報は消去されます。`cleanSession=false` の場合、前回のセッションが再開されます。前回のセッションが存在しない場合、新規セッションが開始されます。

注: WebSphere MQ Administrator は開かれているセッションを強制的に閉じ、すべてのセッション情報を削除します。クライアントが `cleanSession=false` でセッションを再び開く場合、新規セッションが開始されます。

## パブリケーション

デフォルトの `MqttConnectOptions` を使用するか、クライアントに接続する前に `MqttConnectOptions.cleanSession` を `true` に設定すると、クライアントの接続時にそのクライアントで保留になっているすべてのパブリケーションの送達が削除されます。

クリーン・セッションの設定は、QoS=0 で送信されるパブリケーションには影響を与えません。QoS=1 と QoS=2 の場合、cleanSession=true を使用するとパブリケーションが失われる可能性があります。

## サブスクリプション

クライアントに接続する前に、デフォルトの MqttConnectOptions を使用するか、MqttConnectOptions.cleanSession を true に設定すると、クライアントの古いサブスクリプションはクライアントの接続時に削除されます。セッション中にクライアントによって作成される新規サブスクリプションはすべて、切断時に削除されます。

接続前に MqttConnectOptions.cleanSession を false に設定した場合、クライアントが作成するサブスクリプションは、接続前にクライアントに存在していたすべてのサブスクリプションに追加されます。クライアントが切断する際、サブスクリプションはすべてアクティブのままとなります。

cleanSession 属性がサブスクリプションに与える影響を知る別の方法は、それをモーダル属性と見なすことです。そのデフォルト・モード cleanSession=true では、クライアントはセッションの有効範囲内でのみサブスクリプションを作成し、パブリケーションを受信します。それに代わる cleanSession=false モードでは、サブスクリプションは永続的になります。クライアントは接続および切断を行うことができ、そのサブスクリプションはアクティブのままとなります。クライアントは、再接続時にすべての未送達パブリケーションを受信します。接続中、クライアントはアクティブになっているサブスクリプションのセットを代わりに変更することができます。

接続する前に cleanSession モードを設定する必要があります。このモードは、セッション全体にわたって存続します。その設定を変更するには、クライアントを切断し、再接続する必要があります。

cleanSession=false を使用するモードを cleanSession=true に変更すると、クライアントの以前のサブスクリプション、および受信されていないパブリケーションはすべて破棄されます。

## クライアント ID

クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。クライアント ID は、サーバーに接続するすべてのクライアントで固有である必要があります。サーバー上のキュー・マネージャー名と同じであってはなりません。この制約内で、任意の ID ストリングを使用できます。クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

クライアント ID は、MQTT システムの管理に使用されます。管理対象のクライアントは幾十万にもなる可能性があるため、特定のクライアントを迅速に識別する必要があります。例えば、ある装置に誤動作が生じて、カスタマーがヘルプ・デスクに電話してそれを通知したとします。カスタマーはどのようにして装置を識別し、管理者はどのようにその識別をクライアントが通常接続しているサーバーに対応させるのでしょうか。各装置をクライアント ID およびサーバーにマップするデータベースを調べる必要がありますか。装置の名前から、それが接続するサーバーを識別できますか。MQTT クライアント接続を参照するとき、各接続にはクライアント ID のラベルが付いています。クライアント ID を物理装置にマップするテーブルを検索する必要がありますか。

クライアント ID は、特定の装置、ユーザー、またはそのクライアントで実行しているアプリケーションを識別しますか。カスタマーが障害のある装置を新しいものと置き換えた場合、新しい装置の ID は古い装置のものと同じですか。管理者は新しい ID を割り振りますか。物理装置を変更しても同じ ID を使用する場合、未送信のパブリケーションおよびアクティブなサブスクリプションは新しい装置に自動的に転送されます。

クライアント ID が固有であることをどのように確認しますか。固有の ID を生成するシステムとともに、クライアントに ID を設定するための信頼できるプロセスを持つ必要があります。クライアント装置はユーザー・インターフェースのない「ブラック・ボックス」である可能性があります。装置を製造するときに、MAC アドレスを使用するなどして、クライアント ID を設定しますか。あるいは、装置がアクティブ化される前に装置を構成するソフトウェアのインストールおよび構成のプロセスがありますか。

クライアント ID を 48 ビットの装置 MAC アドレスから作成して、ID を短くしかも固有に保つことができます。伝送サイズが重要な問題でなければ、残りの 17 バイトを使用してアドレスの管理を容易にすることができます。

## 送達トークン

クライアントがトピックにパブリッシュするときに、新しい送達トークンが作成されます。送達トークンは、パブリケーションの送達をモニターしたり、送達が完了するまでクライアント・アプリケーションをブロックしたりするために使用します。

トークンは `MqttDeliveryToken` オブジェクトです。これは `MqttTopic.publish()` メソッドを呼び出すことによって作成されます。クライアント・セッションが切断され、送達が完了するまで MQTT クライアントによって保存されます。

トークンは通常、送達が完了しているかどうかを調べるために使用します。戻されるトークンを使用して `token.waitForCompletion` を呼び出すことにより、送達が完了するまでクライアント・アプリケーションをブロックします。または、`MqttCallback` ハンドラーを提供します。パブリケーションの送達の一部として予期するすべての確認応答を MQTT クライアントが受信すると、クライアントは `MqttCallback.deliveryComplete` を呼び出して送達トークンをパラメーターとして渡します。

送達が完了するまで、戻される送達トークンを使用して `token.getMessage` を呼び出すことにより、パブリケーションを調べることができます。

### 完了送達

送達の完了は非同期で、パブリケーションに関連付けられているサービス品質によって異なります。

#### 最高 1 回

QoS=0

送達は、`MqttTopic.publish` からの戻り時に即時に完了します。  
`MqttCallback.deliveryComplete` が即時に呼び出されます。

#### 最低 1 回

QoS=1

送達は、パブリケーションに対する確認応答をキュー・マネージャーから受信したときに完了します。確認応答を受信すると、`MqttCallback.deliveryComplete` が呼び出されます。通信が遅かったり不安定であったりする場合は、`MqttCallback.deliveryComplete` が呼び出される前にメッセージが複数回送達される可能性があります。

#### 正確に 1 回

QoS=2

パブリケーションがサブスクライバーにパブリッシュされたという完了メッセージをクライアントが受け取ると、送達が完了します。パブリケーション・メッセージを受信した後すぐに `MqttCallback.deliveryComplete` が呼び出されます。完了メッセージを待ちません。

まれに、クライアント・アプリケーションが `MqttCallback.deliveryComplete` から MQTT クライアントに正常に戻らないことがあります。送達が完了したことは、`MqttCallback.deliveryComplete` が呼び出されたことで分かります。クライアントが同じセッションを再始動する場合、`MqttCallback.deliveryComplete` は再び呼び出されません。

### 未完了送達

送達が完了せずにクライアント・セッションが切断された場合、クライアントを再接続して送達を完了することができます。メッセージの送達は、`MqttConnectionOptions` 属性が `false` に設定されたセッションでメッセージがパブリッシュされた場合のみ完了できます。

同じクライアント ID およびサーバー・アドレスを使ってクライアントを作成してから接続します。その際、`cleanSession` `MqttConnectionOptions` 属性は再び `false` に設定します。`cleanSession` を `true` に設定すると、保留中の送達トークンが廃棄されてしまいます。

`MqttClient.getPendingDeliveryTokens` を呼び出すことにより、保留中の送達があるかどうかを調べることができます。`MqttClient.getPendingDeliveryTokens` はクライアントに接続する前に呼び出すことができます。

## 遺言パブリケーション

MQTT クライアント接続が予期せずに終了した場合は、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成できます。パブリケーションの内容とその送信先のトピックを事前定義します。「遺言」は接続プロパティです。遺言はクライアントに接続する前に作成しておきます。

遺言用のトピックを作成します。MQTTManagement/Connections/server URI/client identifier/Lost などのトピックを作成することができます。

MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained) メソッドを使用して、「遺言」をセットアップします。

lastWillPayload メッセージにタイム・スタンプを作成することを検討します。クライアントおよび接続環境の識別に役立つその他のクライアント情報を組み込みます。MqttClient コンストラクターに MqttConnectionOptions オブジェクトを渡します。

lastWillQos を 1 または 2 に設定して、メッセージを IBM WebSphere MQ で永続化し、確実に配信できるようにします。最後に失われた接続情報を保存するには、lastWillRetained を true に設定します。

接続が予期せず終了した場合に、「遺言」パブリケーションがサブスクライバーに送信されます。これは、クライアントが MqttClient.disconnect メソッドを呼び出さずに接続が終了した場合に送信されません。

接続をモニターするには、「遺言」パブリケーションを他のパブリケーションで補完して、接続およびプログラムされた切断を記録します。

## MQTT クライアントでのメッセージ持続性

パブリケーション・メッセージは、「最低 1 回」または「正確に 1 回」のサービス品質で送信される場合に、持続的になります。持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。持続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

MQTT において、メッセージ持続性には 2 つの面があります。メッセージがどのように転送されるか、およびそれが持続メッセージとして MQTT サーバーでキューに入れられるかどうかの 2 つです。

1. MQTT クライアントは、メッセージ持続性とサービス品質とを結合させます。メッセージのために選択するサービス品質に応じて、メッセージは永続的になります。メッセージ永続性は、必要なサービス品質を実装するために必要です。

「最高 1 回」(QoS=0) を指定した場合、クライアントはパブリッシュの直後にメッセージを廃棄します。メッセージのアップストリーム処理に障害が生じた場合、メッセージは再送信されません。クライアントがアクティブのままであっても、メッセージは再送信されません。QoS=0 のメッセージの動作は、IBM WebSphere MQ の高速非持続メッセージと同じです。

メッセージは、QoS が 1 または 2 に指定されてクライアントによってパブリッシュされる場合、持続的になります。そのメッセージはローカルに保管されて、「最低 1 回」(QoS=1) または「正確に 1 回」(QoS=2) の送達を保証する必要がなくなったときにのみ、クライアントから廃棄されます。

2. QoS 1 または 2 のマークが付いている場合、メッセージは持続メッセージとしてキューに入れられます。QoS=0 のマークが付いている場合は、非持続メッセージとしてキューに入れられます。IBM WebSphere MQ では、メッセージ・チャンネルの NPMSPEED 属性が FAST に設定されていない限り、非持続メッセージはキュー・マネージャー間で「正確に 1 回」転送されます。

持続パブリケーションはクライアント・アプリケーションによって受け取られるまでクライアント上に保管されます。QoS=2 の場合、アプリケーションのコールバックから制御が戻ったときに、パブリケーションはクライアントから廃棄されます。QoS=1 の場合、障害が生じたときは、アプリケーションがパブリケーションを再び受け取ることがあります。QoS=0 の場合、コールバックがパブリケーションを受け取る回数は 1 回以内となります。パブリケーション時に障害が生じている場合やクライアントが切断されている場合には、パブリケーションを受け取らないことがあります。

トピックにサブスクライブするときに、サブスクライバーがメッセージを受け取る際の QoS を、その持続性の能力に合うように低くすることができます。より大きな QoS で作成されたパブリケーションは、サブスクライバーが要求したもののの中で最高の QoS で送信されます。

## メッセージの保管

小さな装置にデータ・ストレージを実装する方法は、さまざまに大きく異なります。MQTT クライアントが管理するストレージに持続メッセージを一時保存するモデルは、遅すぎたりストレージ要求が大きすぎたりすることがあります。モバイル・デバイスでは、モバイル・オペレーティング・システムによって MQTT メッセージに適したストレージ・サービスが提供される場合があります。

小型の装置の制約に適合するように柔軟性を提供するために、MQTT クライアントには 2 つの持続性インターフェースがあります。インターフェースは永続メッセージの保管に関連した操作を定義します。これらのインターフェースは、Java 用の MQTT クライアントの API ドキュメンテーションで説明されています。MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、[MQTT クライアント・プログラミング・リファレンス](#)を参照してください。装置に適合するようにインターフェースを実装できます。Java SE で実行される MQTT クライアントには、永続メッセージをファイル・システムに保管するインターフェースのデフォルト実装があります。それは `java.io` パッケージを使用します。クライアントには、Java ME のデフォルト実装 `MqttDefaultMIDPPersistence` もあります。

## パーシスタンス (持続性) クラス

### MqttClientPersistence

`MqttClientPersistence` の実装のインスタンスを、`MqttClient` コンストラクターのパラメーターとして MQTT クライアントに渡します。`MqttClientPersistence` パラメーターを `MqttClient` コンストラクターから省略した場合、MQTT クライアントはクラス `MqttDefaultFilePersistence` または `MqttDefaultMIDPPersistence` を使用して持続メッセージを保管します。

### MqttPersistable

`MqttClientPersistence` は、ストレージ・キーを使用して `MqttPersistable` オブジェクトを取得および配置します。`MqttDefaultFilePersistence` も `MqttDefaultMIDPPersistence` も使用していない場合は、`MqttPersistable` の実装および `MqttClientPersistence` の実装を提供する必要があります。

### MqttDefaultFilePersistence

MQTT クライアントは、`MqttDefaultFilePersistence` クラスを提供します。クライアント・アプリケーションで `MqttDefaultFilePersistence` をインスタンス化する場合、持続メッセージを保管するディレクトリーを `MqttDefaultFilePersistence` コンストラクターのパラメーターとして提供できます。

あるいは、MQTT クライアントが `MqttDefaultFilePersistence` をインスタンス化して、ファイルをデフォルトのディレクトリーに入れることもできます。ディレクトリーの名前は `client identifier-tcp hostname portnumber` です。"`\`", "`\\`", "`/`", "`:`" および "`"`" は、ディレクトリー名ストリングから削除されます。

ディレクトリーへのパスは、システム・プロパティー `rcp.data` の値です。`rcp.data` が設定されていない場合、パスはシステム・プロパティー `usr.data` の値です。

`rcp.data` は、OSGi または Eclipse リッチ・クライアント・プラットフォーム (RCP) のインストールに関連付けられたプロパティーです。

`usr.data` は、アプリケーションを開始した Java コマンドが起動されたディレクトリーです。

### MqttDefaultMIDPPersistence

`MqttDefaultMIDPPersistence` にはデフォルトのコンストラクターがあり、パラメーターはありません。これは `javax.microedition.rms.RecordStore` パッケージを使用してメッセージを保管します。

## パブリケーション

パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

`MqttMessage` には、ペイロードとしてのバイト配列があります。メッセージはできる限り小さくしてください。MQTT プロトコルで許可されるメッセージの最大長は 250 MB です。

通常、MQTT クライアント・プログラムは `java.lang.String` または `java.lang.StringBuffer` を使用してメッセージ・コンテンツを扱います。`MqttMessage` クラスには、ペイロードをストリングに変換するための便利な `toString` メソッドがあります。`java.lang.String` または `java.lang.StringBuffer` からバイト配列のペイロードを作成するには、`getBytes` メソッドを使用します。

`getBytes` メソッドは、ストリングをプラットフォームのデフォルト文字セットに変換します。デフォルト文字セットは、通常 UTF-8 です。テキストだけを含む MQTT パブリケーションは、通常 UTF-8 でエンコードされています。デフォルト文字セットをオーバーライドするには、メソッド `getBytes("UTF8")` を使用します。

IBM WebSphere MQ で、MQTT パブリケーションは `.jms-bytes` メッセージとして受け取られます。メッセージには、`<mqtt>` フォルダーおよび `<mqps>` フォルダーのある `MQRFH2` フォルダーが含まれます。`<mqtt>` フォルダーには `clientId` および `qos` が含まれますが、この内容は将来変更される可能性があります。

`MqttMessage` には 3 つの追加的な属性 (サービス品質、保持されるかどうか、および重複であるかどうか) があります。重複フラグは、サービス品質が「最低 1 回」または「正確に 1 回」のときにだけ設定されます。このメッセージが既に送信されたが、MQTT クライアントによって素早く認知されない場合、重複属性が `true` に設定された状態でそのメッセージが再送信されます。

## パブリッシュ

MQTT クライアント・アプリケーションでパブリケーションを作成するには、`MqttMessage` を作成します。そのペイロード、サービスの品質、および保持されるかどうかを設定して、`MqttTopic.publish(MqttMessage message)` メソッドを呼び出します。`MqttDeliveryToken` が返され、パブリケーションの完了は非同期です。

あるいは、MQTT クライアントがパブリケーションを作成するときに、`MqttTopic.publish(byte [] payload, int qos, boolean retained)` メソッドのパラメーターから一時メッセージ・オブジェクトを作成することもできます。

パブリケーションのサービス品質が「最低 1 回」または「正確に 1 回」、つまり `QoS=1` または `QoS=2` である場合、MQTT クライアントは `MqttClientPersistence` インターフェースを呼び出します。送達トークンをアプリケーションに戻す前に、メッセージを保管するために `MqttClientPersistence` を呼び出します。

`MqttDeliveryToken.waitForCompletion` メソッドを使用することで、アプリケーションは、メッセージがサーバーに送達されるまでブロックするよう選択できます。あるいは、アプリケーションはブロックなしで続行することもできます。ブロックなしでパブリケーションが送達されたかどうかを確認するには、MQTT クライアントに `MqttCallback` を実装するコールバック・クラスのインスタンスを登録します。MQTT クライアントは、パブリケーションが送達された直後に `MqttCallback.deliveryComplete` メソッドを呼び出します。サービス品質に応じて、`QoS=0` であれば送達はほぼ即時に行われ、`QoS=2` であれば幾らかの時間がかかります。

送達が完了したかどうかポーリングするには、`MqttDeliveryToken.isComplete` メソッドを使用します。`MqttDeliveryToken.isComplete` の値が `false` である間は、

`MqttDeliveryToken.getMessage` を呼び出してメッセージの内容を取得できます。

`MqttDeliveryToken.isComplete` を呼び出した結果が `true` の場合、メッセージは既に廃棄されているので、`MqttDeliveryToken.getMessage` を呼び出すと NULL ポインター例外がスローされます。

MqttDeliveryToken.getMessage と MqttDeliveryToken.isComplete とが同期するような機能は組み込まれていません。

保留中の送達トークンをすべて受け取る前にクライアントが切断した場合、クライアントの新しいインスタンスは、接続の前に保留中の送達トークンを照会することができます。クライアントが接続するまで新しい送達は完了しないので、MqttDeliveryToken.getMessage を安全に呼び出すことができます。どのパブリケーションがまだ送達されていないかを検出するには、MqttDeliveryToken.getMessage メソッドを使用します。MqttConnectOptions.cleanSession をデフォルト値 true に設定して接続すると、保留中の送達トークンは廃棄されます。

## サブスクリイブ

キュー・マネージャーまたは IBM MessageSight は、MQTT サブスクライバーに送るパブリケーションを作成します。キュー・マネージャーは、MQTT クライアントによって作成されたサブスクリプション内のトピック・フィルターが、パブリケーション内のトピック・ストリングと一致するかどうかを確認します。この一致は完全一致にすることも、または一致にワイルドカードを含めることもできます。パブリケーションがキュー・マネージャーによってサブスクライバーに転送される前に、キュー・マネージャーは、パブリケーションに関連付けられたトピック属性を調べます。ワイルドカード文字を含むトピック・ストリングを使用したサブスクライブ操作で説明されている検索手順に従い、管理トピック・オブジェクトによってサブスクライブ権限がユーザーに与えられているかどうかを識別します。

MQTT クライアントは、「最低 1 回」のサービス品質でパブリケーションを受け取ると、MqttCallback.messageArrived メソッドを呼び出してパブリケーションを処理します。パブリケーションのサービス品質が「正確に 1 回」つまり QoS=2 である場合、MQTT クライアントは、メッセージを受け取ると MqttClientPersistence インターフェースを呼び出してそれを保管します。その後、MqttCallback.messageArrived を呼び出します。

## MQTT クライアントによって提供されるサービス品質

MQTT クライアントは、WebSphere MQ および MQTT クライアントにパブリケーションを送達するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」の 3 つのサービス品質を提供します。MQTT クライアントが IBM WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

パブリケーションのサービス品質は、MqttMessage の属性です。これはメソッド MqttMessage.setQos によって設定されます。

メソッド MqttClient.subscribe は、トピックのクライアントに送信されるパブリケーションに適用されるサービス品質を下げるすることができます。サブスクライバーに転送されるパブリケーションのサービス品質は、パブリケーションのサービス品質と異なる場合があります。2 つのうちの低い方の値がパブリケーションの転送に使用されます。

### 最高 1 回 QoS=0

メッセージは最高 1 回送信されるか、まったく送信されません。ネットワークでのメッセージの送信は確認応答されません。

メッセージは保管されません。クライアントが切断されるか、サーバーで障害が発生すると、メッセージが失われる可能性があります。

QoS=0 が最も速い転送モードです。これは「応答不要送信」と呼ばれることがあります。

MQTT プロトコルは、QoS=0 でパブリケーションをクライアントに転送するためにサーバーを必要としません。サーバーがパブリケーションを受信したときにクライアントが切断される場合、サーバーによってはパブリケーションが廃棄される可能性があります。テレメトリー (MQXR) サービスは、QoS=0 で送信されるメッセージを廃棄しません。そのメッセージは非持続メッセージとして保管され、キュー・マネージャーが停止する場合にのみ廃棄されます。

### 最低 1 回 QoS=1

QoS=1 はデフォルトの転送モードです。

メッセージは常に、最低 1 回送信されます。送信側が確認応答を受信しない場合、確認応答が受信されるまで、DUP フラグが設定されたメッセージが再送信されます。結果として、受信側に同じメッセージが複数回送信されて、受信側がそれを複数回処理することになる可能性があります。

メッセージは処理されるまで、送信側と受信側でローカルに保管しておく必要があります。

メッセージは、受信側によって処理された後、受信側から削除されます。受信側がブローカーの場合、メッセージはそのサブスクライバーにパブリッシュされます。受信側がクライアントの場合、メッセージはサブスクライバー・アプリケーションに送信されます。メッセージを削除した後、受信側は送信側に確認応答を送信します。

送信側が受信側から確認応答を受信した後、そのメッセージは送信側から削除されます。

## 正確に 1 回

QoS=2

メッセージは常に、正確に 1 回送信されます。

メッセージは処理されるまで、送信側と受信側でローカルに保管しておく必要があります。

QoS=2 は最も安全ですが、最も遅い転送モードです。メッセージを送信側から削除する前に、送信側と受信側の間で少なくとも 2 組の伝送を取ります。最初の伝送の後に受信側のメッセージを処理することができます。

1 組目の伝送で、送信側はメッセージを送信し、メッセージが保管されたという確認応答を受信側から取得します。送信側が確認応答を受信しない場合、確認応答が受信されるまで、DUP フラグが設定されたメッセージが再送信されます。

2 組目の伝送で、送信側はメッセージ "PUBREL" の処理を完了できることを受信側に伝えます。送信側が "PUBREL" メッセージの確認応答を受信しない場合、確認応答が受信されるまで "PUBREL" メッセージが再送信されます。送信側は、"PUBREL" メッセージに対する確認応答を受信すると、保存していたメッセージを削除します。

受信側は、メッセージを再処理しない場合には、最初または 2 番目のフェーズでメッセージを処理することができます。受信側がブローカーの場合、サブスクライバーにメッセージをパブリッシュします。受信側がクライアントの場合、サブスクライバー・アプリケーションにメッセージを送信します。受信側は、メッセージの処理を完了したという完了メッセージを送信側に送り戻します。

## 保存パブリケーションおよび MQTT クライアント

保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

MqttMessage.setRetained メソッドを使用して、トピックのパブリケーションが保存されるかどうかを指定します。

IBM WebSphere MQ で保存パブリケーションを削除するには、**CLEAR TOPICSTRCLEAR TOPICSTR MQSC** コマンドを実行します。

NULL ペイロードでパブリケーションを作成した場合、空のパブリケーションがサブスクライバーに転送されます。その他の MQTT ブローカーは、空のパブリケーションをサブスクライバーに転送しない可能性があります。

保存パブリケーションを持つトピックに非保存パブリケーションをパブリッシュした場合、保存パブリケーションは影響を受けません。現在のサブスクライバーは新しいパブリケーションを受信します。新しいサブスクライバーは、まず保存パブリケーションを受信し、次に新しいパブリケーションがあればそれを受信します。

保存パブリケーションを作成または更新する場合、QoS または 1 または 2 を指定してパブリケーションを送信します。QoS を 0 の値で送信すると、IBM WebSphere MQ は非永続保存パブリケーションを作成します。キュー・マネージャーが停止する場合、パブリケーションは保存されません。

保存パブリケーションを使用して最新の測定値を記録します。保存トピックへの新規サブスクライバーは、即時に最新の測定値を受信します。サブスクライバーが最後にパブリケーション・トピックにサブスクライブした後に新しい測定が行われていない場合、サブスクライバーが再度サブスクライブすると、そのトピックの最新の保存パブリケーションを再び受信します。



## サブスクリプション

トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む1つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。クライアントが再接続される時、パブリケーションがそこに送信されます。

`MqttClient.subscribe` メソッドを使用してサブスクリプションを作成します。その際に、1つ以上のトピック・フィルターおよびサービス品質パラメーターを渡します。サービス品質パラメーターは、サブスクライバーがメッセージを受け取るために使用する用意のある最大のサービス品質を設定します。このクライアントに送るメッセージを、それよりも高いサービス品質で送信することはできません。サービス品質は、メッセージがパブリッシュされたときの元の値およびサブスクリプションに指定されたレベルよりも低く設定されます。メッセージを受け取るためのデフォルトのサービス品質は、`QoS=1` つまり「最低1回」です。

サブスクリプション要求そのものは、`QoS=1` で送信されます。

MQTT クライアントが `MqttCallback.messageArrived` メソッドを呼び出すときに、パブリケーションはサブスクライバーによって受け取られます。`messageArrived` メソッドは、メッセージをサブスクライバーにパブリッシュするときに使用されたトピック・ストリングも渡します。

`MqttClient.unsubscribe` メソッドを使用して、サブスクリプション (またはサブスクリプションのセット) を削除できます。

WebSphere MQ コマンドはサブスクリプションを削除できます。WebSphere MQ エクスプローラーを使用するか、`runmqsc` または `PCF` コマンドを使用して、サブスクリプションをリストします。すべての MQTT クライアントのサブスクリプションの名前が示されます。`ClientIdentifier:Topic name` という形式の名前が付けられます。

クライアントに接続する前に、デフォルトの `MqttConnectOptions` を使用するか、`MqttConnectOptions.cleanSession` を `true` に設定すると、クライアントの古いサブスクリプションはクライアントの接続時に削除されます。セッション中にクライアントによって作成される新規サブスクリプションはすべて、切断時に削除されます。

接続前に `MqttConnectOptions.cleanSession` を `false` に設定した場合、クライアントが作成するサブスクリプションは、接続前にクライアントに存在していたすべてのサブスクリプションに追加されます。クライアントが切断する際、サブスクリプションはすべてアクティブのままとなります。

`cleanSession` 属性がサブスクリプションに与える影響を知る別の方法は、それをモダル属性と見なすことです。そのデフォルト・モード `cleanSession=true` では、クライアントはセッションの有効範囲内でのみサブスクリプションを作成し、パブリケーションを受信します。それに代わる `cleanSession=false` モードでは、サブスクリプションは永続的になります。クライアントは接続および切断を行うことができ、そのサブスクリプションはアクティブのままとなります。クライアントは、再接続時にすべての未送達パブリケーションを受信します。接続中、クライアントはアクティブになっているサブスクリプションのセットを代わりに変更することができます。

接続する前に `cleanSession` モードを設定する必要があります。このモードは、セッション全体にわたって存続します。その設定を変更するには、クライアントを切断し、再接続する必要があります。

`cleanSession=false` を使用するモードを `cleanSession=true` に変更すると、クライアントの以前のサブスクリプション、および受信されていないパブリケーションはすべて破棄されます。

アクティブなサブスクリプションに一致するパブリケーションは、パブリッシュされるとすぐにクライアントに送信されます。クライアントが切断されている場合は、そのクライアントが同じサーバーに同じクライアント ID を使用して再接続すれば、そして `MqttConnectOptions.cleanSession` が `false` に設定されていれば送信されます。

特定のクライアントのサブスクリプションは、クライアント ID によって識別されます。クライアントを別のクライアント装置から同じサーバーに再接続して、同じサブスクリプションの処理を続行し、未送達のパブリケーションを受け取ることもできます。

## MQTT クライアントのトピック・ストリングおよびトピック・フィルター

トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

トピック・ストリングは、パブリケーションをサブスクライバーに送信するために使われます。メソッド `MqttClient.getTopic(java.lang.String topicString)` を使用して、トピック・ストリングを作成します。

トピック・フィルターは、トピックにサブスクライブし、パブリケーションを受信するために使われます。トピック・フィルターにはワイルドカードを含めることができます。ワイルドカードを使用すると、複数のトピックにサブスクライブできます。サブスクリプション方式を使用してトピック・フィルターを作成します (例: `MqttClient.subscribe(java.lang.String topicFilter)`)。

### トピック・ストリング

IBM WebSphere MQ トピック・ストリングの構文については、「[トピック・ストリング](#)」で説明されています。MQTT トピック・ストリングの構文は、Java 用の MQTT クライアント、MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、[MQTT クライアント・プログラミング・リファレンス](#)を参照してください。の API 資料の `MqttClient` クラスで説明されています。

それぞれの種類のトピック・ストリングの構文は、ほとんど同じです。ただし、小さな違いが 4 つあります。

1. MQTT クライアントによって IBM WebSphere MQ に送信されるトピック・ストリングは、キュー・マネージャー名の規則に従う必要があります。特に、トピック・ストリングにはハイフンを含めることができないことに注意してください。
2. 最大長が異なります。IBM WebSphere MQ トピック・ストリングは 10,240 文字に制限されます。MQTT クライアントは、最大 65535 バイトのトピック・ストリングを作成することができます。
3. MQTT クライアントによって作成されるトピック・ストリングには、ヌル文字を含めることができません。
4. WebSphere Message Broker では、ヌルのトピック・レベル `'...//...'` が無効でした。IBM WebSphere MQ ではヌルのトピック・レベルがサポートされています。

IBM WebSphere MQ パブリッシュ/サブスクライブとは異なり、`mqttv3` プロトコルには管理トピック・オブジェクトという概念がありません。トピック・オブジェクトおよびトピック・ストリングからトピック・ストリングを構成することはできません。ただし、トピック・ストリングは WebSphere MQ の管理トピックにマップされます。管理トピックに関連付けられているアクセス制御は、パブリケーションがトピックにパブリッシュされるか、廃棄されるかを決定します。サブスクライバーへの転送時にパブリケーションに適用される属性は、管理トピックの属性の影響を受けます。

### トピック・フィルター

IBM WebSphere MQ トピック・フィルターの構文については、「[トピック・ベースのワイルドカード・スキーム](#)」で説明されています。MQTT クライアントで構成できるトピック・フィルターの構文は、Java 用の MQTT クライアントの API 資料の `MqttClient` クラスで説明されています。MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、[MQTT クライアント・プログラミング・リファレンス](#)を参照してください。

それぞれの種類のトピック・フィルターの構文はほとんど同じです。唯一の違いは、さまざまな MQTT ブローカーがトピック・フィルターを解釈する方法です。WebSphere Message Broker V6 では、マルチレベル・ワイルドカードはトピック・フィルターの末尾でのみ使用可能でした。WebSphere MQ では、マルチレベル・ワイルドカードはトピック・ツリーのあらゆるレベルで使用可能です (例えば `USA/#/Dutchess County`)。

## MQTT クライアントのプログラミング・リファレンス

Mobile Messaging and M2M クライアント・パックおよび関連したクライアント API 文書に対するリンクを以下に示します。

Mobile Messaging and M2M クライアント・パックには、MQTT クライアント・ライブラリーが、生成された API 文書と共に組み込まれています。[IBM メッセージング・コミュニティのダウンロード](#)からクライアント・パックをダウンロードできます。

以下の [Eclipse Paho](#) プロジェクトへのリンクから、最新の API 文書をオンラインで参照できます。

- [Java 用の MQTT クライアントのクラス](#)
- [C 用の MQTT クライアント・ライブラリー](#)
- [C 用の非同期 MQTT クライアント・ライブラリー](#)

注:

1. MQTT Java アプリケーションを `com.ibm.micro.client.mqttv3` ではなく `org.eclipse.paho.client.mqttv3` パッケージにリンクします。パッケージ `com.ibm.micro.client.mqttv3` パッケージは、既存の MQTT Java アプリケーションをサポートするために提供されています。
2. **V7.5.0.1** C 用の MQTT クライアント・アプリケーションを、MQTTClient ライブラリーではなく MQTTAsync ライブラリーにリンクします。C 用の既存の MQTT アプリケーションをサポートするために、MQTTClient が提供されています。
3. JavaScript 用の MQTT メッセージング・クライアントには、WebSockets をサポートする MQTT サーバーが必要です。例えば、IBM WebSphere MQ Version 7.5 以降のバージョンが該当します。

## MQTT サーバーの概要

MQTT トランスポート・プロトコルをサポートするメッセージング・サーバーは、IBM および他の会社から入手可能です。最も基本的な MQTT サーバーでは、メッセージを交換するための (MQTT クライアント・ライブラリーでサポートされる) モバイル・アプリケーションおよびデバイスが使用可能になります。IBM WebSphere MQ および IBM MessageSight は、IBM の MQTT サーバーです。これらは、基本的な MQTT サーバーとして機能するだけでなく、MQTT クライアント・アプリケーションおよびエンタープライズ・アプリケーション間のメッセージ交換も行います。IBM から提供されるすべての MQTT サーバーは、MQTT version 3.1 プロトコル、および WebSocket protocol を介した MQTT をサポートします。

### IBM から現在提供されている MQTT サーバー

#### **IBM WebSphere MQ**

- IBM WebSphere MQ ではエンタープライズ・グレードのメッセージングが提供されます。テレメトリ・コンポーネントは、IBM WebSphere MQ を有効にし、MQTT サーバーとしても機能させます。
- これにより、モバイル、M2M (machine-to-machine) およびデバイス・ベースのアプリケーションがサポートされ、これらのアプリケーションで、IBM WebSphere MQ および JMS アプリケーションなどのエンタープライズ・メッセージング・アプリケーションとメッセージを交換できるようになります。
- IBM WebSphere MQ のインストールには、IBM からの MQTT SDK のコピーが含まれています。この SDK では、サンプル MQTT クライアント・アプリケーション、およびこれらのアプリケーションをサポートする MQTT クライアント・ライブラリーが提供されます。

注: この SDK の最新バージョンを取得するには、[Mobile Messaging and M2M クライアント・パック](#) をダウンロードします。詳しくは、[11 ページの『MQTT クライアントの概要』](#)を参照してください。

- MQTT サポートは当初、IBM WebSphere MQ Version 7.0.1 に含まれていました。IBM WebSphere MQ のリリースごとの詳細については、以下の製品資料を参照してください。

- [WebSphere MQ Telemetry バージョン 7.5](#)
- [WebSphere MQ Telemetry バージョン 7.1](#)

IBM WebSphere MQ の概要および IBM WebSphere MQ Telemetry コンポーネントでの作業を開始する手順については、[141 ページの『MQTT サーバーとしての IBM WebSphere MQ』](#)を参照してください。

### **IBM MessageSight**

- IBM MessageSight はアプライアンス・ベースの MQTT サーバーです。これにより、非常に多くの MQTT クライアントを同時に接続することができ、増え続ける多数のモバイル・デバイスとセンサーに対応するために必要なパフォーマンスとスケーラビリティが提供されます。MQTT version 3.1 プロトコル、および WebSocket protocol を介した MQTT がサポートされます。



- IBM MessageSight の MQTT サーバーとしてのメイン・フィーチャーおよび利点は次のとおりです。
  - ハイパフォーマンス、信頼性、およびスケーラブル・メッセージング。
  - 同時に接続されたエンドポイントの多数のコミュニティをサポートすることによる、M2M (machine-to-machine) および Internet of Things シナリオ専用の設計。
  - インストールと使用が容易。30 分足らずで稼働させることができます。
  - Android および iOS を含むネイティブ・モバイル・アプリケーションをサポート。
  - パブリッシュ/サブスクライブ・ブローカーとしての IBM WebSphere MQ との統合。
- [IBM MessageSight の簡単な紹介](#)については、[MessageSight introduction \(YouTube\)](#) および [MessageSight announcement](#) を参照してください。詳しい技術情報については、[MessageSight の製品資料](#)を参照してください。

### **IBM WebSphere MQ Telemetry daemon for devices**

- これは IBM WebSphere MQ Telemetry advanced client for C とも呼ばれます。これは小フットプリントの MQTT サーバーであり、通常は、セット・トップ・ボックス、リモート・テレメトリー・ユニット、POS 端末など、ネットワークの端に近い位置にあるサテライト・ロケーションやデバイスで実行されます。
- 標準的な用途は、多数の MQTT クライアント接続を集中させることです。これらの接続は、単一の MQTT 接続でインターネット経由で IBM WebSphere MQ に接続されます。例えば、1つのビル内に多数のセンサーを取り付け、それらを IBM WebSphere MQ Telemetry daemon for devices に接続し、デーモンを IBM WebSphere MQ に接続することが考えられます。
- IBM WebSphere MQ Telemetry daemon for devices は IBM WebSphere MQ に含まれています。これを IBM WebSphere MQ に接続するには、別個のライセンスが必要です。[IBM United States Software Announcement 212-091](#) を参照してください。

### **Really Small Message Broker**

- Really Small Message Broker (RSMB) は IBM WebSphere MQ Telemetry daemon for devices のバージョンの 1 つです。主な違いは使用法です。RSMB は IBM alphaWorks® から入手可能な小規模のテスト・サーバーであり、MQTT ベース・ソリューションの評価時または試行時に使用するためのものです。RSMB は、多くの Linux プラットフォーム、Windows XP、Apple Mac OS X Leopard、および Unslung (Linksys NSLU12) で MQTT をサポートします。

## IBM から以前提供されていた MQTT サーバー

### WebSphere Message Broker (現在は *IBM Integration Bus* と呼ばれる)

- WebSphere Message Broker バージョン 6 はこの製品自体の MQTT サーバーを提供していました。このサポートは、WebSphere Message Broker バージョン 7 で IBM WebSphere MQ のテレメトリー・コンポーネントに置き換えられました。

## その他の MQTT サーバー

MQTT.org の Software ページには、オープン・ソースのサーバーも含め、MQTT サーバーおよびブローカーのリストが維持されています。

### 関連タスク

#### 11 ページの『MQTT クライアントの概要』

MQTT クライアント・ライブラリーを使用するサンプル MQTT クライアント・アプリケーションを作成して実行することにより、モバイル・アプリケーションまたは M2M (machine-to-machine) アプリケーションの開発を開始できます。サンプル・アプリケーションおよび関連するクライアント・ライブラリーは、IBM の Mobile Messaging and M2M クライアント・パックで入手できます。Java、JavaScript、および C で作成されたアプリケーションとクライアント・ライブラリーのバージョンがあります。これらのアプリケーションは、Apple の Android デバイスや製品など、ほとんどのプラットフォームおよびデバイスで実行できます。

## MQTT サーバーとしての IBM WebSphere MQ

IBM WebSphere MQ に含まれている MQTT サーバーの使用の概要。

作業を開始する場合は、以下の記事に示されているステップに従ってください。

- 141 ページの『のインストール IBM WebSphere MQ』
- 143 ページの『コマンド・ラインからの MQTT サービスの構成』
- 146 ページの『IBM WebSphere MQ Explorer を使用した MQTT サービスの構成』

注：コマンド行インターフェースの例を使用することで、作業を迅速に開始することができます。ただし、ご使用の構成が例と大幅に異なる場合は、コマンド行インターフェースを効果的に使用するための知識とスキルがさらに必要になります。IBM WebSphere MQ Explorer インターフェースは、作業を開始する場合と、標準的な構成タスクを簡単に行う場合の両方に使用します。

IBM WebSphere MQ Telemetry コンポーネントに関する重要な概念情報については、IBM WebSphere MQ 製品資料で以下の記事を参照してください。

- キュー・マネージャーへの遠隔測定装置の接続
- 遠隔測定 (MQXR) サービス
- 遠隔測定チャンネル

### 関連情報

Linux および AIX でのテレメトリー用のキュー・マネージャーの構成

Windows 上のテレメトリー用キュー・マネージャーの構成

メッセージを MQTT クライアントに送信するように分散キューイングを構成

WebSphere MQ Telemetry の管理

## のインストール IBM WebSphere MQ

Windows または Linux 上で IBM WebSphere MQ を取得してインストールし、IBM WebSphere MQ Telemetry を構成するには、以下の手順に従います。

## 始める前に

IBM WebSphere MQ 上で実行される MQTT サービスによってサポートされるオペレーティング・システムについては、IBM WebSphere MQ Telemetry のシステム要件を参照してください。

IBM WebSphere MQ インストール素材のコピーとライセンスを、次のいずれかの方法で取得します。

1. IBM WebSphere MQ 管理者にインストール素材を依頼し、使用許諾契約書に同意してよいか確認します。
2. IBM WebSphere MQ の 90 日間の評価コピーを取得します。 [評価: IBM WebSphere MQ](#) を参照してください。
3. IBM WebSphere MQ を購入します。 [IBM WebSphere MQ 製品ページ](#) を参照してください。

## このタスクについて

Linux では root として、Windows では管理者として IBM WebSphere MQ をインストールします。インストール時に、追加オプションの「テレメトリー・サービス」と「テレメトリー・クライアント」を選択して、IBM WebSphere MQ Telemetry コンポーネントをインストールします。IBM WebSphere MQ を管理するユーザー ID を作成し、ゲスト・ユーザー ID が定義されていることを確認します。ゲスト・ユーザー ID は、IBM WebSphere MQ への MQTT アクセスを許可するために、サンプル MQTT サービス構成で使用されます。

IBM WebSphere MQ のインストール後に MQTT サービスを始動するため、[143 ページ](#)の『[コマンド・ラインからの MQTT サービスの構成](#)』または [146 ページ](#)の『[IBM WebSphere MQ Explorer を使用した MQTT サービスの構成](#)』の手順を実行します。

## 手順

1. Linux の場合は root として、Windows の場合は管理者としてログオンします。
2. IBM WebSphere MQ をインストールします。

「[Installing WebSphere MQ server on Linux](#)」または「[Installing WebSphere MQ server on Windows](#)」の指示に従ってください。「テレメトリー・サービス」と「テレメトリー・クライアント」を選択して、IBM WebSphere MQ Telemetry コンポーネントをインストールします。

Linux では、「次の作業」セクションの手順をメモして、インストール済み環境をプライマリーにします。当該インストール済み環境がご使用のワークステーションで唯一の IBM WebSphere MQ インストール済み環境であっても、これをプライマリーにします。[プライマリー・インストールとして構成されている WebSphere MQ バージョン 7.1 以降の単一インストール](#)を参照してください。

サンプル構成の指示に正確に従うためには、インストール済み環境をプライマリーにすることが必要です。

**複数のインストール:** プライマリー以外のインストール済み環境で処理を行う場合、`setmqenv` コマンドを実行します。これにより、ワークステーションのコマンド・ウィンドウで IBM WebSphere MQ 環境がセットアップされます。[複数のインストール](#)を参照してください。

インストール・プログラムによって示されるデフォルトのインストール場所を受け入れた場合、IBM WebSphere MQ は以下のディレクトリーにインストールされます。

### Linux 64 ビット

```
/opt/mqm
```

### Windows 32 ビット

```
C:\Program Files\IBM\WebSphere MQ
```

### Windows 64 ビット

```
C:\Program Files (x86)\IBM\WebSphere MQ
```

インストール・ディレクトリーは `MQ_INSTALLATION_PATH` として示します。

3. オプション: IBM WebSphere MQ の管理に使用する予定のユーザーを、使用するワークステーションの `mqm` グループに追加します。

IBM WebSphere MQ は Windows 管理者として管理できるため、このステップは Windows ではオプションです。 [UNIX および Windows システム上の WebSphere MQ を管理する権限](#)を参照してください。

Windows ワークステーションがドメインのメンバーである場合は、[デフォルト以外のセキュリティー権限を持つ Windows 2000 ドメインとデフォルト・セキュリティー権限を持つ Windows 2003 および Windows Server 2008 ドメイン](#)を参照してください。

Linux では、インストール・プログラムにより、グループ mqm のメンバーとしてユーザー mqm が作成されます。このユーザーにパスワードを指定するか、または、別のユーザーを作成してその 1 次グループを mqm にしてください。

4. オプション: mqm グループのメンバーにしたユーザーでサインオンします。

IBM WebSphere MQ は Windows 管理者として管理できるため、このステップは Windows ではオプションです。

5. ワークステーションにゲスト・ユーザー ID が定義されていることを確認します。

ゲスト・ユーザー ID は、Windows では "guest"、Linux では "nobody" です。ゲスト・ユーザー ID にはオペレーティング・システムの許可あるいは権限は必要ありません。

## タスクの結果

これで、ワークステーションに IBM WebSphere MQ がプライマリー IBM WebSphere MQ インストール済み環境としてインストールされ、グループ mqm が作成されました。インストールによって、IBM WebSphere MQ を管理するアクセス権が mqm グループのメンバーに与えられます。Windows の Administrators グループのメンバーにも IBM WebSphere MQ を管理する権限があります。

## 次のタスク

1. MQTT サービスを、コマンド・ラインまたは IBM WebSphere MQ Explorer から構成します。 [146 ページの『IBM WebSphere MQ Explorer を使用した MQTT サービスの構成』](#) または [143 ページの『コマンド・ラインからの MQTT サービスの構成』](#) を参照してください。
2. Android、iOS、WebSockets、Java、および "C" MQTT クライアントをテストします。
3. テストが完了したら、コマンド `MQ_INSTALLATION_PATH\mqxr\samples\CleanupMQM.bat` (Windows の場合) および `MQ_INSTALLATION_PATH/mqxr/samples/CleanupMQM.sh` (Linux の場合) を実行して、キュー・マネージャーと MQTT サービスを削除します。

## 関連情報

[WebSphere MQ Telemetry のインストール](#)

[Linux での WebSphere MQ サーバーのインストール](#)

[Windows での WebSphere MQ サーバーのインストール](#)

## コマンド・ラインからの MQTT サービスの構成

コマンド行を使用してサンプル IBM WebSphere MQ Telemetry アプリケーションを実行するには、以下の手順に従って IBM WebSphere MQ を構成します。このステップでは、MQXR\_SAMPLE\_QM という名前の新しいキュー・マネージャーで MQTT サービスを作成するスクリプトを実行する方法を示します。

## 始める前に

MQTT サービスをセットアップするには、IBM WebSphere MQ キュー・マネージャーに対する管理アクセス権限が必要です。キュー・マネージャーへのアクセス権限を得るにはいくつかの方法があります。

1. IBM WebSphere MQ のコピーを取得して、使用する Linux または Windows ワークステーションにキュー・マネージャーを作成します。 [141 ページの『のインストール IBM WebSphere MQ』](#) の指示に従って、IBM WebSphere MQ を取得してインストールします。インストール時には「テレメトリー・サービス」と「テレメトリー・クライアント」も選択する必要がある点に注意してください。既存のインストール済み環境を変更してこれらのオプションを追加することもできます。

2. IBM WebSphere MQ 管理者に連絡して、オプションとして IBM WebSphere MQ Telemetry をインストールしたサーバー上のキュー・マネージャーに対する管理アクセス権限の付与を依頼します。

**V7.5.0.1** キュー・マネージャーの名前に加え、WebSockets を介した MQTT 用と MQTT 用に少なくとも 2 つの TCP/IP ポートが必要です。セキュア・クライアントへの接続を計画している場合、さらに少なくとも 2 つのポートが必要です。

説明どおりに正確にタスクの手順を実行するには、MQXR\_SAMPLE\_QM というキュー・マネージャーの作成が可能になっている必要があります。また、TCP/IP ポート 1883 が未使用でなければなりません。

## このタスクについて

このタスクでは、キュー・マネージャーを作成するスクリプトを実行し、ポート 1883 で MQTT V3.1 クライアント接続を listen するように MQTT サービスを構成します。この構成により、すべてのユーザーに任意のトピックのパブリッシュとサブスクライブを行う許可が与えられます。セキュリティ構成とアクセス制御は最小限であり、制限されたアクセスを行うセキュア・ネットワーク上のキュー・マネージャーのみを対象としています。非セキュア環境で IBM WebSphere MQ と MQTT を実行するには、セキュリティの構成が必要です。IBM WebSphere MQ と MQTT のセキュリティを構成するには、このタスクの末尾にある関連リンクを参照してください。

## 手順

1. IBM WebSphere MQ に対する管理権限を持つユーザー ID でログオンします。

IBM WebSphere MQ に対する管理権限を持つユーザー ID を定義するには、[141 ページの『のインストール IBM WebSphere MQ』のステップ 3](#) を参照してください。

2. コマンド・ウィンドウを開いてサンプル・コマンド・スクリプトを実行し、MQXR\_SAMPLE\_QM というサンプル・キュー・マネージャーと MQTT サービスを作成し、開始します。

サンプル・スクリプトのパスは、%MQ\_FILE\_PATH%\mqxr\samples\SampleMQM.bat (Windows の場合) および MQ\_INSTALLATION\_PATH/mqxr/samples/SampleMQM.sh (Linux の場合) です。

次のコマンドを入力して、キュー・マネージャーを作成して構成します。

### Windows

```
"%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
```

### Linux

```
MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
```

## タスクの結果

このサンプルは、Windows 上に、以下のプロパティを持つ PlainText という名前の MQTT チャネルを作成します。

```
com.ibm.mq.MQXR.channel/PlainText: \  
com.ibm.mq.MQXR.Protocol=MQTT;\br/>com.ibm.mq.MQXR.Port=1883;\br/>com.ibm.mq.MQXR.Backlog=4096;\br/>com.ibm.mq.MQXR.UserName=Guest;\br/>com.ibm.mq.MQXR.StartWithMQXRService=true
```

Linux でのチャネル・プロパティも Windows と同様ですが、com.ibm.mq.MQXR.UserName=nobody とします。

ポート 1883 に接続する MQTT V3.1 クライアントは、変数 com.ibm.mq.MQXR.UserName に設定されたユーザー ID を使用して IBM WebSphere MQ にアクセスします。サンプル・スクリプトは次の IBM WebSphere MQ コマンドを使用してこのユーザー ID に権限を与えます。

```
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
```

1 番目のコマンドは、ベース・トピックからアクセス権を継承したトピックに対してパブリッシュとサブスクライブを行う権限を、ユーザーに与えます。2 番目のコマンドは、SYSTEM.MQTT.TRANSMIT.QUEUE 伝送キューにメッセージを書き込む権限を、ユーザーに与えます。MQTT サービスは SYSTEM.MQTT.TRANSMIT.QUEUE 上のメッセージを MQTT サブスクライバーにパブリケーションとして送信します。

スクリプトはキュー・マネージャー上の MQTT サービスを開始し、ポート 1883 上の接続を listen します。

## 次のタスク

以下の手順に従ってサンプル MQTT V3.1 Java アプリケーションを実行することにより、接続をテストします。

サンプル Java アプリケーションのソースは、MQTTV3Sample.java ファイルにあります。

サンプルの実行には 2 つのコマンド・ウィンドウが必要です。サンプルを、1 つのウィンドウではサブスクライバーとして、もう 1 つのウィンドウではパブリッシャーとして実行します。

- **Windows** サブスクライバーを開始するため次のコマンドを実行します。

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
```

パブリッシュを行うため次のコマンドを実行します。

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
```

- **Linux** サブスクライバーを開始するため次のコマンドを実行します。

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
```

パブリッシュを行うため次のコマンドを実行します。

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
```

パブリッシャーとサブスクライバーは、出力をそれぞれのコマンド・ウィンドウに書き込みます。

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

図 27. パブリッシャーからの出力

```
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
```

図 28. サブスクライバーからの出力

これで、MQTT V3.1 アプリケーションをテストするためのサーバーの準備ができました。

## 関連タスク

[WebSphere MQ エクスプローラーによる MQTT サービスの構成](#)

IBM WebSphere MQ Explorer を使用してサンプル IBM WebSphere MQ Telemetry クライアントを実行するように IBM WebSphere MQ を構成するには、以下の手順に従います。このステップでは、Define sample 構成ウィザードを実行して MQTT サービスを作成する方法を示します。

## 関連情報

[WebSphere MQ Telemetry](#)

[WebSphere MQ Telemetry のアプリケーションの開発](#)

[WebSphere MQ Telemetry の管理](#)

[WebSphere MQ Telemetry におけるセキュリティー](#)

## IBM WebSphere MQ Explorer を使用した MQTT サービスの構成

IBM WebSphere MQ Explorer を使用してサンプル IBM WebSphere MQ Telemetry クライアントを実行するように IBM WebSphere MQ を構成するには、以下の手順に従います。このステップでは、Define sample 構成ウィザードを実行して MQTT サービスを作成する方法を示します。

### 始める前に

MQTT サービスをセットアップするには、IBM WebSphere MQ キュー・マネージャーに対する管理アクセス権限が必要です。キュー・マネージャーへのアクセス権限を得るにはいくつかの方法があります。

1. IBM WebSphere MQ のコピーを取得して、使用する Linux または Windows ワークステーションにキュー・マネージャーを作成します。[141 ページの『のインストール IBM WebSphere MQ』](#)の指示に従って、IBM WebSphere MQ を取得してインストールします。インストール時には「テレメトリー・サービス」と「テレメトリー・クライアント」も選択する必要がある点に注意してください。既存のインストール済み環境を変更してこれらのオプションを追加することもできます。
2. IBM WebSphere MQ 管理者に連絡して、オプションとして IBM WebSphere MQ Telemetry をインストールしたサーバー上のキュー・マネージャーに対する管理アクセス権限の付与を依頼します。  
**V7.5.0.1** キュー・マネージャーの名前に加え、WebSockets を介した MQTT 用と MQTT 用に少なくとも 2 つの TCP/IP ポートが必要です。セキュア・クライアントへの接続を計画している場合、さらに少なくとも 2 つのポートが必要です。

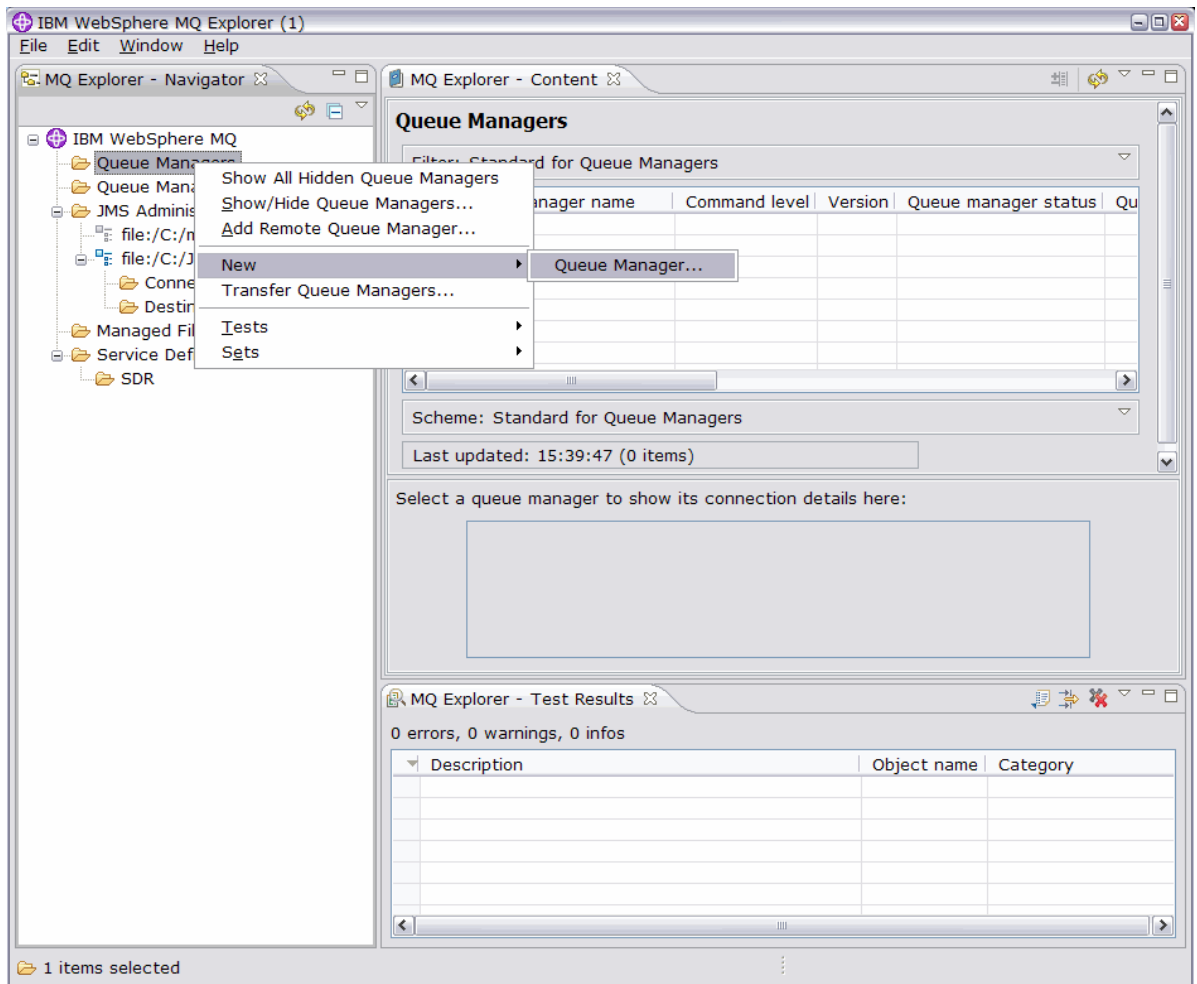
説明どおりに正確にタスクの手順を実行するには、MQXR\_SAMPLE\_QM というキュー・マネージャーの作成が可能になっている必要があります。また、TCP/IP ポート 1883 が未使用でなければなりません。

### このタスクについて

このタスクでは、IBM WebSphere MQ Explorer Define sample 構成ウィザードを実行して、ポート 1883 で MQTT V3.1 クライアント接続を listen する MQTT サービスを作成します。この構成により、すべてのユーザーに任意のトピックのパブリッシュとサブスクライブを行う許可が与えられます。セキュリティー構成とアクセス制御は最小限であり、制限されたアクセスを行うセキュア・ネットワーク上のキュー・マネージャーのみを対象としています。非セキュア環境で IBM WebSphere MQ と MQTT を実行するには、セキュリティーの構成が必要です。IBM WebSphere MQ と MQTT のセキュリティーを構成するには、このタスクの末尾にある関連リンクを参照してください。

### 手順

1. IBM WebSphere MQ に対する管理権限を持つユーザー ID でログオンします。  
IBM WebSphere MQ に対する管理権限を持つユーザー ID を定義するには、[141 ページの『のインストール IBM WebSphere MQ』](#)のステップ 3 を参照してください。
2. コマンド・ウィンドウを開き、IBM WebSphere MQ Explorer コマンド **strmqcfg** を実行して IBM WebSphere MQ Explorer を開始します。
3. キュー・マネージャーの作成
  - a) 「新規キュー・マネージャー」ウィザードを開始します。



- b) 「キュー・マネージャー名」と「送達不能キュー」の名前を入力します。便宜上、これをデフォルトのキュー・マネージャーにしてください。終了をクリックする

**Create Queue Manager**

**Queue Manager**  
Enter basic values

Queue manager name: \* MQXR\_SAMPLE\_QM

Make this the default queue manager

Default transmission queue:

Dead-letter queue: SYSTEM.DEAD.LETTER.QUEUE

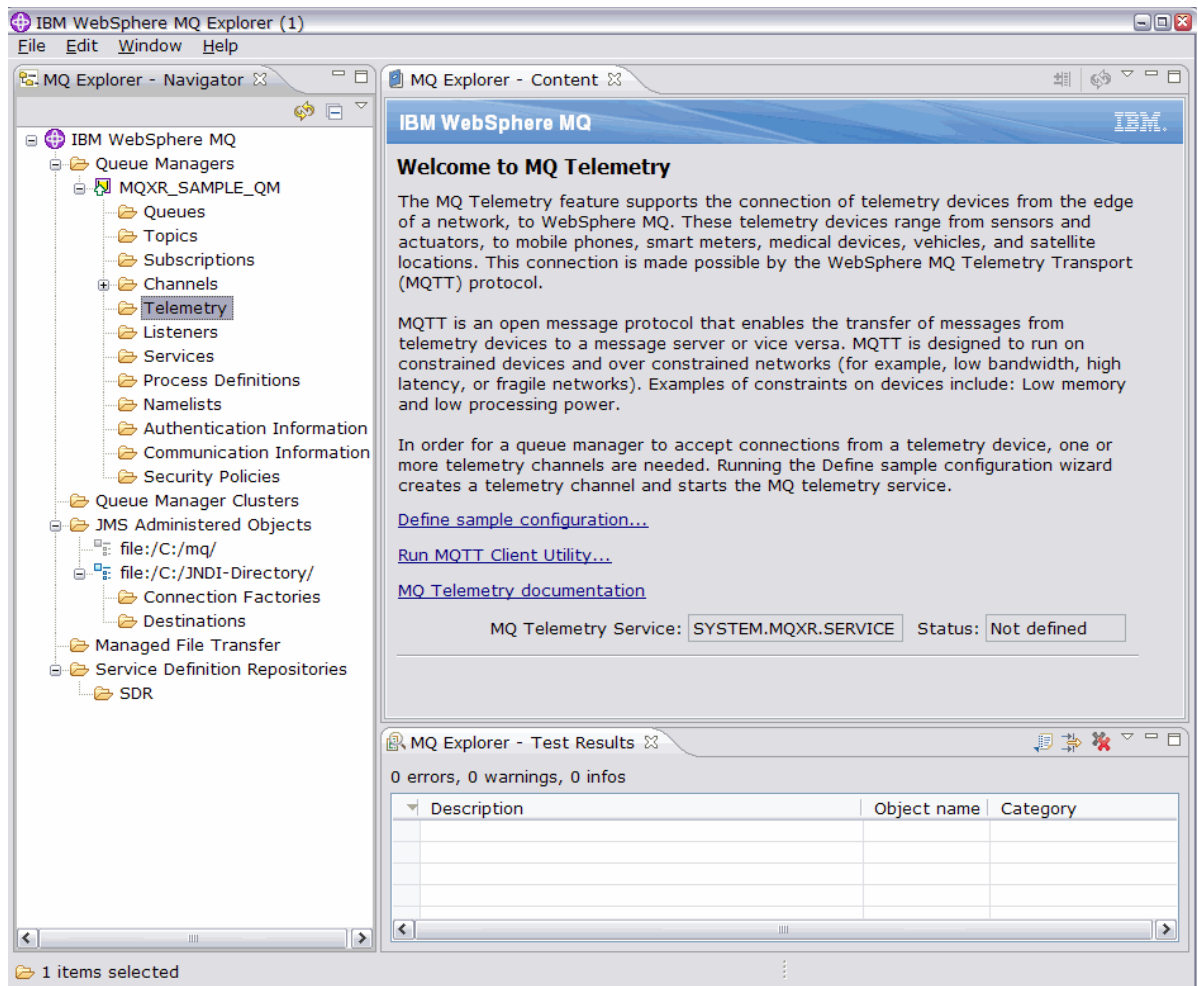
Max handle limit: 256

Trigger interval: 999999999

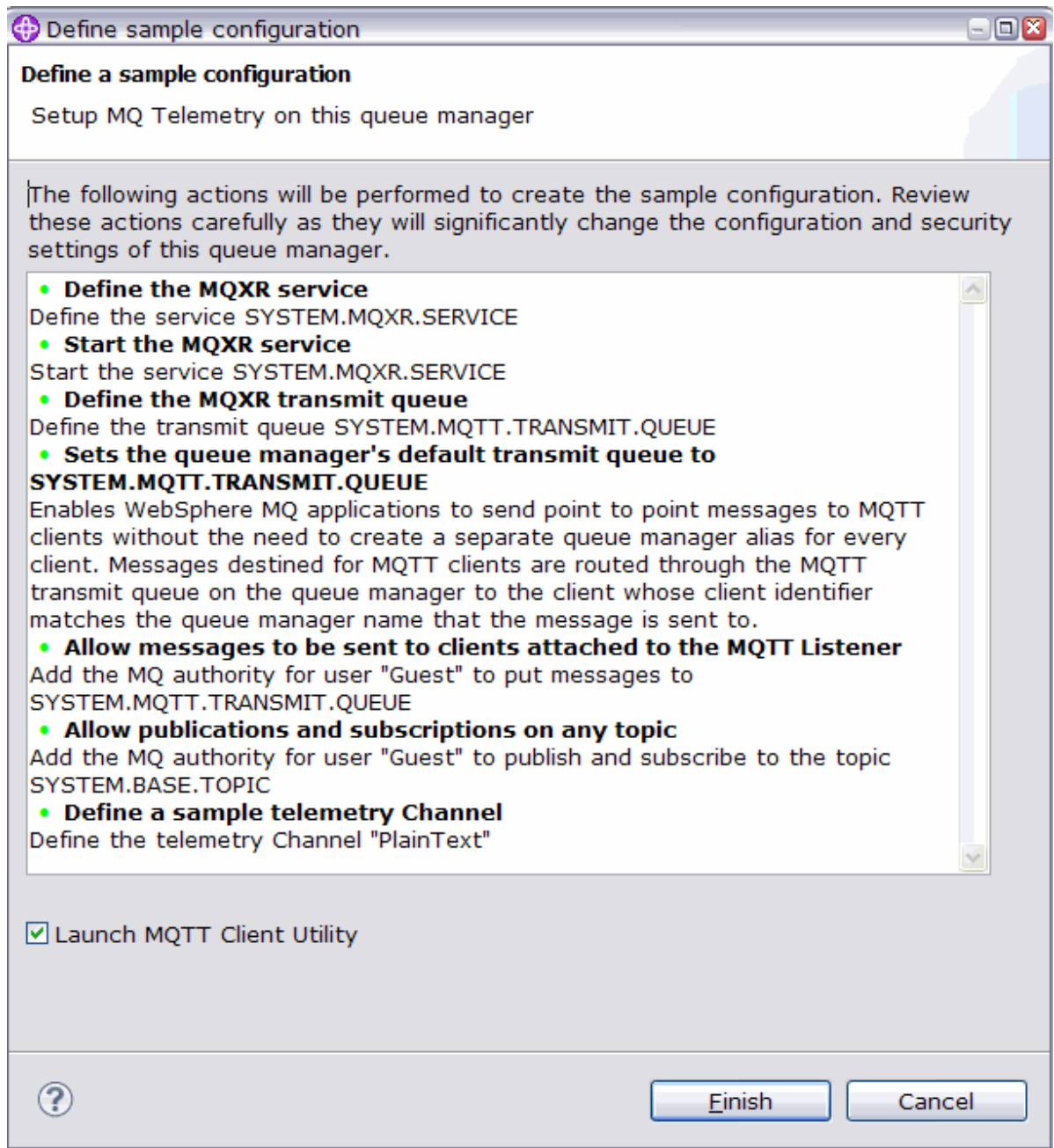
Max uncommitted messages: 10000

? < Back Next > Finish Cancel

- IBM WebSphere MQ Explorer によってキュー・マネージャーが作成され、開始されます。
4. Telemetry 「サンプル構成の定義」ウィザードを実行します。
    - a) キュー・マネージャーの「テレメトリー」フォルダーを開きます。

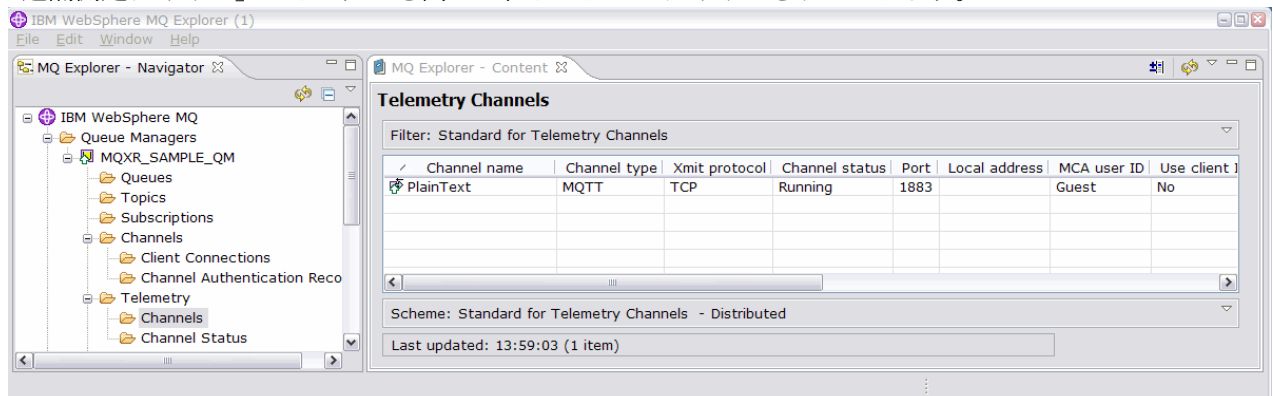


- b) 「サンプル構成の定義」をクリックしてウィザードを開始します。
- c) 「完了」をクリックして、Telemetry サービスを作成し、MQTT クライアント・ユーティリティーを実行します。



## タスクの結果

「遠隔測定チャンネル」フォルダーを開いて、サンプル・チャンネルをリストします。



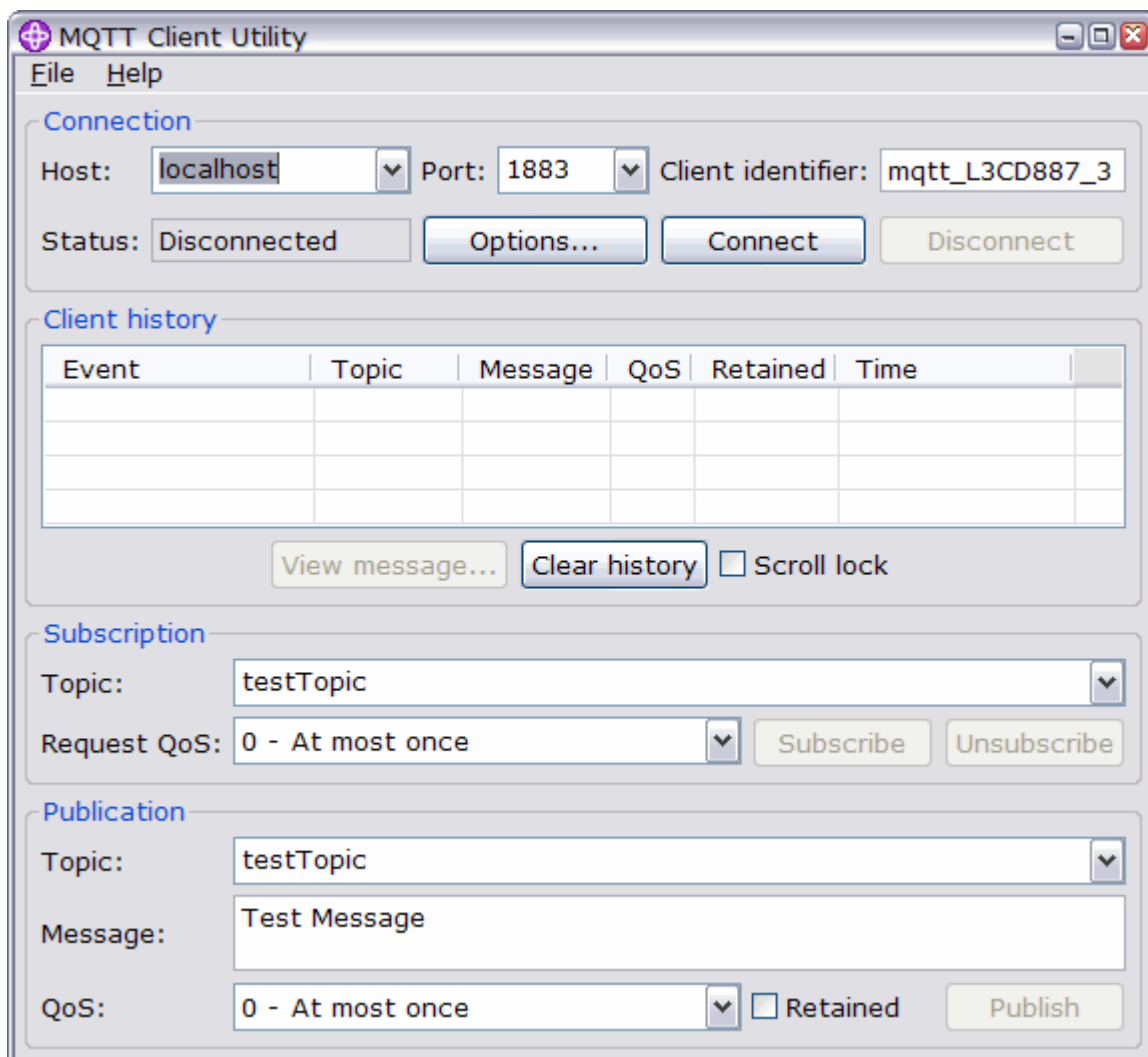
このウィンドウで、このチャンネルのプロパティーの変更、およびチャンネルの追加と削除を行うことができます。

## 次のタスク

MQTT クライアント・ユーティリティーを実行することにより、接続をテストします。

1. クライアント・ユーティリティーを開始するには、「テレメトリー」フォルダーを開いて、「MQTT クライアント・ユーティリティーを実行」を2回クリックします。

「MQTT クライアント・ユーティリティー」ウィンドウが2つ開きます。同じウィンドウですが、クライアント ID が異なっています。



2. 両方のウィンドウで「接続」をクリックします。
3. 両方のウィンドウで「サブスクライブ」をクリックします。
4. どちらかのウィンドウで「パブリッシュ」をクリックします。結果を [152 ページの図 29](#) に示します。

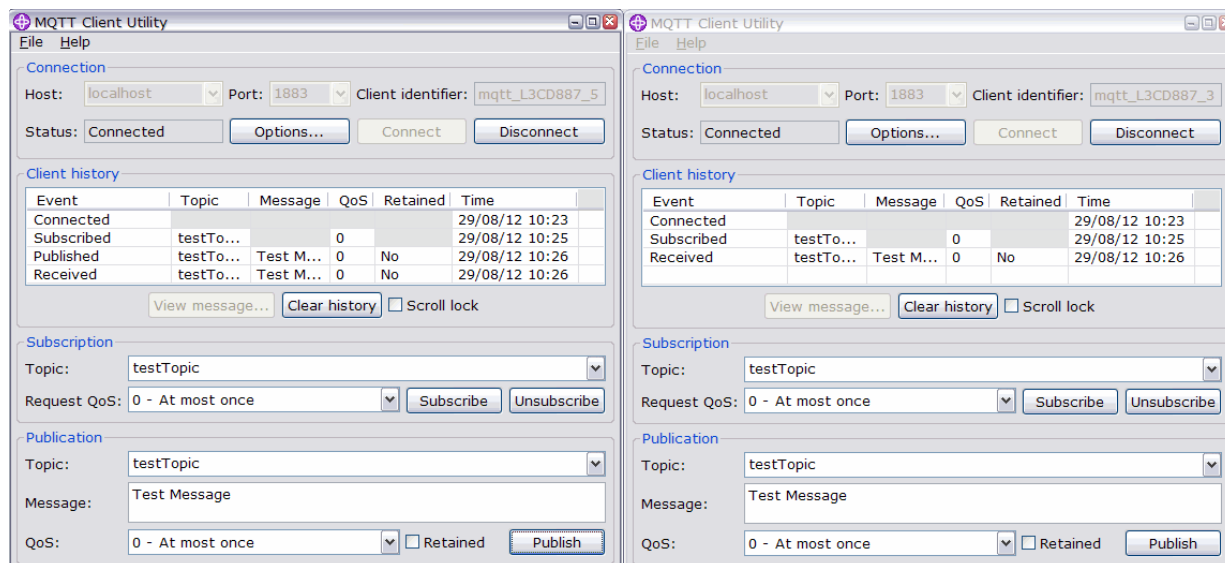


図 29. 結果

5. 両方のウィンドウで「切斷」をクリックします。

これで、MQTT V3.1 アプリケーションをテストするためのサーバーの準備ができました。

#### 関連タスク

[コマンド・ラインからの MQTT サービスの構成](#)

コマンド行を使用してサンプル IBM WebSphere MQ Telemetry アプリケーションを実行するには、以下の手順に従って IBM WebSphere MQ を構成します。このステップでは、MQXR\_SAMPLE\_QM という名前の新しいキュー・マネージャーで MQTT サービスを作成するスクリプトを実行する方法を示します。

[WebSphere MQ Telemetry の管理](#)

#### 関連情報

[WebSphere MQ Telemetry](#)

[WebSphere MQ エクスプローラーを使用した WebSphere MQ Telemetry の管理](#)

[WebSphere MQ Telemetry のアプリケーションの開発](#)

[機密保護](#)

[WebSphere MQ Telemetry におけるセキュリティー](#)

## IBM WebSphere MQ Telemetry デーモン(デバイス用)の概念

IBM WebSphere MQ Telemetry デーモン(デバイス用)は、拡張 MQTT V3 クライアント・アプリケーションです。これは、他の MQTT クライアントからのメッセージをストア・アンド・フォワードするために使用します。MQTT クライアントのように IBM WebSphere MQ に接続しますが、他の MQTT クライアントを接続することもできます。

デーモンはパブリッシュ/サブスクライブ・ブローカーです。MQTT V3 クライアントはこれに接続して、パブリッシュの場合はトピック・ストリングを使用し、サブスクライブの場合はトピック・フィルターを使用して、トピックへのパブリッシュおよびサブスクライブを行います。トピック・ストリングは階層型をしており、そのトピック・レベルは / で分割されます。トピック・フィルターは、単一レベルの + ワイルドカードとトピック・ストリングの最後の部分としてマルチレベルの # ワイルドカードを含めることができるトピック・ストリングです。

**注:** デーモンのワイルドカードは、WebSphere Message Broker v6 のより厳しい規則に従います。IBM WebSphere MQ の場合とは異なります。それは、複数のマルチレベル・ワイルドカードをサポートします。ワイルドカードは、トピック・ストリングの任意の場所の、任意の数の階層レベルで使用可能です。

複数の MQTT v3 クライアントが、リスナー・ポートを使用してデーモンに接続されます。デフォルトのリスナー・ポートは変更可能です。複数のリスナー・ポートを定義して、それらに異なるネーム・スペースを割り振ることができます(160 ページの『WebSphere MQ Telemetry デーモン(デバイス用)のリスナー・ポート』を参照)。デーモン自体が MQTT v3 クライアントとなります。デーモンのブリッジ接続を構成し



て、あるデーモンを別のデーモンのリスナー・ポートまたは WebSphere MQ のテレメトリー (MQXR) サービスに接続することができます。

WebSphere MQ Telemetry デーモン (デバイス用) に複数のブリッジを構成することができます。それらのブリッジを使用して、パブリケーションを交換できるデーモンのネットワークも一緒に接続します。

各ブリッジで、そのローカル・デーモンのトピックにパブリッシュおよびサブスクライブできます。また、別のデーモン、WebSphere MQ パブリッシュ/サブスクライブ・ブローカー、または接続されている他の MQTT v3 ブローカーのトピックにパブリッシュおよびサブスクライブすることもできます。トピック・フィルターを使用することで、ブローカー間で伝搬するパブリケーションを選択できます。パブリケーションは、いずれの方向にも伝搬可能です。つまり、パブリケーションは、あるローカル・デーモンからそれに接続されている各リモート・ブローカーに伝搬することも、接続されているいずれかのブローカーからローカル・デーモンに伝搬することもできます (153 ページの『IBM WebSphere MQ Telemetry デーモン (デバイス用) のブリッジ』を参照)。

## IBM WebSphere MQ Telemetry デーモン (デバイス用) のブリッジ

IBM WebSphere MQ Telemetry デーモン (デバイス用) のブリッジでは、MQTT v3 プロトコルを使用して、2つのパブリッシュ/サブスクライブ・ブローカーを接続します。このブリッジによって、ブローカー間でのパブリケーションの両方向の伝搬が行われます。一方の終端は WebSphere MQ Telemetry デーモン (デバイス用) のブリッジ接続で、もう一方の終端はキュー・マネージャー、または別のデーモンのブリッジ接続の場合があります。キュー・マネージャーは、テレメトリー・チャンネルを使用して、ブリッジ接続に接続されます。デーモンは、デーモン・リスナーを使用してブリッジ接続に接続されます。

IBM WebSphere MQ Telemetry デーモン (デバイス用) では、他のブローカーへの1つ以上の同時接続がサポートされます。デーモンからの接続はブリッジ接続と呼ばれ、デーモン構成ファイル内の接続項目によって定義されます。IBM WebSphere MQ への接続は、次の図に示されているように、IBM WebSphere MQ のテレメトリー・チャンネルを使用して行われます。

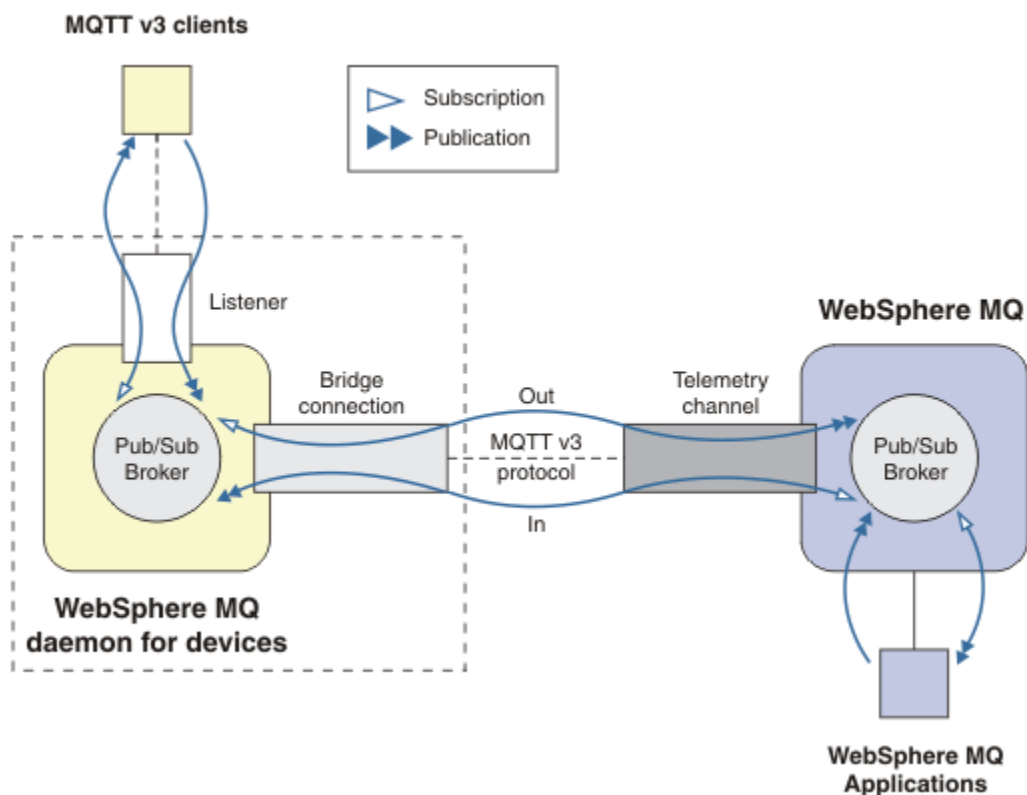


図 30. IBM WebSphere MQ への IBM WebSphere MQ Telemetry daemon for devices の接続

ブリッジは、別のブローカーを MQTT v3 クライアントとしてデーモンに接続します。ブリッジ・パラメーターは、MQTT v3 クライアントの属性をミラーリングします。

ブリッジは接続を行うだけではありません。2つのパブリッシュ/サブスクライブ・ブローカー間でパブリッシュおよびサブスクライブ・エージェントとして機能します。ローカル・ブローカーは IBM WebSphere MQ Telemetry デーモン (デバイス用) であり、リモート・ブローカーは MQTT v3 プロトコルをサポートするパブリッシュ/サブスクライブ・ブローカーです。通常、リモート・ブローカーは別のデーモンまたは IBM WebSphere MQ となります。

ブリッジの仕事は、2つのブローカー間でパブリケーションを伝搬することです。ブリッジは双方向です。パブリケーションをいずれの方向にも伝搬します。153 ページの図 30 に、ブリッジが IBM WebSphere MQ Telemetry デーモン (デバイス用) を IBM WebSphere MQ に接続する方法が示されています。155 ページの『ブリッジのトピック設定例』では、例を挙げて、トピック・パラメーターを使用してブリッジを構成する方法を示しています。

153 ページの図 30 の In 矢印と Out 矢印は、ブリッジが双方向であることを示しています。矢印の一方の終端で、サブスクリプションが作成されます。サブスクリプションと一致するパブリケーションは、矢印の反対側の終端でブローカーにパブリッシュされます。矢印には、パブリケーションのフローに応じてラベルが付けられています。パブリケーションのフローは、In の場合はデーモンに入り、Out の場合はデーモンから出て行きます。ここで重要なのは、ラベルがコマンド構文で使用されることです。In と Out はパブリケーションのフロー方向を示すものであり、サブスクリプションの送信方向を示すものではないということに注意してください。

他のクライアント、アプリケーション、またはブローカーを、IBM WebSphere MQ または WebSphere MQ Telemetry デーモン (デバイス用) に接続できます。これらは接続先のブローカーのトピックにパブリッシュおよびサブスクライブします。ブローカーが IBM WebSphere MQ である場合、トピックがクラスター化または分散されている可能性があり、ローカル・キュー・マネージャーで明示的に定義されていない可能性があります。

## ブリッジの用途

複数のブリッジ接続と複数のリスナーを使用して、複数のデーモンと一緒に接続します。複数のブリッジ接続と複数のテレメトリー・チャンネルを使用して、複数のデーモンと複数のキュー・マネージャーと一緒に接続します。複数のブローカーと一緒に接続する場合は、ループを作成することができます。その際に注意すべき点は、パブリケーションがブローカーのループを無限に循環し、それが検出されない可能性があることです。

IBM WebSphere MQ にブリッジ接続されるデーモンを使用する理由をいくつか以下に示します。

### WebSphere MQ への MQTT クライアントの接続数の削減

デーモンの階層を使用することで、単一のキュー・マネージャーが一度に接続できるクライアントの数よりも多くの数のクライアントを WebSphere MQ に接続できます。

### MQTT クライアントと WebSphere MQ 間でのメッセージのストア・アンド・フォワード

クライアントに独自のストレージがない場合は、ストア・アンド・フォワードを使用することで、クライアントと IBM WebSphere MQ 間で接続が継続されないようにすることができます。その場合、MQTT クライアントと WebSphere MQ の間で複数のタイプの接続を使用することができます (モニターと制御に関するテレメトリーの概念とシナリオを参照)。

### MQTT クライアントと WebSphere MQ 間で交換されるパブリケーションのフィルター

通常、パブリケーションでは、メッセージがローカルで処理されるものと、他のアプリケーションに関係するものに分けられます。ローカル・パブリケーションにはセンサーとアクチュエーター間の制御フローが含まれる場合があり、リモート・パブリケーションには測定値、状況、および構成コマンドの要求が含まれる場合があります。

### パブリケーションのトピック・スペースの変更

異なるリスナー・ポートに接続されているクライアントからのトピック・ストリングが、互いに競合しないようにします。示されている例では、異なる建物から得られたメーター測定値のラベル付けにデーモンを使用しています (異なるクライアント・グループのトピック・スペースの分離を参照)。

## ブリッジのトピック設定例

### リモート・ブローカーにすべてパブリッシュする (デフォルトを使用)

デフォルトの方向は `out` と呼ばれ、ブリッジはトピックをリモート・ブローカーにパブリッシュします。 `topic` パラメーターは、トピック・フィルターを使用して、伝搬されるトピックを制御します。

ブリッジは、155 ページの図 31 にある `topic` パラメーターを使用して、MQTT クライアントまたはその他のブローカーによってローカル・デーモンにパブリッシュされたすべてのものをサブスクライブします。ブリッジは、ブリッジによって接続されているリモート・ブローカーにトピックをパブリッシュします。

```
connection Daemon1
topic #
```

図 31. リモート・ブローカーにすべてパブリッシュする

### リモート・ブローカーにすべてパブリッシュする (明示的)

以下のコード・フラグメントにある `topic` の設定値を使用すると、デフォルトを使用した場合と同じ結果になります。唯一の違いは、**`direction`** パラメーターが明示的である点です。 `out` 方向を使用して、ローカル・ブローカーであるデーモンにサブスクライブし、リモート・ブローカーにパブリッシュします。ブリッジがサブスクライブしたローカル・デーモンで作成されたパブリケーションは、リモート・ブローカーでパブリッシュされます。

```
connection Daemon1
topic # out
```

図 32. リモート・ブローカーにすべてパブリッシュする (明示的)

### ローカル・ブローカーにすべてパブリッシュする

`out` 方向を使用する代わりに、逆方向の `in` を設定することができます。以下のコード・フラグメントでは、ブリッジによって接続されたリモート・ブローカーでパブリッシュされたすべてのものにサブスクライブするように、ブリッジが構成されています。ブリッジは、ローカル・ブローカーであるデーモンにトピックをパブリッシュします。

```
connection Daemon1
topic # in
```

図 33. ローカル・ブローカーにすべてパブリッシュする

### ローカル・ブローカーのエクスポート・トピックからリモート・ブローカーのインポート・トピックにすべてのものをパブリッシュする

2つの追加トピック・パラメーター (**`local_prefix`** と **`remote_prefix`**) を使用して、前の例のトピック・フィルター `#` を変更します。1つのパラメーターはサブスクリプションで使用されるトピック・フィルターの変更で使用され、もう1つのパラメーターはパブリケーションがパブリッシュされるトピックの変更で使用されます。その結果、一方のブローカーで使用されるトピック・ストリングの先頭が、もう一方のブローカーの別のトピック・ストリングに置き換えられます。

トピック・コマンドの方向に応じて、**`local_prefix`** と **`remote_prefix`** の意味は逆になります。方向が `out` の場合 (デフォルト) は、**`local_prefix`** がトピック・サブスクリプションの一部として使用され、リモート・パブリケーションのトピック・ストリングの **`local_prefix`** 部分が **`remote_prefix`** に置き換えられます。方向が `in` の場合は、**`remote_prefix`** がリモート・サブスクリプションの一部になり、トピック・ストリングの **`remote_prefix`** 部分が **`local_prefix`** に置き換えられます。

トピック・ストリングの最初の部分は、多くの場合、トピック・スペースを定義するものと見なされます。追加パラメーターを使用して、トピックがパブリッシュされるトピック・スペースを変更します。この操作を行うことで、伝搬されるトピックがターゲット・ブローカーの別のトピックと競合しないようにしたり、マウント・ポイント・トピック・ストリングを削除したりすることができます。

例えば、以下のコード・フラグメントでは、デーモン上のトピック・ストリング `export/#` へのすべてのパブリケーションが、リモート・ブローカー上の `import/#` にリパブリッシュされます。

```
topic # out export/ import/
```

図 34. ローカル・ブローカーのエクスポート・トピックからリモート・ブローカーのインポート・トピックにすべてのものをパブリッシュする

### リモート・ブローカーのエクスポート・トピックからローカル・ブローカーのインポート・トピックにすべてのものをパブリッシュする

以下のコード・フラグメントは、逆の構成を示しています。つまり、ブリッジは、リモート・ブローカーで `export/#` トピック・ストリングを使用してパブリッシュされたものすべてにサブスクライブし、それをローカル・ブローカーの `import/#` にパブリッシュします。

```
connection Daemon1
topic # in import/ export/
```

図 35. リモート・ブローカーのエクスポート・トピックからローカル・ブローカーのインポート・トピックにすべてのものをパブリッシュする

### 元のトピック・ストリングを使用して、1884/ マウント・ポイントからリモート・ブローカーにすべてのものをパブリッシュする

以下のコード・フラグメントでは、ブリッジは、ローカル・デーモンでマウント・ポイント `1884/` に接続されたクライアントによってパブリッシュされたものすべてにサブスクライブします。ブリッジは、マウント・ポイントにパブリッシュされたすべてのものをリモート・ブローカーにパブリッシュします。マウント・ポイントのストリング `1884/` は、リモート・ブローカーにパブリッシュされたトピックから削除されます。`local_prefix` はマウント・ポイントのストリング `1884/` と同じであり、`remote_prefix` はブランク・ストリングです。

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

図 36. 元のトピック・ストリングを使用して、1884/ マウント・ポイントからリモート・ブローカーにすべてのものをパブリッシュする。

### 異なるデーモンに接続されている異なるクライアントのトピック・スペースを分離する

電力メーターから建物のメーター測定値をパブリッシュするためのアプリケーションを作成したとします。測定値は、MQTT クライアントを使用して、同じ建物にホストされているデーモンにパブリッシュされます。パブリケーション対象として選択されたトピックは `power` です。同じアプリケーションが、複合施設内のいくつかの建物にデプロイされます。複合施設全体をモニターしデータを保管するために、すべての建物から得られた測定値は、ブリッジ接続を使用して集約されます。この接続により、建物のデーモンが中央の WebSphere MQ にリンクされます。

すべての建物で同じクライアント・アプリケーションが使用されます。このアプリケーションはトピック `power` にパブリッシュします。しかし、データは建物ごとに区別される必要があります。この区別は、建物ごとにデーモンによって行われます。デーモンはトピック名に接頭部として建物番号を追加

します。複合施設内の最初の建物からのブリッジでは、接頭部として `meters/building01/` を使用し、2つ目の建物からのブリッジの場合、接頭部は `meters/building02/` となります。他の建物からの測定値も同じパターンに従います。したがって、WebSphere MQ は、`meters/building01/power` のようなトピックの読み取り値を受け取ります。

各デーモンの構成ファイルには、以下のコード・フラグメントのパターンに従ったトピック・ステートメントがあります。

```
connection Daemon1
topic power out "" meters/building01/
```

図 37. 異なるデーモンに接続されているクライアントのトピック・スペースを分離する

前述のコード・フラグメントで、空ストリングは、使用しない `local_prefix` パラメーターのプレースホルダーです。

**注:** この例は多少手を加えており、単に図示することが目的です。実際には、アプリケーションがパブリッシュするトピック・スペースを構成可能なものにするのが多いでしょう。

### 同じデーモンに接続されているクライアントのトピック・スペースを分離する

すべての電力メーターを接続する際に単一のデーモンを使用すると仮定します。複数の異なるポートに接続するようにアプリケーションを構成できると想定すると、以下のコード・フラグメントにあるように、建物ごとに異なるメーターを異なるリスナー・ポートに接続することにより、建物を区別できます。ここでは、マウント・ポイントを使用する可能性がある例を示します。

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

図 38. 同じデーモンに接続されているクライアントのトピック・スペースを分離する

### 両方向に流れるパブリケーションの異なるトピックをリマップする

以下のコード・フラグメントの構成では、ブリッジはリモート・ブローカーの単一トピック `b` にサブスクライブし、`b` に関するパブリケーションをローカル・デーモンに転送して、トピックを `a` に変更します。また、ブリッジは、ローカル・ブローカーの単一トピック `x` にサブスクライブし、`x` に関するパブリケーションをリモート・ブローカーに転送して、トピックを `y` に変更します。

```
connection Daemon1
topic "" in a b
topic "" out x y
```

図 39. 両方向に流れるパブリケーションの異なるトピックをリマップする

この例での重要な点は、異なるトピックが両方のブローカーでサブスクライブおよびパブリッシュされることです。両方のブローカーのトピック・スペースはそれぞれ独立したものです。

### 両方向に流れるパブリケーションの同じトピックをリマップする (ループ)

前の例とは異なり、[158 ページの図 40](#) の構成では、通常、ループになります。トピック・ステートメント `topic "" in a b` の場合、ブリッジはリモートで `b` にサブスクライブし、ローカルで `a` にパブリッシュします。もう一方のトピック・ステートメントの場合、ブリッジはローカルで `a` にサブスクライブし、リモートで `b` にパブリッシュします。同じ構成を、[158 ページの図 41](#) にあるように作成することができます。

一般的な結果として、クライアントがリモート側で b にパブリッシュした場合、パブリケーションはトピック a のパブリケーションとしてローカル・デーモンに転送されます。ただし、ブリッジによってトピック a のローカル・デーモンにパブリッシュされると、パブリケーションは、ブリッジによってローカル・トピック a に対して作成されたサブスクリプションと一致します。サブスクリプションは topic "" out a b です。結果として、パブリケーションはトピック b のパブリケーションとしてリモート・ブローカーに戻されます。これで、ブリッジはリモート・トピック b にサブスクライブされ、サイクルが再び開始されます。

一部のブローカーでは、ループが発生しないようにループ検出が実装されています。しかし、ループ検出メカニズムは、異なるタイプのブローカーと一緒にブリッジ接続される場合にも機能する必要があります。WebSphere MQ が WebSphere MQ Telemetry デーモン (デバイス用) にブリッジ接続される場合は、ループ検出は機能しません。2 つの IBM WebSphere MQ Telemetry デーモン (デバイス用) が一緒にブリッジ接続される場合には機能します。デフォルトでは、ループ検出はオンに設定されています ([try\\_private](#) を参照)。

```
connection Daemon1
topic "" in a b
topic "" out a b
```

図 40. !両方向に流れるパブリケーションについて同じトピックを再マップする

```
connection Daemon1
topic "" both a b
```

図 41. !both を使用して、両方向に流れるパブリケーションについて同じトピックを再マップします。

157 ページの図 39 の構成は 158 ページの図 40 と同じです。

## IBM WebSphere MQ Telemetry daemon for devices のブリッジ接続の可用性

最初に使用可能なリモート・ブローカーに接続するように、複数の IBM WebSphere MQ Telemetry daemon for devices のブリッジ接続アドレスを構成します。ブローカーが複数インスタンスのキュー・マネージャーの場合は、その両方の TCP/IP アドレスを指定します。使用可能な場合は、1 次サーバーに接続または再接続するように 1 次接続を構成します。

接続ブリッジ・パラメーター [addresses](#) は、TCP/IP ソケット・アドレスのリストです。ブリッジは、正常に接続されるまで、各アドレスへの接続を順に試行します。[round\\_robin](#) および [start\\_type](#) 接続パラメーターは、正常に接続された後のアドレスの使用方法を制御します。

[start\\_type](#) が auto、manual、または lazy の場合、接続に失敗すると、ブリッジは再接続を試行します。ブリッジは、アドレスを 1 つずつ順に使用します。この場合の接続試行の間隔は約 20 秒となります。[start\\_type](#) が once の場合、接続に失敗すると、ブリッジは自動的に再接続を試行しません。

[round\\_robin](#) が true の場合、ブリッジによる接続試行はリストの最初のアドレスから開始され、リストのアドレスを 1 つずつ順に試みます。リストの最後のアドレスでの試行が終わると、再び、最初のアドレスから試行が開始されます。リストにアドレスが 1 つしかない場合は、20 秒ごとにそのアドレスで再試行されます。

[round\\_robin](#) が false の場合、リストの最初のアドレス (1 次サーバーと呼ぶ) が優先されます。1 次サーバーへの最初の接続試行に失敗した場合、ブリッジは引き続き、バックグラウンドで 1 次サーバーへの再接続を試行します。同時に、ブリッジは、リスト内の他のアドレスを使用して、接続を試行します。バックグラウンドでの 1 次サーバーへの接続試行に成功した場合、ブリッジは現行の接続を切断し、1 次サーバー接続に切り替えます。

接続が、例えば [connection\\_stop](#) コマンドを実行して、任意に切断された場合、接続が再開されると、再度、同じアドレスを使用して接続が試行されます。接続の失敗、またはリモート・ブローカーでの接続の中断が原因で接続が切断された場合、ブリッジは 20 秒間待機します。その後、リスト内の次のアドレスへの接続を試行するか、リストにアドレスが 1 つしかない場合は同じアドレスへの接続を試行します。

## 複数インスタンス・キュー・マネージャーへの接続

複数インスタンス・キュー・マネージャーの構成では、キュー・マネージャーは異なる IP アドレスを持つ 2 つの異なるサーバーで実行されます。通常、テレメトリー・チャンネルは特定の IP アドレスを指定せずに構成されます。これらのチャンネルはポート番号のみを指定して構成されます。テレメトリー・チャンネルが開始されると、デフォルトで、ローカル・サーバー上の最初の使用可能なネットワーク・アドレスが選択されます。

ブリッジ接続の `addresses` パラメーターは、キュー・マネージャーで使用する 2 つの IP アドレスで構成します。`round_robin` は `true` に設定します。

アクティブ・キュー・マネージャー・インスタンスで障害が発生した場合、キュー・マネージャーはスタンバイ・インスタンスに切り替わります。デーモンは、アクティブ・インスタンスへの接続が切断したことを検出し、スタンバイ・インスタンスへの再接続を試行します。その場合、ブリッジ接続用に構成されたアドレスのリストにある他の IP アドレスが使用されます。

ブリッジの接続先のキュー・マネージャーは、引き続き同じキュー・マネージャーです。キュー・マネージャーの状態はキュー・マネージャー自体が回復させます。`cleansession` が `false` に設定されている場合、ブリッジ接続セッションは、フェイルオーバーの前と同じ状態に復元されます。接続は遅延後に再開されます。"少なくとも 1 回" または "最大 1 回" のサービス品質を持つメッセージは失われず、サブスクリプションは引き続き機能します。

再接続時間は、スタンバイ・インスタンスの開始時に再開するクライアントとチャンネルの数、および未完了のメッセージ数によって異なります。ブリッジ接続の場合、両方の IP アドレスへの再接続は、接続が再確立されるまで何度でも試行できます。

複数インスタンス・キュー・マネージャーのテレメトリー・チャンネルは、特定の IP アドレスを使用して構成しないでください。IP アドレスは 1 つのサーバーでのみ有効です。

IP アドレスを管理する代替の高可用性ソリューションを使用するのであれば、特定の IP アドレスを使用してテレメトリー・チャンネルを構成するのが適切な場合があります。

### cleansession

ブリッジ接続は MQTT v3 クライアント・セッションです。接続で新規セッションを開始するか、既存のセッションを復元するかを制御できます。既存のセッションを復元する場合、ブリッジ接続では、前のセッションからのサブスクリプションと保存パブリケーションが保持されます。

`addresses` に複数の IP アドレスがリストされ、しかもそれらの IP アドレスが、異なる複数のキュー・マネージャーでホストされる遠隔測定チャンネル、または異なる複数の遠隔測定デーモンに接続される場合は、`cleansession` を `false` に設定しないでください。セッション状態は、キュー・マネージャー間やデーモン間では転送されません。異なるキュー・マネージャーまたはデーモンで既存のセッションを再開しようとすると、新規セッションが開始されます。未確定メッセージは失われ、サブスクリプションは予期したとおりに動作しない可能性があります。

### notifications

アプリケーションは、`notifications` を使用して、ブリッジ接続が実行されているかどうかを把握できます。通知は、値が 1 (接続) または 0 (切断) のパブリケーションです。これは、`notification_topic` パラメーターによって定義された `topicString` にパブリッシュされます。`topicString` のデフォルト値は `$$SYS/broker/connection/clientIdentifier/state` です。デフォルトの `topicString` には接頭部 `$$SYS` が含まれます。`$$SYS` で始まるトピックには、`$$SYS` で始まるトピック・フィルターを定義してサブスクライブします。トピック・フィルター # (すべてにサブスクライブ) の場合、デーモンの `$$SYS` で始まるトピックにはサブスクライブしません。`$$SYS` は、アプリケーションのトピック・スペースとは異なる特別なシステムのトピック・スペースを定義するものと考えてください。

`notifications` により、IBM WebSphere MQ Telemetry daemon for devices はブリッジの接続時または切断時に MQTT クライアントへ通知できます。

## keepalive\_interval

keepalive\_interval ブリッジ接続パラメーターは、ブリッジが TCP/IP ping をリモート・サーバーに送信する間隔を設定します。デフォルトの間隔は 60 秒です。ping は、リモート・サーバーまたはファイアウォールで接続の非アクティブ期間が検出されて、TCP/IP セッションが終了されないようにします。

## clientid

ブリッジ接続は MQTT v3 クライアント・セッションであり、ブリッジ接続パラメーター clientid で設定される clientIdentifier を持ちます。cleansession パラメーターを false に設定して、前のセッションを再開するために再接続する場合は、各セッションで使用する clientIdentifier が同じでなければなりません。clientid のデフォルト値は hostname.connectionName で、この値は変わりません。

## WebSphere MQ Telemetry デーモン (デバイス用) のインストール、検査、構成、および制御

デーモンのインストール、構成、および制御はファイル・ベースで行います。

デーモンをインストールするには、Software Development Kit を、デーモンを実行するデバイスにコピーします。

例えば、MQTT クライアント・ユーティリティーを実行し、パブリッシュ/サブスクライブ・ブローカーとして WebSphere MQ Telemetry デーモン (デバイス用) に接続します。『[デバイス用の WebSphere MQ Telemetry デーモンをパブリッシュ/サブスクライブ・ブローカーとして使用する](#)』を参照してください。

デーモンの構成は、構成ファイルを作成して行います。[WebSphere MQ Telemetry デーモン \(デバイス用\) の構成ファイル](#)を参照してください。

実行中のデーモンの制御は、amqtddd.upd ファイルにコマンドを作成して行います。デーモンは、5 秒ごとにこのファイルを読み取り、コマンドを実行して、このファイルを削除します。[WebSphere MQ Telemetry デーモン \(デバイス用\) のコマンド・ファイル](#)を参照してください。

## WebSphere MQ Telemetry デーモン (デバイス用) のリスナー・ポート

リスナー・ポートを使用して、MQTT V3 クライアントを WebSphere MQ Telemetry デーモン (デバイス用) に接続します。リスナー・ポートには、マウント・ポイントと最大接続数を指定できます。

リスナー・ポートは、このポートに接続するクライアントの MQTT クライアント connect(serverURI) メソッドで指定されたポート番号に対応している必要があります。デフォルトでは、クライアントとデーモンの両方で 1883 に設定されています。

デーモンのデフォルト・ポートは、デーモン構成ファイルでグローバル定義 port を設定することにより変更可能です。listener 定義をデーモン構成ファイルに追加することで、特定のポートを設定できます。

デフォルト・ポート以外の各リスナー・ポートでは、マウント・ポイントを指定して、クライアントを分離することができます。マウント・ポイントを指定したポートに接続されているクライアントは、他のクライアントから分離されます ([161 ページの『WebSphere MQ Telemetry デーモン \(デバイス用\) のマウント・ポイント』](#)を参照)。

任意のポートに接続可能なクライアントの数を制限することができます。グローバル定義 max\_connections を設定し、デフォルト・ポートへの接続を制限します。または、各リスナー・ポートに max\_connections を指定します。

### 例

構成ファイルの例を以下に示します。デフォルト・ポートを 1883 から 1880 に変更し、ポート 1880 への接続数を 10000 に制限しています。ポート 1884 への接続数を 1000 に制限しています。ポート 1884 に接続されているクライアントは、他のポートに接続されているクライアントから分離されます。

```
port 1880
max_connections 10000
listener 1884
```



## WebSphere MQ Telemetry デーモン (デバイス用) のマウント・ポイント

MQTT クライアントで使用されているリスナー・ポートにマウント・ポイントに関連付けることで、WebSphere MQ Telemetry デーモン (デバイス用) に接続することができます。マウント・ポイントにより、あるリスナー・ポートを使用する MQTT クライアントで交換されるパブリケーションおよびサブスクリプションを、別のリスナー・ポートに接続されている MQTT クライアントから分離します。

マウント・ポイントを指定したリスナー・ポートに接続されたクライアントは、その他のリスナー・ポートに接続されているクライアントと直接トピックを交換することはできません。マウント・ポイントを指定していないリスナー・ポートに接続されたクライアントは、どのクライアントのトピックにもパブリッシュおよびサブスクライブを行うことができます。クライアントは、マウント・ポイントを介して接続されているかどうかを認識しないため、クライアントで作成されたトピック・ストリングに影響はありません。

マウント・ポイントは、パブリケーションおよびサブスクリプションのトピック・ストリングの接頭部として付けられるテキストのストリングです。マウント・ポイントを指定したリスナー・ポートに接続されたクライアントで作成されたすべてのトピック・ストリングに、接頭部として付けられます。このテキストのストリングは、リスナー・ポートに接続されているクライアントに送信されたすべてのトピック・ストリングから削除されます。

リスナー・ポートにマウント・ポイントが指定されていない場合、ポートに接続されているクライアントで作成および受信されたパブリケーションとサブスクリプションのトピック・ストリングは変更されません。

末尾に / が付いたマウント・ポイント・ストリングを作成します。これにより、マウント・ポイントが、そのマウント・ポイントのトピック・ツリーの親トピックとなります。

### 例

構成ファイルには以下のリスナー・ポートが含まれています。

```
listener 1883  
mount_point 1883/  
listener 1884 127.0.0.1  
mount_point 1884/  
listener 1885
```

ポート 1883 に接続されているあるクライアントが、MyTopic へのサブスクリプションを作成します。デーモンはそのサブスクリプションを 1883/MyTopic として登録します。ポート 1883 に接続されている別のクライアントは、トピック MyTopic でメッセージをパブリッシュします。デーモンは、トピック・ストリングを 1883/MyTopic に変更し、一致するサブスクリプションを検索します。ポート 1883 のこのサブスクライバーは、元のトピック・ストリングが MyTopic のパブリケーションを受信します。デーモンはトピック・ストリングからマウント・ポイント接頭部を削除します。

ポート 1884 に接続されている別のクライアントも、トピック MyTopic でパブリッシュを行います。この場合、デーモンはトピックを 1884/MyTopic として登録します。ポート 1883 のサブスクライバーはパブリケーションを受信しません。これは、異なるマウント・ポイントでは異なるトピック・ストリングのサブスクリプションが作成されるためです。

ポート 1885 に接続されているクライアントが、トピック 1883/MyTopic でパブリッシュを行います。この場合、デーモンはトピック・ストリングを変更しません。ポート 1883 のサブスクライバーは MyTopic へのパブリケーションを受信します。

## デバイス用 WebSphere MQ Telemetry デーモンのサービスの品質、永続サブスクリプション、および保存パブリケーション

サービスの品質の設定は、実行中のデーモンに対してのみ適用されます。デーモンが停止すると、その停止が制御下で行われたものであっても、障害が原因であっても、送信中のメッセージの状態は失われます。デーモンが停止した場合、メッセージが少なくとも 1 回、あるいは多くても 1 回送達されるかどうかは保

証されません。デバイス用 WebSphere MQ Telemetry デーモンがサポートするのは、制限付きの持続性です。デーモンがシャットダウンしたときに保存パブリケーションとサブスクリプションを保存するには、**retained\_persistence** 構成パラメーターを設定します。

WebSphere MQ とは異なり、デバイス用 WebSphere MQ Telemetry デーモンは、持続データをジャーナル処理しません。セッション状態、メッセージ状態、および保存パブリケーションは、トランザクションとして保存されません。デフォルトでは、このデーモンは、停止するとすべてのデータを廃棄します。オプションを設定することにより、定期的にサブスクリプションと保存パブリケーションのチェックポイントを指定できます。このデーモンが停止すると、メッセージの状況は常に失われます。非保存パブリケーションはすべて失われます。

保存パブリケーションをファイルに定期的に保存するには、デーモン構成オプション **Retained\_persistence** を **true** に設定します。このデーモンが再始動すると、最後に自動保存された保存パブリケーションが復元されます。デフォルトでは、クライアントによって作成された保存メッセージは、デーモンの再始動時には復元されません。

持続セッションで作成されたサブスクリプションを定期的にファイルに保存するには、デーモン構成オプション **Retained\_persistence** を **true** に設定します。**Retained\_persistence** が **true** に設定されている場合、**CleanSession** が **false** ("持続セッション") に設定されたセッションでクライアントが作成したサブスクリプションが復元されます。このデーモンは、再始動時にそれらのサブスクリプションを復元し、復元されたサブスクリプションがパブリケーションの受信を開始します。クライアントがパブリケーションを受信するのは、**CleanSession** を **false** に設定してクライアントが再始動したときです。デフォルトでは、デーモンが停止した場合、クライアント・セッション状態は保存されないため、クライアントが **CleanSession** を **false** に設定していても、サブスクリプションは復元されません。

**Retained\_persistence** は自動保存のための仕組みです。直近の保存パブリケーションあるいはサブスクリプションは保存されない場合があります。保存パブリケーションおよびサブスクリプションの保存頻度は変更できます。次の保存までの間隔、または次の保存までに行われた変更の数を、構成オプション **autosave\_on\_changes** および **autosave\_interval** を使用して設定します。

### 持続性を設定するための構成例

```
# 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
```

## WebSphere MQ Telemetry デーモン (デバイス用) のセキュリティー

WebSphere MQ Telemetry デーモン (デバイス用) では、接続するクライアントの認証、他のブローカーに接続するための資格情報の使用、およびトピックへのアクセスの制御を行えます。デーモンで提供されるセキュリティーは、WebSphere MQ Telemetry C クライアントを使用してビルドされることにより限定的なものになっています。これは SSL サポートを提供していません。したがって、デーモンとの接続は暗号化されず、証明書を使用して認証することはできません。

デフォルトでは、セキュリティーはオンになっていません。

### クライアントの認証

MQTT クライアントでは、`MqttConnectOptions.setUsername` メソッドと `MqttConnectOptions.setPassword` メソッドを使用して、ユーザー名およびパスワードを設定することができます。

デーモンに接続されているクライアントの認証は、クライアントが提供するユーザー名およびパスワードを、パスワード・ファイル内の項目と照合することにより行います。認証を使用可能にするには、パスワード・ファイルを作成し、デーモン構成ファイルで `password_file` パラメーターを設定します (`password_file` を参照)。

デーモン構成ファイルに `allow_anonymous` パラメーターを設定すると、ユーザー名やパスワードを設定せずに接続したクライアントを、認証を確認するデーモンに接続することができます (`allow_anonymous` を参照)。クライアントでユーザー名またはパスワードが提供され、`password_file` パラメーターが設定されている場合は、常にパスワード・ファイルとの照合が行われます。

デーモン構成ファイルに `clientid_prefixes` パラメーターを設定すると、接続を特定のクライアントに制限します。接続するクライアントは、`clientid_prefixes` パラメーターにリストされているいずれかの接頭部で始まる `clientIdentifiers` を持たなければなりません (`clientid_prefixes` を参照)。

## ブリッジ接続セキュリティ

ブリッジ接続される各 WebSphere MQ Telemetry デーモン (デバイス用) が MQTT V3 クライアントとなります。各ブリッジ接続に対して、デーモン構成ファイルでブリッジ接続パラメーターとしてユーザー名とパスワードを設定することができます (`username` および `password` を参照)。その結果、ブリッジ自体をブローカーに対して認証できるようになります。

## トピックへのアクセス制御

クライアントが認証されている場合、デーモンで、トピックへのアクセス制御を各ユーザーに対して行うこともできます。デーモンは、クライアントがパブリッシュまたはサブスクライブするトピックと、アクセス制御ファイル内のアクセス・トピック・ストリングとのマッチングに基づいて、アクセス制御を許可します (`acl_file` を参照)。

アクセス制御リストは2つの部分に分かれています。1つ目の部分では、匿名クライアントを含むすべてのクライアントからのアクセスを制御します。2つ目の部分には、パスワード・ファイル内の任意のユーザーに関するセクションがあります。ここでは、各ユーザーに対する特定のアクセス制御がリストされます。

### 例

以下に、セキュリティ・パラメーターの例を示します。

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

図 42. デーモン構成ファイル

```
Fred:Fredpassword
Barney:Barneypassword
```

図 43. パスワード・ファイル `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/#
```

図 44. アクセス制御ファイル `acl.txt`

## MQTT クライアントのトラブルシューティング

MQTT クライアントの実行に関する問題の解決に役立つトラブルシューティング・タスクを探します。

### 関連タスク

#### [172 ページの『MQTT \(Paho\) Java クライアントのトレースおよびデバッグ』](#)

デフォルト・ロガーは、`java.util.logging (JSR47)` と呼ばれる標準 Java ロギング機能を使用します。これは、構成ファイルの使用によって構成するか、プログラムで構成することができます。

#### [175 ページの『MQTT JavaScript クライアントのトレース』](#)

接続されたクライアント・オブジェクトでメソッドを呼び出すようクライアント Web アプリケーションを変更することにより、JavaScript クライアントを使用してトレースを収集することができます。

#### [168 ページの『テレメトリー \(MQXR\) サービスのトレース』](#)

以下の指示に従って、テレメトリー・サービスのトレースを開始し、トレースを制御するパラメーターを設定し、トレース出力を見つけます。

#### [169 ページの『MQTT v3 Java クライアントのトレース』](#)

以下の指示に従って、MQTT Java クライアント・トレースを作成し、その出力を制御します。

#### [171 ページの『C 用の MQTT クライアントのトレース』](#)

環境変数 `MQTT_C_CLIENT_TRACE` を設定し、MQTT クライアント C アプリケーションをトレースします。

#### [181 ページの『問題の解決: MQTT クライアントが接続しない』](#)

MQTT クライアント・プログラムがテレメトリー (MQXR) サービスへの接続に失敗するという問題を解決します。

#### [183 ページの『問題の解決: MQTT クライアントの接続が切断される』](#)

正常に接続され、短時間または長時間実行された後で、予期しない `ConnectionLost` 例外をクライアントがスローする原因となっているものを突き止めます。

#### [184 ページの『問題の解決: MQTT アプリケーションで失われたメッセージ』](#)

メッセージが失われる問題を解決します。メッセージは非持続メッセージですか、間違った場所に送信されましたか、それともまったく送信されませんでしたか? クライアント・プログラムのコード化に誤りがあると、メッセージが失われる可能性があります。

#### [186 ページの『問題の解決: テレメトリー \(MQXR\) サービスが開始しない』](#)

テレメトリー (MQXR) サービスの開始に失敗するという問題を解決します。WebSphere MQ Telemetry のインストールを確認し、ファイルが欠落および移動していないこと、あるいはそのファイルの権限が間違っていないことを確認します。テレメトリー (MQXR) サービスによって使用されるパスに、テレメトリー (MQXR) サービスのプログラムが含まれていることを確認します。

#### [188 ページの『問題の解決: JAAS ログイン・モジュールがテレメトリー・サービスによって呼び出されない』](#)

JAAS ログイン・モジュールがテレメトリー (MQXR) サービスによって呼び出されていないかどうかを突き止め、問題が修正されるように JAAS を構成します。

#### [191 ページの『問題の解決: デーモンの開始または実行』](#)

デーモンに関する問題のトラブルシューティングを行うには、IBM WebSphere MQ Telemetry デーモンのデバイス・コンソール・ログを参照するか、トレースをオンにするか、あるいはこのトピック内の症状表を使用します。

#### [191 ページの『問題の解決: MQTT クライアントがデーモンに接続していない』](#)

クライアントがデーモンに接続していないか、あるいはデーモンが他のデーモンまたは WebSphere MQ Telemetry チャンネルに接続していません。

### 関連資料

#### [165 ページの『テレメトリー・ログ、エラー・ログ、および構成ファイルの場所』](#)

IBM WebSphere MQ Telemetry によって使用されるログ、エラー・ログ、および構成ファイルを探します。

#### [167 ページの『MQTT v3 Java クライアントの理由コード』](#)

MQTT v3 Java クライアントの例外またはスロー可能な例外で理由コードの原因を調べてください。

#### [176 ページの『MQTT クライアントで SHA-2 暗号スイートを使用する場合のシステム要件』](#)

IBM SR13 以降の Java 6 では、SHA-2 暗号スイートを使用して、MQTT チャンネルおよびクライアント・アプリケーションを保護することができます。ただし、SHA-2 暗号スイートは、IBM、SR4 以降の Java 7

以降ではデフォルトで有効になりません。そのため、以前のバージョンでは、必要なスイートを指定する必要があります。独自の JRE で MQTT クライアントを実行する場合、SHA-2 暗号スイートをサポートしていることを確認する必要があります。クライアント・アプリケーションで SHA-2 暗号スイートを使用する場合は、クライアントの SSL コンテキストを、Transport Layer Security (TLS) バージョン 1.2 をサポートする値に設定する必要があります。

177 ページの『[SSL を介したモバイル・メッセージング Web アプリケーションのブラウザー・サポートに関する制約事項](#)』

ブラウザーの違いや、ブラウザーが稼働するプラットフォームの違いにより、機能に違いが生じます。これらの違いを理解することで、JavaScript 用の MQTT メッセージング・クライアント over SSL および WebSockets を使用して接続するようにアプリケーション、認証局 (CA)、およびクライアント証明書を構成することができます。

## テレメトリー・ログ、エラー・ログ、および構成ファイルの場所

IBM WebSphere MQ Telemetry によって使用されるログ、エラー・ログ、および構成ファイルを探します。

注: 例は Windows 用にコーディングされています。Linux で例を実行するように構文を変更します。

### サーバー・サイドのログ

IBM WebSphere MQ Telemetry のインストール・ウィザードは、メッセージをそのインストール・ログに書き込みます。

```
WMQ program directory\mqxr
```

テレメトリー (MQXR) サービスは、メッセージを WebSphere MQ キュー・マネージャーのエラー・ログに書き込み、FDC ファイルを IBM WebSphere MQ エラー・ディレクトリーに書き込みます。

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG  
WMQ data directory\errors\AMQnnn.n.FDC
```

テレメトリー (MQXR) サービスのログも作成されます。ログには、サービスが開始されたときのプロパティーや、MQTT クライアントのプロキシとして機能していて見つけたエラーが表示されます。例えば、クライアントが作成しなかったサブスクリプションからのアンサブスクライブなどです。ログのパスは次のとおりです。

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

IBM WebSphere MQ エクスプローラーによって作成される IBM WebSphere MQ 遠隔測定サンプル構成は、コマンド **runMQXRService** を使って遠隔測定サービスを開始します。**runMQXRService** は *WMQ Telemetry install directory\bin* にあります。以下に書き込みます。

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout  
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

テレメトリー (MQXR) サービスに対して構成されているパスを表示するか、テレメトリー (MQXR) サービスを開始する前に初期設定をエコー出力するように **runMQXRService** を変更します。

### サーバー・サイドの構成ファイル

#### テレメトリー・チャンネルおよびテレメトリー (MQXR) サービス

**制約事項:** テレメトリー・チャンネルの構成ファイルの形式、場所、内容、および解釈は将来のリリースで変更される可能性があります。テレメトリー・チャンネルを構成するには、IBM WebSphere MQ エクスプローラーを使用する必要があります。

IBM WebSphere MQ エクスプローラーは、テレメトリー構成を Windows の場合は *mqxr\_win.properties* ファイルに、Linux の場合は *mqxr\_unix.properties* ファイルに保存します。プロパティー・ファイルは、以下のテレメトリー構成ディレクトリーに保存されます。

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

図 45. Windows の遠隔測定構成ディレクトリー

```
/var/mqm/qmgrs/qMgrName/mqxr
```

図 46. Linux のテレメトリー構成ディレクトリー

## JVM

`java.properties` ファイル内のテレメトリー (MQXR) サービスに引数として渡される Java プロパティを設定します。このファイルのプロパティは、テレメトリー (MQXR) サービスを実行している JVM に直接渡されます。これらは、Java コマンド行で追加の JVM プロパティとして渡されます。コマンド行に設定されたプロパティは、`java.properties` ファイルからコマンド行に追加されたプロパティよりも優先されます。

テレメトリー構成と同じフォルダーにある `java.properties` ファイルを見つけます。166 ページの図 45 および 166 ページの図 46 を参照してください。

各プロパティを個別の行として指定して、`java.properties` を変更します。プロパティを JVM に引数として渡す場合と同じように、各プロパティをフォーマットします。例えば、以下のようになります。

```
-Xmx1024m  
-Xms1024m
```

## JAAS

JAAS 構成ファイルについてはテレメトリー・チャンネルの JAAS 構成で説明されています。これにはサンプル JAAS 構成ファイル `JAAS.config` が含まれており、これは IBM WebSphere MQ Telemetry に付属しています。

JAAS を構成する場合はほぼ確実に、ユーザーを認証するクラスを作成して、標準的な JAAS 認証手順を置き換えることとなります。

テレメトリー (MQXR) サービス・クラスパスで使用されるクラスパスに `Login` クラスを組み込むには、`WebSphere MQ service.env` 構成ファイルを指定します。

`service.env` に JAAS `LoginModule` のクラスパスを設定します。`service.env` では変数 `%classpath%` を使用できません。`service.env` のクラスパスは、テレメトリー (MQXR) サービス定義に既に設定されているクラスパスに追加されます。

`echo set classpath` を `runMQXRService.bat` に追加して、テレメトリー (MQXR) サービスで使用されているクラスパスを表示します。出力は `mqxr.stdout` に送信されます。

`service.env` ファイルのデフォルトの場所は、次のとおりです。

```
WMQ data directory\service.env
```

これらの設定を、以下の場所にある各キュー・マネージャーの `service.env` ファイルでオーバーライドします。

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

166 ページの図 47 は、サンプル `LoginModule.class` を使用するためのサンプル `service.env` ファイルを示しています。

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

注：`service.env` に変数を含めてはなりません。`WMQ Install Directory` は、実際の値に置換してください。

図 47. Windows 用のサンプル `service.env`

## トレース

トレースを構成するように IBM サービス・エンジニアから依頼されることがあります。168 ページの『[テレメトリー \(MQXR\) サービスのトレース](#)』を参照してください。構成されるトレースへのパラメーターは、以下の 2 つのファイルに保管されます。

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

## クライアント・サイドのログ・ファイル

IBM WebSphere MQ Telemetry に付属する Java SE MQTT クライアントのデフォルトのファイル・パーシスタンス・クラスは、以下の名前フォルダーを作成します。 *clientIdentifier*-クライアント作業ディレクトリーの *tcpHostNameport* または *clientIdentifier-sslHostNameport*。フォルダー名によって、接続試行に使用される *hostName* および *port* が分かります。このフォルダーには、パーシスタンス・クラスによって保管されたメッセージが含まれています。正常に送達された後、メッセージは削除されます。

クリーン・セッションによってクライアントが終了すると、フォルダーは削除されます。

クライアント・トレースがオンになると、フォーマットされていないログが、デフォルトでクライアントの作業ディレクトリーに保管されます。トレース・ファイルの名前は *mqtt-n.trc* です。

## クライアント・サイドの構成ファイル

Java プロパティ・ファイルを使用して MQTT Java クライアントのトレース・プロパティと SSL プロパティを設定するか、プロパティをプログラマチックに設定します。JVM `-D` スイッチを使用して、MQTT Java クライアントにプロパティを渡します。以下に例を示します。

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
-Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

169 ページの『[MQTT v3 Java クライアントのトレース](#)』を参照してください。MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、[MQTT クライアント・プログラミング・リファレンス](#)を参照してください。

## MQTT v3 Java クライアントの理由コード

MQTT v3 Java クライアントの例外またはスロー可能な例外で理由コードの原因を調べてください。

理由コード	値	原因
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	クライアントは既に接続されています。
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	クライアントは既に切断されています。
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	MqttClient.disconnect の呼び出し試行が MqttCallback のメソッド内から行われているときにスローされます。
REASON_CODE_CLIENT_DISCONNECTING	32102	クライアントは現在切断中であり、新しい作業を受け入れることができません。
REASON_CODE_CLIENT_EXCEPTION	0	クライアントで例外が発生しました。

表 5. MQTT v3 Java クライアントの理由コード (続き)		
理由コード	値	原因
REASON_CODE_CLIENT_NOT_CONNECTED	32104	クライアントはサーバーに接続されていません。
REASON_CODE_CLIENT_TIMEOUT	32000	サーバーからの応答の待機中にクライアントがタイムアウトになりました。
REASON_CODE_FAILED_AUTHENTICATION	4	ユーザー名またはパスワードが正しくないために、サーバーの認証に失敗しました。
REASON_CODE_INVALID_CLIENT_ID	2	サーバーは、指定されたクライアント ID を拒否しました。
REASON_CODE_INVALID_PROTOCOL_VERSION	1	要求されたプロトコル・バージョンはサーバーでサポートされていません。
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	新規メッセージ ID を使用できないために内部エラーが発生しました。
REASON_CODE_NOT_AUTHORIZED	5	要求された操作の実行は許可されていません。
REASON_CODE_SERVER_CONNECT_ERROR	32103	サーバーに接続できません。
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	サーバー URI と指定の SocketFactory が一致しません。
REASON_CODE_SSL_CONFIG_ERROR	32106	SSL 構成エラー。
REASON_CODE_UNEXPECTED_ERROR	6	予期しないエラーが発生しました。

## テレメトリー (MQXR) サービスのトレース

以下の指示に従って、テレメトリー・サービスのトレースを開始し、トレースを制御するパラメーターを設定し、トレース出力を見つけます。

### 始める前に

トレースは、サポート機能の一つです。IBM サービス・エンジニアによってテレメトリー (MQXR) サービスのトレースを依頼される場合は以下の指示に従ってください。製品資料には、トレース・ファイルの形式や、それを使用してクライアントをデバッグする方法は記載されていません。

### このタスクについて

IBM WebSphere MQ **strmqtrc** および **endmqtrc** コマンドを使用して、IBM WebSphere MQ トレースを開始および停止することができます。**strmqtrc** は、テレメトリー (MQXR) サービスのトレースをキャプチャーします。**strmqtrc** を使用すると、テレメトリー・サービス・トレースが開始されるまでに、最大で数秒の遅延が発生します。IBM WebSphere MQ のトレースについて詳しくは、[トレースの使用](#)を参照してください。あるいは、以下の手順に従って遠隔測定 (MQXR) サービスをトレースすることもできます。

### 手順

1. トレース・オプションを設定して、トレースの詳細の量とサイズを制御します。オプションは、**strmqtrc** または **controlMQXRChannel** のいずれかのコマンドで開始されたトレースに適用されません。

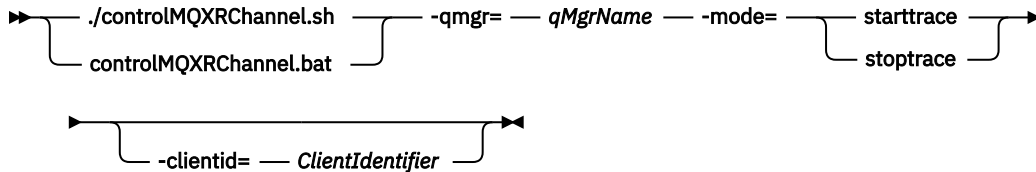
以下のファイルでトレース・オプションを設定します。



```
mqxrtrace.properties
trace.config
```

これらのファイルは以下のディレクトリーにあります。

- Windows の場合は、*WebSphere MQ data directory\qmgrs\qMgrName \mqxr*。
  - Linux の場合は、*var/mqm/qmgrs/ qMgrName/mqxr*。
2. 以下のディレクトリーでコマンド・ウィンドウを開きます。
- Windows システムの場合は、*WebSphere MQ installation directory\mqxr\bin*。
  - Linux システムの場合 */opt/mqm/mqxr/bin*。
3. 以下のコマンドを実行して、SYSTEM.MQXR.SERVICE トレースを開始します。



#### 必須パラメーター

##### **qmgr=qmgrName**

*qmgrName* をキュー・マネージャー名に設定します。

##### **mode=starttrace| stoptrace**

トレースを開始する場合は *starttrace* を設定し、トレースを終了する場合は *stoptrace* を設定します。

#### オプション・パラメーター

##### **clientid=ClientIdentifier**

*ClientIdentifier* をクライアントの *ClientIdentifier* に設定します。 *clientid* は、単一クライアントへのトレースをフィルタリングします。複数のクライアントをトレースする場合は、トレース・コマンドを複数回実行します。

以下に例を示します。

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

## タスクの結果

トレース出力を表示するには、以下のディレクトリーに移動します。

- Windows の場合は、*WebSphere MQ data directory\trace*。
- Linux の場合は、*/var/mqm/trace*。

トレース・ファイルの名前は *mqxr\_PPPPP.trc* です。PPPPP はプロセス ID です。

#### 関連資料

[strmqtrc](#)

## MQTT v3 Java クライアントのトレース

以下の指示に従って、MQTT Java クライアント・トレースを作成し、その出力を制御します。

### 始める前に

このトピックは、IBM WebSphere MQ バージョン 7.5.0.0 にのみ該当します。それ以降のバージョンの Java クライアントのトレースについては、[172 ページの『MQTT \(Paho\) Java クライアントのトレースおよびデバッグ』](#)を参照してください。

トレースは、サポート機能の一つです。IBM サービス・エンジニアによって MQTT Java クライアントのトレースを依頼される場合は以下の指示に従ってください。製品資料には、トレース・ファイルの形式や、それを使用してクライアントをデバッグする方法は記載されていません。

トレースは WebSphere MQ Telemetry Java クライアントでのみ機能します。

## このタスクについて

注：例は Windows 用にコーディングされています。Linux で例を実行するように構文を変更します。<sup>2</sup>

### 手順

1. トレース構成が含まれる Java プロパティ・ファイルを作成します。

プロパティ・ファイルで、以下のオプション・プロパティを指定します。プロパティ・キーが複数回指定される場合、最後に指定されたものがプロパティを設定します。

a) `com.ibm.micro.client.mqttv3.trace.outputName`

トレース・ファイルの書き込み先のディレクトリ。これはデフォルトでクライアント作業ディレクトリになります。トレース・ファイルの名前は `mqtt-n.trc` です。

```
java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
```

b) `com.ibm.micro.client.mqttv3.trace.count`

作成するトレース・ファイルの数。デフォルトのファイルの数は 1 つで、サイズの制限はありません。

```
java com.ibm.micro.client.mqttv3.trace.count=5
```

c) `com.ibm.micro.client.mqttv3.trace.limit`

作成するファイルの最大サイズ。デフォルトは 500000 です。複数のトレース・ファイルが要求される場合のみ、制限が適用されます。

```
java com.ibm.micro.client.mqttv3.trace.limit=100000
```

d) `com.ibm.micro.client.mqttv3.trace.client.clientIdentifier.status`

クライアントごとにトレースをオンまたはオフにします。 `clientIdentifier=*` の場合、トレースはすべてのクライアントでオンまたはオフにされます。デフォルトで、トレースはすべてのクライアントでオフにされます。

```
java com.ibm.micro.client.mqttv3.trace.client.*.status=on
```

```
java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
```

2. システム・プロパティを使ってトレース・プロパティ・ファイルを JVM に渡します。

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. クライアントを実行します。

4. トレース・ファイルをバイナリー・エンコードからテキストまたは `.html` に変換します。以下のコマンドを使用します。

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o  
outputFile] [-h] [-d  
time]
```

引数は次のとおりです。

---

<sup>2</sup> Java は正しいパス区切り文字を使用します。区切り文字は、プロパティ・ファイル内で '/' または '\'、\ ' としてコーディングできます。 '\' はエスケープ文字です。

-?

ヘルプを表示します。

**-i traceFile**

必須。入力ファイルを受け渡します (例: mqtt-0.trc)。

**-o outputFile**

必須。出力ファイルを定義します (例: mqtt-0.trc.html または mqtt-0.trc.txt)。

**-h**

HTML として出力します。出力ファイルの拡張子は .html にする必要があります。これを指定しない場合、出力はプレーン・テキストになります。

**-d time**

ミリ秒単位の時間差が time 以上 (>=) の場合に、行を \* で字下げします。HTML 出力の場合は適用外です。

次の例では、トレース・ファイルを HTML 形式で出力します。

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

この 2 番目の例では、トレース・ファイルをプレーン・テキストで出力し、50 ミリ秒以上の時間差のあるタイム・スタンプが連続している場合は、アスタリスク (\*) で字下げします。

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

この最後の例では、トレース・ファイルをプレーン・テキストで出力します。

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

## C 用の MQTT クライアントのトレース

環境変数 MQTT\_C\_CLIENT\_TRACE を設定し、MQTT クライアント C アプリケーションをトレースします。

### 始める前に

C 用の MQTT クライアントのトレースは、事前に作成された Windows および Linux の C 用の MQTT クライアント・ライブラリーとユーザーがビルドする iOS ライブラリーの両方に使用できます。

### このタスクについて

トレース出力を格納するファイルへのパスを環境変数 MQTT\_C\_CLIENT\_TRACE に設定します。トレース出力は、このファイルに書き込まれます。

### 手順

MQTT クライアント C アプリケーションを実行する前に MQTT\_C\_CLIENT\_TRACE=mqtccclient.log を設定します。

- a) 例えば、[27 ページの『C 用の MQTT クライアントの概要』](#)のサンプル・スクリプトを次のように変更します。

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

```
pause
endlocal
```

b) %sdkroot%/sdk/client/c/samples ディレクトリーからこのスクリプトを実行します。

## タスクの結果

トレース出力ファイルは、以下の行で始まります。

```
=====
                          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
```

### 関連タスク

[27 ページの『C 用の MQTT クライアントの概要』](#)

C ソースをコンパイルできる任意のプラットフォームで C 用のサンプル MQTT クライアントを起動して稼働状態にします。C 用のサンプル MQTT クライアントを IBM MessageSight または IBM WebSphere MQ のいずれかを MQTT サーバーとしてで実行できることを確認します。

## MQTT (Paho) Java クライアントのトレースおよびデバッグ

デフォルト・ロガーは、`java.util.logging` (JSR47) と呼ばれる標準 Java ログ機能を使用します。これは、構成ファイルの使用によって構成するか、プログラムで構成することができます。

### このタスクについて

注: Paho Java クライアントは、IBM WebSphere MQ バージョン 7.5.0.1 以降のバージョンにのみ適用できます。IBM WebSphere MQ バージョン 7.5.0.0 での Java クライアントのトレースについて詳しくは、[169 ページの『MQTT v3 Java クライアントのトレース』](#)を参照してください。

注: トレースは、サポート機能の一つです。IBM サービス・エンジニアから MQTT Java クライアントをトレースするように依頼された場合は、以下の指示に従ってください。製品資料には、トレース・ファイルの形式や、それを使用してクライアントをデバッグする方法は記載されていません。トレースは、IBM WebSphere MQ Telemetry Java クライアントに対してのみ機能します。

構成ファイルを使用する最も単純な方法は、その名前をプロパティ `java.util.logging.config.file` で指定する方法です。

作業プロパティ・ファイル `jsr47min.properties` は、パッケージ `org.eclipse.paho.client.mqttv3.logging` で提供されています。

JSR47 ログ機能の使用について、いくつかの点を挙げます。

- 選択されたパッケージのセットからメッセージを収集することができます
- ログ・レベル以下からメッセージを収集することができます
- ログ・メッセージの複数の宛先を選択することができます
- ファイルに書き込んで、使用されるファイルのサイズと数を制御する組み込みロガーを提供することによって使用できます
- メモリーに書き込んで、トリガーに基づいて書き出されるメモリー内のメッセージを有効にする組み込みロガーを提供することによって使用できます
- MQTT クライアント・ライブラリーを使用するアプリケーションも JSR47 の使用によって装備されている場合、アプリケーションおよびクライアント・ライブラリーからのメッセージが混合されます

デバッグ情報の収集に役立つユーティリティー・クラスが提供されています。このクラスには前述のログ・メッセージとトレース・メッセージが含まれますが、Paho クライアント内部から Java システム・プロパティや変数の値などの情報を収集することができます。

デバッグ機能は、パブリック・クラス `Debug` で提供されます。これは、パッケージ `org.eclipse.paho.client.mqttv3.util` の一部です。非同期と同期の両方の MQTT クライアント・オブジェクトでメソッド `getDebug()` を使用することで、`Debug` のインスタンスを入手することができます。

以下に例を示します。

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

メソッド `dumpClientDebug()` は、最大のデバッグ情報量をダンプします。完全なデバッグ情報(ログに書き込まれる)を収集するには、ログ機能を有効にする必要があります。完全なデバッグ情報を収集するには、問題発生タイミングが分かっている(例えば、特定の例外発生後など)、ダンプ・メソッドを呼び出します。

## 手順

1. 構成ファイルを作成するか、提供される `jsr47min.properties` を使用します。

提供されるプロパティ・ファイルを使用する場合、プッシュ・トリガーが正しいエラー・レベルに設定されていることを確認してください。デフォルトでは、このエラー・レベルは `SEVERE` に設定されていますが、エラーが発生するまでトレースをメモリーに保持するのではなく、トレースをファイルに書き込み続ける必要があるかもしれません。これを行うには、以下の変更を行います。

```
java.util.logging.MemoryHandler.push=SEVERE
```

終了

```
java.util.logging.MemoryHandler.push=ALL
```

2. システム・プロパティを使ってトレース構成ファイルを JVM に渡します。

`jsr4min.properties` を使用する場合、次のようになります:

```
java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
```

3. クライアントを実行します。

## タスクの結果

例外または問題が発生した場合、Paho Debug クラスは、構成されたファイル・ターゲットにメモリー内のトレースを書き込みます。

トレースは、生成時に自動的にファイルに書き込まれません。書き込みは、プッシュ・トリガーがヒットするか、デバッグ・クラスによってトレースの書き込みが行われる場合にのみ発生します。後者の場合、アプリケーション・コードの変更が必要になる可能性があります。

各行は、作成時にファイル・ハンドラーに書き込まれます。FileHandler を構成することにより、メッセージが書き出される形式を制御できます。Paho でカスタム・ファイル・ハンドラーが提供されています。これは SimpleHandler よりも多く書き込みを行います。JRE で提供されている XMLHandler ほどではありません。Paho のログ・フォーマッターを使用するトレース・レコードは、以下のいずれかの形式になります。

Level	Data and Time	Class	Method	Thread	clientID	Message
-------	---------------	-------	--------	--------	----------	---------

### 例

作業プロパティ・ファイル `jsr47min.properties` が提供されています。このファイルには、Paho MQTT クライアントに関連した問題の解決に役立つトレースの収集用に推奨されている構成が含まれています。これは、パフォーマンスへの影響は最小限に抑えつつ、トレースがメモリー内に継続的に収集されるよう構成します。プッシュ・トリガーが発生するか、特定のプッシュ要求が出されると、構成されたターゲット・ハンドラーにメモリー内のトレースがプッシュされます。デフォルトのプッシュ・トリガーは SEVERE レベルのメッセージですが、これは中断された接続です。デフォルトでは、メモリー内に収集されるトレースは、この時点で指定されたファイルに書き込まれます。デフォルトでは、このファイルは標準の `java.util.logging.FileHandler` です。Paho Debug クラスを使用して、メモリー・トレースをそのターゲットにプッシュすることができます。

JSR47 の完全な詳細は、Javadoc のパッケージ `java.util.logging` にあります。

```
# 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%.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.SimpleLogFormatter
```

## 次のタスク

トレースをプログラムで収集するために、デバッグ情報の収集に役立つユーティリティー・クラスが提供されています。このクラスには前述のログ・メッセージとトレース・メッセージが含まれますが、Paho クライアント内部から Java システム・プロパティーや変数の値などの情報を収集することができます。

デバッグ機能は、パブリック・クラス `Debug` で提供されます。これは、パッケージ `org.eclipse.paho.client.mqttv3.util` の一部です。非同期と同期の両方の MQTT クライアント・オブジェクトでメソッド `getDebug()` を使用することで、`Debug` のインスタンスを入手することができます。

以下に例を示します。

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

メソッド `dumpClientDebug()` は、最大のデバッグ情報量をダンプします。完全なデバッグ情報 (ログに書き込まれる) を収集するには、ログ機能を有効にする必要があります。完全なデバッグ情報を収集するには、問題発生タイミングが分かっている (例えば、特定の例外発生後など)、ダンプ・メソッドを呼び出します。

## MQTT JavaScript クライアントのトレース

接続されたクライアント・オブジェクトでメソッドを呼び出すようクライアント Web アプリケーションを変更することにより、JavaScript クライアントを使用してトレースを収集することができます。

### このタスクについて

トレースを収集するには、以下のメソッドを使用できます。

- `client.startTrace()` は、クライアントのトレースを開始します。
- `client.stopTrace()` は、クライアントのトレースを停止します。
- `client.getTraceLog()` は、現行のトレース・バッファーを返します。

トレース・バッファーを出力して、IBM ソフトウェア・サポートに送信することができます。これはいくつかの方法で行えます。例えば、トレースを開始した後、出力をコンソールと指定された E メール・アドレスの両方に送信し、最後にトレースを停止することができます。

```
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("onConnectionLost:"+responseObject.errorMessage);
    console.log(client.getTraceLog());
    window.location="mailto:helpdesk@"+location.hostname+
        "?Subject=Web%20Messaging%20Utility%20Trace&body="+
        client.getTraceLog().join("%0A");
    client.stopTrace();
};
```

出力例は次のとおりです。

```
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,
```

## V7.5.0.2 MQTT クライアントで SHA-2 暗号スイートを使用する場合のシステム要件

IBM SR13 以降の Java 6 では、SHA-2 暗号スイートを使用して、MQTT チャネルおよびクライアント・アプリケーションを保護することができます。ただし、SHA-2 暗号スイートは、IBM、SR4 以降の Java 7 以降ではデフォルトで有効になりません。そのため、以前のバージョンでは、必要なスイートを指定する必要があります。独自の JRE で MQTT クライアントを実行する場合、SHA-2 暗号スイートをサポートしていることを確認する必要があります。クライアント・アプリケーションで SHA-2 暗号スイートを使用する場合は、クライアントの SSL コンテキストを、Transport Layer Security (TLS) バージョン 1.2 をサポートする値に設定する必要もあります。

IBM SR4 以降の Java 7 では、SHA-2 暗号スイートがデフォルトで有効になります。IBM、SR13 以降のサービス・リリースの Java 6 では、暗号スイートを指定せずに MQTT チャネルを定義すると、チャネルは SHA-2 暗号スイートを使用したクライアントからの接続を受け入れません。SHA-2 暗号スイートを使用するには、チャネル定義で必要なスイートを指定する必要があります。これにより、MQTT サーバーでは接続前にスイートが使用可能になります。これは、指定したスイートを使用するクライアントのみがこのチャネルに接続できることも意味します。

Java 用の MQTT クライアントには同じような制約事項があります。クライアント・コードが IBM の Java 1.6 JRE で実行されている場合、必要な SHA-2 暗号スイートを明示的に使用可能にする必要があります。これらのスイートを使用するには、バージョン 1.2 の Transport Layer Security (TLS) プロトコルをサポートする値に SSL コンテキストを設定する必要もあります。以下に例を示します。

```
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);
```

2013 年 6 月現在、Internet Explorer 10 は JavaScript 用の MQTT メッセージング・クライアントで動作する唯一のブラウザであり、TLS 1.2 プロトコルもサポートしているため、JavaScript クライアントとの SHA-2 接続を行う場合に使用できる唯一のブラウザです。

現在サポートされている暗号スイートのリストについては、関連リンクを参照してください。

### 関連概念

#### 106 ページの『SSL を使用したクライアント認証のための MQTT クライアントの構成』

SSL を使用する MQTT クライアントを認証するには、クライアントは SSL を使用して遠隔測定チャンネルに接続します。クライアントは SSL クライアントを認証するように構成された遠隔測定チャンネルに対応する TCP ポートを指定する必要があります。

#### 109 ページの『SSL を使用したチャンネル認証のための MQTT クライアントの構成』

SSL を使用する遠隔測定チャンネルを認証するには、クライアントは SSL を使用して遠隔測定チャンネルに接続する必要があります。クライアントは SSL のために構成された遠隔測定チャンネルに対応するポートを指定する必要があります。この構成には、サーバーの私的に署名されたデジタル証明書を含む、パスフレーズで保護された鍵ストアが含まれている必要があります。



## V7.5.0.1 SSL を介したモバイル・メッセージング Web アプリケーションのブラウザ・サポートに関する制約事項

ブラウザの違いや、ブラウザが稼働するプラットフォームの違いにより、機能に違いが生じます。これらの違いを理解することで、JavaScript 用の MQTT メッセージング・クライアント over SSL および WebSockets を使用して接続するようにアプリケーション、認証局 (CA)、およびクライアント証明書を構成することができます。

SSL を介した JavaScript を使用するモバイル・メッセージングはかなり新しいものなので、ブラウザとプラットフォームの組み合わせが違えば、機能が実装される方法が多少異なったり、どの程度の機能が実装されるかが異なるのは意外なことではありません。以下の表は、ブラウザ (Firefox、Chrome、Internet Explorer、および Safari) とプラットフォーム (Windows、Linux、Mac、iOS、および Android) の組み合わせごとに、現在行える作業と行えない作業を概説しています。

表 6. プラットフォームおよびブラウザ別の SSL サポート. この表は、ブラウザとプラットフォームの組み合わせごとに、SSL 匿名および非匿名接続がサポートされているかどうか、またブラウザがすべての認証局 (CA) とクライアントの証明書をどの程度扱うかを示しています。

ブラウザ	SSL サポート (あり/なし)	SSL による CA の処理 (あり/なし)	詳細情報
Firefox デスクトップ	SSL 匿名 - あり SSL 非匿名 - あり	SSL 匿名 - あり SSL 非匿名 - あり	<p>CA とクライアントの証明書をブラウザに追加してください。</p> <p>Firefox は独自の証明書ストアを使用します。</p> <p>CA 証明書をインポートするには、<b>ツール</b> &gt; <b>「オプション」</b> &gt; <b>「拡張」</b> &gt; <b>暗号化</b> &gt; <b>証明書の表示</b> &gt; <b>認証局</b> &gt; <b>「インポート」</b> をクリックします。</p> <p>クライアント証明書をインポートするには、「<b>ツール</b>」 &gt; <b>「オプション」</b> &gt; <b>「拡張」</b> &gt; <b>暗号化</b> &gt; <b>証明書の表示</b> &gt; <b>証明書の表示</b> &gt; <b>「インポート」</b> をクリックします。</p> <p>セキュア接続を使用可能にするには、URL に <b>https://</b> を指定します。Firefox には、証明書を自動的に選択するか、毎回尋ねるかを選ぶオプションがあります。また Firefox には、SSL 3.0 と TLS 1.0 のどちらを使用するかを選ぶオプションもあります。両方とも選択していることを確認してください。</p>

表 6. プラットフォームおよびブラウザ別の SSL サポート. この表は、ブラウザとプラットフォームの組み合わせごとに、SSL 匿名および非匿名接続がサポートされているかどうか、またブラウザがすべての認証局 (CA) とクライアントの証明書をどの程度扱うかを示しています。(続き)

ブラウザ	SSL サポート (あり/なし)	SSL による CA の処理 (あり/なし)	詳細情報
Chrome デスクトップ	SSL 匿名 - あり SSL 非匿名 - あり	SSL 匿名 - あり SSL 非匿名 - あり	<p>このブラウザを使用して、CA とクライアントの証明書をオペレーティング・システムの証明書ストアに追加してください。このストアは他のソフトウェアと共有されます。</p> <p>CA 証明書をインポートするには、<b>設定 &gt; 詳細設定の表示 &gt; 「証明書の管理」 &gt; 「信頼されたルート証明機関」 &gt; 「インポート」</b> をクリックします。</p> <p>クライアント証明書をインポートするには、<b>設定 &gt; 詳細設定の表示 &gt; 「証明書の管理」 &gt; 個人 &gt; 「インポート」</b> をクリックします。</p> <p>セキュア接続を使用可能にするには、URL に <b>https://</b> を指定します。Chrome は複数の選択項目に関するプロンプトを出します。匿名接続と非匿名接続のどちらを構成しているかに応じて、正しい項目を選択してください。</p>

表 6. プラットフォームおよびブラウザ別の SSL サポート. この表は、ブラウザとプラットフォームの組み合わせごとに、SSL 匿名および非匿名接続がサポートされているかどうか、またブラウザがすべての認証局 (CA) とクライアントの証明書をどの程度扱うかを示しています。(続き)

ブラウザ	SSL サポート (あり/なし)	SSL による CA の処理 (あり/なし)	詳細情報
Internet Explorer.	SSL 匿名 - あり SSL 非匿名 - あり	SSL 匿名 - あり SSL 非匿名 - あり	<p>非匿名 SSL 接続を行う際には、正しいクライアント証明書を選択するよう促すプロンプトが出されます。</p> <p>Internet Explorer は Windows 証明書ストアを使用します。このストアは他のソフトウェアと共有されます。</p> <p>CA 証明書をインポートするには、「ツール」&gt;「インターネットオプション」&gt;「コンテンツ」&gt;「証明書」&gt;「信頼されたルート証明機関」&gt;「インポート」をクリックします。</p> <p>クライアント証明書をインポートするには、「ツール」&gt;「インターネットオプション」&gt;「コンテンツ」&gt;「証明書」&gt;「個人」&gt;「インポート」をクリックします。</p>
Safari デスクトップ	SSL 匿名 - あり SSL 非匿名 - あり	SSL 匿名 - あり SSL 非匿名 - あり	<p>このブラウザを使用して、CA とクライアントの証明書をオペレーティング・システムの証明書ストアに追加してください。このストアは他のソフトウェアと共有されます。</p>

表 6. プラットフォームおよびブラウザ別の SSL サポート. この表は、ブラウザとプラットフォームの組み合わせごとに、SSL 匿名および非匿名接続がサポートされているかどうか、またブラウザがすべての認証局 (CA) とクライアントの証明書をどの程度扱うかを示しています。(続き)

ブラウザ	SSL サポート (あり/なし)	SSL による CA の処理 (あり/なし)	詳細情報
Firefox on Android	SSL 匿名 - あり SSL 非匿名 - あり	SSL 匿名 - あり SSL 非匿名 - なし	<p>非匿名の場合: Firefox のリストにご使用の CA を追加する要件を満たせないで、クライアント証明書は処理できません。</p> <p>クライアント証明書をインポートするには、「設定」&gt;セキュリティ&gt;「資格情報ストレージ」をクリックします。ご使用の証明書がリスト内の信頼された CA によって署名されている場合は、セキュア接続を行うことができます。</p>
Chrome on Android	SSL 匿名 - あり SSL 非匿名 - あり	SSL 匿名 - あり SSL 非匿名 - なし	<p>非匿名の場合: Chrome のリストにご使用の CA を追加する要件を満たせないで、クライアント証明書は処理できません。</p> <p>注: Google 社は、Chrome のバージョン 27 でこのサポートを計画しています。バージョン 18 以降、公開された問題があります。</p> <p>クライアント証明書をインポートするには、「設定」&gt;セキュリティ&gt;「資格情報ストレージ」をクリックします。ご使用の証明書がリスト内の信頼された CA によって署名されている場合は、セキュア接続を行うことができます。</p>

表 6. プラットフォームおよびブラウザ別の SSL サポート. この表は、ブラウザとプラットフォームの組み合わせごとに、SSL 匿名および非匿名接続がサポートされているかどうか、またブラウザがすべての認証局 (CA) とクライアントの証明書をどの程度扱うかを示しています。(続き)

ブラウザ	SSL サポート (あり/なし)	SSL による CA の処理 (あり/なし)	詳細情報
Safari on iOS	SSL 匿名 - あり SSL 非匿名 - あり	SSL 匿名 - あり SSL 非匿名 - なし	<p>非匿名の場合: CA 証明書を同時にインストールしても、デバイスはクライアント証明書を信頼しません。</p> <p>Safari はデバイスの証明書ストアを使用します。このストアにインポートするには、<b>設定</b> &gt; 「<b>一般</b>」 &gt; 「<b>プロファイル</b>」をクリックし、Web ページから CA 証明書またはクライアント証明書を提供するか、自分に E メールで送信します。</p>
Chrome on iOS	SSL 匿名 - あり SSL 非匿名 - なし	SSL 匿名 - なし SSL 非匿名 - なし	<p>匿名の場合: Apple アプリケーションのみ iOS システム・ルート・ストアにアクセスできます。したがって、Chrome は独自の CA リストを使用しなければなりません。このリストに追加することはできません。</p> <p>非匿名の場合: リストにご使用の CA を追加する要件を満たさないで、クライアント証明書は処理できません。</p>

#### 関連タスク

79 ページの『[SSL を介した JavaScript 用の MQTT メッセージング・クライアント と WebSockets の接続](#)』  
SSL を使用する JavaScript 用の MQTT メッセージング・クライアント サンプル HTML ページと WebSocket protocol を使用して、Web アプリを IBM WebSphere MQ に安全に接続します。

#### 関連情報

Mozilla: (SSL) Firefox は Android CA ストレージと独自の CA ストレージのどちらを使用するか?

Chromium: [問題 134418 - クライアント証明書サポートの実装](#)

ie10 で信頼された証明書のない https サイトをオープンできない

## 問題の解決: MQTT クライアントが接続しない

MQTT クライアント・プログラムがテレメトリー (MQXR) サービスへの接続に失敗するという問題を解決します。

## 始める前に

この問題は、サーバー側の問題でしょうか、クライアント側の問題でしょうか、それとも接続の問題でしょうか? 独自の MQTT v3 プロトコル処理クライアント、または C または Java WebSphere MQTT クライアントを使用する MQTT クライアント・アプリケーションを作成しましたか?

WebSphere MQ Telemetry に付属する検査アプリケーションをサーバー上で実行し、テレメトリー・チャンネルとテレメトリー (MQXR) サービスが正しく実行されているかどうか確認してください。次に、その検査アプリケーションをクライアントに転送し、そこで検査アプリケーションを実行してください。

## このタスクについて

テレメトリー・サーバーに MQTT クライアントを接続できない、または接続されていないという結論に達する理由はいくつかあります。

## 手順

1. テレメトリー (MQXR) サービスが `MqttClient.Connect` に返した理由コードから引き出せる推論について検討します。それはどのタイプの接続の失敗ですか?

オプション	説明
<b>REASON_CODE_INVALID_PROTOCOL_VERSION</b>	ソケット・アドレスがテレメトリー・チャンネルに対応し、別のブローカーと同じソケット・アドレスを使用していないことを確認します。
<b>REASON_CODE_INVALID_CLIENT_ID</b>	クライアント ID が 23 バイト以下であり、A-Z, a-z, 0-9, '._/% の範囲の文字のみが含まれていることを確認してください。
<b>REASON_CODE_INVALID_DESTINATION</b>	クライアント ID がキュー・マネージャー名と同じでないことを確認します。
<b>REASON_CODE_SERVER_CONNECT_ERROR</b>	テレメトリー (MQXR) サービスとキュー・マネージャーが正常に実行されていることを確認します。 <b>netstat</b> を使用して、ソケット・アドレスが別のアプリケーションに割り振られていないことを確認します。

IBM WebSphere MQ Telemetry に付属のいずれかのライブラリーを使用する代わりに、MQTT クライアント・ライブラリーを作成した場合は、CONNACK 戻りコードを調べてください。

これら 3 つのエラーから、クライアントがテレメトリー (MQXR) サービスに接続されたものの、このサービスがエラーを検出したと推測できます。

2. テレメトリー (MQXR) サービスが応答しないときにクライアントが生成する理由コードから引き出せる推論について検討します。

オプション	説明
<b>REASON_CODE_CLIENT_EXCEPTION</b> <b>REASON_CODE_CLIENT_TIMEOUT</b>	サーバーにある FDC ファイルを探します。165 ページの『サーバー・サイドのログ』を参照してください。テレメトリー (MQXR) サービスは、クライアントがタイムアウトになったことを検出すると、First Failure Data Capture (FDC) ファイルを作成します。このサービスは、予期せずに接続が中断するたびに毎回 FDC ファイルを作成します。

テレメトリー (MQXR) サービスがクライアントに回答しなかった可能性があり、クライアント側でタイムアウトになります。WebSphere MQ Telemetry Java クライアントは、アプリケーションが無期限のタイムアウトを設定した場合にのみハングします。このクライアントは `MqttClient.Connect` に対し

て設定されたタイムアウトを過ぎると、未診断の接続の問題とともに、これらの例外の1つをスローします。

接続の失敗と関連している FDC ファイルが見つからない限り、クライアントがサーバーとの接続を試みたことを推測することはできません。

a) クライアントが接続要求を送信したことを確認します。

<https://java.net/projects/tcpmon> から入手できる **tcpmon** などのツールを使用して、TCPIP 要求を確認します。

b) このクライアントによって使用されるリモート・ソケット・アドレスは、テレメトリー・チャンネルに対して定義されているソケット・アドレスと一致しますか？

IBM WebSphere MQ Telemetry に付属する Java SE MQTT クライアントのデフォルトのファイル・パーシスタンス・クラスは、以下の名前のフォルダーを作成します。 *clientIdentifier*-クライアント作業ディレクトリーの *tcpHostNameport* または *clientIdentifier-sslHostNameport*。フォルダー名によって、接続試行に使用される *hostName* および *port* が分かります。[167 ページの『クライアント・サイドのログ・ファイル』](#)を参照してください。

c) リモート・サーバー・アドレスを ping できますか？

d) サーバー上の **netstat** は、クライアントが接続しているポートでテレメトリー・チャンネルが実行されていることを示していますか？

3. テレメトリー (MQXR) サービスがクライアント要求で問題を検出したかどうか調べます。

テレメトリー (MQXR) サービスは、検出したエラーを *mqxr.log* に書き込み、キュー・マネージャーはエラーを *AMQERR01.LOG* に書き込みます。

4. 別のクライアントを実行して、問題の分離を試みます。

- 同じテレメトリー・チャンネルを使用して、MQTT サンプル・アプリケーションを実行します。
- **wmqttSample** GUI クライアントを実行して、接続を検査します。 [SupportPac IA92](#) をダウンロードして、**wmqttSample** を入手します。

注：旧バージョンの IA92 には、MQTT v3 Java クライアント・ライブラリーが含まれていません。

サーバー・プラットフォーム上でサンプル・プログラムを実行してネットワーク接続に関するあいまいさを排除し、クライアント・プラットフォーム上でそれらのサンプル・プログラムを実行します。

5. さらに、以下の点を確認します。

a) 何万もの MQTT クライアントが同時に接続を試行していますか？

テレメトリー・チャンネルには、着信接続のバックログをバッファーに入れるためのキューがあります。1秒間に10,000を上回る接続が処理されます。IBM WebSphere MQ エクスプローラーのテレメトリー・チャンネル・ウィザードを使って、バックログ・バッファーのサイズを構成することができます。そのデフォルトのサイズは4096です。バックログが低い値に構成されていないことを確認します。

b) テレメトリー (MQXR) サービスとキュー・マネージャーは引き続き実行されていますか？

c) クライアントは、TCPIP アドレスに切り替えられた高可用性キュー・マネージャーに接続されましたか？

d) ファイアウォールはアウトバウンド・データ・パケットまたは戻りデータ・パケットを選択的にフィルタに掛けていますか？

## 問題の解決: MQTT クライアントの接続が切断される

正常に接続され、短時間または長時間実行された後で、予期しない `ConnectionLost` 例外をクライアントがスローする原因となっているものを突き止めます。

## 始める前に

MQTT クライアントの接続は正常に行われました。クライアントの接続はしばらく続く可能性があります。クライアントが短い間隔でのみ開始している場合、接続が成功してから接続が切断するまでの時間は短い可能性があります。

切断された接続と、正常に行われた後に切断された接続を見分けるのは難しくありません。ドロップされた接続は、`MqttCallback.ConnectionLost` メソッドを呼び出す MQTT クライアントによって定義されます。このメソッドは、接続が正常に確立されてからでなければ呼び出されません。症状は、否定応答を受信するかタイムアウトになった後で例外をスローする `MqttClient.Connect` とは異なります。

IBM WebSphere MQ に付属の MQTT クライアント・ライブラリーを MQTT クライアント・アプリケーションが使用していない場合、症状はクライアントによって異なります。MQTT v3 プロトコルでは、サーバーへの要求に対する応答がタイムリーに行われず、あるいは TCP/IP 接続が失敗する、という症状が現れます。

## このタスクについて

MQTT クライアントは、接続の肯定応答を受信した後に発生したサーバー・サイドの問題に対して、スロー可能な例外とともに `MqttCallback.ConnectionLost` を呼び出します。MQTT クライアントが `MqttTopic.publish` および `MqttClient.subscribe` から戻ると、メッセージを送受信する役割を持つ MQTT クライアント・スレッドに要求を転送します。サーバー・サイドのエラーは、スロー可能な例外を `ConnectionLost` コールバック・メソッドに渡すことによって非同期で報告されます。

テレメトリー (MQXR) サービスは、接続を切断する場合には常に、`First Failure Data Capture` ファイルを作成します。

## 手順

1. 同じ `ClientIdentifier` を使用する別のクライアントが開始しましたか？

同じ `ClientIdentifier` を使用して、2 番目のクライアントが開始した場合、または同じクライアントが再始動した場合、最初のクライアントとの最初の接続が切断されます。

2. クライアントはパブリッシュまたはサブスクライブを行う許可が与えられていないトピックにアクセスしましたか？

テレメトリー・サービスがクライアントに代わってアクションを行い、そのアクションが `MQCC_FAIL` を戻した場合、サービスはクライアント接続を切断します。

理由コードはクライアントに返されません。

- `mqxr.log` および `AMQERR01.LOG` ファイルでログ・メッセージを探し、クライアントが接続されているキュー・マネージャーがないか調べてください ([165 ページの『サーバー・サイドのログ』](#)を参照)。

3. TCP/IP 接続は切断されましたか？

TCP/IP 接続を非アクティブとしてマーキングするためのファイアウォールのタイムアウト設定が低かったために接続が切断された可能性があります。

- `MqttConnectOptions.setKeepAliveInterval` を使って、非アクティブの TCP/IP 接続時間を短くしてください。

## 問題の解決: MQTT アプリケーションで失われたメッセージ

メッセージが失われる問題を解決します。メッセージは非持続メッセージですか、間違った場所に送信されましたか、それともまったく送信されませんでしたか？ クライアント・プログラムのコード化に誤りがあると、メッセージが失われる可能性があります。



## 始める前に

送信したメッセージが失われたというのはどれほど確実なことですか? メッセージが失われるのはメッセージが受信されなかったためであると推測できますか? メッセージがパブリケーションの場合、失われるメッセージは、パブリッシャーによって送信されるメッセージですか、それともサブスクライバーに送信されるメッセージですか?あるいは、サブスクリプションが失われましたか? ブローカーはそのサブスクリプションのパブリケーションをサブスクライバーに送信していませんか?

解決策にクラスターまたはパブリッシュ/サブスクライブ階層を使用する分散パブリッシュ/サブスクライブが関係する場合、多くの構成上の問題の結果として、メッセージが失われたように見える可能性があります。

サービス品質が「最低1回」または「最高1回」のメッセージを送信した場合、失われたと思われるメッセージは、期待どおりに送達されなかった可能性があります。メッセージが誤ってシステムから削除された可能性は低いかもしれませんが、期待していたパブリケーションまたはサブスクリプションの作成に失敗した可能性があります。

失われたメッセージの問題判別を行う際に取りべき最も重要なステップは、メッセージが失われたことを確認することです。このシナリオを再現し、さらにメッセージを失わせます。システムがメッセージをすべて破棄してしまうことがないようにするために、サービスの品質「最低1回」または「最高1回」を使用してください。

## このタスクについて

失われたメッセージは、以下の4つのレグで診断します。

1. 設計どおりに機能する「応答不要送信」メッセージ。「応答不要送信」メッセージはシステムによって廃棄されることもあります。
2. 構成: 分散環境において正しい権限でパブリッシュ/サブスクライブをセットアップするのは、簡単ではありません。
3. クライアントのプログラミング・エラー: メッセージ送達の責任は、IBMによって作成されたコードだけの責任ではありません。
4. すべての可能性が消えたら、IBM サービスに援助を依頼することができます。

## 手順

1. 失われたメッセージのサービス品質が「応答不要送信」だった場合、サービス品質「最低1回」または「最高1回」を設定します。もう一度メッセージを失ってみます。
  - 「応答不要送信」のサービス品質で送信されるメッセージは、次のような多くの状況において IBM WebSphere MQ によって破棄されます。
    - 通信が失われて、チャンネルが停止した。
    - キュー・マネージャーがシャットダウンした。
    - メッセージが多すぎる。
  - 「応答不要送信」メッセージの送達は、TCP/IP の信頼性に依存します。TCP/IP は、送達が確認されるまでデータ・パケットの再送信を続けます。TCP/IP セッションが中断されると、サービス品質が「応答不要送信」のメッセージは失われます。セッションは、クライアントまたはサーバーの終了、通信の問題、またはファイアウォールでのセッションの切断により中断される可能性があります。
2. サービス品質が「最低1回」または「最高1回」の未配布メッセージを再送信するために、クライアントが前回のセッションを再始動していることを確認します。
  - a) クライアント・アプリケーションが Java SE MQTT クライアントを使用している場合は、`MqttClient.CleanSession` が `false` に設定されていることを確認します。
  - b) 別のクライアント・ライブラリーを使用している場合は、セッションが正しく再始動していることを確認します。
3. クライアント・アプリケーションが間違っって別のセッションを開始するのではなく、同じセッションを再開することを確認します。

同じセッションをもう一度開始するには、`cleanSession = false` とし、`Mqttclient.clientIdentifier` と `MqttClient.serverURI` は前回のセッションと同じでなければなりません。

4. セッションが予定より早く閉じる場合、クライアント側のパーシスタンス・ストアにあるメッセージを再送信できることを確認します。
  - a) クライアント・アプリケーションが Java SE MQTT クライアントを使用している場合は、メッセージがパーシスタンス・フォルダーに保存されていることを確認します。167 ページの『[クライアント・サイドのログ・ファイル](#)』を参照してください。
  - b) 別のクライアント・ライブラリーを使用している場合、または独自の持続性メカニズムを実装した場合、それが正しく機能していることを確認します。
5. メッセージが送達される前に、誰もそのメッセージを削除していないことを確認します。

MQTT クライアントへの送達を待つ未配布メッセージは `SYSTEM.MQTT.TRANSMIT.QUEUE` に保管されます。テレメトリー・サーバーへの送達を待つメッセージは、クライアントの持続性メカニズムによって保管されます ([MQTT クライアントでのメッセージ持続性](#)を参照)。

6. クライアントが、受信することを期待しているパブリケーションのサブスクリプションを持っていることを確認します。

WebSphere MQ エクスプローラーを使用するか、`runmqsc` または PCF コマンドを使用して、サブスクリプションをリストします。すべての MQTT クライアントのサブスクリプションの名前が示されます。`ClientIdentifier:Topic name` という形式の名前が付けられます。

7. パブリッシュを行う権限がパブリッシャーにあり、パブリケーション・トピックにサブスクライブする権限がサブスクライバーにあることを確認します。

```
dspmqaut -m qMgr -n topicName -t topic -p user ID
```

クラスター化されたパブリッシュ/サブスクライブ・システムでは、サブスクライバーの接続先のキュー・マネージャー上のトピックに対する権限がサブスクライバーに付与されている必要があります。パブリケーションがパブリッシュされるキュー・マネージャー上のトピックにサブスクライブするためにサブスクライバーに権限を付与する必要はありません。キュー・マネージャー間のチャンネルには、プロキシー・サブスクリプションを渡し、パブリケーションを転送するための権限が正しく付与されている必要があります。

IBM WebSphere MQ エクスプローラーを使って同じサブスクリプションを作成し、そのサブスクリプションにパブリッシュします。クライアント・ユーティリティーを使って、パブリッシュおよびサブスクライブを行うアプリケーション・クライアントをシミュレートします。IBM WebSphere MQ エクスプローラーからユーティリティーを開始し、クライアント・アプリケーションによって採用されるものと一致するよう、そのユーザー ID を変更します。

8. パブリケーションを `SYSTEM.MQTT.TRANSMIT.QUEUE` に置く権限がサブスクライバーにあることを確認します。

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

9. IBM WebSphere MQ Point-to-Point アプリケーションに、メッセージを `SYSTEM.MQTT.TRANSMIT.QUEUE` に書き込む権限があることを確認します。

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

[MQTT クライアントにメッセージを送信するための分散キューイングの構成](#)の「"クライアントへの直接送信"」を参照してください。

## 問題の解決: テレメトリー (MQXR) サービスが開始しない

テレメトリー (MQXR) サービスの開始に失敗するという問題を解決します。WebSphere MQ Telemetry のインストールを確認し、ファイルが欠落および移動していないこと、あるいはそのファイルの権限が間違っていないことを確認します。テレメトリー (MQXR) サービスによって使用されるパスに、テレメトリー (MQXR) サービスのプログラムが含まれていることを確認します。

## 始める前に

WebSphere MQ Telemetry フィーチャーをインストールしておきます。IBM WebSphere MQ エクスプローラーの **IBM WebSphere MQ > キュー・マネージャー > qMgrName > Telemetry** に Telemetry フォルダがあります。このフォルダが存在しない場合は、インストールに失敗しています。

テレメトリー (MQXR) サービスを開始するには、このサービスを作成しておく必要があります。テレメトリー (MQXR) サービスが作成されていない場合は、**サンプル構成の定義 ...** を実行します。Telemetry フォルダ内のウィザード。

遠隔測定 (MQXR) サービスが事前に開始されている場合は、Telemetry フォルダの下に追加の **Channels** および **Channel Status** フォルダが作成されています。遠隔測定サービス SYSTEM.MQXR.SERVICE は、**Services** フォルダに入ります。このフォルダは、エクスプローラーで「システム・オブジェクト」を表示するためのラジオ・ボタンをクリックすると可視になります。

SYSTEM.MQXR.SERVICE を右クリックしてサービスを開始および停止し、その状況を表示し、ユーザー ID にサービスを開始する権限があるかどうかを表示します。

## このタスクについて

SYSTEM.MQXR.SERVICE テレメトリー (MQXR) サービスの開始が失敗します。開始に失敗した場合の症状の現れ方には、以下の 2 つがあります。

1. 開始コマンドが即時に失敗する。
2. 開始コマンドは成功するが、その直後にサービスが停止する。

## 手順

1. サービスを開始します。

### 結果

サービスが即時に停止します。次の例に示したようなエラー・メッセージがウィンドウに表示されます。

```
WebSphere MQ cannot process the request because the executable specified cannot be started. (AMQ4160)
```

### 理由

インストールにファイルが欠落しているか、インストールされているファイルの権限の設定が間違っています。

IBM WebSphere MQ Telemetry フィーチャーは、高可用性キュー・マネージャーのペアの一方だけにインストールされます。キュー・マネージャー・インスタンスは、スタンバイに切り替わると、SYSTEM.MQXR.SERVICE の開始を試みます。テレメトリー (MQXR) サービスはスタンバイにはインストールされていないため、サービスを開始するコマンドは失敗します。

### 検査

エラー・ログを調べます。[165 ページの『サーバー・サイドのログ』](#)を参照してください。

### Actions

WebSphere MQ Telemetry フィーチャーをインストールするか、またはアンインストールして再インストールします。

2. サービスを開始します。30 秒間待機します。エクスプローラーを最新表示し、サービス状況を確認します。

### 結果

サービスが開始した後、停止します。

### 理由

SYSTEM.MQXR.SERVICE は **runMQXRService** コマンドを開始しましたが、このコマンドは失敗しました

## 検査

エラー・ログを調べます。[165 ページの『サーバー・サイドのログ』](#)を参照してください。定義済みのサンプル・チャンネルでのみ問題が発生しているかどうか調べます。`WMQ data directory\Qmgrs\qMgrName\mqxr\` ディレクトリーの内容をバックアップし、クリアします。サンプル構成ウィザードを実行し、サービスの開始を試みます。

## Actions

権限およびパスの問題を調べます。

## 問題の解決: JAAS ログイン・モジュールがテレメトリー・サービスによって呼び出されない

JAAS ログイン・モジュールがテレメトリー (MQXR) サービスによって呼び出されていないかどうかを突き止め、問題が修正されるように JAAS を構成します。

### 始める前に

`WMQ installation directory\mqxr\samples>LoginModule.java` を変更して、独自の認証クラス `WMQ installation directory\mqxr\samples\samples>LoginModule.class` を作成しました。あるいは、独自の JAAS 認証クラスを作成し、選択したディレクトリーにそれを配置しておきます。テレメトリー・サービス (MQXR) で少し初期テストしたところ、認証クラスがテレメトリー (MQXR) サービスによって呼び出されていないのではないかと疑いがあります。

注: WebSphere MQ に適用されている保守によって認証クラスが上書きされないように保護します。WebSphere MQ ディレクトリー・ツリー内のパスではなく、認証クラスの独自のパスを使用します。

### このタスクについて

タスクでは、シナリオを使用して問題を解決する方法を示します。このシナリオでは、`security.jaas` というパッケージに、`JAASLogin.class` という JAAS 認証クラスが含まれています。これは、パス `C:\WMQTelemetryApps\security\jaas` に保管されます。IBM WebSphere MQ Telemetry の JAAS の構成については、[テレメトリー・チャンネル JAAS 構成](#) を参照してください。[189 ページの『JAAS の構成例』](#)の例は、サンプル構成です。

### 手順

1. `javax.security.auth.login.LoginException` によってスローされた例外については、`mqxr.log` を参照してください。  
`mqxr.log` へのパスについては [165 ページの『サーバー・サイドのログ』](#) を参照し、このログでリストされる例外の例については [190 ページの図 54](#) を参照してください。
2. JAAS 構成を [189 ページの『JAAS の構成例』](#) で扱った例と比較して修正します。
3. サンプルの `JAASLoginModule` をリファクタリングして認証パッケージに入れた後に、このサンプルによってログイン・クラスを置き換え、同じパスを使ってその認証パッケージをデプロイします。`loggedIn` の値を `true` と `false` の間で切り替えます。  
`loggedIn` が `true` のときには問題が発生せず、`loggedIn` が `false` のときには同じように問題が発生する場合、問題はログイン・クラスにあります。
4. この問題が、認証の問題ではなく許可の問題であるかどうかを調べます。
  - a) 固定されたユーザー ID を使って許可検査を実行するように、テレメトリー・チャンネル定義を変更します。`mqm` グループのメンバーであるユーザー ID を選択します。
  - b) クライアント・アプリケーションを再実行します。  
問題が発生しなくなった場合、解決策は、許可のために渡されるユーザー ID にあります。渡されているユーザー名は何ですか? ユーザー名をログイン・モジュールからファイルに出力してください。

IBM WebSphere MQ エクスプローラーまたは **dspmqauth** を使用して、そのユーザー名のアクセス権限を調べます。

## JAAS の構成例

WebSphere MQ エクスプローラーの「新規テレメトリー・チャンネル」ウィザードを使用して、テレメトリー・チャンネルを構成します。クライアントはポート 1884 で接続し、JAASMCUser テレメトリー・チャンネルに接続します。189 ページの図 48 に、このテレメトリー・ウィザードによって作成されたテレメトリー・プロパティ・ファイルの一例を示します。このファイルを直接編集しないでください。遠隔測定チャンネルは JAAS を使って認証を行います。その際、JAASConfig という構成を使用します。クライアントは、認証を行った後、ユーザー ID Admin を使って、IBM WebSphere MQ オブジェクトへのアクセスを許可します。

```
com.ibm.mq.MQXR.channel/JAASMCUser: \  
com.ibm.mq.MQXR.Port=1884;\  
com.ibm.mq.MQXR.JAASConfig=JAASConfig;\  
com.ibm.mq.MQXR.UserName=Admin;\  
com.ibm.mq.MQXR.StartWithMQXRService=true
```

図 48. WMQ Installation directory\data\mqgrs\qMgrName\mqxr\mqxr\_win.properties

JAAS 構成ファイルには、Java クラス security.jaas.JAASLogin を指定する JAASConfig というスタンプがあります。JAAS はこのスタンプを使用してクライアントを認証します。

```
JAASConfig {  
    security.jaas.JAASLogin required debug=true;  
};
```

図 49. WMQ Installation directory\data\mqgrs\qMgrName\mqxr\jaas.config

SYSTEM.MQTT.SERVICE が開始すると、189 ページの図 50 のパスがそのクラスパスに追加されます。

```
CLASSPATH=C:\WMQTelemetryApps;
```

図 50. WMQ Installation directory\data\mqgrs\qMgrName\service.env

189 ページの図 51 は、テレメトリー (MQXR) サービスに対してセットアップされているクラスパスに追加された、189 ページの図 50 の追加パスを示しています。

```
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\..\lib\java\lib\com.ibm.mq.jmqi.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\lib\java\lib\com.ibm.mqjms.jar;  
C:\IBM\MQ\Program\mqxr\bin\..\lib\java\lib\com.ibm.mq.jar;  
C:\WMQTelemetryApps;
```

図 51. runMQXRService.bat からのクラスパス出力

190 ページの図 52 の出力は、189 ページの図 48 で示されているチャンネル定義を使ってテレメトリー (MQXR) サービスが開始されたことを示しています。

```
21/05/2010 15:32:12 [main] com.ibm.mq.MQXRService.MQXRPropertiesFile
AMQXR2011I: Property com.ibm.mq.MQXR.channel/JAASMCUser 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
```

図 52. WMQ Installation directory\data\qmgrs\qMgrName\errors\mqxr.log

クライアント・アプリケーションが JAAS チャネルに接続する際、`com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` が `jaas.config` ファイル内の JAAS スタンザの名前と一致しない場合は接続が失敗し、クライアントは戻りコード 0 と共に例外をスローします (190 ページの図 53 を参照)。2 番目の例外「Client is not connected (32104)」がスローされたのは、クライアントが、接続されていないのに切断を試みたからです。

```
C:\WMQTelemetryApps>java com.ibm.mq.id.PubAsyncRestartable
Starting a clean session for instance "Admin_PubAsyncRestartab"
Publishing "Hello World Fri May 21 17:23:23 BST 2010" on topic "MQTT Example"
for client instance: "Admin_PubAsyncRestartab" using QoS=1 on address tcp://localhost:1884"
userid: "Admin", Password: "Password"
Delivery token "528752516" has been received: false
Connection lost on instance "Admin_PubAsyncRestartab" with cause "MqttException"
MqttException (0) - java.io.EOFException
    at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:118)
    at java.lang.Thread.run(Thread.java:801)
Caused by: java.io.EOFException
    at java.io.DataInputStream.readByte(DataInputStream.java:269)
    at
com.ibm.micro.client.mqttv3.internal.wire.MqttInputStream.readMqttWireMessage(MqttInputStream.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(ExceptionHelper.java:33)
)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:100)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNowait(ClientComms.java:117)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.disconnect(ClientComms.java:229)
    at com.ibm.micro.client.mqttv3.MqttClient.disconnect(MqttClient.java:385)
    at com.ibm.mq.id.PubAsyncRestartable.main(PubAsyncRestartable.java:49)
```

図 53. `com.ibm.mq.id.PubAsyncRestartable` の接続時にスローされる例外

`mqxr.log` には、190 ページの図 53 に示されている追加の出力が含まれています。

このエラーは、「No LoginModules configured for JAAS (JAAS 用に構成されている LoginModules が無い)」が原因で `javax.security.auth.login.LoginException` をスローする JAAS によって検出されます。このエラーは、190 ページの図 54 に示されているように、構成名が不適切であることが原因である可能性があります。また、JAAS 構成のロード時に JAAS で発生した他の問題が原因である可能性もあります。

例外が JAAS によって報告されない場合は、`JAASConfig` スタンザで指定されている `security.jaas.JAASLogin` クラスが JAAS によって正常にロードされています。

```
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
```

図 54. `mqxr.log` - JAAS 構成のロード・エラー

## 問題の解決: デーモンの開始または実行

デーモンに関する問題のトラブルシューティングを行うには、IBM WebSphere MQ Telemetry デーモンのデバイス・コンソール・ログを参照するか、トレースをオンにするか、あるいはこのトピック内の症状表を使用します。

### 手順

1. コンソール・ログを確認します。

デーモンがフォアグラウンドで実行されている場合、コンソール・メッセージは端末ウィンドウに表示されます。デーモンがバックグラウンドで開始された場合、コンソールは stdout のリダイレクト先です。

2. デーモンを再始動します。

構成ファイルに対する変更は、デーモンを再始動するまでアクティブになりません。

3. [191 ページの表 7](#) を参照してください。

問題	推奨される解決法
Windows でデーモンを開始すると、次のメッセージが表示されます。  システムは、指定されたプログラムを実行できません または アプリケーションの開始に失敗しました。 理由: 横並び構成が間違っています。	Microsoft Visual C++ 2008 再頒布可能パッケージをインストールします。
2 つ以上のデーモンまたは MQTT 対応サーバーがブリッジによって相互接続されており、プロセッサに過大負荷がかかっています。	あるサーバーから別のサーバーへ 1 つ以上のメッセージが繰り返し渡されて、メッセージ・ループになっている可能性があります。構成ファイル内のトピック・パラメーターを調べてください。可能などころでは、より特定のなトピックを使用してください。大ざっぱなワールドカード文字を両方向に使用することが、接続ループの最も一般的な原因となります。
ブリッジは、他の MQTT クライアントが接続可能なリモート MQTT 対応サーバーに接続できません。	リモート・サーバーは、そのリモート・サーバーが WebSphere MQ Telemetry デーモン (デバイス用) でもあるかどうかの判別を試行できない場合があります。 <b>try_private</b> を <b>off</b> に設定して、メッセージ・ループを除去する特別な処理を使用不可にしてください。
ブリッジを構成するときに以下のメッセージが出力されます。  警告: 接続はソケット 1888 上の最初のパケットではありませんでした。CONNACK を取得しました。	ローカル・デーモンにループバックするようにブリッジが構成されていることが考えられます。ループバックはサポートされていません。

## 問題の解決: MQTT クライアントがデーモンに接続していない

クライアントがデーモンに接続していないか、あるいはデーモンが他のデーモンまたは WebSphere MQ Telemetry チャンネルに接続していません。

### このタスクについて

デーモンが送受信する各 MQTT パケットをトレースします。

## 手順

デーモン構成ファイルで **trace\_output** パラメーターを **protocol** に設定するか、あるいは **amqtdd.upd** ファイルを使用してコマンドをデーモンに送信します。

**amqtdd.upd** ファイルの使用例については、[IBM WebSphere MQ Telemetry デーモン \(デバイス用\) と IBM WebSphere MQ 間のメッセージの転送](#) を参照してください。

プロトコル設定を使用すると、デーモンは送受信する各 MQTT パケットについて説明したメッセージをコンソールに出力します。



## 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 103-8510

東京都中央区日本橋箱崎町 19 番 21 号

日本アイ・ビー・エム株式会社

日本アイ・ビー・エム株式会社

法務・知的財産

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

〒 103-8510

103-8510

東京 103-8510、日本

**以下の保証は、国または地域の法律に沿わない場合は、適用されません。** INTERNATIONAL BUSINESS MACHINES CORPORATION は、法律上の瑕疵担保責任、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。"" 国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

東京都中央区日本橋箱崎町 19 番 21 号

日本アイ・ビー・エム株式会社

Software Interoperability Coordinator, Department 49XA

3605 Highway 52 N

Rochester, MN 55901

U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性があります。その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、名前や住所が類似する個人や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほめかしたり、保証することはできません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

## プログラミング・インターフェース情報

プログラミング・インターフェース情報 (提供されている場合) は、このプログラムで使用するアプリケーション・ソフトウェアの作成を支援することを目的としています。

本書には、プログラムを作成するユーザーが IBM WebSphere MQ のサービスを使用するためのプログラミング・インターフェースに関する情報が記載されています。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

**重要:** この診断、修正、およびチューニング情報は、変更される可能性があるため、プログラミング・インターフェースとして使用しないでください。

## 商標

IBM、IBM ロゴ、ibm.com® は、世界の多くの国で登録された IBM Corporation の商標です。現時点での IBM の商標リストについては、"Copyright and trademark information" [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml) をご覧ください。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

この製品には、Eclipse Project (<http://www.eclipse.org/>) により開発されたソフトウェアが含まれています。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。







部品番号:

(1P) P/N: