9.4

IBM MQ の技術概要

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、309 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® MQ バージョン 9 リリース 4、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。

目次

IBM MQ の技術概要

IBM MQ を使用して、アプリケーションに接続し、組織全体での情報の分散を管理します。

IBM MQ により、複数のプログラムは、一貫性のあるアプリケーション・プログラミング・インターフェースを使用して、異なるコンポーネント (プロセッサー、オペレーティング・システム、サブシステム、および通信プロトコル) のネットワーク間で相互に通信することができます。 このインターフェースを使用して設計および作成されるアプリケーションは、メッセージ・キューイング・アプリケーションとして知られています。

以下のサブトピックを使用して、IBM MQ で提供されるメッセージ・キューイングおよび他の機能についての情報を得ることができます。

関連概念

IBM MQ の概要

製品の要件とサポート情報を確認できる場所

関連タスク

IBM MQ アーキテクチャーの計画

関連資料

7 ページの『メッセージ・キューイングの主な機能とメリット』

ここでは、メッセージ・キューイングの主な機能とメリットを中心に説明します。 メッセージ・キューイングのセキュリティーやデータ保全性などについて説明します。

メッセージ・キューイングの概要

IBM MQ 製品では、整合性のあるアプリケーション・プログラミング・インターフェースを使用して、性質の異なるコンポーネント (プロセッサー、オペレーティング・システム、サブシステム、および通信プロトコル) のネットワーク内でプログラムが相互に通信できるようにしています。

メッセージング とキューイング のスタイルをとるため、このインターフェースを使用して設計、作成されるアプリケーションをメッセージ・キューイング・アプリケーションといいます。

- メッセージングでは、プログラムは、相互に直接呼び出す代わりに、メッセージでデータを送信し合うことにより通信します。
- キューイングでは、メッセージがストレージ内のキューに配置されるので、複数のプログラムが互いに独立して、異なるスピードで、異なる時刻に、異なる場所で、プログラム間の論理接続をもたずに稼働できるようになります。

メッセージ・キューイングはデータ処理の分野で長年用いられてきました。 現在では、電子メールの分野で広く使われています。 キューイングがない場合、電子メッセージをリモート側で送信するには、経路上のすべてのノードがメッセージを転送でき、受信局をログオンさせて、メッセージを送信しようとしていることを常に認識させることが必要です。 キューイング・システムでは、メッセージはシステムがそれを転送できるようになるまで中間ノードに格納されます。 最終の宛先では、受信局が読み取り可能になるまで、メッセージは電子メールボックスに格納されます。

しかし、今日では多くの複雑な業務トランザクションがキューイングを使用しないで処理されています。 大規模ネットワークでは、システムが膨大な数の接続を使用可能な状態にしていなければなりません。 システムの一部で問題が発生すると、システムの多くの部分が使用できなくなります。

メッセージ・キューイングは、プログラム用の電子メールと考えることができます。 メッセージ・キューイング環境では、アプリケーション・スイートの各部分を構成するそれぞれのプログラムが、特定の要求に応答する形で、明確に定義された自己完結型の機能を実行します。 他のプログラムと通信するには、プログラムは事前定義キューにメッセージを書き込む必要があります。 他のプログラムは、そのメッセージをキューから取り出し、そこに入っている要求や情報を処理します。 このため、メッセージ・キューイングはプログラム間通信の 1 つのスタイルと言えます。

キューイングとは、アプリケーションがメッセージの処理が行えるようになるまでメッセージを保留する機能のことです。 キューイングによって次のことが可能になります。

- 通信コードを記述することなく、プログラム間で通信を行うことができる (これらのプログラムはそれぞれ異なった環境で実行中でも構わない)。
- プログラムがメッセージを処理する順序を選択することができる。
- メッセージの数がしきい値を超えるときは、複数のプログラムが 1 つのキューをサービスするようにして、システム上の負荷のバランスをとることができる。
- 基本システムが使用不可のときは、代替システムがキューをサービスするようにして、アプリケーションの使用可能度を向上させることができる。

メッセージ・キューとは

メッセージ・キューは、単にキューと呼ばれており、メッセージを送信することができる名前付きの宛先です。 メッセージは、それらのキューを取り扱うプログラムによって取り出されるまで、キューに蓄積されます。

キューは、キュー・マネージャー内にあり、キュー・マネージャーによって管理されます (9 ページの『メッセージ・キューイングの用語』を参照)。 キューの物理的な特性は、キュー・マネージャーが実行されているオペレーティング・システムに依存します。 キューはコンピューターのストレージ内の揮発性バッファー域にあっても、あるいはディスクなどの永続記憶装置上のデータ・セットにあっても構いません。 キューの物理的な管理は、キュー・マネージャーの役目であり、関連するアプリケーション・プログラムには明らかにされません。

プログラムは、キュー・マネージャーの外部サービスを通じてのみキューにアクセスします。 外部サービスは、キューのオープン、キューへのメッセージの書き込み、キューからのメッセージの読み取り、およびキューのクローズを行うことができます。 また、キューの属性を設定したり、照会したりすることもできます。

さまざまなスタイルのメッセージ・キューイング

Point-to-Point

1 つのメッセージがキューに入れられ、1 つのアプリケーションがそのメッセージを受け取ります。

Point-to-Point メッセージングでは、送信側アプリケーションが受信側アプリケーションにメッセージを送信する前に、送信側アプリケーションが受信側アプリケーションについての情報を認識している必要があります。 例えば、送信側アプリケーションは、情報を送信するキューの名前を認識していて、キュー・マネージャー名を指定しなければならない場合があります。

パブリッシュ/サブスクライブ

パブリッシング・アプリケーションによってパブリッシュされた各メッセージのコピーは、各インタレスト・アプリケーションに送信されます。 インタレスト・アプリケーションは多数ある場合も、1 つある場合も、まったくない場合もあります。 パブリッシュ/サブスクライブでは、インタレスト・アプリケーションはサブスクライバーと呼ばれ、メッセージはサブスクリプションによって識別されるキューに入れられます。

パブリッシュ/サブスクライブ・メッセージングを使用すると、情報の提供者とその情報の利用者 を分離することができます。 送信側アプリケーションと受信側のアプリケーションは、情報を送受信するために、お互いについて何も知る必要がありません。 詳しくは、63 ページの『パブリッシュ/サブスクライブ・メッセージング』を参照してください。

アプリケーションの設計担当者および開発者にとってのメッセージ･キューイングの利点

IBM MQ により、アプリケーション・プログラムは、メッセージ・キューイング を使用してメッセージ・ドリブン処理に加わることができます。 適切なメッセージ・キューイング・ソフトウェア製品を使用すれば、プラットフォームが異なっていても、アプリケーション・プログラム相互間で通信することができます。 例えば、z/OS® アプリケーションは IBM MQ for z/OS を介して通信できます。 アプリケーションは、基礎となる通信機構からは保護されています。 その他に、メッセージ・キューイングには次のような利点があります。

- 多くのアプリケーション間で共用できる小規模のプログラム群を使用して、アプリケーションを設計できる。

- これらの構築ブロックを再使用することによって、新しいアプリケーションを速やかに構築できる。
- メッセージ・キューイング技法を用いるため作成されたアプリケーションは、キュー・マネージャーの作業の仕方が変更されても影響を受けない。
- 通信プロトコルを使用する必要がない。 キュー・マネージャーが通信のすべての局面を処理します。
- メッセージを受信するプログラムは、メッセージが送信されてくる時に実行中である必要はない。 これらのメッセージはキューに保存されます。

設計担当者はアプリケーションのコストを下げることができます。なぜなら、開発の速度が速くなり、開発担当者の必要人数が少なくてすみ、しかもメッセージ・キューイングを使わないアプリケーションの場合よりもプログラミング・スキルの要求レベルが低いからです。

IBM MQ は、 メッセージ・キュー・インターフェース (または MQI) と呼ばれる共通アプリケーション・プログラミング・インターフェースを、アプリケーションがどこで実行されていても実装します。 このため、アプリケーション・プログラムを、あるプラットフォームから 別のプラットフォームに移植するのが容易になります。

MQI の詳細については、Message Queue Interface の概要を参照してください。

メッセージ・キューイングの主な機能とメリット

ここでは、メッセージ・キューイングの主な機能とメリットを中心に説明します。 メッセージ・キューイングのセキュリティーやデータ保全性などについて説明します。

メッセージ・キューイング方法を用いるアプリケーションの主な機能は、次のとおりです。

- 7 ページの『プログラム間に直接の接続がない』
- 8 ページの『時間に依存しない通信』
- 8 ページの『小規模のプログラム』
- 8 ページの『メッセージ・ドリブン型の処理』
- 9 ページの『イベント主導型の処理』
- 9 ページの『メッセージ優先順位』
- 9 ページの『セキュリティー』
- 9 ページの『データ整合性』
- 9 ページの『リカバリー・サポート』

注 : IBM MQ クライアントおよびサーバーについて検討する際に、新しいプラットフォームの追加の IBM MQ MQI clients をサポートするためにサーバー・アプリケーションを変更する必要はありません。 同様に、IBM MQ MQI client は、変更しなくても、追加タイプのサーバーで機能することができます。

プログラム間に直接の接続がない

メッセージ・キューイングは、間接的なプログラム間通信の技法です。 これは、プログラムが相互に通信し合うアプリケーション内で用いることができます。 1 つのプログラムがメッセージを (キュー・マネージャーが所有する) キューに書き込んで、次に別のプログラムがそのメッセージをキューから読み取ることによって通信が行われます。

プログラムは、他のプログラムによってキューに書き込まれたメッセージを読み取ることができます。 他のプログラムは、受信プログラムとして同じキュー・マネージャーに接続することも、または別のキュー・マネージャーに接続することもできます。 この別のキュー・マネージャーは別のシステム、異なったコンピューター・システム、またはたとえ異なった企業にあっても構いません。

メッセージ・キューを用いて通信するプログラム間には、物理的な接続はありません。 一方のプログラムが、キュー・マネージャーが所有するキューにメッセージを送信し、もう一方のプログラムがキューからメッセージを取り出します (8 ページの図 1 を参照)。

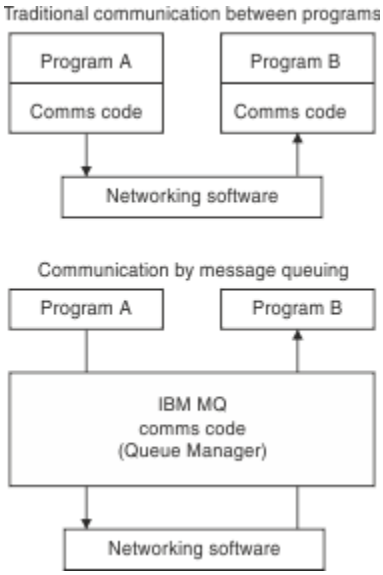


図 *1.* 従来の通信方式と比較したメッセージ・キューイング

電子メールの場合と同様に、トランザクションの一部である個々のメッセージは、蓄積交換方式に基づいてネットワーク内を移動します。基盤。 ノード間のリンクに障害が起こった場合は、そのリンクが復旧するか、オペレーターまたはプログラムがそのメッセージの宛先を変更するまで、メッセージは保留されます。

メッセージがキューからキューへ移動する機構はプログラムには表れません。 そのため、プログラムはより単純化されます。

時間に依存しない通信

他のプログラムに作業を要求したプログラムは、要求に対する応答を待つ必要はありません。 これらのプログラムは他の作業を行うことができ、応答の処理をそれが到着したとき、または後で行うことができます。 メッセージング・アプリケーションを作成するときは、プログラムがいつメッセージを送信するか、または相手がいつメッセージを受信できるかを知る (あるいは関与する) 必要はありません。 メッセージが失われることはありません。メッセージは、相手がそれを処理できるようになるまで、キュー・マネージャーによって保存されます。 メッセージは、プログラムが削除するまでキューにとどまっています。 これは、送信と受信のアプリケーション・プログラムは分離されていることを意味します。送信側は、メッセージの受信に関する受信側の応答を待たずに処理を継続できます。 ターゲット・アプリケーションが、メッセージの送信時に、実行中である必要もありません。 ターゲット・アプリケーションは、始動後にメッセージを取り出すことができます。

小規模のプログラム

メッセージ・キューイングには、小規模の自己完結型のプログラムを使用できるという利点があります。単一の大規模なプログラムに作業のすべての部分を順次に実行させる代わりに、その作業をいくつかの小規模で独立したプログラムに分散させることができます。 要求側プログラムは、個別の各プログラムにメッセージを送信して、それぞれの機能を実行するよう要求します。各プログラムは、機能を完了すると、1つまたは複数のメッセージとしてその結果を戻します。

メッセージ・ドリブン型の処理

トリガー操作 と呼ばれるメカニズムを使用して、メッセージがキューに到着したときに自動的にアプリケーションを開始できます。 必要に応じて、メッセージの処理が完了したときに、アプリケーションを停止させることもできます。

イベント主導型の処理

プログラムを、キューの状態に応じて制御できます。 例えば、メッセージがキューに到着した直後にプログラムを開始するようにしたり、またはある優先順位より上あるいはあらゆる優先順位のメッセージが、例えば 10 個キューに入るまではプログラムを開始しないようにすることもできます。

メッセージ優先順位

プログラムは、メッセージをキューに書き込むときに優先順位を割り当てることができます。 これにより、新しいメッセージが追加されるキュー内の位置が決まります。

プログラムがキューからメッセージを取得する方法は、キューに入っているメッセージの順番に取得するか、特定のメッセージを取得するかのいずれかです。 (プログラムが以前に送信した要求に対する応答を探している場合は、特定のメッセージを取得する必要が生じることがあります。)

セキュリティー

キュー・マネージャーを使用する際のアプリケーションの認証、キュー・マネージャー上のキューなどのリソースを使用する際の許可検査、ネットワークを介して転送されるメッセージ・データおよびキュー上に存在するメッセージ・データの暗号化、といったセキュリティー機能が用意されています。 セキュリティーの詳細については、セキュリティーの概要を参照してください。

データ整合性

データ保全性は、作業単位によって提供されます。 各 MQGET や MQPUT では、作業単位の開始と終了の同期は、オプションとして完全にサポートされており、作業単位の結果をコミットしたり、ロールバックしたりすることができます。 同期点のサポートは、そのアプリケーションに選択された同期点調整の形式によって、IBM MQ の外側または内側のどちらかで動作します。

リカバリー・サポート

リカバリーを可能にするため、IBM MQ の持続的な更新はすべてログに記録されます。 リカバリーが必要な場合、すべてのパーシスタント・メッセージがリストアされ、すべての未完了トランザクションがロールバックされ、すべての同期点コミットおよびバックアウトは制御中の同期点マネージャーが通常ハンドルする方法で処理されます。 持続メッセージの詳細については、メッセージの持続性を参照してください。

メッセージ・キューイングの用語

ここでは、メッセージ・キューイングで使われている一部の用語について説明します。

次の方法があります。

- チャネル
- クラスター
- IBM MQ MQI client
- z/OS グループ内キューイング
- メッセージ
- メッセージ・チャネル・エージェント
- メッセージ記述子
- Point-to-Point
- publish/subscribe
- キュー
- キュー・マネージャー
- z/OS キュー共用グループ
- z/OS 共用キュー

- サブスクリプション
- トピック

チャネル

あるキュー・マネージャーから別のキュー・マネージャーにメッセージを移すためにチャネルが使用され、それによってアプリケーションは基礎をなす通信プロトコルから遮蔽されます。 キュー・マネージャーは、同一のシステム上に存在する場合もあれば、同一のプラットフォーム上の異なるシステムや、異なるプラットフォーム上に存在する場合もあります。 メッセージの送信元は次のように多岐にわたります。

- あるノードから別のノードにデータを転送するユーザー作成のアプリケーション・プログラム。
- PCF コマンドまたは MQAI を使用するユーザー作成の管理アプリケーション。
- IBM MQ Explorer。
- インスツルメンテーション・イベント・メッセージを別のキュー・マネージャーに送信するキュー・マネージャー。
- リモート管理コマンドを別のキュー・マネージャーに送信するキュー・マネージャー。 例えば、MQSC コマンドや administrative REST API を使用します。

チャネルについて詳しくは、32 ページの『チャネル定義』を参照してください。

クラスター

クラスター とは、何らかの方法で論理的に関連付けられているキュー・マネージャーのネットワークです。

クラスター化せずに分散キューイングを使用する IBM MQ のネットワークでは、キュー・マネージャーはすべて独立しています。 あるキュー・マネージャーから別のキュー・マネージャーにメッセージを送信する必要がある場合、送信元のキュー・マネージャーには、伝送キューとリモート・キュー・マネージャーへのチャネルが定義されていなければなりません。

クラスターの使用には、システム管理の単純化、および可用性とワークロードの平衡化の向上という 2 つの異なる目的があります。

設定したクラスターが最小のものであっても、システム管理の単純化は実現されます。 1 つのクラスターを構成しているキュー・マネージャーは多くの定義を必要としないので、間違った定義を作成する状況は少なくなります。

クラスタリングの詳細については、クラスターを参照してください。

IBM MQ MQI client

IBM MQ MQI クライアントは、独立してインストール可能な IBM MQ のコンポーネントです。 MQI クライアントを使用することにより、通信プロトコルを使って IBM MQ アプリケーションを実行し、他のプラットフォーム上の 1 つ以上の Message Queue Interface (MQI) サーバーと対話して、それらのキュー・マネージャーに接続することができます。

IBM MQ MQI client コンポーネントをインストールして使用する方法の詳細については、以下のトピックを参照してください。

- AIX AIX®での IBM MQ クライアントのインストール
- Linux Linux®での IBM MQ クライアントのインストール
- Windows Windows での IBM MQ クライアントのインストール
- IBM i IBM i での IBM MQ クライアントのインストール

サーバーとクライアント間の接続の構成

グループ内キューイング

z/OS

キュー共用グループ中のキュー・マネージャーは、通常のチャネルを使って通信できます。または、グループ内キューイング (IGQ) と呼ばれる、チャネルを定義しなくても高速でメッセージ転送を実行できる技法を使うことができます。 これは、IBM MQ for z/OS のみに適用されます。

グループ内キューイングについて詳しくは、218 ページの『Intra-group queuing』を参照してください。

メッセージ

メッセージ・キューイングでのメッセージとは、1 つのプログラムから別のプログラムに向けて送られるデータの集合です。 IBM MQ メッセージ を参照してください。

メッセージ・タイプの詳細については、メッセージのタイプを参照してください。

MSG チャネル・エージェント

メッセージ・チャネル・エージェントは、チャネルの一端です。 1 組のメッセージ・チャネル・エージェント (送信エージェントと受信エージェント) によって 1 つのチャネルが構成され、キュー・マネージャーから別のキュー・マネージャーにメッセージが移動されます。

メッセージ・チャネル・エージェントの使用方法については、分散キュー管理の概要を参照してください。

メッセージ記述子

IBM MQ メッセージは制御情報とアプリケーション・データから構成されています。

制御情報は、メッセージ記述子構造体 (MQMD) で定義され、次のような項目で構成されています。

- メッセージのタイプ
- メッセージの ID
- メッセージ送達の優先順位

アプリケーション・データの構造と内容は、IBM MQ ではなく、関連するプログラムによって決定されます。

詳しくは、 MQMD を参照してください。

Point-to-Point メッセージング

Point-to-Point メッセージングでは、各メッセージは、これを生成する 1 つのアプリケーションから、これをコンシュームする 1 つのアプリケーションへ移動します。 メッセージは、生成側のアプリケーションがメッセージをキューに入れることで転送され、コンシューム側のアプリケーションはそのキューからメッセージを取得します。

パブリッシュ/サブスクライブ・メッセージング

パブリッシュ/サブスクライブ・メッセージングでは、パブリッシング・アプリケーションによってパブリッシュされた各メッセージのコピーは、各インタレスト・アプリケーションに送信されます。 インタレスト・アプリケーションは多数ある場合も、1 つある場合も、まったくない場合もあります。 パブリッシュ/サブスクライブでは、インタレスト・アプリケーションはサブスクライバーと呼ばれ、メッセージはサブスクリプションによって識別されるキューに入れられます。

詳細については、63 ページの『パブリッシュ/サブスクライブ・メッセージング』を参照してください。

キュー

メッセージの送信先になる指定された宛先です。 メッセージは、それらのキューを取り扱うプログラムによって取り出されるまで、キューに蓄積されます。

詳細については、19 ページの『キュー』を参照してください。

キュー・マネージャー

キュー・マネージャー は、アプリケーションに対してキューイング・サービスを提供するシステム・プログラムです。

これは、アプリケーション・プログラミング・インターフェースを提供し、それによってプログラムはメッセージのキューへの書き込みおよびキューからのメッセージの読み取りができます。 キュー・マネージャーは管理者が新しいキューを生成したり、既存のキューの属性を変更したり、キュー・マネージャーの操作を制御したりできる付加的な機能も提供します。

IBM MQ メッセージ・キューイング・サービスをシステムで利用する場合、キュー・マネージャーを実行する必要があります。 1 つのシステムに複数のキュー・マネージャーを稼働させることができます (例えば、テスト・システムと実働 システムを区別するため)。 各キュー・マネージャーは、アプリケーションに対して、接続ハンドル (*Hconn*) で識別される。

種々のアプリケーションがキュー・マネージャーのサービスを同時に使用でき、しかもこれらのアプリケーションは互いにまったく無関係でも構いません。 プログラムがキュー・マネージャーのサービスを利用するためには、そのキュー・マネージャーとの接続を確立する必要があります。

アプリケーションが他のキュー・マネージャーに接続しているアプリケーションにメッセージを送信するためには、そのキュー・マネージャーはこれらのキュー・マネージャー間で通信できなければなりません。 IBM MQ は、蓄積交換 プロトコルを実装することにより、このようなアプリケーションの間で安全にメッセージが送達されるようにしています。

詳細については、28 ページの『キュー・マネージャー』を参照してください。

キュー共用グループ

共用キューの同じセットにアクセスできるキュー・マネージャーは、キュー共用グループ (QSG) というグループを形成します。 これらは、共用キューを格納するカップリング・ファシリティー (CF) を使って互いに通信を行います。 これは、IBM MQ for z/OS のみに適用されます。

詳細については、174 ページの『Shared queues and queue sharing groups』を参照してください。

共用キュー

共用キューとはローカル・キューのタイプで、シスプレックス内の 1 つ以上のキュー・マネージャーによってメッセージにアクセスすることができます。 これは、同一のキュー・マネージャーを使用して複数のアプリケーションによって共用されるキューとは異なります。 これは、IBM MQ for z/OS のみに適用されます。

サブスクリプション

パブリッシュ/サブスクライブ・アプリケーションは、特定のトピックに関して、インタレストをメッセージに登録することができます。 これを行ったアプリケーションはサブスクライバーと呼ばれ、マッチング・メッセージが処理のためにキューに入れられる方法を、用語サブスクリプションが定義します。

サブスクリプションには、サブスクライバーの ID、パブリケーションを配置する宛先キューの ID についての情報が含まれます。 また、パブリケーションを宛先キューに配置する方法についての情報も含まれます。

詳細については、66 ページの『サブスクライバーとサブスクリプション』を参照してください。

トピック

トピックは、パブリッシュ/サブスクライブ・メッセージでパブリッシュされる情報の主題を示す文字ストリングです。

トピックは、パブリッシュ/サブスクライブ・システムでメッセージを正常に送達するうえで、重要な役割を果たします。 パブリッシャーは、各メッセージに特定の宛先アドレスを含める代わりに、各メッセージにトピックを割り当てます。 キュー・マネージャーは、トピックと、そのトピックをサブスクライブして

いるサブスクライバーのリストとを突き合わせ、それらのサブスクライバーの各々にメッセージを送達します。

詳細については、69 ページの『トピック』を参照してください。

メッセージとキュー

メッセージとキューは、メッセージ・キューイング・システムの基本構成要素です。

メッセージとは

メッセージ を使用するアプリケーションにとって、メッセージは意味のあるバイト・ストリングです。 メッセージは、ある 1 つのアプリケーションから別のアプリケーションに (または、同じアプリケーションの異なる部分の間で) 情報を転送するために使用されます。 伝達に関係するアプリケーションは、 同じプラットフォーム上で実行されていても、別のプラットフォーム上で実行されていてもかまいません。

IBM MQ メッセージの構成要素を次に示します。

- アプリケーション・データ。アプリケーション・データの内容と構造は、そのデータを使用するアプリケーション・プログラムによって定義されます。
- メッセージ記述子。メッセージ・ディスクリプターは、メッセージを識別し、 メッセージのタイプおよび送信側アプリケーションによってメッセージに割り当てられた優先順位などの 追加の制御情報を含んでいます。

  メッセージ記述子の形式は、IBM MQ によって定義されます。 メッセージ記述子の詳細な説明については、MQMD - メッセージ記述子を参照してください。
- メッセージ・プロパティー。 メッセージに関するメタデータです。 メッセージ・プロパティーの内容は、それらを使用するアプリケーション・プログラムが定義します。 詳細については、メッセージ・プロパティーを参照してください。

メッセージ長

デフォルトの最大メッセージ長は 4 MB ですが、この最大長を 100 MB まで増やすことができます (1 MB は 1 048 576 バイトです)。 ただし、実際には、次のものによってメッセージ長は制限されます。

- 受信側のキュー用に定義した最大メッセージ長
- キュー・マネージャー用に定義した最大メッセージ長
- キューによって定義された最大メッセージ長
- 送信側のアプリケーションまたは受信側のアプリケーションで定義した 最大メッセージ長
- メッセージ用として使用可能なストレージの量

1 つのアプリケーションが必要とするすべての情報を送るには、複数のメッセージが必要な場合があります。

アプリケーションによるキューの送信/受信方法

アプリケーションでは、**MQI 呼び出し**を使用してメッセージを送受信します。

例えば、キューにメッセージを書き込むために、アプリケーションでは以下の処理が行われます。

1. MQOPEN 呼び出しを発行して必要なキューをオープンする。
2. MQPUT 呼び出しを発行してキューにメッセージを書き込む。

別のアプリケーションがそのキューからメッセージを取り出すときには、MQI MQGET 呼び出しを発行します。

MQI 呼び出しについての詳細は、MQI 呼び出しを参照してください。

キューとは

キュー とは、メッセージを保管するためのデータ構造体です。

キューはすべて、キュー・マネージャー に属しています。 キュー・マネージャーは、所有するキューを管理し、受信したすべてのメッセージを適切なキューに格納します。 メッセージは、アプリケーション・プログラムまたはキュー・マネージャーによって、通常の操作の一部としてキューに書き込まれます。

事前定義キューと動的キュー

キューは、その作成方法によって次のような特徴があります。

- **事前定義キュー**は、管理者が該当する MQSC または PCF コマンドを使用して作成します。事前定義キューは、永続キューであって、それらを使用するアプリケーションとは無関係に存在し、IBM MQ が再始動しても存続します。
- **動的キュー**が作成されるのは、アプリケーションが モデル・キュー の名前を指定して MQOPEN 要求を出した場合です。 作成されたキューは、モデル・キューというテンプレート・キュー定義 に基づきます。 MQSC コマンド DEFINE QMODEL を使用してモデル・キューを作成することができます。 モデル・キューの属性 (例えば、キューに保管できるメッセージの最大数) は、そのモデル・キューから作成される動的キューが継承します。

  モデル・キューは、作成される動的キューが永続キューになるか一時キュー になるかを指定する属性を持っています。 永続キューは、アプリケーションやキュー・マネージャーが 再始動しても存続しますが、一時キューは、再始動すると失われます。

キューからのメッセージの取り出し

MQSeries では、アプリケーションに適切な許可が与えられている場合、次の取り出しアルゴリズ ムに従ってキューからメッセージを取り出すことができます。

- 先入れ先出し法 (first-in-first-out (FIFO))。
- メッセージ記述子に定義されたメッセージ優先順位。 同じ優先順位を持つメッセージは、FIFO 順に取り出されます。
- 特定のメッセージについてのプログラム要求。

アプリケーションからの MQGET 要求によって、使用される方式が決まります。

IBM MQ オブジェクト

キュー・マネージャーは、IBM MQ オブジェクトのプロパティーを定義します。 それらのプロパティーの値は、IBM MQ がこれらのオブジェクトを処理する方法に影響します。 IBM MQ のコマンドとインターフェースを使用して、オブジェクトを作成したり管理したりできます。 アプリケーションから、Message Queue Interface (MQI) を使用してオブジェクトを制御します。 プログラムからアドレッシングされるとき、オブジェクトは IBM MQ オブジェクト記述子 (MQOD) によって識別されます。

オブジェクト管理

オブジェクトの管理には、以下のタスクが含まれます。

- キュー・マネージャーの始動および停止
- アプリケーション用のオブジェクト (特にキュー) の作成
- オブジェクトの属性の表示または変更
- オブジェクトの削除
- チャネルを使用して、他の (リモート) システムにあるキュー・マネージャーへの通信パスを作成。
- キュー・マネージャーのクラスター を作成することによって、管理プロセス全体を簡易化し、ワークロードのバランスをとる。

動的キューの場合を除き、オブジェクト (動的キューを除く) は、処理の前にキュー・マネージャーに定義されていなければなりません。

IBM MQ コマンドを使用してオブジェクト管理操作を行う場合、キュー・マネージャーは、ユーザーがそれらの操作を実行するのに必要なレベルの権限を持っているかどうかを検査します。 同様に、アプリケーションが MQOPEN 呼び出しを使用してオブジェクトをオープンするとき、キュー・マネージャーは、そのオブジェクトへのアクセスを許可する前に、アプリケーションが必要なレベルの権限を持っているかどうかを検査します。 検査は、オープンされているオブジェクトの名前に対して行われます。

以下のメソッドを使用してオブジェクトを定義したり管理したりできます。

- プログラマブル・コマンド・フォーマット・リファレンスおよび管理タスクの自動化で説明されている PCF コマンド
- MQSC コマンドで説明されている MQSC コマンド
- z/OS IBM MQ for z/OS 操作および制御パネル (IBM MQ for z/OS の管理 を参照)
- Windows Linux IBM MQ Explorer (Windows および Linux システム専用の Intel)。 詳しくは、MQ エクスプローラーの概要を参照してください。

以下のメソッドを使用してオブジェクトを管理することもできます。

- キーボードから入力する制御コマンド。 制御コマンドを使用した IBM MQ for Multiplatforms の管理を参照してください。
- プログラムからの IBM MQ 管理インターフェース (MQAI) の呼び出し。 IBM MQ 管理インターフェース (MQAI) を参照してください。

ALW AIX, Linux, and Windows 上の IBM MQ コマンドのシーケンスについては、MQSC 機能を使用して、ファイルに保持されている一連のコマンドを実行できます。 詳しくは、 MQSC コマンドを使用した IBM MQ の管理を参照してください。

IBM i 定期的に使用する IBM MQ for IBM i コマンドのシーケンスについては、制御言語プログラムを作成することができます。 詳しくは、CL コマンドを使用した IBM MQ for IBM i の管理を参照してください。

z/OS 通常使用する一連の IBM MQ for z/OS コマンドの場合、コマンドの入ったメッセージを作成して、システム・コマンドの入力キューにこれらのメッセージを書き込む管理プログラムを作成することもできます。 キュー・マネージャーは、このキュー上のメッセージを、コマンド・ラインまたは操作および制御パネルから入力されたコマンドを処理するのと同じ方法で処理します。 この手法は、IBM MQ 管理のためのプログラムの作成に説明されており、IBM MQ for z/OS と共に提供されている Mail Manager サンプル・アプリケーションで示されています。 このサンプルについては、IBM MQ for z/OS 用のサンプル・プログラムに説明があります。

オブジェクトの属性

オブジェクトのプロパティーは、その属性によって定義されます。 属性の一部はユーザーが指定できますが、他の属性は表示のみ可能です。

例えば、キューが収容できる最大メッセージ長は、**MaxMsgLength** 属性によって定義されます。この属性は、キューの作成時に指定できます。 **DefinitionType** 属性は、キューが作成された方法を指定します。この属性は表示のみできます。

IBM MQ では、属性を参照する方法としては、次の 2 とおりの方法があります。

- 属性の PCF 名 (例えば、**MaxMsgLength**) を使用 する方法
- 属性の MQSC コマンド名 (例えば、MAXMSGL) を使用する方法

キュー共有グループ

z/OS

共用キューの同じセットにアクセスできるキュー・マネージャーは、キュー共用グループ (QSG) と呼ばれるグループを形成し、共用キューを保管するカップリング・ファシリティー (CF) を使用して互いに通信します。 QSG は厳密にはオブジェクトではないことに注意してください。

共用キューとは、キュー共用グループ内にある 1 つ以上のキュー・マネージャーがアクセスできるメッセージを持つ、一種のローカル・キューです。 これは、同一のキュー・マネージャーを使用して複数のアプリケーションによって共用されるキューとは異なります。

キュー共用グループには、最大 4 文字の名前があります。 この名前はネットワーク内で固有であり、かつ、キュー・マネージャー名とは異なるものである必要があります。

重要 : 共用キューおよびキュー共用グループは、IBM MQ for z/OS でのみサポートされます。

詳しくは、 174 ページの『Shared queues and queue sharing groups』 を参照してください。

システム・デフォルト・オブジェクト

システム・デフォルト・オブジェクト とは、キュー・マネージャーの作成時に各キュー・マネージャーごとに自動的に作成される 1 組のオブジェクト定義のことです。 ご使用のシステムのアプリケーションで使用するために、これらのオブジェクト定義はすべてコピーしたり、修正することができます。

デフォルト・オブジェクト名には、語幹である SYSTEM が付いています。例えば、デフォルト・ローカル・キューは SYSTEM.DEFAULT.LOCAL.QUEUE であり、デフォルト受信チャネルは SYSTEM.DEF.RECEIVER です。 これらのオブジェクトの名前を変更することはできません。これらの名前を 持つデフォルト・オブジェクトは必須です。

オブジェクトを定義する際に、明示的に指定されなかった属性は該当するデフォルト・オブジェクトからコピーされます。 例えば、ローカル・キューを定義する場合、指定しなかった属性は、デフォルト・キュー SYSTEM.DEFAULT.LOCAL.QUEUE から取られます。

詳しくは、 システムおよびデフォルト・オブジェクト を参照してください。

オブジェクト・タイプ

管理タスクの多くには、さまざまな種類の IBM MQ オブジェクト の操作が関係します。

IBM MQ オブジェクトの命名については、37 ページの『IBM MQ オブジェクトの命名』を参照してください。

キュー・マネージャー上に作成されるデフォルトのオブジェクトについては、16 ページの『システム・デフォルト・オブジェクト』を参照してください。

さまざまなタイプの IBM MQ オブジェクトについては、以下を参照してください。

認証情報オブジェクト

認証情報オブジェクトは、証明書取り消し検査を実行するために必要な定義を提供します。

キュー・マネージャー認証情報オブジェクトは、Transport Layer Security (TLS) の IBM MQ サポートの一部を構成しています。 このオブジェクトには、取り消された証明書の検査に必要な定義があります。 認証局は、信頼できなくなった証明書を取り消します。

認証情報オブジェクトを定義するには、MQSC コマンド **DEFINE AUTHINFO** を使用できます。 認証情報オブジェクトの属性について詳しくは、**DEFINE AUTHINFO** を参照してください。

認証情報オブジェクトでは、以下の IBM MQ 制御コマンドを使用できます。

- **setmqaut** (権限の付与または取り消し)
- **dspmqaut** (オブジェクト権限の表示)
- **dmpmqaut** (権限のダンプ)
- **rcrmqobj** (オブジェクトの再作成)
- **rcdmqimg** (メディア・イメージの記録)
- **dspmqfls** (ファイル名の表示)

TLS の概要、および認証情報オブジェクトの使用については、 Transport Layer Security (TLS) の概念 および IBM MQ での TLS セキュリティー・プロトコルを参照してください。

チャネル

チャネルとは、あるキュー・マネージャーと別のキュー・マネージャーを結ぶ通信パスを提供するオブジェクトのことです。

詳しくは、 30 ページの『チャネル』 を参照してください。

通信情報オブジェクト

IBM MQ Multicast は、待ち時間が短く、ファンアウトが大きい、高信頼性マルチキャスト・メッセージングです。 マルチキャスト送信を使用するには、通信情報 (COMMINFO) オブジェクトが必要です。

詳しくは、 110 ページの『IBM MQ マルチキャスト』 を参照してください。

COMMINFO オブジェクトは、マルチキャスト伝送に関連付けられた属性が含まれる IBM MQ オブジェクトです。 これらの属性の詳細については、DEFINE COMMINFO を参照してください。 COMMINFO オブジェクトの作成について詳しくは、マルチキャストの概要を参照してください。

リスナー

リスナー は、他のキュー・マネージャーまたはクライアント・アプリケーションからネットワーク要求を受け取るプロセスで、関連付けられたチャネルを始動します。

リスナー・プロセス は、 **runmqlsr** 制御コマンドを使用して開始できます。

リスナー・オブジェクト は、キュー・マネージャーの有効範囲内からリスナー・プロセスの開始と停止を管理することができるようにする、IBM MQ オブジェクトです。 リスナー・オブジェクトの属性を定義することにより、次の操作を実行できます。

- リスナー・プロセスを構成する。
- キュー・マネージャーが開始および停止したときに、リスナー・プロセスも自動的に開始および停止させるかどうかを指定する。

重要 : z/OS リスナー・オブジェクトは、IBM MQ for z/OS ではサポートされていません。 チャネル・イニシエーターを使用して、IBM MQ for z/OS が listen 機能を実装する方法について詳しくは、170 ページの『The channel initiator on z/OS』 を参照してください。

名前リスト

名前リスト は、クラスター名、キュー名、または認証情報オブジェクト名のリストが含まれた IBM MQ オブジェクトです。 クラスターでは、このリストを使用して、キュー・マネージャーがリポジトリーを保持しているクラスターのリストを識別することができます。

名前リストは、IBM MQ オブジェクトの 1 つで、他の IBM MQ オブジェクトのリストが格納されています。 通常、名前リストはトリガー・モニターなどのアプリケーションによって、キュー・グループの識別に使用されます。 名前リストを使用した場合の利点は、アプリケーションとは別個に管理できるということです。更新時に、その名前リストを使用しているアプリケーションを停止する必要はありません。 また、あるアプリケーションで障害が起こった場合でも、名前リストには影響はなく、他のアプリケーションは引き続きその名前リストを使用できます。

名前リストは、複数の IBM MQ オブジェクトによって参照されるクラスターのリストを維持するために、キュー・マネージャー・クラスターでも使用されます。

MQSC コマンド DEFINE NAMELIST および ALTER NAMELIST を使用して、名前リストを定義および変更することができます。

注 : z/OS あるいは、 z/OS では、 IBM MQ for z/OS 操作および制御パネルを使用することもできます。

プログラムは MQI を使用することにより、これらの名前リストにどのキューが登録されているかを検知できます。 名前リストの編成は、アプリケーション設計担当者およびシステム管理者が行います。

使用可能な名前リスト属性のリストについては、 名前リストの属性を参照してください。

プロセス定義

プロセス定義オブジェクトを使用すると、キュー・マネージャーが使用するアプリケーションの属性を定義することによって、オペレーターによる介入がなくてもアプリケーションを開始することができます。

プロセス定義オブジェクトは、IBM MQ キュー・マネージャーでのトリガー・イベントに応答して開始されるアプリケーションを定義します。 プロセス定義の属性には、アプリケーション ID、アプリケーション・タイプ、およびアプリケーション特有のデータがあります。 詳しくは、27 ページの『IBM MQ によって特定の目的で使用されるキュー』の『開始キュー』を参照してください。

アプリケーションをオペレーターの介入なしで開始させる (これについては、トリガーによる IBM MQ アプリケーションの開始を参照) には、アプリケーションの属性をキュー・マネージャーに通知する必要があります。 これらの属性はプロセス定義オブジェクト で定義されます。

ProcessName 属性はオブジェクトの作成時に固定されます。 ただし、他の属性は、IBM MQ コマンドを使用して変更できます。

注 : z/OS あるいは、 z/OS では、 IBM MQ for z/OS 操作パネルおよび制御パネルを使用することもできます。

すべての属性の値について問い合わせるには、MQINQ - オブジェクト属性の照会を使用します。

使用可能なプロセス定義属性のリストについては、 プロセス定義の属性を参照してください。

キュー

IBM MQ キュー は名前付きオブジェクトで、アプリケーションはそこにメッセージを書き込んだり、そこからメッセージを読み取ったりできます。

詳しくは、 19 ページの『キュー』 を参照してください。

キュー・マネージャー

IBM MQ キュー・マネージャーは、アプリケーションにキューイング・サービスを提供し、そのキューを管理します。

詳しくは、 28 ページの『キュー・マネージャー』 を参照してください。

サービス

サービス・オブジェクトは、キュー・マネージャーが開始または停止したときに実行するプログラムを定義するための方法です。

プログラムは次のいずれかのタイプになります。

サーバー
サーバー は、SERVTYPE パラメーターが SERVER に指定されているサービス・オブジェクトです。 サーバー・サービス・オブジェクトは、指定したキュー・マネージャーの開始時に実行されるプログラムの定義です。 サーバー・プロセスの 1 つのインスタンスだけを並行して実行できます。 実行中は、サーバー・プロセスの状況を MQSC コマンド DISPLAY SVSTATUS を使用してモニターできます。 通常、サーバー・サービス・オブジェクトは、送達不能ハンドラーまたはトリガー・モニターなどのプログラムの定義ですが、実行可能なプログラムは IBM MQ によって提供されるプログラムに限定されません。 また、サーバー・サービス・オブジェクトを定義して、指定されたキュー・マネージャーがシャットダウンしてプログラムが終了したときに実行されるコマンドを含めることができます。

コマンド
コマンド は、SERVTYPE パラメーターが COMMAND に指定されているサービス・オブジェクトです。 コマンド・サービス・オブジェクトは、指定されたキュー・マネージャーが開始または停止したときに実行されるプログラムの定義です。 コマンド・プロセスの複数のインスタンスを並行して実行できます。 コマンド・サービス・オブジェクトは、プログラムが実行されるとキュー・マネージャーがプログラムをモニターしなくなる点で、サーバー・サービス・オブジェクトと異なっています。 通常、コマンド・サービス・オブジェクトは存続期間の短いプログラムの定義で、1 つ以上の他のタスクを開始するなどの特定のタスクを実行します。

重要 : z/OS サービス・オブジェクトは、IBM MQ for z/OS ではサポートされていません。

詳しくは、「サービスの操作」を参照してください。

ストレージ・クラス

z/OS

記憶域クラスは、1 つ以上のキューをページ・セットにマップします。

つまり、そのキューのメッセージがそのページ・セットに (バッファリングを条件として) 保管されます。

記憶域クラスは、IBM MQ for z/OS でのみサポートされています。

ストレージ・クラスについて詳しくは、 z/OS での IBM MQ 環境の計画を参照してください。

トピック・オブジェクト

トピック・オブジェクト は、特定の非デフォルト属性をトピックに割り当てることができるようにする IBM MQ オブジェクトです。

トピック は、特定のトピック・ストリング にパブリッシュまたはサブスクライブするアプリケーションによって定義されます。 トピック・ストリングでは、スラッシュ記号 (/) でトピックを分離することにより、トピックの階層を指定できます。階層はトピック・ツリーで視覚化されます。 例えば、アプリケーションがトピック・ストリングである `/Sport/American Football` および `/Sport/Soccer` にパブリッシュする場合、作成されるトピック・ツリーは、`Sport` という親ノードと、`American Football` および `Soccer` という 2 つの子ノードで構成されます。

各トピックは、トピック・ツリー内にある最初の親管理ノードから属性を継承します。 特定のトピック・ツリーに管理トピック・ノードが存在しない場合、全トピックは基本トピック・オブジェクトである SYSTEM.BASE.TOPIC から属性を継承します。

トピック・ツリー内の任意のノードでトピック・オブジェクトを作成できます。これは、トピック・オブジェクトの TOPICSTR 属性でそのノードのトピック・ストリングを指定することにより可能となります。また、管理トピック・ノードの他の属性も定義することができます。 これらの属性について詳しくは、『MQSC コマンド』または『PCF コマンドによる管理の自動化』を参照してください。 デフォルトでは、各トピック・オブジェクトは直近の親管理トピック・ノードから属性を継承します。

さらに、トピック・オブジェクトを使用することにより、アプリケーション開発者からトピック・ツリー全体を隠すことも可能です。 トピック `/Sport/American Football` に対して FOOTBALL.US という名前のトピック・オブジェクトが作成された場合、アプリケーションは、同じ結果を持つストリング `/Sport/American Football` ではなく、 FOOTBALL.US という名前のオブジェクトにパブリッシュまたはサブスクライブすることができます。

トピック・オブジェクトのトピック・ストリング中に文字 #、+、/、または * を 入力する場合、その文字はストリング中では通常文字として扱われ、 トピック・オブジェクトが関連付けられているトピック・ストリングの一部であると 見なされます。

トピック・オブジェクトについて詳しくは、63 ページの『パブリッシュ/サブスクライブ・メッセージング』を参照してください。

関連概念

5 ページの『メッセージ・キューイングの概要』
IBM MQ 製品では、整合性のあるアプリケーション・プログラミング・インターフェースを使用して、性質の異なるコンポーネント (プロセッサー、オペレーティング・システム、サブシステム、および通信プロトコル) のネットワーク内でプログラムが相互に通信できるようにしています。

関連資料

MQSC コマンド

キュー

IBM MQ のキューとキュー属性の紹介。

メッセージはキューに入れられるので、メッセージを書き込んだアプリケーションは、そのメッセージに対する応答を期待する場合でも、応答を待っている間に他の作業を自由に行うことができます。 アプリケーションは Message Queue Interface (MQI) を用いてキューにアクセスします。MQI については、Message Queue Interface の概要で説明します。

メッセージをキューに書き込むには、その前にキューが作成されている必要があります。 キューはキュー・マネージャーによって所有され、そのキュー・マネージャーは多数のキューを所有できます。 ただし、各キューはそのキュー・マネージャー内で固有の名前を持っていなければなりません。

キューはキュー・マネージャーを通じて保守されます。 ほとんどの場合、各キューはそのキュー・マネージャーによって物理的に管理されますが、このことはアプリケーション・プログラムからは認識されません。 IBM MQ for z/OS 共用キューは、キュー共用グループ内の任意のキュー・マネージャーによって管理できます。

キューを作成するには、IBM MQ コマンド (MQSC)、PCF コマンド、またはプラットフォーム固有のインターフェースを使用できます。 例えば、IBM MQ for z/OS 操作および制御パネルはプラットフォーム固有のものです。

一時的なジョブ用のローカル・キューを独自のアプリケーションから動的に 作成できます。 例えば、応答先 キュー (アプリケーションが終了したら必要なくなる) を作成できます。 詳しくは、25 ページの『動的キューとモデル・キュー』 を参照してください。

キューを使用する前に、そのキューで何を行いたいかを指定して、キューをオープンする必要があります。例えば、以下の目的でキューをオープンすることができます。

- メッセージのブラウズのみ (取り出しは行わない)
- メッセージの取り出し (他のプログラムとの共用アクセスか、または排他的アクセスで)
- キューへのメッセージの書き込み
- キューの属性の照会
- キューの属性の設定

キューを開くときに指定できるオプションの完全なリストについては、MQOPEN - オブジェクトのオープンを参照してください。

キューの属性

キューの属性には、そのキューが定義される時に指定され、後からは変更できないものがあります (例えば、キューのタイプ)。 キューのその他の属性は、次のいずれかの方法で変更可能な属性にグループ分けできます。

- キューの処理中にキュー・マネージャーによって (例えば、キューの現在のサイズ)
- コマンドによってのみ (例えば、キューのテキスト記述)
- アプリケーションが MQSET 呼び出しを使用して (例えば、そのキューに書き込み操作ができるかどうか)

MQINQ 呼び出しを用いてすべての属性の値を知ることができます。

複数のキュー・タイプに共通な属性として次のものがあります。

QName
キューの名前。

QType
キューのタイプ。

QDesc
キューのテキスト記述。

InhibitGet
プログラムがキューからのメッセージの取得を許可されているかどうか。 ただし、リモート・キューからメッセージを取得することはできません。

InhibitPut
プログラムがそのキューにメッセージを書き込むことができるかどうか。

DefPriority
キューに書き込まれたメッセージのデフォルト優先順位。

DefPersistence
キューに書き込まれたメッセージのデフォルト持続性

範囲
このキューに対するエントリーが名前サービスでも存在するか制御します。

z/OS **Scope** 属性は、z/OS ではサポートされません。

これらの属性の詳細な説明については、キューの属性を参照してください。

キューの定義方法

IBM MQ にキューを定義するには、MQSC の DEFINE コマンドまたは PCF の Create Queue コマンドを使用します。 これらのコマンドは、キューのタイプおよびキューの属性を指定します。 例えば、ローカル・キュー・オブジェクトは、アプリケーションがその キューを MQI 呼び出しで参照したときに何が発生するかを指定する属性を 持っています。 属性には、次のものがあります。

- アプリケーションがメッセージをキューから取り出せるかどうか (読み取り (GET) 可能)
- アプリケーションがメッセージをキューに書き込めるかどうか (書き込み (PUT) 可能)
- キューへのアクセスが、1 つのアプリケーション専用になるか、または複数のアプリケーションで共用されるか
- 同時にキューに保管できるメッセージの最大数 (キューの最大サイズ)
- キューに書き込むことのできる最大メッセージ長

また、キューの定義に使用できるプラットフォーム固有の各種インターフェースもあります。

関連概念
59 ページの『クラスター・キュー』
クラスター・キューとは、クラスター・キュー・マネージャーでホストされ、同じクラスター内の別のキュー・マネージャーで使用できるキューです。

50 ページの『送達不能キュー』
送達不能キュー (または未配布メッセージ・キュー) は、正しい宛先にメッセージを送信できない場合にそのメッセージが送信されるキューです。 一般的に、送達不能キューはキュー・マネージャーごとに 1 つ存在します。

PCF コマンドによる管理の自動化
IBM MQ Console: キューの処理

関連タスク
MQSC コマンドを使用した IBM MQ の管理
MQ Explorer を使用したキュー・マネージャーおよびオブジェクトの作成と構成
IBM i CL コマンドを使用した IBM MQ for IBM i の管理
z/OS IBM MQ for z/OS で MQSC コマンドおよび PCF コマンドを発行できるソース

関連資料
59 ページの『Comparison between shared queues and cluster queues』
This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

関連情報
174 ページの『What is a shared queue?』

ローカル・キュー

ローカル・キューのタイプは、伝送、開始、送達不能、コマンド、デフォルト、チャネル、およびイベント・キューです。

キューは、プログラムが接続されているキュー・マネージャーに所有される場合、プログラムではローカル と認知されます。 メッセージは、ローカル・キューから取得し、ローカル・キューに入れることができます。

キュー定義オブジェクトは、キューに入る物理メッセージと同様、そのキューの定義情報を保持します。

各キュー・マネージャーは、ある特別な目的のために用いられる次のようなローカル・キューを持つことができます。

伝送キュー

アプリケーションがメッセージをリモート・キューに送信するとき、ローカル・キュー・マネージャーはそのメッセージを、伝送キュー と呼ばれる特別のローカル・キューに保管します。 アプリケーションは、メッセージを伝送キューに直接書き込むことも、リモート・キュー定義を介して間接的に書き込むこともできます。

キュー・マネージャーは、メッセージをリモート・キュー・マネージャーに送信するときに、次の順序で伝送キューを決定します。

1. リモート・キューのローカル定義の XMITQ 属性に名前が指定されている伝送キュー。
2. リモート・キュー・マネージャーと同じ名前の伝送キュー。 この値は、リモート・キューのローカル定義の XMITQ のデフォルト値です。
3. ローカル・キュー・マネージャーの DEFXMITQ 属性に名前が指定されている伝送キュー。

メッセージ・チャネル・エージェントは、伝送キューと関連付けられたチャネル・プログラムで、次の宛先にメッセージを送達します。 次の宛先とは、メッセージ・チャネルが接続されるキュー・マネージャーです。 それはメッセージの最終宛先と同じキュー・マネージャーである必要はありません。 メッセージは、次の宛先に送達されると伝送キューから削除されます。 メッセージは、最終宛先までの経路上でいくつものキュー・マネージャーを通過しなければならない場合があります。 経路上の各キュー・マネージャーには、伝送キューを定義する必要があります。各伝送キューは、次の宛先に伝送されるのを待っているメッセージを保持します。 通常の伝送キューは、メッセージの最終的な宛先が異なる場合でも、次の宛先へのメッセージを保持します。 クラスター伝送キューは、複数の宛先へのメッセージを保持します。 各メッセージの `correlID` は、次の宛先に転送するためにメッセージが入れられるチャネルを識別します。

キュー・マネージャーには、複数の伝送キューを定義できます。 同じ宛先に対して、それぞれが異なるサービスのクラスに用いられる伝送キューをいくつか定義する場合もあります。 例えば、同じ宛先に送る小規模メッセージと大規模メッセージに異なる伝送キューを作成したい場合があります。 そうすれば異なるメッセージ・チャネルを使用してメッセージを転送できるため、大規模メッセージが小規模メッセージを遅らせてしまうことがありません。 デフォルトでは、クラスター・キューまたはクラスター・トピックへのメッセージはすべて、単一のクラスター伝送キュー `SYSTEM.CLUSTER.TRANSMIT.QUEUE` に入れられます。 オプションとして、デフォルトを変更し、異なるクラスター・キュー・マネージャーから異なるクラスター伝送キューに行くようメッセージ・トラフィックを分離することができます。 キュー・マネージャー属性 `DEFCLXQ` を `CHANNEL` に設定すると、各クラスター送信側チャネルが別々のクラスター伝送キューを作成します。 代わりに、クラスター送信側チャネルに使用するクラスター伝送キューを手動で定義することもできます。

伝送キューは、メッセージ・チャネル・エージェントを起動して、メッセージを転送することができます。トリガーによる IBM MQ アプリケーションの開始を参照してください。

z/OS IBM MQ for z/OS では、グループ内キューイングを使用している場合は、伝送キューはグループ内キューイング・エージェントによりサービスされます。 共用伝送キューは、IBM MQ for z/OS でグループ内キューイングを使用する際に使われます。

開始キュー

開始キュー は、トリガー・イベントがアプリケーション・キューで起きたときに、キュー・マネージャーがトリガー・メッセージを書き込むローカル・キューです。

トリガー・イベントとは、プログラムにキューの処理を開始させることを目的とするイベントのことです。 例えば、到着メッセージが 10 件を超えるというイベントなどがあります。 トリガー操作の詳細については、トリガーによる IBM MQ アプリケーションの開始を参照してください。

送達不能 (未配布メッセージ) キュー

送達不能 (未配布メッセージ) キュー は、キュー・マネージャーが送達不能なメッセージを書き込むローカル・キューです。

キュー・マネージャーは、メッセージを送達不能キューに書き込むときに、メッセージにヘッダーを追加します。 ヘッダー情報には、キュー・マネージャーがそのメッセージを送達不能キューに書き込んだ理由が入っています。 また、元のメッセージの宛先、キュー・マネージャーがそのメッセージを送達不能キューに書き込んだ日時なども入っています。

また、アプリケーションも送達不能のメッセージ用にキューを使用できます。 詳細については、送達不能 (未配布メッセージ) キューの使用を参照してください。

システム・コマンド・キュー

システム・コマンド・キュー は、適正な許可を持つアプリケーションが IBM MQ コマンドを送信できるキューです。 これらのキューはプラットフォームでサポートされている PCF、MQSC および CL コマンドを受信し、キュー・マネージャーがそれらを処理できるようにします。

z/OS IBM MQ for z/OS では、キューは SYSTEM.COMMAND.INPUT と呼ばれます。 他のプラットフォームでは、 SYSTEM.ADMIN.COMMAND.QUEUE と呼ばれます。 受け入れられるコマンドはプラットフォームによって変わります。 詳細については、プログラマブル・コマンド・フォーマット・リファレンスを参照してください。

システム・デフォルト・キュー

システム・デフォルト・キュー には、ご使用のシステム用のキューの初期定義が格納されています。 キュー定義を作成すると、キュー・マネージャーは該当のシステム・デフォルト・キューからその定義をコピーします。 キュー定義の作成は、動的キューの作成とは異なります。 動的キューの定義は、動的キューのテンプレートとして選択したモデル・キューに基づきます。

イベント・キュー

イベント・キュー はイベント・メッセージを保持します。 これらのメッセージは、キュー・マネージャーまたはチャネルによって報告されます。

リモート・キュー

プログラムが接続されているキュー・マネージャーとは別のキュー・マネージャーが所有するキューは、そのプログラムにとってリモート です。

通信リンクが確立されている場合、プログラムはリモート・キューにメッセージを送信できます。 ただし、プログラムはリモート・キューからメッセージを読み取ることはできません。

キュー定義オブジェクトはリモート・キューを定義する際に作成され、ローカル・キュー・マネージャーがメッセージの送信先のキューを見つけるために必要な情報だけを保持します。 このオブジェクトは、リモート・キューのローカル定義 と呼ばれます。 リモート・キューの属性は、すべてそれを所有するキュー・マネージャーによって保持されますが、これは、リモート・キューが、そのキュー・マネージャーにとってはローカル・キューになるためです。

リモート・キューをオープンするときは、キューを識別するために次のどちらかを指定しなければなりません。

- リモート・キューを定義するローカル定義の名前。 アプリケーションの視点からは、これはローカル・キューのオープンと同じです。 アプリケーションは、キューがローカルかリモートかを認識する必要はありません。

  IBM i 以外のすべてのプラットフォームでリモート・キューのローカル定義を作成するには、 DEFINE QREMOTE コマンドを使用します。

IBM i では、CRTMQMQ コマンドを使用します。

- リモート・キュー・マネージャーの名前と、そのリモート・キュー・マネージャーに認識されているキューの名前。

20 ページの『キューの属性』で説明した共通属性のほかに、リモート・キューのローカル定義には 3 つの属性があります。 以下の 3 つの属性が該当します。

RemoteQName
キューを所有するキュー・マネージャーが、そのキューを識別する名前。

RemoteQMgrName
所有キュー・マネージャーの名前。

XmitQName
メッセージを他のキュー・マネージャーに転送するときに使用するローカル伝送キューの名前。

これらの属性の詳細については、キューの属性を参照してください。

リモート・キューのローカル定義に対し MQINQ 呼び出しを使用する場合、キュー・マネージャーが戻すのは、リモート・キュー名、リモート・キュー・マネージャー名、および伝送キュー名のローカル定義の属性だけで、リモート・システム内で一致するローカル・キューの属性は戻しません。

伝送キューも参照してください。

別名キュー

別名キュー とは、別のキューまたはトピックにアクセスするために使用できる IBM MQ オブジェクトです。これは、複数のプログラムがその同じキューを別の名前でアクセスして、作業できることを意味しています。

別名を解決した結果として得られるキュー (基本キューと呼ばれる) には、 プラットフォームでサポートされる、以下のいずれかの種類のキューを指定できます。

- ローカル・キュー
- リモート・キューのローカル定義。
- 共用キュー。これは、IBM MQ for z/OS でのみ使用可能なローカル・キューのタイプです。
- 事前定義キュー
- 動的キュー

別名を解決した結果がトピックになることもあります。 現在はキューにメッセージを 書き込むアプリケーションがある場合、キューの名前をトピックの別名にすれば、そのアプリケーションはトピックへのパブリッシュを行う ようになります。 アプリケーション・コードを変更する 必要はありません。

注 : 別名を同一キュー・マネージャー上の別の別名に直接解決することはできません。

別名キューの使用例は、システム管理者が、基本キューの名前 (つまり、別名が解決された結果のキュー名) と別名キューの名前に異なったアクセス許可を与える場合です。 これは、プログラムまたはユーザーが、基本キューではなく、別名キューを使用するように許可されることを意味します。

または、別名の書き込み操作は禁止し、基本キューの書き込み操作は可能になるように許可を設定できます。

あるアプリケーションでは、別名キューの使用により、システム管理者がアプリケーションを変更しなくても、別名キュー・オブジェクトの定義を容易に変更できるようになります。

プログラムが別名を使用しようとすると、IBM MQ は、別名に対する許可検査を行います。 ただし、プログラムに、その別名が解決された結果の名前にアクセスする許可が与えられているかどうかは検査しません。 したがって、プログラムには別名キューの名前にアクセスする許可は与えられますが、解決された結果のキュー名にアクセスする許可は与えられません。

19 ページの『キュー』に記述されている一般的なキュー属性のほかに、別名キューは **BaseQName** 属性を持っています。 これは、別名が解決されたときの基本キューの名前です。 この属性の詳細については、BaseQName (MQCHAR48) を参照してください。

別名キューの *InhibitGet* 属性および **InhibitPut** 属性 (19 ページの『キュー』を参照) は、別名に属します。 例えば、別名キュー名 ALIAS1 が基本キュー名 BASE に解決される場合は、ALIAS1 に対する禁止事項は ALIAS1 だけを対象にし、BASE は禁止されません。 ただし、BASE に対する禁止事項は ALIAS1 にも影響を及ぼします。

DefPriority 属性および **DefPersistence** 属性も別名に属します。 したがって、例えば、同じ基本キューの異なった別名に異なるデフォルト優先順位を割り当てることができます。 さらに、別名を使用するアプリケーションを変更しなくても、これらの優先順位を変えることもできます。

動的キューとモデル・キュー

この情報は、動的キュー、一時および永続の動的キューのプロパティー、動的キューの使用方法、動的キューを使用する際の考慮事項、およびモデル・キューについて詳しく説明するものです。

アプリケーション・プログラムがモデル・キューをオープンするために MQOPEN 呼び出しを出すと、キュー・マネージャーは動的に、そのモデル・キューと同じ属性をもつローカル・キューのインスタンスを作成します。 モデル・キューの *DefinitionType* フィールドの値に応じて、キュー・マネージャーは一時または永続の動的キューを作成します (動的キューの作成を参照)。

一時動的キューの特性

一時動的キュー は次のような特性をもっています。

- z/OS キュー共用グループ内のキュー・マネージャーがアクセスできる共用キューではない。

  キュー共用グループが使用可能なのは、IBM MQ for z/OS のみです。
- 非持続メッセージだけを保持する。
- リカバリー不能である。
- キュー・マネージャーの始動時に削除される。
- キューを作成した MQOPEN 呼び出しを発行したアプリケーションがキューをクローズしたときか、そのアプリケーションが終了したときに、削除される。
  - コミットされたメッセージがキューにある場合には削除される。
  - この時に、キューに対して未解決で、コミットされない MQGET、MQPUT、または MQPUT1 呼び出しがある場合は、キューは論理的に削除されたものとしてマーク付けされる。さらに、このキューは、(これらの呼び出しがコミットされたあとに) クローズ処理の一部として、またはアプリケーションが終了した時に、物理的に削除される。
  - この時 (作成中のアプリケーションまたは他のアプリケーションが) キューを使用している場合、そのキューは物理的に削除されたものとしてマーク付けされ、それを使用する最後のアプリケーションによってクローズされたときに、物理的に削除される。
  - 論理的に削除されたキューに (クローズ以外の目的で) アクセスしようとすると、MQRC_Q_DELETED の理由コードで失敗する。
  - MQCO_NONE、MQCO_DELETE、および MQCO_DELETE_PURGE は、キューを作成した MQOPEN 呼び出しに対応する MQCLOSE 呼び出しで指定すると、すべて MQCO_NONE として処理される。

永続動的キューの特性

永続動的キュー は、次のような特性を持っています。

- 持続メッセージまたは非持続メッセージを保持する。
- システム障害が生じた場合にリカバリー可能である。
- アプリケーション (必ずしも、キューを作成した MQOPEN 呼び出しを発行したアプリケーションでなくてもよい) が、MQCO_DELETE または MQCO_DELETE_PURGE オプションを使用してキューを正常にクローズしたときに、削除される。
  - MQCO_DELETE オプションを指定したクローズ要求は、キュー上にメッセージ (コミットされた、またはコミットされない) がまだ残っている場合は、失敗する。 MQCO_DELETE_PURGE オプションを指定したクローズ要求は、コミットされたメッセージがキュー上にある場合でも、正常に実行される (メッセージはクローズの一部として削除される)。ただし、キューに対して未解決で、コミットされない MQGET、MQPUT、または MQPUT1 呼び出しがある場合は、失敗する。

  - 削除要求が成功しても (作成中の、または他のアプリケーションによって)、キューが使用中の場合は、そのキューは、論理的に削除されたものとしてマーク付けされ、それを使用する最後のアプリケーションによってクローズされた時に、物理的に削除される。
- キューを削除する権限のないアプリケーションがキューをクローズした場合は、キューをクローズしたアプリケーションがキューを作成した MQOPEN 呼び出しを発行しない限り、キューは削除されません。 許可検査は、対応する MQOPEN 呼び出しの妥当性検査に使用されたユーザー ID (MQOO_ALTERNATE_USER_AUTHORITY が指定されていた場合は、代替ユーザー ID) に対して行われる。
- 通常のキューと同様に削除できる。

動的キューの使用方法

次のような場合に動的キューを使用できます。

- 処理を終了した後、キューを保持する必要がないアプリケーション
- メッセージに対する応答を別のアプリケーションに処理させるアプリケーション。 このようなアプリケーションは、モデル・キューをオープンして応答先キューを動的に作成できます。 例えば、クライアント・アプリケーションは次のようにすることができます。

  1. 動的キューを作成する。
  2. 要求メッセージのメッセージ記述子構造体の **ReplyToQ** フィールドにその名前を指定する。
  3. サーバーによって処理されるキューにその要求を入れる。

これで、サーバーは応答メッセージを応答先キューに入れることができるようになります。 最後に、クライアントはその応答を処理し、削除オプションを使用して応答先キューをクローズできます。

動的キューを使用する際の考慮事項

動的キューを使用するときは次の点を考慮してください。

- クライアント/サーバー・モデルでは、各クライアントが各自の動的応答先キューを作成して使用しなければならない。 動的応答先キューが複数のクライアントで共用されている場合は、そのキューに対して未解決でコミットされない活動があったり、そのキューが他のクライアントによって使用されていたりして、応答先キューの削除が遅れることがあります。 さらに、キューは論理的に削除されたというマークが付けられたために、後続の (MQCLOSE 以外の) API 要求でアクセス不能になることもあります。
- 使用しているアプリケーションの環境により、動的キューをアプリケーション間で共用しなければならない場合は、必ず、そのキューに対するすべての活動がコミットされてから (削除オプションで)、そのキューがクローズされるようにする。 これは、最後のユーザーが行う必要があります。 これによって、キューの削除が遅れないようになり、論理的に削除されたものとしてマーク付けされたためにキューがアクセス不能になっている期間が最小限になります。

モデル・キュー

モデル・キュー はキュー定義のテンプレートで、動的キューを作成する際に使用します。

IBM MQ プログラムからローカル・キューを動的に作成し、キュー属性のテンプレートとして使用したいモデル・キューを命名できます。 この時点で、新規のキューのいくつかの属性を変更することができます。ただし、**DefinitionType** を変更することはできません。 例えば、永続キューが必要な場合には、定義タイプが永続的に設定されたモデル・キューを選択してください。 ある会話型のアプリケーションでは、動的キューを使用して照会に対する応答を入れておくことができます。なぜなら、多くの場合、応答を処理し終わったらそれらのキューを保持する必要がなくなるからです。

MQOPEN 呼び出しのオブジェクト記述子 (MQOD) に、モデル・キューの名前を指定します。 キュー・マネージャーは、指定されたモデル・キューの属性を使用して、ローカル・キューを動的に作成します。

動的キューの名前を (完全名で) 指定するか、名前の語幹 (例えば、ABC) を指定して、キュー・マネージャーが固有の部分をこれに追加するようにできます。または、キュー・マネージャーがユーザーの代わりに完全な固有名を割り当てるようにすることもできます。 キュー・マネージャーが名前を割り当てる場合、それを MQOD 構造体に書き込みます。

モデル・キューに直接 MQPUT1 呼び出しを発行することはできませんが、モデル・キューをオープンしたときに作成された動的キューに対して MQPUT1 を発行することは可能です。

モデル・キューに対して MQSET や MQINQ を発行することはできません。 MQOO_INQUIRE か MQOO_SET を使用してモデル・キューを開くと、それ以降の MQINQ 呼び出しや MQSET 呼び出しは、動的に作成されたキューに対して行われます。

モデル・キューの属性は、ローカル・キューの属性のサブセットです。 詳細については、キューの属性を参照してください。

IBM MQ によって特定の目的で使用されるキュー

IBM MQ は、一部のローカル・キューを、その操作に関連する特定の目的のために使用します。

これらのキューは、IBM MQ によって使用される前に、必ず定義する必要があります。

開始キュー

開始キュー は、トリガー操作に使用するキューです。 キュー・マネージャーは、トリガー・イベントが発生すると、開始キュー にトリガー・メッセージを書き込みます。 トリガー・イベントとは、キュー・マネージャーによって検出される条件の論理的 組み合わせのことです。 例えば、キュー上のメッセージの数が、あらかじめ定義されたキューのサイズに達したときにトリガー・イベントが生成される場合があります。 このイベントが発生すると、キュー・マネージャーは指定の開始キューに トリガー・メッセージを入れることになります。 このトリガー・メッセージは、開始キューをモニターする特殊アプリケーションであるトリガー・モニター によって取り出されます。 次に、トリガー・モニターは、トリガー・メッセージに指定されているアプリケーション・プログラムを開始します。

キュー・マネージャーでこのトリガー操作を使用する場合は、少なくとも 1 つの 開始キューを、そのキュー・マネージャー用に定義する必要があります。 トリガー操作のためのオブジェクトの管理、runmqtrm、およびトリガーによる IBM MQ アプリケーションの開始を参照してください。

伝送キュー

伝送キューは、リモート・キュー・マネージャー宛てのメッセージを一時的に保管するキューです。 ローカル・キュー・マネージャーがメッセージを直接送信する各リモート・ キュー・マネージャーごとに、少なくとも 1 つの伝送キューを定義する必要が あります。 これらのキューは、リモート管理にも使用されます (ローカル・キュー・マネージャーからのリモート管理を参照)。 分散キューイングでの伝送キューの使用については、 IBM MQ 分散キューイング技法を参照してください。

各キュー・マネージャーには、1 つのデフォルト伝送キューがあります。 クラスター外のキュー・マネージャーがリモート・キューにメッセージを書き込む場合、デフォルトでは、デフォルト伝送キューが使用されます。 宛先のキュー・マネージャーと同名の伝送キューがあれば、その伝送キューにメッセージが入れられます。 キュー・マネージャーの別名定義で、**RQMNAME** パラメーターが宛先キュー・マネージャーと一致し、**XMITQ** パラメーターが指定されている場合は、**XMITQ** により指定された伝送キューにメッセージが入れられます。 **XMITQ** パラメーターがない場合は、メッセージで指定されたローカル・キューにメッセージが入れられます。

クラスター伝送キュー

クラスター内のそれぞれのキュー・マネージャーには、SYSTEM.CLUSTER.TRANSMIT.QUEUE というクラスター伝送キューと、SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE というモデル・クラスター伝送キューがあります。 これらのキューの定義は、キュー・マネージャーを定義するときにデフォルトとして作成されます。 キュー・マネージャーの属性、**DEFCLXQ** が CHANNEL に設定された場合、作成された各クラスター送信側チャネルに、永続動的クラスター伝送キューが自動的に作成されます。 キューは SYSTEM.CLUSTER.TRANSMIT. *ChannelName* と呼ばれます。 クラスター伝送キューを手動で定義することもできます。

クラスター内のキュー・マネージャーは、同じクラスター内の他の キュー・マネージャーに、これらのキュー上のメッセージを送ります。

名前の解決時には、クラスター伝送キューがデフォルト伝送キューより優先され、特定のクラスター伝送キューが SYSTEM.CLUSTER.TRANSMIT.QUEUE より優先されます。

送達不能キュー
送達不能 (未配布メッセージ) キューは、正しい宛先に渡すことができないメッセージを保管します。 例えば、宛先キューが満杯である場合には、メッセージは転送されません。 システムに提供された送達不能キューは、SYSTEM.DEAD.LETTER.QUEUE と呼ばれます。

分散キューイングでは、関係する各キュー・マネージャーごとに 1 つの送達不能キューを定義してください。

コマンド・キュー
コマンド・キュー SYSTEM.ADMIN.COMMAND.QUEUE は、適切な許可を与えられたアプリケーションが処理対象の MQSC コマンドを送信する先のローカル・キューです。 次に、これらのコマンドは、コマンド・サーバーと呼ばれる IBM MQ コンポーネントによって取り出されます。 コマンド・サーバーは、それらのコマンドを検査し、正しいものをキュー・マネージャーの処理用に渡し、該当する応答先キューに応答を戻します。

キュー・マネージャーを作成すると、各キュー・マネージャーごとにコマンド ・キューが自動的に作成されます。

応答先キュー
あるアプリケーションが要求メッセージを送信した場合、そのメッセージを 受信するアプリケーションは、送信側のアプリケーションに応答メッセージを 戻すことができます。 この応答メッセージは、応答先キューと呼ばれるキューに書き込まれます。このキューは、通常は送信側のアプリケーションのローカル・キューです。 応答先キューの名前は、送信側のアプリケーションによってメッセージ記述子の 一部として指定されます。

イベント・キュー
観測イベントは、MQI アプリケーションとは無関係にキュー・マネージャーをモニターするときに使用できます。

観測イベントが発生すると、キュー・マネージャーはイベント・キューにイベント・メッセージを書き込みます。 書き込まれたこのメッセージは、モニター・アプリケーションにより読み取られます。このアプリケーションは、イベントが問題を提示すると、管理者に通知したり何らかの矯正処置を開始します。

注 : トリガー・イベントは観測イベントとは異なります。 トリガー・イベントは、同じ条件では発生せず、イベント・メッセージが生成されません。

観測イベントの詳細については、観測イベントを参照してください。

キュー・マネージャー

キュー・マネージャー およびキュー・マネージャーがアプリケーションに提供するキューイング・サービスの概要です。

プログラムは、キュー・マネージャーのサービスを使用する前に、そのキュー・マネージャーへ接続していなければなりません。 この接続は、プログラムが明示的に (MQCONN または MQCONNX 呼び出しを使用する) 行うこともできますし、暗示的に行われることもあります (これはプログラムを実行するプラットフォームおよび環境に依存します)。

IBM MQ キュー・マネージャーは、以下のアクションを実行します。

- オブジェクトの属性は、受け取ったコマンドに応じて変更されます。
- 該当する条件が満たされたときに、トリガー・イベントや観測イベントなどの特殊イベントが生成されます。
- メッセージが、MQPUT 呼び出しを行ったアプリケーションの要求により、正しいキューに書き込まれます。 正しいキューに入れられなかった場合には、アプリケーションに通知され、該当する理由コードが戻されます。

それぞれのキューは、1 つのキュー・マネージャーに属しており、そのキュー・マネージャーに対してローカル・キュー であるといいます。 アプリケーションが接続されているキュー・マネージャーは、そのアプリケーションに対してローカル・キュー・マネージャー であるといいます。 アプリケーションのローカル・キュー・マネージャーに属しているキューは、そのアプリケーションのためのローカル・キューです。

リモート・キュー とは、別のキュー・マネージャーに属しているキューのことです。 リモート・キュー・マネージャー とは、ローカル・キュー・マネージャー 以外の任意のキュー・マネージャーのことです。 リモート・キュー・マネージャーは、ネットワーク内のリモート・マシン上にある場合と、ローカル・キュー・マネージャーと同じマシン上にある場合があります。 IBM MQ では、1 つのマシン上で複数のキュー・マネージャーを使用することができます。

キュー・マネージャー・オブジェクトは、一部の MQI 呼び出しで使用することができます。 例えば、キュー・マネージャー・オブジェクトの属性について、MQI 呼び出しの MQINQ を使用して問い合わせることができます。

キュー・マネージャーの属性

各キュー・マネージャーに関連付けられているのは、その特性を定義する一連の属性 (または特性) です。 キュー・マネージャーの属性のいくつかは、作成される時に固定されます。他のものについては、IBM MQ コマンドを使用して変更できます。 すべての属性の値は、Transport Layer Security (TLS) 暗号化で使用されるものを除いて、MQINQ 呼び出しを使って問い合わせることができます。

固定されている属性は、次のとおりです。

- キュー・マネージャーの名前
- キュー・マネージャーが稼働するプラットフォーム (例えば、Windows)
- キュー・マネージャーがサポートするシステム制御コマンドのレベル
- キュー・マネージャーが処理するメッセージに割り当てられる最高優先順位
- プログラムが IBM MQ コマンドを送信できるキューの名前
- キュー・マネージャーが処理できるメッセージの最大長 z/OS (IBM MQ for z/OS でのみ固定)
- プログラムがメッセージを書き込んだり読み取ったりするときに、キュー・マネージャーが同期点機能をサポートするかどうか

変更可能な 属性は、次のとおりです。

- キュー・マネージャーのテキスト記述子
- キュー・マネージャーが MQI 呼び出しを処理するときに文字ストリングに使用する文字セットの ID
- トリガー・メッセージの数を制限するためにキュー・マネージャーが使用する時間間隔
- z/OS キュー・マネージャーがキュー内の期限切れメッセージをスキャンする頻度を決定するのに使用される時間間隔 (IBM MQ for z/OS のみ)
- キュー・マネージャーの送達不能 (未配布メッセージ) キュー
- キュー・マネージャーのデフォルトの伝送キューの名前
- 任意の接続のためのオープン・ハンドルの最大数
- イベント報告の各種カテゴリーの設定または解除
- 1 つの作業単位内のコミットされていないメッセージの最大数

キュー・マネージャーとワークロード管理

1 つのキューに対して複数の定義が作成されている場合は、いくつかのキュー・マネージャーからなる 1 つのクラスターを設定することができます (例えば、クラスター内のキュー・マネージャーは、それぞれが他のキュー・マネージャーの複製として機能できます)。 特定のキューに対するメッセージは、そのキューのインスタンスのホストになっているすべてのキュー・マネージャーで処理できます。 ワークロード管理アルゴリズムは、どのキュー・マネージャーがメッセージを処理するか決定し、キュー・マネージャー間でワークロードを分散します。詳細については、クラスター・ワークロード管理アルゴリズムを参照してください。

チャネル

チャネルは、分散キュー・マネージャーが使用する IBM MQ MQI client と IBM MQ サーバー間、または 2 つの IBM MQ サーバー間の論理通信リンクのことです。

あるキュー・マネージャーから別のキュー・マネージャーにメッセージを移すためにチャネルが使用され、それによってアプリケーションは基礎をなす通信プロトコルから遮蔽されます。 キュー・マネージャーは、同一のシステム上に存在する場合もあれば、同一のプラットフォーム上の異なるシステムや、異なるプラットフォーム上に存在する場合もあります。 メッセージの送信元は次のように多岐にわたります。

- あるノードから別のノードにデータを転送するユーザー作成のアプリケーション・プログラム。
- PCF コマンドまたは MQAI を使用するユーザー作成の管理アプリケーション。
- IBM MQ Explorer。
- インスツルメンテーション・イベント・メッセージを別のキュー・マネージャーに送信するキュー・マネージャー。
- リモート管理コマンドを別のキュー・マネージャーに送信するキュー・マネージャー。 例えば、MQSC コマンドや administrative REST API を使用します。

1 つのチャネルには、2 つの定義があり、それらは接続の両端に 1 つずつあります。 キュー・マネージャーが相互に通信できるようにするには、メッセージを送信するキュー・マネージャーに 1 つのチャネル・オブジェクトを定義し、メッセージを受信するキュー・マネージャーに別の補完的なチャネル・オブジェクトを定義する必要があります。 接続の両端には、同じチャネル名 を使用する必要があり、使用するチャネル・タイプ には互換性がなければなりません。

IBM MQ のチャネルは、次に示すように 3 つのカテゴリーに分かれます (これらのカテゴリー内に異なるチャネル・タイプがあります)。

- メッセージ・チャネル。単一方向のもので、あるキュー・マネージャーから他のキュー・マネージャーへメッセージを転送します。
- MQI チャネル。双方向のもので、IBM MQ MQI client からキュー・マネージャーに MQI 呼び出しを転送し、キュー・マネージャーから IBM MQ クライアントに応答を転送します。
- AMQP チャネルは両方向であり、AMQP クライアントをサーバー・マシン上のキュー・マネージャーに接続します。 IBM MQ は AMQP チャネルを使用して、AMQP アプリケーションとキュー・マネージャー間の AMQP 呼び出しおよび応答を転送します。

メッセージ・チャネル

メッセージ・チャネルの目的は、1 つのキュー・マネージャーから別のキュー・マネージャーにメッセージを移すことです。 メッセージ・チャネルは、クライアント・サーバー環境で必須ではありません。

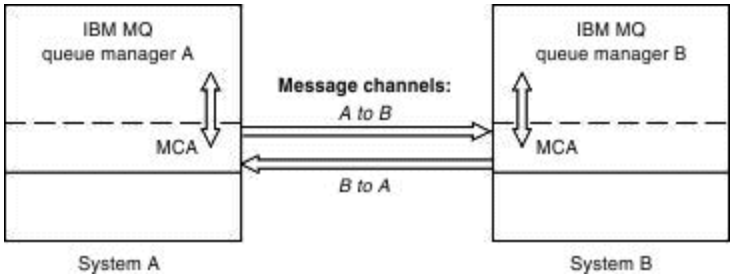


図 *2. 2* つのキュー・マネージャー間のメッセージ・チャネル

メッセージ・チャネルは、単一方向のリンクです。 ローカル・キュー・マネージャーから送信されたメッセージにリモート・キュー・マネージャーを応答させるには、ローカル・キュー・マネージャーに応答を返す別のチャネルをセットアップする必要があります。

メッセージ・チャネルは、メッセージ・チャネル・エージェント (MCA) を使用して 2 つのキュー・マネージャーを接続します。 チャネルの両側に、メッセージ・チャネル・エージェントが 1 つずつあります。 MCA は、複数のスレッドを使用してメッセージを転送できます。 このプロセスを、パイプライン と呼びま

す。 パイプラインを使用すると、MCA のメッセージ転送効率が向上し、その結果、チャネルのパフォーマンスが向上します。 パイプラインについて詳しくは、チャネルの属性を参照してください。

チャネルについて詳しくは、チャネル出口呼び出しおよびデータ構造体と、47 ページの『分散キューイング・コンポーネント』を参照してください。

MQI チャネル

メッセージ・キュー・インターフェース (MQI) チャネルは、 IBM MQ MQI client をサーバー・マシン上のキュー・マネージャーに接続し、IBM MQ MQI client アプリケーションから MQCONN または MQCONNX 呼び出しを発行すると確立されます。

これは両方向のリンクであり、メッセージ・データが入った MQPUT 呼び出しや、メッセージ・データが戻される結果となる MQGET 呼び出しなど、MQI 呼び出しおよび応答だけを転送するのに使用されます。 チャネル定義の作成と使用方法は 2 つあります (MQI チャネルの定義を参照してください)。

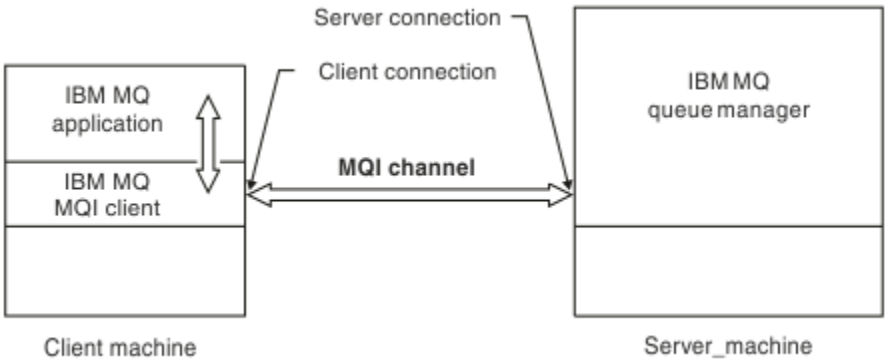


図 *3. MQI* チャネル上のクライアント接続およびサーバー接続

z/OS MQI チャネルを使用して、単一のキュー・マネージャー、あるいはキュー共用グループの一部となっているキュー・マネージャーにクライアントを 接続することができます (キュー共用グループへのクライアントの接続を参照)。

MQI チャネル定義のチャネル・タイプには 2 種類あります。 これらのチャネル・タイプでは、両方向 MQI チャネルを定義します。

クライアント接続チャネル
このタイプは、IBM MQ MQI client 用です。

サーバー接続チャネル
このタイプは、 IBM MQ MQI client 環境で実行されている IBM MQ アプリケーションが通信するキュー・マネージャーを実行しているサーバー用です。

AMQP チャネル

Multi

AMQP チャネルのタイプは 1 つだけです。

チャネルを使用して AMQP メッセージング・アプリケーションをキュー・マネージャーと接続し、アプリケーションが IBM MQ アプリケーションとメッセージを交換できるようにします。 AMQP チャネルにより、MQ Light を使用してアプリケーションを開発し、IBM MQ によって提供されるエンタープライズ・レベルの機能を利用して、そのアプリケーションをエンタープライズ・アプリケーションとしてデプロイすることができます。

クライアント接続チャネル

クライアント接続チャネル とは、IBM MQ MQI client からキュー・マネージャーへの通信パスを提供するオブジェクトのことです。

クライアント接続チャネルは、キュー・マネージャーとクライアントとの間でメッセージを移動するために、分散キューイングで使用されます。 チャネルは、基礎をなす通信プロトコルからアプリケーションを保護します。 クライアントは、キュー・マネージャーと同一のプラットフォーム上に存在する場合もあれば、異なるプラットフォーム上に存在する場合もあります。

チャネル定義

それぞれのチャネルのタイプについては、32 ページの『チャネル定義』を参照してください。

関連概念

43 ページの『分散キューイングとクラスター』
分散キューイングとは、あるキュー・マネージャーから別のキュー・マネージャーにメッセージを送信することです。 受信キュー・マネージャーは、同じマシン上に置くことも別のマシン上に置くこ ともでき、近くにあっても地球の裏側にあってもメッセージを受信できます。 このプログラムは、ローカル・キュー・マネージャーと同じプラットフォーム、または IBM MQ がサポートするいずれかのプラットフォームで実行することができます。 分散キューイング環境内のすべての接続を手動で定義することもできますし、クラスターを作成して IBM MQ で接続詳細の多くが定義されるようにすることもできます。

Message Queue Interface の概要

関連タスク

リモート IBM MQ オブジェクトの管理
MQI チャネルの停止中
サーバーとクライアント間の接続の構成

関連資料

チャネル出口呼び出しおよびデータ構造体

36 ページの『通信』
IBM MQ MQI clients は、サーバーとの通信に MQI チャネルを使用します。

チャネル定義

IBM MQ が使用するメッセージ・チャネルおよび MQI チャネルのさまざまなタイプを説明する表。

メッセージ・チャネルに言及する際、チャネルという言葉はよくチャネル定義の同義語として使用されます。 2 つの側を持つ完全チャネルのことか、または 1 つの側しかないチャネル定義のことなのかは、ふつうはその文脈から明らかです。

メッセージ・チャネル

メッセージ・チャネル定義は、以下のタイプのいずれかです。

メッセージ・チャネル定義のタイプ	説明
送信側	送信側チャネルは、キュー・マネージャーがメッセージを他のキュー・マネージャーに送信するために使用するメッセージ・チャネルです。 送信側チャネルを使用してメッセージを送信するには、 他のキュー・マネージャー上に送信側チャネルと同じ名前の受信側チャネルも作成しなければなりません。 「コールバック」メカニズムを実装している場合は、送信側チャネルを要求側チャネルと共に使用できます。

メッセージ・チャネル定義のタイプ	説明
サーバー	サーバー・チャネルは、キュー・マネージャーがメッセージを他のキュー・マネージャーに送信するために使用するメッセージ・チャネルです。 サーバー・チャネルを使用してメッセージを送信するには、 他のキュー・マネージャー上にサーバー・チャネルと同じ名前の受信側チャネルも作成しなければなりません。 要求側チャネルを持つサーバー・チャネルを使用することもできます。 その場合、 チャネルの他方の終端の要求側チャネル定義によって、サーバー・チャネル定義の開始が要求されます。 サーバーは要求側にメッセージを送信します。 サーバーにパートナー・チャネルの接続名が知られている限り、サーバーも通信を開始することができます。
受信側	受信側チャネルは、キュー・マネージャーがメッセージを他のキュー・マネージャーから受信するために使用するメッセージ・チャネルです。 受信側チャネルを使用してメッセージを受信するには、他のキュー・マネージャー上にこの受信側チャネルと同じ名前の送信側またはサーバー・チャネルも作成しなければなりません。
リクエスター	要求側チャネルは、キュー・マネージャーが他のキュー・マネージャーからメッセージを受信するために使用するメッセージ・チャネルです。 要求側チャネルは、リモート・エンドで定義されているパートナー・チャネルに開始を要求できます。 パートナー・チャネルがサーバー・チャネルの場合、サーバー・チャネルは開始要求を受け入れて、 サーバー・チャネル定義で識別されている伝送キューから要求側チャネルへのメッセージの送信を開始します。 パートナー・チャネルが送信側チャネルの場合、送信側チャネルは開始要求を受け入れますが、その時点で要求側との接続をクローズします。 その後、送信側チャネルが開始し、要求側であるパートナー・チャネルとのセッションをネゴシエーションして、送信側チャネル定義で識別されている伝送キューからのメッセージの送信を開始します。 要求側チャネルが送信側チャネルにコールバックを要求するという点で、この後者のケースは本質的にコールバック・メカニズムを提供しています。

メッセージ・チャネル定義のタイプ	説明
クラスター送信側	クラスター送信側 (CLUSSDR) チャネル定義は、クラスター・キュー・マネージャーがいずれかの フル・リポジトリーにクラスター情報を送信できるチャネルの送信側を定義します。 クラスター送信側チャネルを使用して、キューの追加や削除など、キュー・マネージャーの状況の変化をリポジトリーに 通知します。 また、このチャネルは、メッセージの送信にも使用されます。 フル・リポジトリー・キュー・マネージャー自体に、お互いを指し示すクラスター送信側チャネルがあります。 フル・リポジトリー・キュー・マネージャーは、このチャネルを使用してクラスター状況の変更を相互に通信します。 キュー・マネージャーの CLUSSDR チャネル定義がどのフル・リポジトリーを指しているかは、あまり重要ではありません。 最初の接続が行われた後に、必要に応じて、自動的にクラスター・キュー・マネージャー・オブジェクトがさらに定義されます。 これで、キュー・マネージャーがすべてのフル・リポジトリーにクラスター情報を送信し、 すべてのキュー・マネージャーにメッセージを送信することができるようになります。
クラスター受信側	クラスター受信側 (CLUSRCVR) チャネル定義は、 クラスター・キュー・マネージャーがクラスター内の他のキュー・マネージャーからメッセージを受信できるチャネルの受信側を定義します。 クラスター受信側チャネルは、リポジトリー宛てのクラスター情報に関する情報も伝送できます。 クラスター受信側チャネルを定義すると、キュー・マネージャーは、 メッセージを受信することができることを他のクラスター・キュー・マネージャーに示します。 各クラスター・キュー・マネージャーごとに、少なくとも 1 つのクラスター受信側チャネルが必要です。

それぞれのチャネルごとに両端を定義し、チャネルのそれぞれの終端にチャネル定義があるようにする必要があります。 チャネルの両端のタイプは互換タイプでなければなりません。

以下のチャネル定義の組み合わせが可能です。

- 送信側 - 受信側
- サーバー - 受信側
- 要求側 - サーバー
- 要求側 - 送信側 (コールバック)
- クラスター送信側 - クラスター受信側

メッセージ・チャネル・エージェント

作成する各チャネル定義は、特定のキュー・マネージャーに属している必要があります。 キュー・マネージャーは、同じまたは異なるタイプのいくつかのチャネルを持つことができます。 チャネルの両端に、プログラム、メッセージ・チャネル・エージェント (MCA) があります。 チャネルの片側で、呼び出し側 MCA が伝送キューからメッセージを取り出し、それらをチャネルを通して送信します。 チャネルのもう一方の側で、応答側 MCA がメッセージを受信し、リモート・キュー・マネージャーへそれらを配信します。

呼び出し側 MCA は、送信側、サーバー、または要求側チャネルと関連している場合があります。 応答側 MCA は、どのタイプのメッセージ・チャネルとも関連できます。

IBM MQ は、接続の両側で次のチャネル・タイプの組み合わせをサポートしています。

呼び出し側		メッセージ・フローの方向	応答側	
チャネル・タイプ	**リスナーが必要か**		**リスナーが必要か**	**チャネル・タイプ**
送信側	いいえ	呼び出し側から応答側	はい	受信側
サーバー	いいえ	呼び出し側から応答側	はい	受信側
サーバー	いいえ	呼び出し側から応答側	はい	リクエスター
リクエスター	いいえ	応答側から呼び出し側	はい	サーバー
リクエスター	はい	応答側から呼び出し側	はい	送信側

MQI チャネル

MQI チャネルは次のタイプの 1 つです。

MQI チャネル・タイプ	説明
サーバー接続	サーバー接続チャネルは、双方向の MQI チャネルで、IBM MQ クライアントを IBM MQ サーバーに接続するために使用されます。 サーバー接続チャネルとは、チャネルのサーバー側です。
クライアント接続	クライアント接続チャネルは、双方向の MQI チャネルで、IBM MQ クライアントを IBM MQ サーバーに接続するために使用されます。 IBM MQ Explorer は、リモート・キュー・マネージャーへの接続に、クライアント接続も使用します。 クライアント接続チャネルとは、チャネルのクライアント側です。 クライアント接続チャネルを作成すると、 キュー・マネージャーをホストするコンピューターにファイルが作成されます。 その後、クライアント接続ファイルを IBM MQ クライアント・コンピューターにコピーする必要があります。

Multi マルチスレッド・サポート - パイプライン

オプションで、メッセージ・チャネル・エージェント (MCA) により、複数のスレッドを使用してメッセージを転送できます。 このプロセスのことをパイプライン といい、このプロセスを使用すると、MCA によるメッセージ転送の効率が上がり、待ち状態が少なくなり、チャネルのパフォーマンスが向上します。 MCA 当たり最大 2 つのスレッドに限定されています。

パイプラインを制御するには、qm.ini ファイル中で *PipeLineLength* パラメーターを使用します。 このパラメーターは、 Channels スタンザに追加されます。

注 : パイプラインは TCP/IP チャネルの場合だけ有効です。

パイプラインを使用する場合は、*PipeLineLength* が 1 より大きくなるようにチャネルの両側のキュー・マネージャーを構成する必要があります。

チャネル出口に関する考慮事項

次の理由で、パイプラインによって一部の出口プログラムが失敗します。

- 出口が逐次に呼び出されない。
- 出口が別のスレッドから代替呼び出しされる。

パイプラインを使用する場合は、その前に以下の点について出口プログラムの設計を確認してください。

- 出口はすべての実行段階で再入可能でなければならない。
- MQI 呼び出しを使用する場合は、別々のスレッドから出口が呼び出されるときは同一の MQI ハンドルを使用できないことを念頭に置かなければならない。

あるメッセージ出口が、キューをオープンし、それ以降のすべての出口呼び出しでこのハンドルを使用して MQPUT 呼び出しを行うとします。 この出口は別のスレッドから呼び出されるので、パイプライン・モードでは失敗します。 失敗しないようにするには、スレッドごとにキュー・ハンドルを保持し、出口が呼び出されるたびにスレッド ID を検査してください。

通信

IBM MQ MQI clients は、サーバーとの通信に MQI チャネルを使用します。

チャネル定義は、IBM MQ MQI client とサーバーの接続の両端で作成する必要があります。 チャネル定義の作成方法については、MQI チャネルの定義で説明されています。

次のテーブルに使用可能な伝送プロトコルを示します。

表 *1. MQI* チャネルの伝送プロトコル

クライアント・プラットフォーム	LU 6.2	TCP/IP	NetBIOS	SPX
IBM i IBM i		はい		
Linux AIX AIX and Linux システム	はい [1]	はい		
Windows Windows	はい	はい	はい	はい

注 :

1. Linux 以下のプラットフォーム上では、LU.2 はサポートされていません。
   - Linux (POWER プラットフォーム)
   - Linux (x86-64 プラットフォーム)
   - Linux (zSeries s390x プラットフォーム)

伝送プロトコル - IBM MQ MQI client とサーバーのプラットフォームの組み合わせに、これらの伝送プロトコルを使用して実現できる IBM MQ MQI client およびサーバーのプラットフォームの組み合わせが示されています。

IBM MQ MQI client 上の IBM MQ アプリケーションは、キュー・マネージャーがローカルの場合と同じ方法ですべての MQI 呼び出しを使用できます。 **MQCONN** または **MQCONNX** は、選択したキュー・マネージャーに IBM MQ アプリケーションを関連付け、接続ハンドル を作成します。 接続ハンドルを使用するその他の呼び出しは、その後、接続されたキュー・マネージャーに処理されます。 IBM MQ MQI client 通信は、クライアントとサーバーの間にアクティブな接続を必要とします。対照的に、キュー・マネージャー間の通信は、接続にも時間にも依存しません。

伝送プロトコルはチャネル定義を使用することで指定され、アプリケーションに影響しません。 例えば、Windows アプリケーションは、TCP/IP を介してあるキュー・マネージャーと接続し、NetBIOS を介して別のキュー・マネージャーと接続することができます。

パフォーマンス上の考慮事項

使用する伝送プロトコルが、IBM MQ クライアント/サーバー・システムのパフォーマンスに影響することがあります。 伝送速度が遅い特定の状況では、 IBM MQ チャネル圧縮を使用できます。

IBM MQ オブジェクトの命名

IBM MQ オブジェクトに適用される命名規則は、オブジェクトごとに異なります。 また、IBM MQ で使用するマシンの名前およびユーザー ID は、いくつかの命名規則に従います。

キュー・マネージャーの各インスタンスは、その名前で見分けられます。 この名前は、相互接続されたキュー・マネージャーのネットワーク内で固有である必要があります。 固有になっていると、あるキュー・マネージャーは、所定のメッセージを送るべきターゲットのキュー・マネージャーを明確に識別することができます。

他のタイプのオブジェクトの場合、各オブジェクトにはそれぞれ関連付けられた名前があり、各オブジェクトはその名前で参照できます。 これらの名前は、1 つのキュー・マネージャーおよびオブジェクト・タイプ内において固有のものである必要があります。 例えば、同じ名前のキューとプロセスを持つことはできますが、同じ名前の 2 つのキューを持つことはできません。

IBM MQ では、名前には最大 48 文字まで使用することができます。ただし、チャネル は例外で、最大 20 文字まで使用できます。 IBM MQ オブジェクトの命名規則の詳細については、37 ページの『IBM MQ オブジェクトの命名規則』 を参照してください。

IBM MQ で使用するマシンの名前およびユーザー ID は、以下のいくつかの命名規則にも従います。

- マシン名にスペースが含まれていないことを確認します。 IBM MQ は、スペースが含まれているマシン名をサポートしていません。 名前にスペースが含まれているマシンに IBM MQ をインストールした場合は、キュー・マネージャーを作成できなくなります。
- IBM MQ 権限のためのユーザー ID およびグループの名前は、20 文字以内にする必要があります (スペースは使用できません)。
- Windows IBM MQ for Windows サーバーでは、@ 文字を含むユーザー ID (例えば abc@d) の下で IBM MQ MQI client が実行されている場合、そのクライアントの接続はサポートされません。

関連概念

40 ページの『IBM MQ ファイル名』
IBM MQ のキュー・マネージャー、キュー、プロセス定義、名前リスト、チャネル、クライアント接続チャネル、リスナー、サービス、および認証情報オブジェクトは、それぞれファイルで表されます。 これらのオブジェクト名は必ずしも有効なファイル名ではないので、キュー・マネージャーは、必要に応じてそのオブジェクト名を有効なファイル名に変換します。

関連資料

37 ページの『IBM MQ オブジェクトの命名規則』
IBM MQ オブジェクト名には最大長があり、大/小文字が区別されます。 すべてのオブジェクト・タイプにすべての文字がサポートされているわけではなく、多くのオブジェクトには名前の固有性に関する規則があります。

IBM MQ オブジェクトの命名規則

IBM MQ オブジェクト名には最大長があり、大/小文字が区別されます。 すべてのオブジェクト・タイプにすべての文字がサポートされているわけではなく、多くのオブジェクトには名前の固有性に関する規則があります。

IBM MQ オブジェクトには多種多様なタイプがあり、オブジェクトはタイプごとに別個のオブジェクト名前空間に存在するので、タイプの異なる複数オブジェクトが同じ名前を持つことができます。例えば、ローカル・キューと送信側チャネルとが同じ名前であってもかまいません。 しかし、あるオブジェクトが同じ名前空間にある別のオブジェクトと同じ名前を持つことはできません。例えば、ローカル・キューはモデル・キューと同じ名前を持つことができず、送信側チャネルは受信側チャネルと同じ名前を持つことができません。

以下の IBM MQ オブジェクトは、別個のオブジェクト名前空間に存在します。

- 認証情報
- チャネル
- クライアント・チャネル

- リスナー
- 名前リスト
- プロセス
- キュー
- サービス
- ストレージ・クラス
- サブスクリプション
- トピック

オブジェクト名の文字長

一般に、IBM MQ オブジェクト名の最大長は 48 文字です。 この規則は、次のオブジェクトに当てはまります。

- 認証情報
- クラスター
- リスナー
- 名前リスト
- プロセス定義
- キュー
- キュー・マネージャー
- サービス
- サブスクリプション
- トピック

以下の制限があります。

1. z/OS z/OS システムでは、キュー・マネージャー名には、最大 4 文字の大文字および数字のみ使用可能です。
2. チャネル・オブジェクト名とクライアント接続チャネル名の最大長は 20 文字です。 チャネルについて詳しくは、チャネルの定義を参照してください。
3. トピック・ストリングに使用できるのは、最大 10240 バイトです。 すべての IBM MQ オブジェクト名は大/小文字を区別します。
4. サブスクリプション名には、最大 10240 バイトのサイズにすることができ、スペースを含めることができます。
5. ストレージ・クラス名の最大長は 8 文字です。
6. CF 構造名の最大長は 12 文字です。

オブジェクト名の文字

IBM MQ オブジェクト名に有効な文字は、次のとおりです。

文字	制約事項
大文字の A から Z	• なし

文字	制約事項
小文字の a から z	• MQSC スクリプトでは、小文字を含む名前を単一引用符で囲む必要があります。 これにより、小文字が大文字に変換されなくなります。 • EBCDIC カタカナを使用するシステムは、オブジェクト名に小文字の a から z の文字を使用できません。 • z/OS z/OS システムで小文字を使用する場合、キュー・マネージャー名に小文字を含めることができないなどの制限があることがあります。 • IBM i IBM i システムで CL コマンドを使用する場合は、小文字を含む名前を単一引用符で囲む必要があります。 これにより、小文字が大文字に変換されなくなります。
数字 0 から 9	• なし
ピリオド (.)	• なし
下線 (_)	• Multi なし • z/OS 名前の前または後に下線を付加することは、IBM MQ for z/OS 操作および制御パネルで処理できないため、避けてください。
スラッシュ (/)	• Windows Windows システムでは、キュー・マネージャー名の最初の文字をスラッシュにすることはできません。 • IBM i IBM i システムで CL コマンドを使用する場合は、スラッシュを含む名前を単一引用符で囲む必要があります。 • z/OS なし
パーセント記号 (%)	• ALW なし • z/OS IBM MQ for z/OS の外部セキュリティー・マネージャーとして RACF® を使用している場合は、オブジェクト名に% を使用しないでください。 これは、RACF 総称プロファイルの使用時に名前がセキュリティー検査に含まれないためです。 • IBM i IBM i システムで CL コマンドを使用する場合は、% 記号を含む名前を単一引用符で囲む必要があります。

オブジェクト名の文字に関しては、いくつかの一般的な規則もあります。

1. 先行ブランクや組み込みブランクは使用できません。
2. 各国語文字は使用不可です。
3. フィールド長全体に長さが満たない名前は、右側にブランクが埋め込まれる場合があります。 キュー・マネージャーから返されるすべてのショート・ネームには、必ず右側にブランクが埋め込まれます。

キュー名

キューの名前には次の 2 つの部分があります。

- キュー・マネージャーの名前。
- そのキュー・マネージャーによって認識されているキューのローカル名

キュー名の各部分の長さは 48 文字です。

ローカル・キューを参照する場合は、キュー・マネージャーの名前を (ブランク文字に置き換えたり、先行ヌル文字を使用したりして) 省略できます。 しかし、IBM MQ によってプログラムに返されるすべてのキュー名には、キュー・マネージャーの名前が含まれます。

z/OS 共用キュー (キュー共用グループ内のどのキュー・マネージャーにもアクセス可能) は、同じキュー共用グループ内のどの非共用ローカル・キューとも同じ名前にすることはできません。 この制限により、アプリケーションがローカル・キューをオープンするはずが間違えて共用キューをオープンしてしまうこと (またはその逆) を防げます。 共用キューおよびキュー共用グループは IBM MQ for z/OS でのみ使用可能です。

リモート・キューを参照する場合、プログラムは完全なキュー名にキュー・マネージャーの名前を含めるか、またはリモート・キューのローカル定義が存在する必要があります。

アプリケーションがキュー名を使用する場合、その名前はローカル・キューの名前 (またはその別名) またはリモート・キューのローカル定義の名前いずれかにすることができますが、アプリケーションがキューからメッセージを受け取る必要がない場合 (キューがローカルでなければならない場合) は、アプリケーションがそのいずれであるかを認識する必要はありません。 アプリケーションがキュー・オブジェクトをオープンすると、MQOPEN 呼び出しはネーム・レゾリューション機能を実行して、それ以降の操作をどのキューに対して実行するかを判別します。 ここで大切な点は、キュー・マネージャーのネットワーク内で特定の場所に定義されている特定のキューに対する依存関係がアプリケーションに組み込まれてはいないということです。 したがって、システム管理者がネットワーク内でキューを再配置し、その定義を変更しても、そのキューを使用するアプリケーションを変更する必要はありません。

予約オブジェクト名

SYSTEM. で始まるオブジェクト名は、キュー・マネージャーで定義されるオブジェクト用に予約されています。 **Alter**、**Define**、および **Replace** コマンドを使用して、ご使用のインストール環境に合わせてこれらのオブジェクト定義を変更できます。 IBM MQ 用に定義されている名前は、キュー名にすべてリストされています。

z/OS IBM MQ for z/OS では、カップリング・ファシリティー・アプリケーション構造名 CSQSYSAPPL は予約済みです。

関連概念
AIX, Linux, and Windows でのインストール名

IBM MQ ファイル名

IBM MQ のキュー・マネージャー、キュー、プロセス定義、名前リスト、チャネル、クライアント接続チャネル、リスナー、サービス、および認証情報オブジェクトは、それぞれファイルで表されます。 これらのオブジェクト名は必ずしも有効なファイル名ではないので、キュー・マネージャーは、必要に応じてそのオブジェクト名を有効なファイル名に変換します。

キュー・マネージャー・ディレクトリーへのデフォルトのパスは、次のものから作られます。

- 接頭部。次のように、IBM MQ 構成情報に定義されています。
  - Linux AIX AIX and Linux では、デフォルトの接頭部は `/var/mqm` です。 これは、mqs.ini 構成ファイルの `DefaultPrefix` スタンザで構成されます。

– Windows Windows 32 ビット・システムでは、デフォルトの接頭部は C:¥Program Files (x86)¥IBM¥WebSphere MQ です。 Windows 64 ビット・システムでは、デフォルトの接頭部は C:¥Program Files¥IBM¥MQ です。 32 ビットおよび 64 ビットのどちらのインストール済み環境でも、データ・ディレクトリーは C:\ProgramData\IBM \MQ にインストールされます。 これは、mqs.ini 構成ファイルの DefaultPrefix スタンザで構成されます。

使用可能な場合、接頭部は「IBM MQ エクスプローラーの IBM MQ プロパティー」ページを使用して変更できます。そうでない場合は、手動で mqs.ini 構成ファイルを編集します。

- キュー・マネージャー名は、有効なディレクトリー名に変換されます。 例えば、次のとおりです。

```
queue.manager
```

このキュー・マネージャーは、次のように表されます。

```
queue!manager
```

この処理を、名前変換 と呼びます。

IBM MQ では、キュー・マネージャーに 48 文字までの名前を付けることが できます。

例えば、キュー・マネージャーに次の名前を付けることができます。

```
QUEUE.MANAGER.ACCOUNTING.SERVICES
```

しかし、各キュー・マネージャーはファイルで表されるため、ファイル名の最大長および名前に使用できる文字に制限があります。 そのため、オブジェクトを表すファイルの名前は、ファイル・システムの要件に合うように自動的に変換されます。

キュー・マネージャー名の変換の規則を以下に示します。

1. 個々の文字が次のように変換されます。
   - 開始。 先へ!
   - / が & に変換される
2. 名前が正しくない場合は次のようになります。
   a. 8 文字に切り捨てられる。
   b. 3 文字の数値接尾部が付加される。

例えば、デフォルト接頭部であるとすると、queue.manager という名前のキュー・マネージャーは次のようになります。

- Windows NTFS または FAT32 の Windows では、キュー・マネージャー名は次のようになります。

```
C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ\qmgrs\queue!manager
```

- Windows FAT の Windows では、キュー・マネージャー名は次のようになります。

```
C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ\qmgrs\queue!ma
```

- Linux AIX AIX and Linux では、キュー・マネージャー名は次のようになります。

```
/var/mqm/qmgrs/queue!manager
```

変換アルゴリズムでは、大文字小文字の区別のないファイル・システム上で、大文字小文字の違いのみの名前を区別することもできます。

オブジェクト名の変換

オブジェクト名は、必ずしも有効なファイル・システム名になっていません。 オブジェクト名を変換することが必要な場合があります。 使用される変換方法は、キュー・マネージャー名の場合とは異なります。 つまり、1 つのマシンにつきキュー・マネージャー名はわずかしかありませんが、各キュー・マネージャーごとに相当数の他のオブジェクトが存在する可能性があるためです。 キュー、プロセス定義、名前リスト、チャネル、クライアント接続チャネル、リスナー、サービス、および認証情報オブジェクトは、ファイル・システム内に表示されます。

変換処理で新しい名前が生成された場合、元のオブジェクト名との関係は簡単には分かりません。 **dspmqfls** コマンドを使用して、本当のオブジェクト名と変換されたオブジェクト名との間での変換を行うことができます。

関連資料

dspmqfls (ファイル名の表示)

関連情報

mqs.ini ファイルの AllQueueManagers スタンザ

IBM i でのオブジェクト名

キュー・マネージャーには、固有の名前を持つ関連するキュー・マネージャー・ライブラリーがあります。 IBM i 統合ファイル・システム (IFS) の要件を満たすために、キュー・マネージャー名およびオブジェクト名の変換が必要になる場合があります。

キュー・マネージャーが作成されると、IBM MQ はキュー・マネージャー・ライブラリーをそれに関連付けます。 このキュー・マネージャー・ライブラリーには、主にユーザーが定義したキュー・マネージャーの名前に基づいた 10 文字以下の固有の名前が付けられます。 キュー・マネージャーとキュー・マネージャー・ライブラリーは共に、接頭部が /QIBM/UserData/mqm であるキュー・マネージャーの名前に基づいたディレクトリーに置かれます。 キュー・マネージャー、キュー・マネージャー・ライブラリー、およびディレクトリーの例を以下に示します。

キュー・マネージャー名	ORANGE
キュー・マネージャー・ライブラリー名	QMORANGE
ディレクトリー	/QIBM/UserData/mqm/ORANGE

すべてのキュー・マネージャー名およびキュー・マネージャー・ライブラリー名は、ファイル /QIBM/UserData/mqm/mqs.ini 内のスタンザに書き込まれます。

IBM MQ IFS ディレクトリーおよびファイル

IBM i Integrated File System (IFS) は、データを保管するために IBM MQ によって広範囲に使用されます。 IFS の詳細については、「*Integrated File System Introduction*」を参照してください。

各 IBM MQ オブジェクト (例えば、チャネルやキュー・マネージャーなど) は、ファイルで表されます。 これらのオブジェクト名は必ずしも有効なファイル名ではないので、キュー・マネージャーは、必要に応じてそのオブジェクト名を有効なファイル名に変換します。

キュー・マネージャー・ディレクトリーへのパスは、次のものから作られます。

- 接頭部は、キュー・マネージャー構成ファイル qm.ini に定義されています。 デフォルトの接頭部は /QIBM/UserData/mqm です。
- リテラル - qmgrs。
- コード化されたキュー・マネージャー名。 これは、有効なディレクトリー名に変換されたキュー・マネージャー名です。 例えば、キュー・マネージャー queue/manager は、queue&manager のように表されます。

この処理のことを、名前変換と呼びます。

IFS キュー・マネージャー名の変換

IBM MQ では、キュー・マネージャーに 48 文字までの名前を付けることが できます。

例えば、キュー・マネージャーには QUEUE/MANAGER/ACCOUNTING/SERVICES という名前を付けることができます。 ライブラリーがそれぞれのキュー・マネージャーに対して作成されるのと同じ方法で、それぞれのキュー・マネージャーもファイルによって表されます。 EBCDIC のコード・ポイントには変種があるため、その名前で使用できる文字には制限があります。 そのため、オブジェクトを表す IFS ファイルの名前は、ファイル・システムの要件に合うように自動的に変換されます。

例えば、queue/manager という名前のキュー・マネージャーで、文字 / を & に変換し、デフォルト接頭部がある場合、IBM MQ for IBM i でのキュー・マネージャー名は /QIBM/UserData/mqm/qmgrs/queue&manager になります。

オブジェクト名の変換

オブジェクト名は、必ずしも有効なファイル・システム名になっていません。 そのため、オブジェクト名を変換しなければならない場合があります。 使用される変換方法は、キュー・マネージャー名の場合とは異なります。 つまり、各マシンにキュー・マネージャー名はわずかしかありませんが、各キュー・マネージャーごとに相当数の他のオブジェクトが存在する可能性があるためです。 ファイル・システムには、プロセス定義、キュー、名前リスト、のみ提示されます。 チャネルは関係しません。

変換処理で新しい名前が生成された場合、元のオブジェクト名との関係は簡単には分かりません。 DSPMQMOBJN コマンドを使用すると、IBM MQ オブジェクトの変換後の名前を表示することができます。

分散キューイングとクラスター

分散キューイングとは、あるキュー・マネージャーから別のキュー・マネージャーにメッセージを送信することです。 受信キュー・マネージャーは、同じマシン上に置くことも別のマシン上に置くこ ともでき、近くにあっても地球の裏側にあってもメッセージを受信できます。 このプログラムは、ローカル・キュー・マネージャーと同じプラットフォーム、または IBM MQ がサポートするいずれかのプラットフォームで実行することができます。 分散キューイング環境内のすべての接続を手動で定義することもできますし、クラスターを作成して IBM MQ で接続詳細の多くが定義されるようにすることもできます。

分散キュー

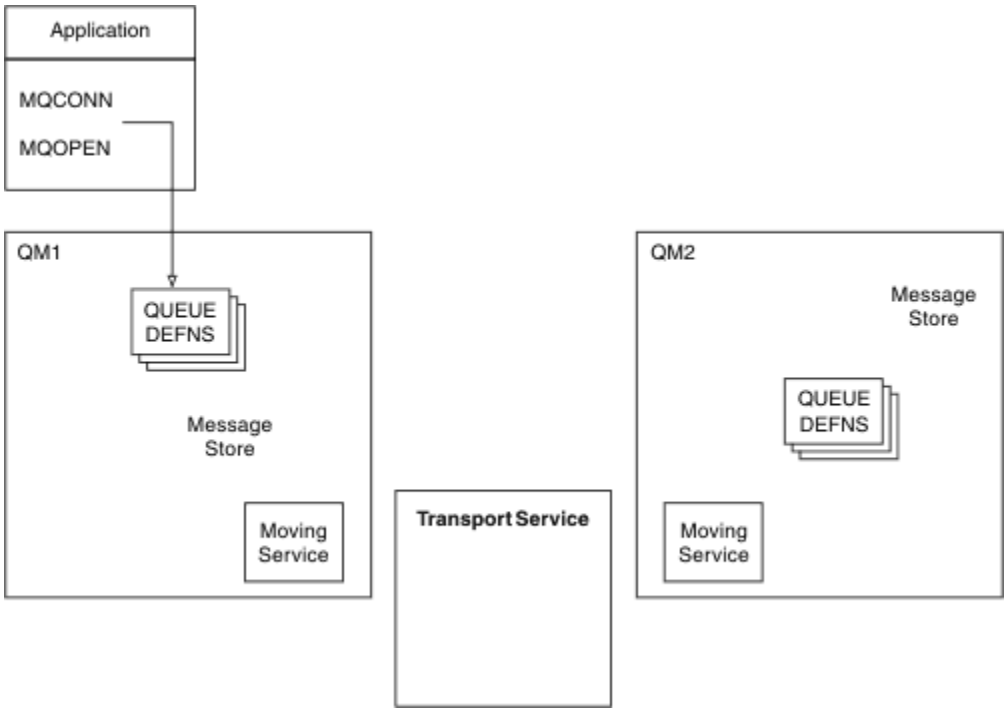


図 4. 分散キューイングのコンポーネントの概要

図の説明:

- アプリケーションは、MQCONN 呼び出しを使用してキュー・マネージャーに接続します。 次に、アプリケーションは、メッセージをキューに書き込めるように MQOPEN 呼び出しを使用してキューをオープンします。
- キュー・マネージャーごとに、所有する各キューの定義を持っています。 ローカル・キュー (このキュー・マネージャーによってホストされるキュー) の定義と、リモート・キュー (他のキュー・マネージャーによってホストされるキュー) の定義を持つことができます。
- リモート・キュー宛のメッセージは、ローカル・キュー・マネージャーが伝送キュー 上に保持し、伝送キューは、それらのメッセージをリモート・キュー・マネージャーに転送できるようになるまでメッセージ・ストアに保持します。
- キュー・マネージャーにはそれぞれ移動サービス と呼ばれる通信ソフトウェアが備えられています。 このソフトウェアを介してキュー・マネージャーは他のキュー・マネージャーと通信することができます。
- トランスポート・サービス は、キュー・マネージャーが関与しないサービスであり、次のいずれかになります (プラットフォームにより異なります)。
  - SNA APPC
  - 伝送制御プロトコル / インターネット・プロトコル (TCP/IP) (Transmission Control Protocol/Internet Protocol (TCP/IP))
  - NetBIOS
  - SPX

メッセージの送信に必要なコンポーネント

メッセージがリモート・キュー・マネージャーに送信される場合、ローカル・キュー・マネージャーには伝送キューとチャネルに関する定義が必要です。 チャネルは、2 つのキュー・マネージャーをつなぐ一方向の通信リンクです。 チャネルを使用して、リモート・キュー・マネージャーの任意の数のキュー宛てのメッセージを送ることができます。

チャネルの両側には、例えば送信側や受信側として定義される独立した定義が備えられています。 簡単なチャネルは、ローカル・キュー・マネージャーの送信側 チャネル定義とリモート・キュー・マネージャーの受信側 チャネル定義から構成されます。 これらの 2 つの定義は同じ名前でなければならず、両方の定義を合わせて 1 つのチャネルが構成されます。

メッセージの送受信を処理するソフトウェアは、メッセージ・チャネル・エージェント (MCA) と呼ばれます。 チャネルの両側に、メッセージ・チャネル・エージェント (MCA) が 1 つずつあります。

それぞれのキュー・マネージャーには 送達不能キュー (未配布メッセージ・キュー とも呼びます) が入っていなければなりません。 メッセージを宛先に送達できない場合、メッセージはこのキューに書き込まれます。

次の図は、キュー・マネージャー、伝送キュー、チャネル、および MCA の関係を示しています。

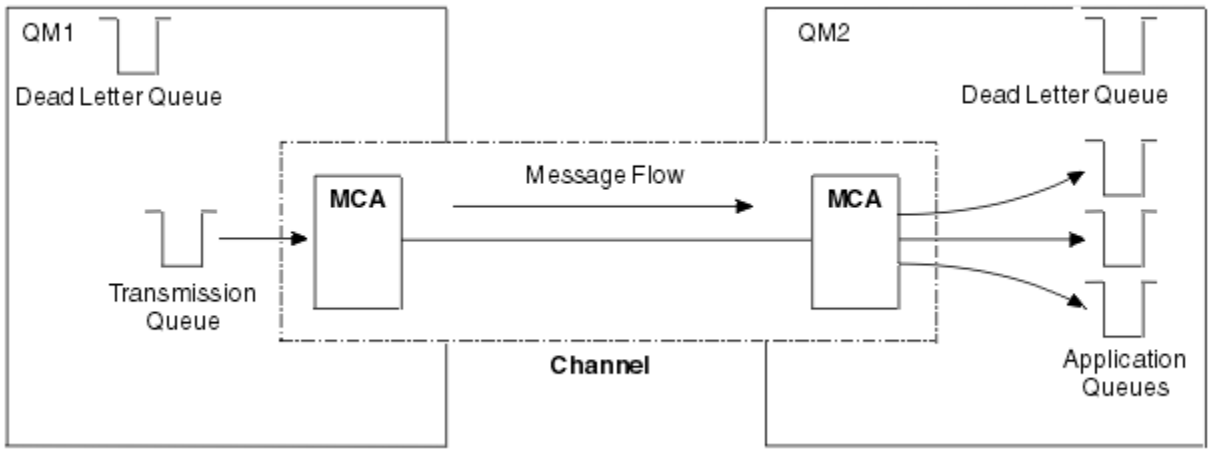


図 5. メッセージの送信

メッセージが戻るために必要なコンポーネント

リモート・キュー・マネージャーからアプリケーションにメッセージを返す必要がある場合は、以下の図に示すように、キュー・マネージャー間で反対方向に機能する別のチャネルを定義する必要があります。

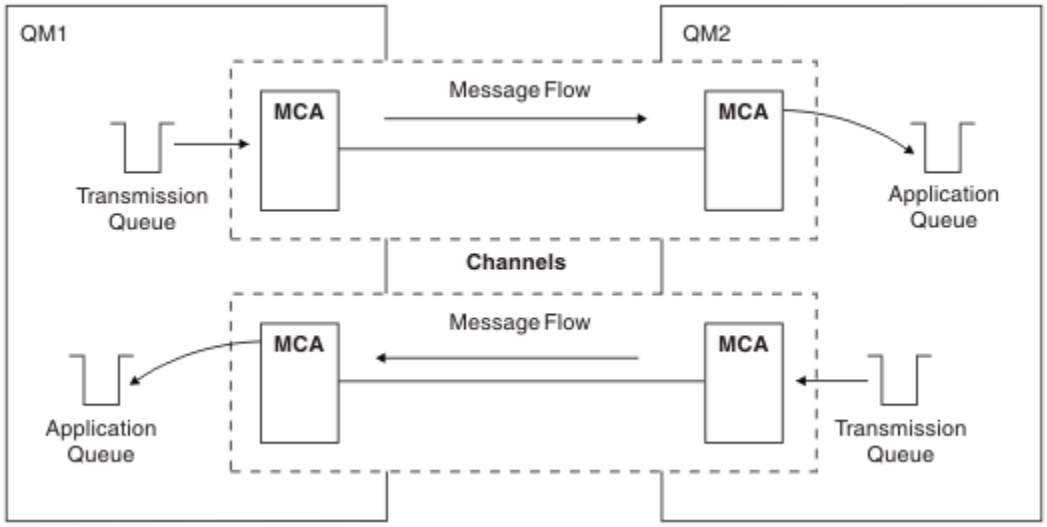


図 6. 両方向のメッセージ送信

クラスター

分散キューイング環境内のすべての接続を手動で定義する代わりに、一連のキュー・マネージャーをクラスターにグループ化することができます。 こうすると、キュー・マネージャーのホストするキューは、宛先ごとに明示的なチャネル定義、リモート・キュー定義、または伝送キューを用意しなくても、クラスター内の他のキュー・マネージャーで使用できるようになります。 クラスター内のすべてのキュー・マネージャーは単一の伝送キューを備えており、その伝送キューによってクラスター内の別のキュー・マネージャーへメッセージが伝送されます。 キュー・マネージャーごとに、1 つのクラスター受信側チャネルと 1 つのクラスター送信側チャネルのみ定義する必要があります。 追加のチャネルはクラスターによって自動的に管理されます。

IBM MQ クライアントは、他のキュー・マネージャーに接続できるのと同じように、クラスターを構成しているキュー・マネージャーに接続できます。 手動で構成した分散キューイングと同様に、任意のキュー・マネージャーのキューにメッセージを書き込むには、MQPUT 呼び出しを使用します。 ローカル・キューからメッセージを取り出すには、MQGET 呼び出しを使用します。

クラスターをサポートするプラットフォーム上にあるキュー・マネージャーは、クラスター構成にしなくても構いません。 クラスターを使用せずに分散キューイングを手動で構成することも、クラスターを使用しながら分散キューイングの構成を行うこともできます。

クラスターを使用する利点

クラスター化には、次の 2 つの主な利点があります。

- クラスター化により、チャネル、伝送キュー、およびリモート・キューの構成のために通常は多くのオブジェクト定義を必要とする IBM MQ ネットワークの管理が簡略化されます。 これは、多数のキュー・マネージャーを相互接続する必要があり、変更される可能性のある大規模ネットワークに特に当てはまります。 そのようなアーキテクチャーでは、構成や頻繁な保守が特に困難です。
- また、クラスターを使用すると、クラスター内のキューおよびキュー・マネージャーの間でメッセージ・トラフィックのワークロードを分散させることができます。 このような分散により、1 つのキューのメッセージ・ワークロードを、複数のキュー・マネージャー上にあるそのキューの複数の同等インスタンスに分散させることができます。 ワークロードの分散は、システム障害に対する回復力、およびシステム内で特にアクティブなメッセージ・フローのスケーリング・パフォーマンスを向上させる上で役立ちます。 そのような環境では、分散キューの各インスタンスに、メッセージを処理するコンシューム側アプリケーションが存在します。 詳細については、クラスターによるワークロードの管理を参照してください。

クラスター内でメッセージが経路指定される方法

クラスターは、信頼できるシステム管理者によって管理されているキュー・マネージャーのネットワークと見なすことができます。 クラスター・キューを定義すると、システム管理者は対応するリモート・キューの定義を必要に応じて他のキュー・マネージャーに対して自動的に作成します。

IBM MQ のクラスター内の各キュー・マネージャーには伝送キューが 1 つ設定されているので、伝送キューを定義する必要はありません。 この単一の伝送キューにより、クラスター内のその他のキュー・マネージャーにメッセージを送信することができます。 伝送キューを 1 つしか使用できないという制限はありません。 キュー・マネージャーは複数のクラスターを使用して、クラスター内の各キュー・マネージャーにメッセージを別々に送信できます。 通常、1 つのキュー・マネージャーは 1 つのクラスター伝送キューを使用します。 キュー・マネージャー属性 DEFCLXQ を変更して、キュー・マネージャーがクラスター内のキュー・マネージャーごとに異なるクラスター伝送キューを使用するように設定することもできます。 クラスター伝送キューを手動で定義することもできます。

クラスターを構成するキュー・マネージャーはすべて、上述のような方法で処理を実行します。 各キュー・マネージャーは、各キュー・マネージャー自体に関する情報とホストしているキューに関する情報を送信し、同じクラスター内にあるその他のキュー・マネージャーに関する情報を受信します。

キュー・マネージャーが使用できなくなった場合に情報の損失を防ぐには、クラスター内の 2 つのキュー・マネージャーをフル・リポジトリー として動作するように指定します。 これらのキュー・マネージャーは、クラスター内のすべてのキュー・マネージャーとキューに関するすべての情報を保管します。 クラスター内の他のすべてのキュー・マネージャーは、これらのキュー・マネージャーに関する情報と、メッセージの交換に使用するキューに関する情報のみを保管します。 このようなキュー・マネージャーは、部分リポジトリー として知られています。 詳しくは、57 ページの『クラスター・リポジトリー』 を参照してください。

クラスターの一部になるために、キュー・マネージャーには、クラスター送信側チャネルおよびクラスター受信側チャネルの 2 つのチャネルが必要です。

- クラスター送信側チャネルとは、送信側チャネルに似た通信チャネルです。 キュー・マネージャーに手動で 1 つのクラスター送信側チャネルを作成し、既にクラスターのメンバーである完全リポジトリーに接続する必要があります。
- クラスター受信側チャネルとは、受信側チャネルに似た通信チャネルです。 クラスター受信側チャネルを 1 つ手動で作成する必要があります。 このチャネルはキュー・マネージャーの仕組みとして機能し、クラスター通信を受信します。

このキュー・マネージャーとクラスターの他のメンバーとの通信に必要となる他のすべてのチャネルは自動的に作成されます。

以下の図は、CLUSTER というクラスターのコンポーネントを示しています。

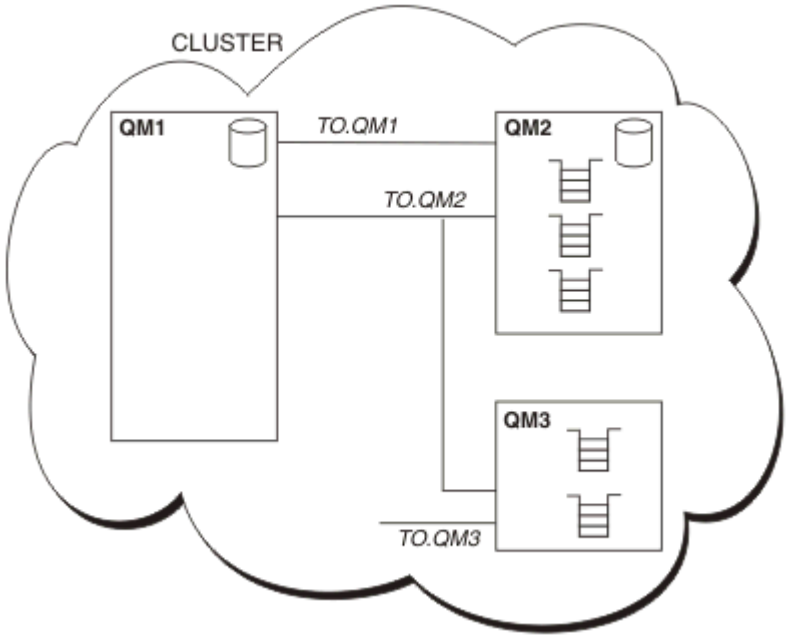


図 7. 複数のキュー・マネージャーで構成されるクラスター

- CLUSTER には、QM1、QM2、および QM3 という 3 つのキュー・マネージャーが備えられています。
- QM1 と QM2 は、クラスター内にあるキュー・マネージャーとキューに関する情報の全リポジトリーのホストです。
- QM2 と QM3 は、いくつかのクラスター・キュー (つまり、このクラスター内にある別のキュー・マネージャーからアクセス可能なキュー) のホストです。
- それぞれのキュー・マネージャーは、メッセージを受信する TO.qmgr というクラスター受信側チャネルを 1 つずつ備えています。
- また、それぞれのキュー・マネージャーは、リポジトリー・キュー・マネージャーの 1 つに情報を送信するクラスター送信側チャネルも 1 つずつ備えています。
- QM1 と QM3 は、QM2 のリポジトリーへ送信し、QM2 は QM1 のリポジトリーへ送信します。

分散キューイング・コンポーネント

分散キューイングのコンポーネントは、メッセージ・チャネル、メッセージ・チャネル・エージェント、伝送キュー、チャネルのイニシエーターとリスナー、およびチャネル出口プログラムです。 メッセージ・チャネルのそれぞれの側の定義は、複数のタイプのいずれか 1 つになります。

メッセージ・チャネルは、あるキュー・マネージャーから別のキュー・マネージャーへメッセージを送るチャネルです。 メッセージ・チャネルと MQI チャネルを混同しないようにしてください。 MQI チャネルには、サーバー接続 (SVRCONN) とクライアント接続 (CLNTCONN) の 2 つのチャネルがあります。 詳しくは、チャネルを参照してください。

メッセージ・チャネルのそれぞれの側の定義は、次のタイプのいずれかです。

- 送信側 (SDR)
- 受信側 (RCVR)
- サーバー (SVR)
- 要求側 (RQSTR)
- クラスター送信側 (CLUSSDR)
- クラスター受信側 (CLUSRCVR)

メッセージ・チャネルは、一方の側で定義されたタイプと、他方の側の互換性のあるタイプを使用して定義されます。 可能な組み合わせは以下のとおりです。

- 送信側と受信側
- 要求側とサーバー
- 要求側と送信側 (コールバック)
- サーバーと受信側
- クラスター送信側とクラスター受信側

送受信チャネル作成の詳しい手順は、チャネルの定義に記載されています。 送受信チャネルのセットアップに必要なパラメーターの例は、ご使用のプラットフォーム用の構成情報の例を参照してください。 チャネルの定義 (タイプを問わず) に必要なパラメーターについては、DEFINE CHANNEL を参照してください。

送信側 - 受信側チャネル

あるシステムで送信側がチャネルを開始すると、他のシステムにメッセージを送信できるようになります。 送信側は、チャネルのもう一方の側の受信側に開始を要求します。 送信側は伝送キューから受信側にメッセージを送信します。 受信側はメッセージを宛先キューに書き込みます。 これは、48 ページの図 8 に示されています。

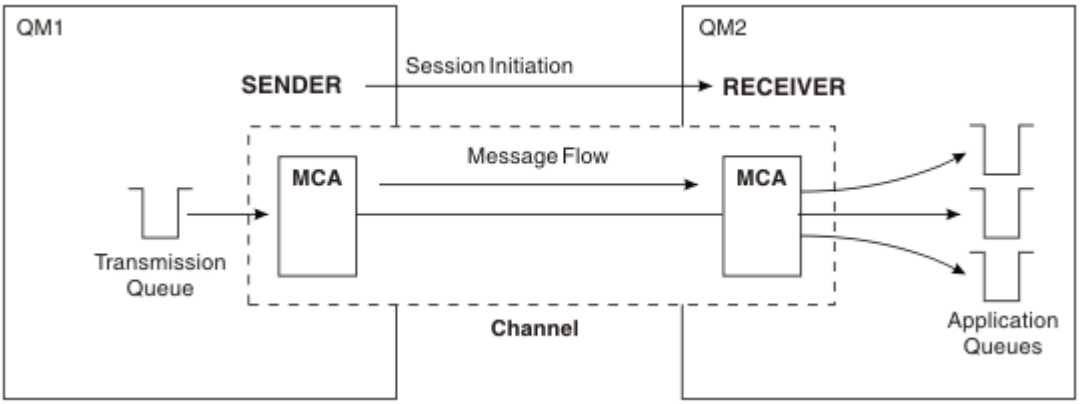


図 *8.* 送信側チャネルと受信側チャネル

要求側 - サーバー・チャネル

あるシステムの要求側がチャネルを開始すると別のシステムからのメッセージを受信できるようになります。 要求側は、チャネルのもう一方の側のサーバーに開始を要求します。 サーバーは、そのチャネル定義に定義された伝送キューから要求側にメッセージを送信します。

サーバー・チャネルは、通信を開始して要求側にメッセージを送信することもできます。 これは完全修飾サーバー、すなわちチャネル定義で指定されたパートナーの接続名を持つサーバー・チャネルにのみ適用されます。 要求側が完全修飾サーバーを開始することができますが、完全修飾サーバーが要求側との通信を開始することもできます。

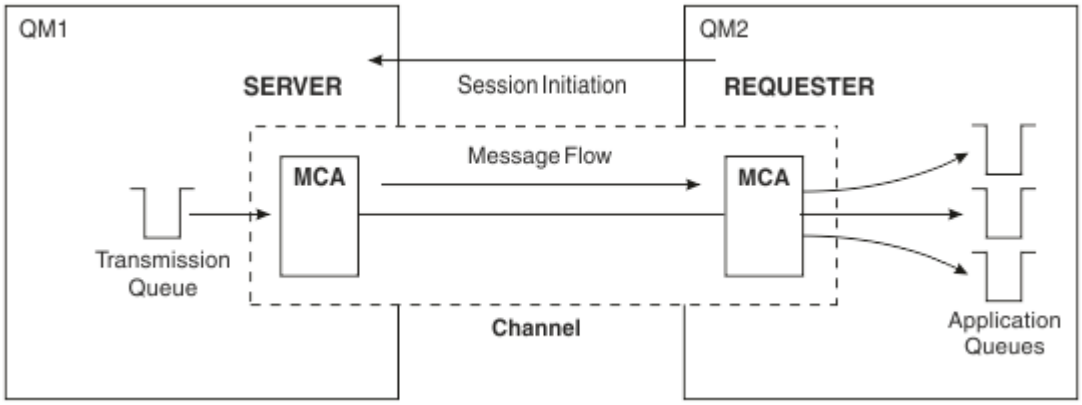


図 *9.* 要求側チャネルとサーバー・チャネル

要求側 - 送信側チャネル

要求側はチャネルを開始し、送信側は呼び出しを終了します。 次に、送信側はチャネル定義に入っている情報によって通信を再開します (コールバック と呼ばれます)。 その後、伝送キューから要求側にメッセージを送ります。

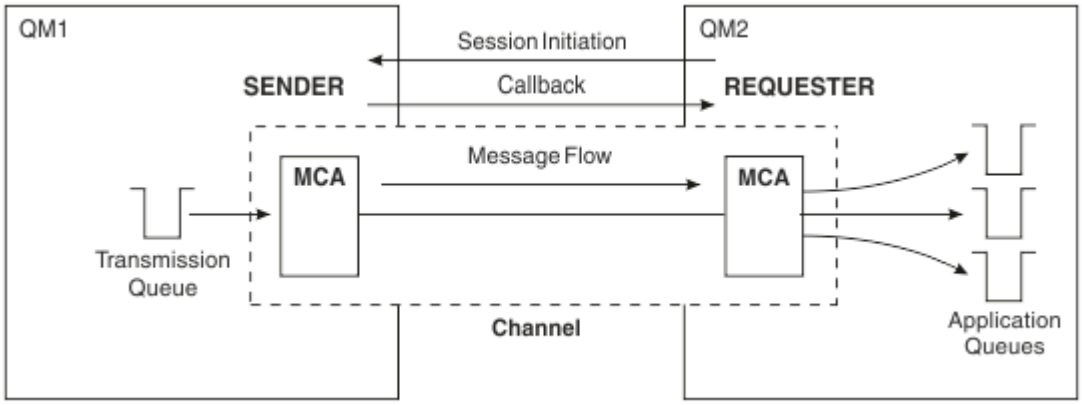


図 *10.* 要求側チャネルと送信側チャネル

サーバー・チャネルと受信側チャネル

これは、送信側チャネルと受信側チャネルに似ていますが、完全修飾 サーバー、すなわちチャネル定義で指定されたパートナーの接続名をもつサーバー・チャネルにだけ適用されます。 チャネルは、リンクのサーバー側で開始する必要があります。 次の図は、48 ページの図 8 に似ています。

クラスター送信側チャネル

クラスター内では、それぞれのキュー・マネージャーは全リポジトリー・キュー・マネージャーの 1 つに情報を送信するクラスター送信側チャネルを 1 つずつ備えています。 キュー・マネージャーは、クラスター送信側チャネル上の別のキュー・マネージャーへメッセージを送信することもできます。

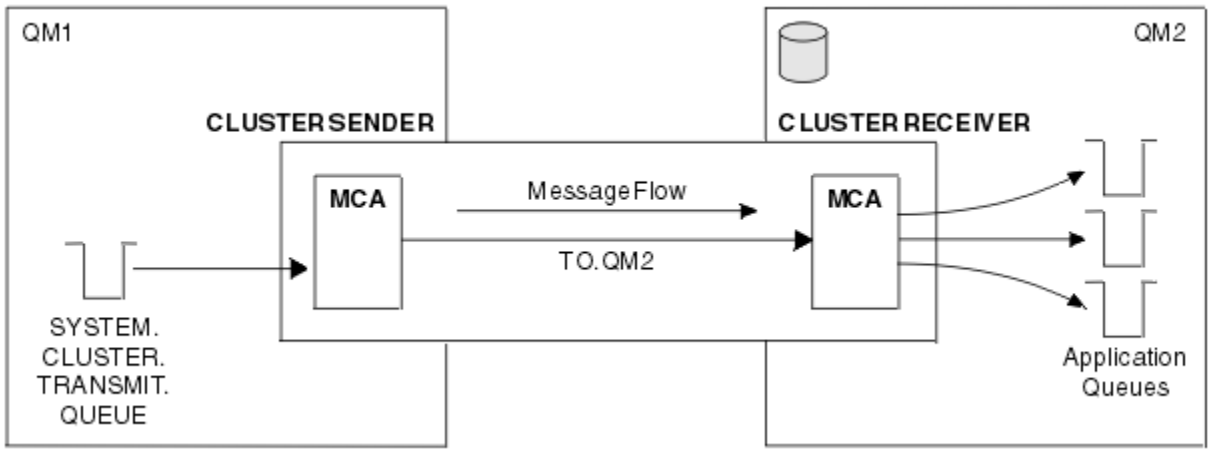


図 *11.* クラスター送信側チャネル

クラスター受信側チャネル

クラスター内の各キュー・マネージャーは、そのクラスターに関する情報とメッセージを受信するクラスター受信側チャネルを 1 つずつ備えています。 次の図は、50 ページの図 11 に似ています。

送達不能キュー

送達不能キュー (または未配布メッセージ・キュー) は、正しい宛先にメッセージを送信できない場合にそのメッセージが送信されるキューです。 一般的に、送達不能キューはキュー・マネージャーごとに 1 つ存在します。

送達不能キュー (DLQ) とは、例えば、宛先キューが存在しなかったり満杯だったりしてメッセージを宛先キューに送達できない場合に、メッセージを入れる保留キューのことで、未配布メッセージ・キュー とも言われます。 送達不能キューは、チャネルの送信側のデータ変換エラーの場合にも使用されます。 一般的には、ネットワーク内のすべてのキュー・マネージャーが、正しい宛先に送達できないメッセージを保管して後で取り出せるように、送達不能キューとして使用するローカル・キューを持っています。

キュー・マネージャー、メッセージ・チャネル・エージェント (MCA)、およびアプリケーションは、メッセージを DLQ に書き込むことができます。 DLQ 上のすべてのメッセージの先頭には、送達不能ヘッダー 構造体 MQDLH を付ける必要があります。 MQDLH 構造体の *Reason* フィールドには、メッセージが DLQ 上にある理由を識別する理由コードが入ります。

一般的には、キュー・マネージャーごとに送達不能キューを定義する必要があります。 定義しない場合、MCA はメッセージを書き込むことができず、伝送キューに残ったままとなり、チャネルは停止してしまいます。 また、高速の非持続メッセージ (Fast, nonpersistent messages を参照) を送達できず、ターゲット・システム上に送達不能キューが存在しない場合、これらのメッセージは破棄されます。

ただし、送達不能キューを使用すると、メッセージが送達される順序に影響するので、これらを使用しなくてもかまいません。

関連タスク

送達不能キューの取り扱い

未配布メッセージのトラブルシューティング

関連資料

runmqdlq (送達不能キュー・ハンドラーの実行)

リモート・キュー定義

リモート・キュー定義とは、別のキュー・マネージャーに所有されるキューの定義です。

アプリケーションはローカル・キューからしかメッセージを取り出せませんが、ローカル・キューまたはリモート・キューのどちらにもメッセージを書き込むことができます。 したがって、キュー・マネージャーには、ローカル・キューごとの定義だけでなくリモート・キュー定義 も備えていることがあります。 リ

モート・キュー定義の利点は、リモート・キューやリモート・キュー・マネージャーの名前、または伝送キューの名前を指定しなくても、アプリケーションがリモート・キューにメッセージを書き込めることです。 リモート・キュー定義を使用すると、ロケーションに依存しなくて済みます。

リモート・キュー定義は、他にも使用できますが、それについては後述します。

リモート・キュー・マネージャーへのアクセス方法

それぞれの発信元キュー・マネージャーとターゲット・キュー・マネージャーをつなぐチャネルは常に 1 つとは限りません。 2 つをつなぐいくつかの別の方法として、マルチ・ホップ、チャネルの共用、個別のチャネルおよびクラスタリングの使用などがあります。

マルチ・ホップ

ソース・キュー・マネージャーとターゲット・キュー・マネージャーをつなぐ直接の通信リンクがない場合、ターゲット・キュー・マネージャーへの経路の途中で 1 つまたは複数の中間キュー・マネージャー を介して伝送することができます。 これは、マルチ・ホップ と呼ばれます。

中間キュー・マネージャーでは、すべてのキュー・マネージャー同士、および伝送キュー同士をつなぐチャネルを定義する必要があります。 これについては、51 ページの図 12 に示してあります。

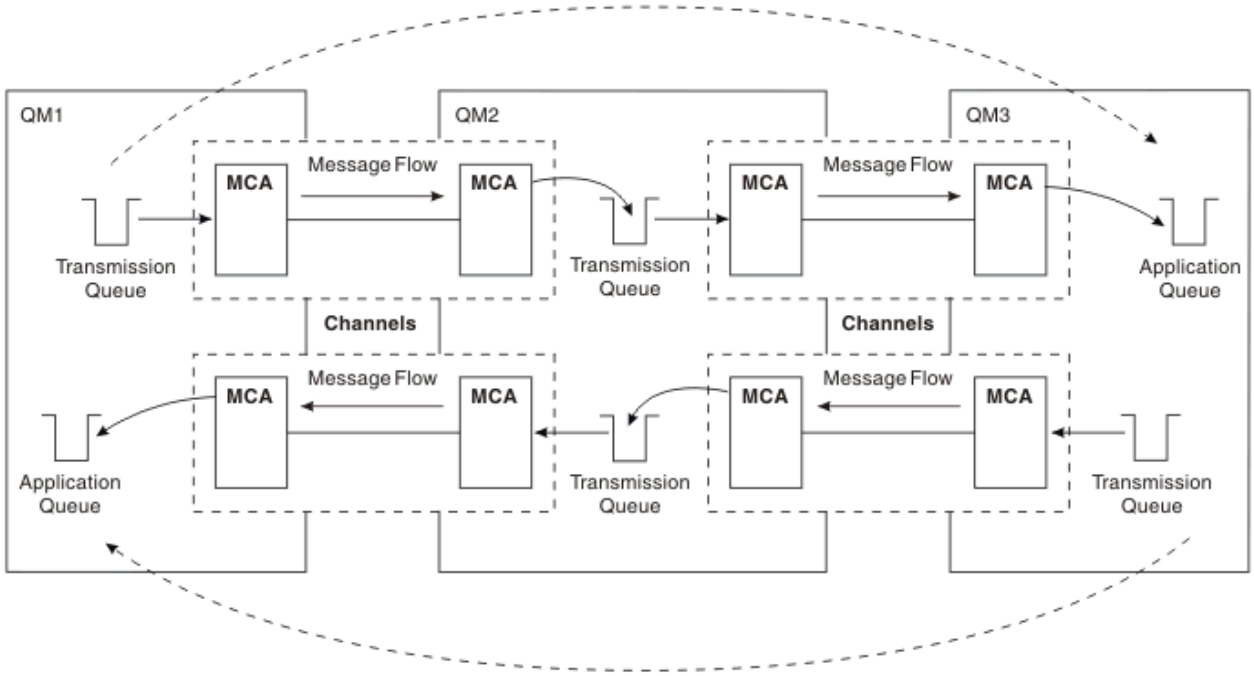


図 *12.* 中間キュー・マネージャーを介した伝送

チャネルの共用

アプリケーションを設計する場合は、キュー名と共にリモート・キュー・マネージャー名をアプリケーションに指定させるか、各リモート・キューごとにリモート・キュー定義 を作成するかを選択できます。 この定義には、リモート・キュー・マネージャー名、キュー名、および伝送キューの名前が入っています。 どちらの方法を使用しても、同じリモート・ロケーションのキューをアドレッシングするすべてのアプリケーションからのすべてのメッセージは、同じ伝送キューを介して送られます。 これについては、52 ページの図 13 に示してあります。

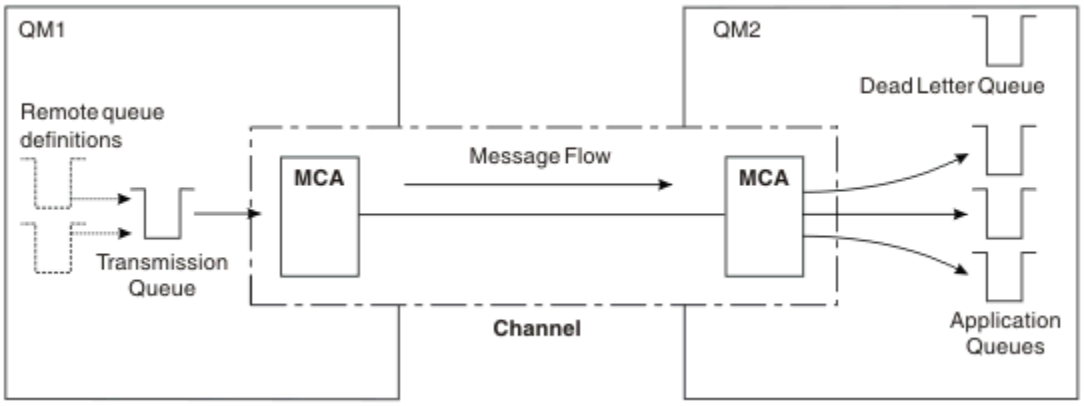


図 *13.* 伝送キューの共用

52 ページの図 13 は、複数のアプリケーションから同じチャネルを使用できる複数のリモート・キューへのメッセージの伝達を表しています。

個別のチャネルの使用

2 つのキュー・マネージャー間で異なるタイプのメッセージを送信する場合、その 2 つのキュー・マネージャー間に複数のチャネルを定義することができます。 セキュリティーのため、あるいは配送速度を犠牲にしてもメッセージ・トラフィックを一括させるために、代替チャネルが必要な場合があります。

2 番目のチャネルをセットアップするには、別のチャネルと伝送キューを定義し、ロケーションと伝送キュー名を指定するリモート・キュー定義を作成する必要があります。 これらを定義した後、アプリケーションではどちらのチャネルも使用できますが、メッセージは依然として同じターゲット・キューに送達されます。 これについては、52 ページの図 14 に示してあります。

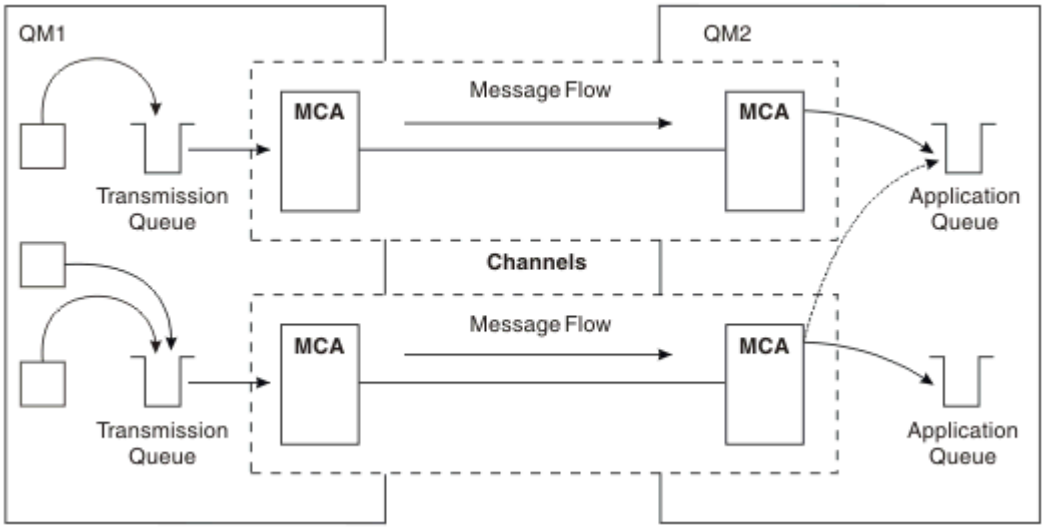


図 *14.* 複数チャネルの使用方法

伝送キューを指定するのにリモート・キュー定義を使用する場合、アプリケーションではロケーション (つまり、ターゲット・キュー・マネージャー) を**指定してはなりません**。 指定した場合、キュー・マネージャーはリモート・キュー定義を使用しません。 リモート・キュー定義を使用すると、ロケーションに依存しなくて済みます。 アプリケーションは、このキューのロケーションを知らなくても、メッセージを論理 キューに書き込むことができます。したがって、アプリケーションを変更しなくても、物理 キューを変更することができます。

クラスターの使用

クラスター内のすべてのキュー・マネージャーは、クラスター受信側チャネルを定義します。 別のキュー・マネージャーは、そのキュー・マネージャーへメッセージを送信するときに自動的にクラスター送信側チャネルを定義します。 例えば、あるキューのインスタンスがクラスター内に 2 つ以上存在する場合、そのキューのホストとなるどのキュー・マネージャーに対してもクラスター送信側チャネルを定義できます。 IBM MQ では、メッセージの経路を指定するのに使えるキュー・マネージャーを選択するため、ラウンドロビン・ルーチンを使用する作業負荷管理アルゴリズムが使用されます。 詳細については、クラスターを参照してください。

アドレッシング情報

アプリケーションがリモート・キュー・マネージャーに定義されているメッセージを書き込むとき、それらを伝送キューに入れる前に、ローカル・キュー・マネージャーは伝送ヘッダーを追加します。 このヘッダーには、宛先キューとキュー・マネージャーの名前、すなわちアドレッシング情報 が収められています。

単一キュー・マネージャー環境では、宛先キューのアドレスは、メッセージを書き込むキューをアプリケーションがオープンしたときに設定されます。 宛先キューは同じキュー・マネージャー上にあるため、アドレッシングのための情報は必要ありません。

分散キューイング環境では、キュー・マネージャーは、宛先キューの名前だけでなくそのキューのロケーション (すなわち、キュー・マネージャー名)、およびそのリモート・ロケーションへの経路 (すなわち、伝送キュー) を知る必要があります。 このアドレッシング情報は、伝送ヘッダーに含まれています。 受信チャネルは、伝送ヘッダーを取り外して、その中の情報を使用して宛先キューの場所を見つけます。

リモート・キュー定義を使用すると、宛先キュー・マネージャーの名前をアプリケーションが指定する必要がなくなります。 この定義は、リモート・キューの名前、メッセージが送られるリモート・キュー・マネージャーの名前、およびメッセージの移送に使用される伝送キューの名前を指定します。

別名について

別名を使用すれば、メッセージ・サービスの質が向上します。 キュー・マネージャーの別名を使用すると、システム管理者はアプリケーションを変更しないでターゲット・キュー・マネージャーの名前を変更することができます。 また、システム管理者は、ターゲット・キュー・マネージャーへの経路を変更したり、多数の他のキュー・マネージャーを介する (マルチ・ホップ) 経路を設定したりできます。 応答キューに別名が付けられるため、サービスが使いやすくなります。

キュー・マネージャーの別名および応答キューの別名は、ブランクの RNAME の入ったリモート・キュー定義を使用して作成されます。 これらの定義は、実際のキューを定義するのではありません。これらの定義は、物理キュー名、キュー・マネージャー名、および伝送キューを解決するためにキュー・マネージャーが使用します。

別名定義はブランクの RNAME を備えているという特徴があります。

キュー名の解決

キュー名の解決は、キューがオープンされるごとにすべてのキュー・マネージャーにおいて生じます。 その目的は、ターゲット・キュー、ターゲット・キュー・マネージャー (ローカルの場合もあります)、およびそのキュー・マネージャーへの経路 (ヌルの場合もあります) の指定です。 解決された名前は、キュー・マネージャー名、キュー名、および伝送キュー (キュー・マネージャーがリモートの場合) の 3 つの部分から構成されます。

リモート・キュー定義がある場合、別名定義は参照されません。 アプリケーションから提供されたキュー名は、ターゲット・キューの名前、リモート・キュー・マネージャー、およびリモート・キュー定義に指定される伝送キューに解決されます。 キュー名の解決についての詳細は、キュー名の解決を参照してください。

リモート・キュー定義がなく、キュー・マネージャーが指定されているか、または名前サービスによって解決されている場合、キュー・マネージャーは、システムに提供されたキュー・マネージャー名と一致するキュー・マネージャー別名定義があるかどうかを探します。 その定義がある場合は、その中の情報を使用して、ターゲット・キュー・マネージャーの名前へのキュー・マネージャーを解決します。 キュー・マ

ネージャー別名定義は、ターゲット・キュー・マネージャーへの伝送キューを決定するために使用することもできます。

解決されたキュー名がローカル・キューではない場合、アプリケーションが伝送キューに入れる各メッセージの伝送ヘッダーには、キュー・マネージャー名とキュー名の両方が収められます。

リモート・キュー定義またはキュー・マネージャー別名定義によって変更されない限り、通常、使用される伝送キューには、解決されたキュー・マネージャーの名前が付けられます。 そのような伝送キューを定義していなくても、デフォルト伝送キューを定義していれば、そのキューが使われます。

z/OS z/OS で実行されるキュー・マネージャーの名前は 4 文字までに制限されています。

キュー・マネージャー別名定義

メッセージを書き込むためにキューをオープンするアプリケーションがキュー名、**および**キュー・マネージャー名を指定するとき、キュー・マネージャー別名定義が適用されます。

キュー・マネージャー別名の定義には、次の 3 つの使用法があります。

- メッセージを送信し、キュー・マネージャー名を再マップするとき
- メッセージを送信し、伝送キューを変更または指定するとき
- メッセージを受信し、ローカル・キュー・マネージャーがこれらのメッセージの宛先を指しているかどうか判定するとき

アウトバウンド・メッセージ - キュー・マネージャー名の再マップ

キュー・マネージャー別名定義は、MQOPEN 呼び出しに指定されるキュー・マネージャー名を再マップするために使用できます。 例えば、MQOPEN 呼び出しは、THISQ のキュー名と YOURQM のキュー・マネージャー名を指定します。 ローカル・キュー・マネージャーには、次の例のようなキュー・マネージャー別名定義があります。

```
DEFINE QREMOTE (YOURQM) RQMNAME(REALQM)
```

これは、アプリケーションがキュー・マネージャー YOURQM にメッセージを書き込むときに、使用される実際のキュー・マネージャーが REALQM であることを示しています。 ローカル・キュー・マネージャーが REALQM の場合、メッセージはローカル・キューの、キュー THISQ に書き込まれます。 ローカル・キュー・マネージャーが REALQM ではない場合、メッセージは REALQM という伝送キューにルーティングされます。 キュー・マネージャーは、伝送ヘッダーを変更して YOURQM の代わりに REALQM を指定します。

アウトバウンド・メッセージ - 伝送キューの変更および指定

55 ページの図 15 は、キュー・マネージャー QM3 でのキュー名を示す伝送ヘッダーを持ったメッセージが、キュー・マネージャー QM1 に到着するシナリオを示しています。 このシナリオでは、QM3 へは QM2 を介するマルチ・ホップを使用して到達できます。

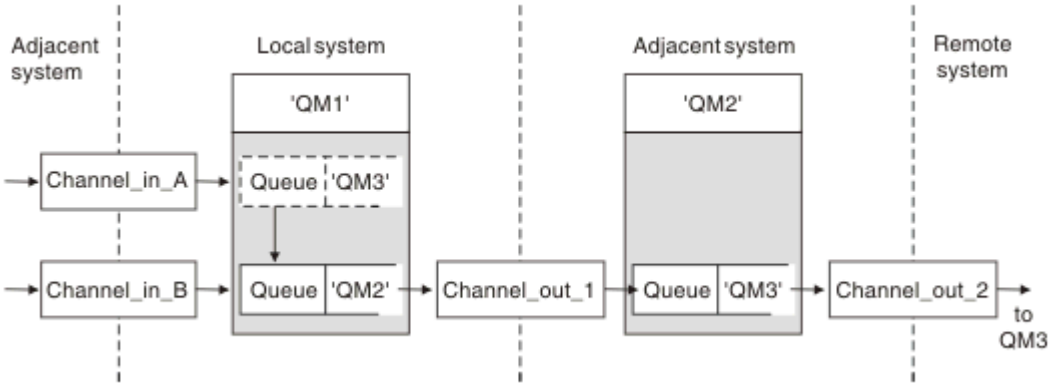


図 *15.* キュー・マネージャー別名

QM3 へのすべてのメッセージは、キュー・マネージャー別名を使用して QM1 で取り込まれます。 キュー・マネージャーの別名は QM3 であり、これには、伝送キュー QM2 を介した定義 QM3 が含まれています。 定義は、以下の例のようになります。

```
DEFINE QREMOTE (QM3) RNAME(' ') RQMNAME(QM3) XMITQ(QM2)
```

キュー・マネージャーはメッセージを伝送キュー QM2 に書き込みますが、宛先キュー・マネージャーの名前 QM3 が変わらないため、伝送キューのヘッダーを変更しません。

QM2 でのキュー名の入った伝送ヘッダーを示すメッセージが QM1 に着信すると、すべて QM2 伝送キューにも書き込まれます。 このようにして、宛先の異なるメッセージが共通の伝送キューに収集され、該当の隣接システムに送られて、さらに宛先に転送されます。

インバウンド・メッセージ - 宛先の決定

受信側 MCA は、伝送ヘッダーで参照されるキューをオープンします。 参照されるキュー・マネージャーと同じ名前のキュー・マネージャー別名定義が存在する場合、伝送ヘッダーに受信されるキュー・マネージャー名は、その定義からの RQMNAME と置き換わります。

このプロセスには、次の 2 つの使用法があります。

- 別のキュー・マネージャーにメッセージを送信する
- キュー・マネージャー名をローカル・キュー・マネージャーと同じ名前に変更する

応答先キュー別名の定義

応答先キューの別名定義は、メッセージ記述子の応答情報の代替名を指定します。 この利点は、アプリケーションを変更しないでキューまたはキュー・マネージャーの名前を変更できることです。

キュー名の解決

アプリケーションは、メッセージに応答するとき、受信したメッセージのメッセージ記述子 内のデータを使用して、応答先のキューの名前を確かめます。 送信側アプリケーションは、応答が送信される場所を示し、この情報をそのメッセージに付加します。 この概念は、アプリケーション設計時に調整する必要があります。

キュー名の解決は、メッセージがキューに書き込まれる前にアプリケーションの送信側で行われます。 キュー名の解決は、メッセージの送信先のリモート・アプリケーションと対話する前に生じます。 これは、キューがオープンされていない時点でネーム・レゾリューションが実行される唯一の状況です。

キュー・マネージャー別名を使用するキュー名の解決

通常、アプリケーションは、応答先キューを指定し、応答先キュー・マネージャー名をブランクにしておきます。 キュー・マネージャーは、書き込み時にその固有の名前を書き込みます。 この方式が唯一首尾よくいかないのは、例えば、伝送キュー QM1 を使用するデフォルト返送チャネルの代わりに、伝送キュー QM1_relief を使うチャネルなどの、代替チャネルを応答で使用したい場合です。 この状況では、伝送キューのヘッダーに指定されるキュー・マネージャー名は「実際の」キュー・マネージャー名と一致しませんが、キュー・マネージャーの別名定義を使用して再定義されます。 代替経路で応答を返すには、応答先キュー別名定義を使用して、応答先キュー・データも同様にマップする必要があります。

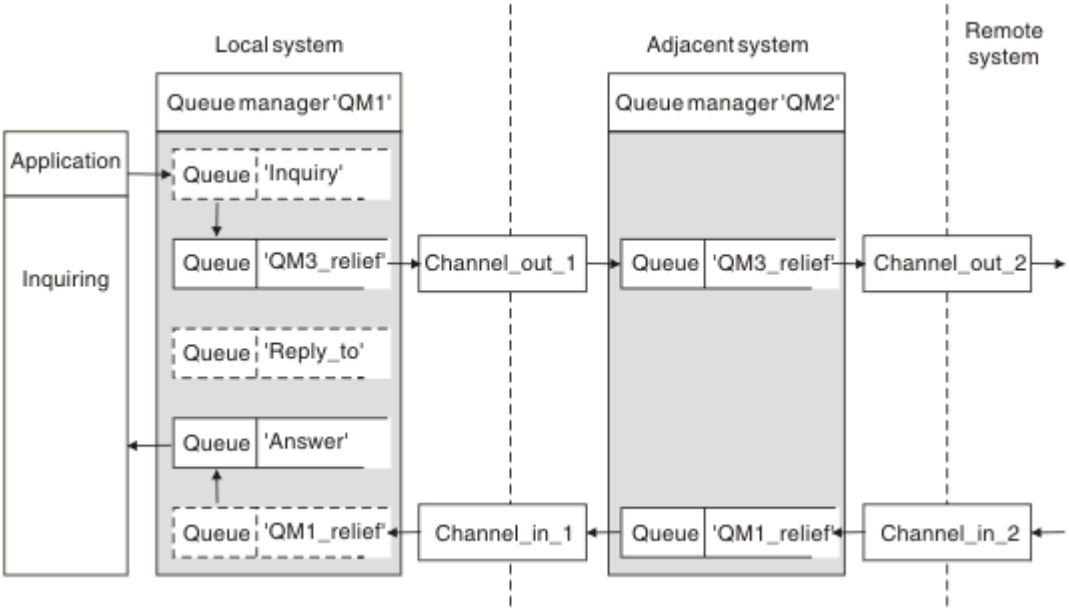


図 16. 応答ロケーションの変更に使用される応答先キューの別名

56 ページの図 16 の例の説明:

1. アプリケーションは、MQPUT 呼び出しを使用し、メッセージ記述子に以下のような情報を指定して、メッセージを書き込みます。

   ```
   ReplyToQ='Reply_to'
   ReplyToQMgr=' '
   ```

   ReplyToQMgr は、使用される応答先キュー別名用で、ブランクにしておく必要があります。
2. Answer という名前と QM1_relief というキュー・マネージャー名の付いた、応答先キュー別名定義 Reply_to を作成します。

   ```
   DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
   RQMNAME ('QM1_relief')
   ```

3. メッセージは、ReplyToQ='Answer' および ReplyToQMgr='QM1_relief' を示すメッセージ記述子付きで送信されます。
4. アプリケーションの仕様には、応答が Reply_to ではなくキュー Answer になければならないという情報が必要です。

応答に備えるには、以下のように定義をして、並列戻りチャネルを作成する必要があります。

- QM2 で、伝送キュー定義 QM1_relief

```
DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
```

- QM1 で、キュー・マネージャー別名 QM1_relief

```
DEFINE QREMOTE ('QM1_relief') RNAME() RQMNAME(QM1)
```

  このキュー・マネージャー別名は、並列戻りチャネルのチェーンを終了させ、QM1 へのメッセージを取り込みます。

実際の運用が将来の場合でも、アプリケーションが最初から別名を使用することを確認してください。 現在は、これは応答先キューへの通常のキュー別名ですが、あとでキュー・マネージャー別名に変更することができます。

応答先キュー名

応答先キューの命名には注意が必要です。 アプリケーションが応答先キュー名をメッセージに組み込むのは、応答の送り先となるキューを指定できるようにするためです。 この名前を使用して応答先キュー別名定義を作成すると、実際の応答先キュー (すなわち、ローカル・キュー定義) に同じ名前を使用できなくなります。 したがって、応答先キュー別名定義には、キュー・マネージャー名の他に新しいキュー名を入れる必要があり、また、この別のキューに応答が入ることを示す情報がアプリケーションの仕様に入っていなければなりません。

こうして、アプリケーションは、元のメッセージを書き込むときに応答先キューとして指定したキューとは別のキューからメッセージを取り出す必要があります。

クラスターのコンポーネント

クラスターは、キュー・マネージャー、クラスター・リポジトリー、クラスター・チャネル、およびクラスター・キューで構成されます。

各クラスター・コンポーネントについて詳しくは、以下のサブトピックを参照してください。

関連概念
クラスター化と分散キューイングとの比較
関連タスク
キュー・マネージャー・クラスターの構成
新規クラスターのセットアップ

クラスター・リポジトリー

リポジトリーとは、クラスターを構成する各キュー・マネージャーについての情報の集まりを指します。

リポジトリー情報には、キュー・マネージャーの名前、場所、チャネル、そのキュー・マネージャーがホストするキュー、および他の情報が含まれます。 この情報は、SYSTEM.CLUSTER.REPOSITORY.QUEUE というキューにメッセージ形式で格納されています。 このキューは、デフォルト・オブジェクトの 1 つです。 Multi マルチプラットフォーム では、IBM MQ キュー・マネージャーを作成するときに定義されます。 z/OS IBM MQ for z/OS では、キュー・マネージャーのカスタマイズの一部として定義されます。

完全リポジトリーおよび部分リポジトリー

通常は、クラスターの中の 2 つのキュー・マネージャーに完全リポジトリーが保持されます。 それ以外のすべてのキュー・マネージャーには、部分リポジトリー が保持されます。

クラスター内の各キュー・マネージャーについての 全情報をホストしているキュー・マネージャーには、完全リポジトリーが含まれます。 クラスター内の他のキュー・マネージャーには、完全リポジトリーに格納されている情報のサブセットが入っている部分リポジトリーが含まれます。

部分リポジトリーには、メッセージを交換する必要があるキュー・マネージャーに関する情報だけが格納されています。 キュー・マネージャーが必要な情報に対する更新情報を要求すると、その情報が変更されている場合には、完全リポジトリー・キュー・マネージャーは要求元のキュー・マネージャーに新しい情報を送信します。 通常、 部分リポジトリーには、キュー・マネージャーがクラスター内で処理を実行するのに 必要な情報がすべて含まれています。 その他の情報が必要になると、キュー・マネージャーは完全リポジトリーにその情報を照会し、これにより、部分リポジトリーを更新します。 キュー・マネージャーは SYSTEM.CLUSTER.COMMAND.QUEUE キューを使用して、リポジトリーに対する更新の情報を要求し、これを受信します。

クラスターのメンバーであるキュー・マネージャーをマイグレーション する場合は、部分リポジトリーをマイグレーションする前に完全リポジトリーをマイグレーションしてください。 これは、新しいリリースで導入された新しい属性は、古いリポジトリーには格納できないためです。 これらは、許容されますが、格納されません。

クラスター・キュー・マネージャー

クラスター・キュー・マネージャーは、クラスターのメンバーであるキュー・マネージャーです。

キュー・マネージャーは、複数のクラスターのメンバーにすることができます。 各クラスター・キュー・マネージャーには、そのクラスターがメンバーとなっているすべてのクラスター全体で固有の名前がなければなりません。

1 つのクラスター・キュー・マネージャーで複数のキューをホストすることができます。 その場合、そのキュー・マネージャーはそれらのキューを同じクラスター内にあるその他のキュー・マネージャーに通知します。 ただし、これを行う必要はありません。 クラスター・キュー・マネージャーは、代わりにメッセージをクラスター内でホストされるキューに送信し、そのメッセージに対する応答のうち、そこに送るよう明示的に指定された応答だけを受信できます。

z/OS IBM MQ for z/OS では、クラスター・キュー・マネージャーはキュー共有グループのメンバーになることができます。 この場合、キュー・マネージャーは同じキュー共有グループ内の他のキュー・マネージャーとキュー定義を共有します。

クラスター・キュー・マネージャーは自律型のプログラムです。 各クラスター・キュー・マネージャーに定義されたキューおよびチャネルをそのキュー・マネージャーが完全に制御することができます。 各キュー・マネージャーの定義を別のキュー・マネージャーによって変更することはできません (同じキュー共有グループ内のキュー・マネージャーであれば可能です)。 リポジトリー・キュー・マネージャーは、クラスター内の他のキュー・マネージャーでの定義を制御しません。 リポジトリー・キュー・マネージャーは、必要に応じて使用できる、すべての定義の完全なセットを保持します。 クラスターは、キュー・マネージャーのフェデレーションです。

クラスター・キュー・マネージャーの定義を作成したり変更したりすると、その情報は完全リポジトリー・キュー・マネージャーに送信されます。 クラスター内の他のリポジトリーは後で更新されます。

完全リポジトリー・キュー・マネージャー

完全リポジトリー・キュー・マネージャーとは、クラスターのリソースの完全な表現を保持するクラスター・キュー・マネージャーです。 可用性を確保するために、各クラスターに 2 つ以上の完全リポジトリー・キュー・マネージャーをセットアップしてください。 完全リポジトリー・キュー・マネージャーは、クラスター内にあるその他のキュー・マネージャーから送信された情報を受信し、そのリポジトリーを更新します。 そして、互いにメッセージを交換することにより、すべての完全リポジトリー・キュー・マネージャーでクラスターに関する情報が常に最新に保たれるようにします。

キュー・マネージャーおよびリポジトリー

すべてのクラスターには、クラスター内のキュー・マネージャー、キュー、およびチャネルに関する情報のフル・リポジトリーを保持する、少なくとも 1 つの (できれば 2 つの) キュー・マネージャーがあります。 また、クラスター内のその他のキュー・マネージャーからこの情報を更新する要求が発行された場合は、その要求も完全リポジトリーに格納されます。

他のキュー・マネージャーは、それぞれ部分リポジトリーを保持します。このリポジトリーには、通信するために必要なキューおよびキュー・マネージャーのサブセットに関する情報が含まれています。 これら

のキュー・マネージャーは、別のキューまたはキュー・マネージャーに初めてアクセスする必要が生じたときに照会を行うことによって、それ自身の部分リポジトリーを作成します。 キュー・マネージャーは、そのキューまたはキュー・マネージャーに関する新しい情報があれば通知するように要求します。

各キュー・マネージャーは、リポジトリー情報を SYSTEM.CLUSTER.REPOSITORY.QUEUE というキューにメッセージ形式で格納します。 そして、各キュー・マネージャー間でメッセージ形式のリポジトリー情報を交換するときには SYSTEM.CLUSTER.COMMAND.QUEUE というキューを使用します。

クラスターに参加する各キュー・マネージャーは、いずれかのリポジトリーに対するクラスター送信側 (CLUSSDR) チャネルを定義します。 キュー・マネージャーはクラスター内のどのキュー・マネージャーが完全リポジトリーを保有しているのかをただちに認識します。 これ以降、キュー・マネージャーはどのリポジトリーに格納されている情報でも要求することができます。 また、キュー・マネージャーが、指定されたリポジトリーに情報を送信すると、その情報は別のリポジトリーがあればそのリポジトリーにも送信されます。

完全リポジトリーをホストしているキュー・マネージャーが、そのプログラムに接続されている別のキュー・マネージャーから新しい情報を受信すると、完全リポジトリーが更新されます。 この新しい情報は別のリポジトリーにも送信されます。そのため、稼働していないリポジトリー・キュー・マネージャーがあると情報の更新に遅れが生じるという可能性も少なくなります。 どの情報も 2 回送信されるので、各リポジトリーでは重複する情報を破棄する必要があります。 各情報には順序番号が付いているので、各リポジトリーではその番号によって重複する情報を識別することができます。 すべてのリポジトリー間では、メッセージを交換することによって互いに同期をとります。

クラスター・キュー

クラスター・キューとは、クラスター・キュー・マネージャーでホストされ、同じクラスター内の別のキュー・マネージャーで使用できるキューです。

クラスター・キュー定義は、クラスター内の他のキュー・マネージャーに通知されます。 クラスター内にあるその他のキュー・マネージャーは、対応するリモート・キュー定義がなくても、クラスター・キューにメッセージを書き込むことができます。 クラスター名前リストを使用して、クラスター・キューを複数のクラスターに通知できます。

キューが通知されると、クラスター内のキュー・マネージャーはそのキューにメッセージを書き込めるようになります。 メッセージを書き込むときには、キュー・マネージャーが、フルリポジトリーで、そのキューがホストされている場所を調べる必要があります。 格納場所が分かったら、宛先情報をメッセージに追加して、クラスター伝送キューにメッセージを書き込みます。

クラスター・キューは、IBM MQ for z/OS でのキュー共用グループの メンバーによって共用されるキューにすることができます。

関連タスク

クラスター・キューの定義

Comparison between shared queues and cluster queues

This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Channel Initiator costs

In cluster queues, messages are sent by channels, so allow for channel initiator costs in addition to application costs. There are costs in the network because channels get and put messages. These costs are not present with shared queues, which therefore use less processing power than cluster queues when moving messages between queue managers in a queue sharing group.

Availability of messages

When putting to a queue, cluster queues send the message to one of the queue managers with active channels connected to your queue manager. On the remote queue manager, if applications used to process the messages are not working, the messages are not processed and wait until the applications

start. Similarly, if a queue manager is shut down, any messages on the queue manager are not made available until the queue manager restarts. These instances show lower message availability than when using shared queues.

When using shared queues, any application in the queue sharing group can get messages that are sent. If you shut down one queue manager in the queue sharing group, messages are available to the other queue managers, providing higher message availability than when using cluster queues.

Capacity

A coupling facility is more expensive than a disk; therefore the cost of storing 1,000,000 messages in a local queue is lower than having a coupling facility with enough capacity to store the same number of messages.

Sending to other queue managers

Shared-queue messages are only available within a queue sharing group. If you want to use a queue manager outside of the queue sharing group, you must use channels. You can use clustering to workload balance between multiple remote distributed queue managers.

Workload balancing

You can use clustering to give weight to which channels and queue managers get a proportion of the messages sent. For example, you can send 60% of messages to one queue manager, and 40% of messages to another queue manager. This instance does not depend on the ability of the remote queue manager to process work. The system with the first queue manager might be overloaded, and the system with the second queue manager might be idle, but most of the messages still go to the first queue manager.

With shared queues, two CICS® systems can get messages. If one system is overloaded, the other system takes over most the workload.

クラスター・チャネル

すべてのフル・リポジトリーで、手動でクラスター受信側チャネルと、クラスター内の他のすべてのフル・リポジトリーに接続するクラスター送信側チャネルのセットを手動で定義します。 部分リポジトリーを追加する場合、クラスター受信側チャネルと、いずれかのフル・リポジトリーに接続する単一のクラスター送信側チャネルを手動で定義します。 その他のクラスター送信側チャネルは、必要になった時にクラスターにより自動的に定義されます。 自動的に定義されたクラスター送信側チャネルの属性は、受信側のキュー・マネージャーの対応するクラスター受信側チャネル定義の属性を基にして設定されます。

クラスター受信側チャネル: CLUSRCVR

CLUSRCVR チャネル定義は、クラスター・キュー・マネージャーが同じクラスター内にある他のキュー・マネージャーからメッセージを受信することができるチャネルの終端を定義します。

CLUSRCVR チャネルは、各クラスター・キュー・マネージャーに 1 つ以上定義しなければなりません。 キュー・マネージャーは、CLUSRCVR チャネルを定義することにより、そのキュー・マネージャーがメッセージを受信可能であることを他のクラスター・キュー・マネージャーに示します。

CLUSRCVR チャネル定義は、他のキュー・マネージャーが対応するクラスター送信側チャネル定義を自動的に定義できるようにもします。 この記事の 61 ページの『自動定義されるクラスター送信側チャネル』 セクションを参照してください。

クラスター送信側チャネル: CLUSSDR

クラスターの各完全リポジトリー・キュー・マネージャーから、他の各完全リポジトリー・キュー・マネージャーへの CLUSSDR チャネルを手動で定義します。 完全リポジトリー間で交換されるすべての更新情

報は、このチャネルでのみ送受信されます。 これらのチャネルを手動で定義して、完全リポジトリーのネットワークを明示的に制御します。

部分リポジトリー・キュー・マネージャーをクラスター追加する場合、いずれかのフル・リポジトリーに接続する単一の CLUSSDR チャネルを手動で定義します。 どのフル・リポジトリーを選択してもあまり変わりはありません。それは、初期接続が確立されれば、キュー・マネージャーのクラスター・キュー・マネージャー・オブジェクトが、CLUSSDR チャネルを含めて、必要に応じて自動的に定義されるからです。このため、キュー・マネージャーは、フル・リポジトリーにクラスター情報を送信し、クラスター内の任意のキュー・マネージャーにメッセージを送信できます。

この記事の該当セクションで説明したように、自動定義される送信側チャネルは、クラスター受信側チャネルの構成に基づきます。 したがって、クラスター・チャネルで設定するチャネル・プロパティーは、対応する CLUSSDR チャネルとクラスター受信側チャネルで同一に設定するか、クラスター受信側チャネルのみに設定する必要があります。

前述した理由で、CLUSSDR チャネルだけは手動で定義しなければなりません。 つまり、最初に部分リポジトリーを完全リポジトリーに接続する場合か、2 つの完全リポジトリーを相互に接続する場合です。 部分リポジトリーやクラスターに含まれないキュー・マネージャーを接続する CLUSSDR チャネルを手動で構成すると、AMQ9427 や AMQ9428 などのエラー・メッセージが出される原因になります。 これは一時的状況として避けられない場合もありますが (例えば完全リポジトリーの位置を変更するとき)、手動定義はできるだけ早く削除する必要があります。

自動定義されるクラスター送信側チャネル

部分リポジトリー・キュー・マネージャーをクラスター追加する場合、通常はキュー・マネージャーで 2 つのクラスター・チャネルだけを手動で定義します。

- クラスターのフル・リポジトリー・キュー・マネージャーへのクラスター送信側 (CLUSSDR) チャネル。
- クラスター受信側 (CLUSRCVR) チャネル。

CLUSSDR チャネルを定義することにより、キュー・マネージャーはクラスターとの初期接続を確立できるようになります。 初期接続が確立された後、他の CLUSSDR チャネルは必要に応じてクラスターにより自動的に定義されます。

自動的に定義される CLUSSDR チャネルは、受信側のキュー・マネージャーの対応する CLUSRCVR チャネル定義から自分の属性を取得します。 手動で定義された CLUSSDR チャネルがある場合でも、自動定義された CLUSSDR チャネルからの属性が使用されます。 例えば、**CONNAME** パラメーターでポート番号を指定せずに CLUSRCVR チャネルを定義し、ポート番号を指定して CLUSSDR チャネルを手動で定義したとします。 手操作で定義した CLUSSDR チャネルが自動定義されたもので置き換えられると、ポート番号 (CLUSRCVR チャネルから設定された) はブランクになります。 デフォルトのポート番号が使用され、チャネルで障害が発生します。

手動で定義された CLUSSDR チャネルと、対応する CLUSRCVR チャネル定義の間に構成の違いがある場合、いくつかの違いは即時に有効になり (例えば、ワークロード・バランシングのパラメーター)、いくつかのものはチャネル再始動 (例えば、TLS 構成) したときにのみ有効になります。

混乱を避けるために、可能な限り、以下のガイドラインを順守してください。

- 手動で定義するのは、完全リポジトリーを指す CLUSSDR チャネルのみにします。
- 手動で定義された CLUSSDR チャネルがある場合、受信側のキュー・マネージャーの対応する CLUSRCVR チャネル定義と完全に一致するように構成します。

自動定義チャネルの処理も参照してください。

関連概念
自動定義チャネルの処理
クラスター伝送キューとクラスター送信側チャネルの操作
関連タスク
新規クラスターのセットアップ
クラスターにキュー・マネージャーを追加する

クラスター・トピック

クラスター・トピックは、 **cluster** 属性が定義されている管理トピックです。 クラスター・トピックに関する情報は、クラスターのすべてのメンバーにプッシュされ、ローカル・トピックと結合されて、複数のキュー・マネージャーにわたるトピック・スペースの一部を形成します。 これにより、あるトピックに対して 1 つのキュー・マネージャーでパブリッシュされたメッセージが、クラスター内の他のキュー・マネージャーのサブスクリプションに配信されます。

キュー・マネージャーにクラスター・トピックを定義すると、そのクラスター・トピック定義が完全リポジトリー・キュー・マネージャーに送信されます。 完全リポジトリーは、そのクラスター・トピック定義をクラスター内のすべてのキュー・マネージャーに伝搬し、クラスター内のあらゆるキュー・マネージャーで、同じクラスター・トピックがパブリッシャーおよびサブスクライバーに使用可能になるようにします。 クラスター・トピックを作成するキュー・マネージャーは、クラスター・トピック・ホストと呼ばれます。 クラスター・トピックはクラスター内の任意のキュー・マネージャーで使用できますが、クラスター・トピックに対する変更は、そのトピックが定義されているキュー・マネージャー (クラスター・トピック・ホスト) で行う必要があります。変更を行った時点で、その変更は完全リポジトリーを介してクラスターのすべてのメンバーに伝搬されます。

直接ルーティング またはトピック・ホスト・ルーティング を使用するためのクラスター・トピックの構成について、またクラスター・トピックの継承やワイルドカード・サブスクリプションについては、クラスター・トピックの定義を参照してください。

クラスター・トピックを表示するために使用するコマンドについては、関連情報を参照してください。

関連概念
管理トピックの操作
サブスクリプションの操作
関連資料
表示トピック
DISPLAYTPSTATUS
表示サブ

デフォルトのクラスター・オブジェクト

Multiplatforms では、キュー・マネージャーを定義するときに自動的に作成されるデフォルト・オブジェクトのセットに、デフォルトのクラスター・オブジェクトが含まれています。 

z/OS の場合、デフォルトのクラスター・オブジェクト定義は、カスタマイズ・サンプルにあります。

注 : デフォルト・チャネル定義は、他のすべてのチャネル定義と同様に、MQSC または PCF コマンドを実行することで変更できます。 SYSTEM.CLUSTER.HISTORY.QUEUE を除き、デフォルトのキュー定義は変更しないでください。

SYSTEM.CLUSTER.COMMAND.QUEUE
クラスター内のそれぞれのキュー・マネージャーには、SYSTEM.CLUSTER.COMMAND.QUEUE という、メッセージを完全リポジトリーに転送するために使用するローカル・キューがあります。 メッセージには、キュー・マネージャーに関する新しい情報や変更された情報、また他のキュー・マネージャーに関する情報の要求が格納されます。 通常、SYSTEM.CLUSTER.COMMAND.QUEUE は空です。

SYSTEM.CLUSTER.HISTORY.QUEUE
クラスター内の各キュー・マネージャーには、SYSTEM.CLUSTER.HISTORY.QUEUE という名前のローカル・キューがあります。 SYSTEM.CLUSTER.HISTORY.QUEUE は、サービス目的でクラスター状態情報の履歴を保管するために使用されます。

デフォルトのオブジェクト設定では、SYSTEM.CLUSTER.HISTORY.QUEUE は PUT (ENABLED) に設定されます。 履歴収集を抑止するには、設定を PUT (DISABLED) に変更します。

SYSTEM.CLUSTER.REPOSITORY.QUEUE
クラスター内の各キュー・マネージャーには、SYSTEM.CLUSTER.REPOSITORY.QUEUE という名前のローカル・キューがあります。 このキューはすべての完全リポジトリー情報の保管に使用します。 このキューは通常は空ではありません。

SYSTEM.CLUSTER.TRANSMIT.QUEUE
各キュー・マネージャーには、SYSTEM.CLUSTER.TRANSMIT.QUEUE というローカル・キューの定義があります。 SYSTEM.CLUSTER.TRANSMIT.QUEUE は、クラスター内のすべてのキューおよびキュー・マネージャーに対するすべてのメッセージのデフォルト伝送キューです。 キュー・マネージャー属性 DEFCLXQ を変更することにより、各クラスター送信側チャネルのデフォルト伝送キューを SYSTEM.CLUSTER.TRANSMIT. *ChannelName* に変更できます。
SYSTEM.CLUSTER.TRANSMIT.QUEUE を削除することはできません。 また、使用されるデフォルト伝送キューが SYSTEM.CLUSTER.TRANSMIT.QUEUE であるか SYSTEM.CLUSTER.TRANSMIT. *ChannelName* であるかの許可検査を定義するためにも使用されます。

SYSTEM.DEF.CLUSRCVR
それぞれのクラスターには、SYSTEM.DEF.CLUSRCVR というデフォルトの CLUSRCVR チャネル定義があります。 SYSTEM.DEF.CLUSRCVR は、クラスター内のキュー・マネージャーにクラスター受信側チャネルを作成するときに指定しないすべての属性にデフォルト値を提供するために使用されます。

SYSTEM.DEF.CLUSSDR
それぞれのクラスターには、SYSTEM.DEF.CLUSSDR というデフォルトの CLUSSDR チャネル定義があります。 SYSTEM.DEF.CLUSSDR は、クラスター内のキュー・マネージャーにクラスター送信側チャネルを作成するときに指定しないすべての属性にデフォルト値を提供するために使用されます。

関連概念
デフォルト・クラスター・オブジェクトの処理

パブリッシュ/サブスクライブ・メッセージング

パブリッシュ/サブスクライブ・メッセージングによって、情報の提供者をその情報の利用者から分離することができます。 送信側および受信側アプリケーションは、情報を送受信するために互いの情報を知っている必要はありません。

Point-to-Point IBM MQ アプリケーションが別のアプリケーションにメッセージを送信できるようにするためには、まずそのアプリケーションについて知る必要があります。 例えば、情報の送信先のキューの名前がわかっていなければなりませんし、場合によってはキュー・マネージャー名を指定する必要もあります。

IBM MQ のパブリッシュ/サブスクライブでは、ご使用のアプリケーションがターゲット・アプリケーションについて何も知る必要はありません。 送信側アプリケーションが行う必要があるのは、以下の処理だけです。

- アプリケーションが必要とする情報を含む IBM MQ メッセージを 書き込み ます。
- 情報のサブジェクトを示すトピックへのこのメッセージの割り当て。
- IBM MQ によるその情報の配布処理。

同様に、ターゲット・アプリケーションも、受け取る情報のソースについて何も知る必要はありません。

以下の図は、最も単純なパブリッシュ/サブスクライブ・システムを示しています。 パブリッシャーが 1 つ、キュー・マネージャーが 1 つ、サブスクライバーが 1 つあります。 サブスクライバーがサブスクリプションをキュー・マネージャー上に作成し、パブリッシャーがパブリケーションをキュー・マネージャーに送信します。 そのパブリケーションを、キュー・マネージャーがサブスクライバーに転送します。

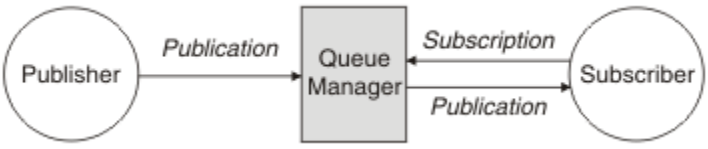


図 17. 簡単なパブリッシュ/サブスクライブの構成

標準的なパブリッシュ/サブスクライブ・システムには、さまざまなトピックに関する複数のパブリッシャーと複数のサブスクライバーがあり、多くの場合、キュー・マネージャーも複数あります。 アプリケーションは、パブリッシャーとサブスクライバーの両方にすることができます。

パブリッシュ/サブスクライブ・メッセージングと Point-to-Point のもう 1 つの重要な相違点は、Point-to-Point キューに送信されたメッセージを処理するのは、単一のコンシューム・アプリケーションのみであるという点です。 複数のサブスクライバーがインタレストを登録しているパブリッシュ/サブスクライブ・トピックにパブリッシュされたメッセージは、すべてのインタレスト・サブスクライバーによって処理されます。

パブリッシュ/サブスクライブの構成要素

パブリッシュ/サブスクライブは、サブスクライバーがパブリッシャーから情報をメッセージの形で受け取るためのメカニズムです。 パブリッシャーとサブスクライバーの間の相互作用は、IBM MQ の標準機能を使用して、キュー・マネージャーによって制御されます。

標準的なパブリッシュ/サブスクライブ・システムには、さまざまなトピックに関する複数のパブリッシャーと複数のサブスクライバーがあり、多くの場合、キュー・マネージャーも複数あります。 アプリケーションは、パブリッシャーとサブスクライバーの両方にすることができます。

情報のプロバイダーをパブリッシャー といいます。 パブリッシャーは主題に関する情報を提供しますが、その情報に関心のあるアプリケーションのことは何も知る必要はありません。 パブリッシャーは、その情報をパブリケーション というメッセージの形で生成します。このようなメッセージのトピックのパブリッシュと定義は、パブリケーションによって行われます。

情報のコンシューマーをサブスクライバー といいます。 サブスクライバーは、サブスクライバーが対象とするトピックを示したサブスクリプション を作成します。 したがって、どのパブリケーションがサブスクライバーに転送されるかは、サブスクリプションで決まります。 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。

パブリッシュされた情報は IBM MQ メッセージで送られ、情報の主題はそのトピック で識別されます。 パブリッシャーは情報をパブリッシュするときにトピックを指定し、サブスクライバーは受け取るパブリケーションのトピックを指定します。 サブスクライバーには、サブスクライバーがサブスクライブしたトピックに関する情報だけが送られます。

Point-to-Point メッセージングでは、メッセージごとに特定の宛先を含める必要がありますが、その必要性を排除してパブリッシュ/サブスクライブ・メッセージングで情報のプロバイダーとコンシューマーを分離できるようにしているのは、トピックの存在です。

パブリッシャーとサブスクライバーの間の相互作用はすべて、キュー・マネージャーによって制御されます。 キュー・マネージャーは、パブリッシャーからメッセージを受け取り、サブスクライバーから (一連のトピックの) サブスクリプションを受け取ります。 キュー・マネージャーのジョブは、パブリッシュされたメッセージを、メッセージのトピックへのインタレストを登録したサブスクライバーにルーティングすることです。

IBM MQ の標準機能を使用してメッセージが配布されるので、アプリケーションは既存の IBM MQ アプリケーションが使用できるすべてのフィーチャーを使用できます。 つまり、持続メッセージを使用して、一度だけ保証される配信を取得することや、メッセージをトランザクション作業単位の一部にして、パブリッシャーがコミットしたメッセージだけがサブスクライバーに配信されるようにすることが可能ということになります。

パブリッシャーとパブリケーション

IBM MQ パブリッシュ/サブスクライブでは、パブリッシャーとは、指定されたトピックに関する情報を、パブリケーションという標準的な IBM MQ メッセージの形式でキュー・マネージャーに対して使用可能にするアプリケーションです。 1 つのパブリッシャーが複数のトピックに関する情報をパブリッシュすることができます。

パブリッシャーは、MQPUT verb を使用して、事前にオープンしたトピックにメッセージを書き込みます。このメッセージがパブリケーションです。 次に、ローカル・キュー・マネージャーは、パブリケーションのトピックへのサブスクリプションを持つ任意のサブスクライバーに、パブリケーションを送付します。複数のサブスクライバーが、パブリッシュされたメッセージをコンシュームできます。

キュー・マネージャーは、適切なサブスクリプションを持つすべてのローカル・サブスクライバーにパブリケーションを配布することに加えて、自分に接続している他のキュー・マネージャーに、直接的にある

いはトピックのサブスクライバーを持つキュー・マネージャーのネットワークを介して、パブリケーションを配布することもできます。

IBM MQ パブリッシュ/サブスクライブ・ネットワークでは、パブリッシュ・アプリケーションはサブスクライバーにもなれます。

同期点の下にあるパブリケーション

パブリッシャーは同期点で MQPUT または MQPUT1 呼び出しを発行して、作業単位内でサブスクライバーに配信されたすべてのメッセージを含めることができます。 MQPMO_RETAIN オプション、または値が ALL または ALLDUR であるトピック配信オプション NPMSGDLV または PMSGDLV が指定された場合、キュー・マネージャーはパブリッシャー MQPUT または MQPUT1 呼び出しの範囲内で、内部的な MQPUT または MQPUT1 呼び出しを同期点で使用します。

状態情報とイベント情報

パブリケーションは、状態パブリケーション (例えば、現在の株価) またはイベント・パブリケーション (例えば、その株の取引) のどちらかに分類できます。

状態パブリケーション

状態パブリケーション には、何かの現在の状態 (例えば、株価やサッカーの試合の現在のスコア) に関する情報が含まれます。 何かが起こると (例えば、株価の変動やサッカーのスコアの変化)、それまでの状態情報は新しい情報に取って代わるので、不要になります。

サブスクライバーは、開始時に現行バージョンの状態情報を受信し、状態が変わるたびに新しい情報が送信されてくることを望むものです。

パブリケーションに状態情報が含まれる場合、そのパブリケーションは多くの場合に保存パブリケーションとしてパブリッシュされます。 新規サブスクライバーは通常、現在の状態情報を直ちに必要とします。 イベントが発生して情報がリパブリッシュされるのを待機することを望んではいません。 サブスクライバーが MQSO_PUBLICATIONS_ON_REQUEST または MQSO_NEW_PUBLICATIONS_ONLY オプションを使用しない限り、サブスクライブしたサブスクライバーは、トピックの保存パブリケーションを自動的に受信します。

イベント・パブリケーション

イベント・パブリケーション には、発生した個々のイベント (例えば、何かの株の取引や特定のゴールの得点) に関する情報が含まれます。 各イベントは他のイベントから独立しています。

サブスクライバーは、イベントが発生すると、そのイベントに関する情報を受信することを望むものです。

保存パブリケーション

デフォルトでは、関心を持つすべてのサブスクライバーにパブリケーションが送信された後、そのパブリケーションは廃棄されます。 ただし、パブリッシャーはパブリケーションのコピーを保存することを指定できます。その結果、そのトピックへのインタレストを登録する今後のサブスクライバーにパブリケーションのコピーを送信することができます。

関心を持つすべてのサブスクライバーにパブリケーションが送信された後にそのパブリケーションを削除するのは、イベント情報に適していますが、状態情報には必ずしも適しているわけではありません。 メッセージを保存することによって、新規サブスクライバーは、初回の状態情報を受信するのに情報が再びパブリッシュされるのを待機する必要がなくなります。 例えば、株価のサブスクリプションを登録したサブスクライバーは、株価が変動する (したがってリパブリッシュされる) のを待たずに、現在の株価を直ちに受信することになります。

キュー・マネージャーは、各トピックのパブリケーションを 1 つだけ保存できます。したがって、新しい保存パブリケーションがキュー・マネージャーに到着すると、トピックの既存の保存パブリケーションが削除されます。 ただし、既存パブリケーションが削除されるのが、新しい保存パブリケーションの到着と同期しない場合もあります。 そのため、どのトピックについても可能な限り、保存パブリケーションを送信するパブリッシャーが 1 つを超えないようにしてください。

MQSO_NEW_PUBLICATIONS_ONLY サブスクリプション・オプションを使用することにより、サブスクライバーは保存パブリケーションを受信しないことを指定できます。 既存のサブスクライバーは、保存パブリケーションの複製コピーが送信されてくるよう要求することができます。

状態情報であっても、以下のように、パブリケーションを保存する必要がない場合があります。

- あるトピックへのすべてのサブスクリプションが行われた後にそのトピックのパブリケーションが行われ、新しいサブスクリプションが見込まれない、または新しいサブスクリプションを許可しない場合は、パブリケーションを保存する必要はありません。パブリケーションが初めてパブリッシュされるときに、サブスクライバーの完全セットにパブリケーションが配信されるからです。
- パブリケーションが頻繁に (例えば毎秒) 行われる場合、新しいサブスクライバー (または障害からの復旧サブスクライバー) は、初期サブスクリプションのほとんど直後に現在の状態を受信します。したがって、このようなパブリケーションを保存する必要はありません。
- 大規模なパブリケーションの場合は、各トピックの保存パブリケーションを保管するためのかなりのストレージ・スペースが最終的に必要になることがあります。 複数キュー・マネージャー環境では、一致サブスクリプションを持っているネットワーク内のすべてのキュー・マネージャーによって、保存パブリケーションが保管されます。

保存パブリケーションを使用するかどうかを決定するときは、サブスクライブ・アプリケーションが障害からどのように復旧するかを考慮してください。 パブリッシャーが保存パブリケーションを使用しない場合は、その現在の状態をサブスクライバー・アプリケーションがローカル保管しなければならないこともあります。

パブリケーションが保存されるようにするには、MQPMO_RETAIN メッセージ書き込みオプションを使用します。 このオプションを使用してもパブリケーションを保持できない場合、メッセージはパブリッシュされず、呼び出しは MQRC_PUT_NOT_RETAINED で失敗します。

メッセージが保存パブリケーションである場合、このことは MQIsRetained メッセージ・プロパティーで示されます。 メッセージの持続性は、それが最初にパブリッシュされた時の状態と同じです。

関連概念

パブリッシュ/サブスクライブ・クラスターでの保存パブリケーションに関する設計上の考慮事項

同期点の下にあるパブリケーション

IBM MQ パブリッシュ/サブスクライブにおいて、同期点はパブリッシャーが使用することも、キュー・マネージャーが内部的に使用することもできます。

パブリッシャーは MQPMO_SYNCPOINT オプション付きの MQPUT/MQPUT1 呼び出しを発行するときに同期点を使用します。 サブスクライバーに送達されるメッセージはすべて、作業単位内でコミットされていないメッセージの最大数までカウントされます。 MAXUMSGS キュー・マネージャー属性はこの上限を指定します。 上限に到達すると、パブリッシャーは 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED 理由コードを受け取ります。

MQPMO_RETAIN オプション付きの MQPMO_NO_SYNCPOINT を使って、またはトピック送達オプション NPMSGDLV/PMSGDLV に値 ALL または ALLDUR を指定してパブリッシャーが MQPUT/MQPUT1 呼び出しを行うと、キュー・マネージャーは内部同期点を使用して、要求のとおりにメッセージが送達されることを保証します。 パブリッシャーの MQPUT/MQPUT1 呼び出しの有効範囲内に制限値に達すると、パブリッシャーは 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED 理由コードを受け取ることができます。

サブスクライバーとサブスクリプション

IBM MQ パブリッシュ/サブスクライブにおけるサブスクライバーは、パブリッシュ/サブスクライブ・ネットワーク内のキュー・マネージャーに特定のトピックに関する情報を要求するアプリケーションです。 サブスクライバーは、同じまたは異なるトピックに関するメッセージを、複数のパブリッシャーから受信できます。

サブスクリプションは、MQSC コマンドを使用して手動で、またはアプリケーションで作成できます。 これらのサブスクリプションは、ローカル・キュー・マネージャーに送出されます。サブスクリプションには、サブスクライバーが受信を望んでいるパブリケーションに関する以下の情報が含まれます。

- サブスクライバーが関心のあるトピック。ワイルドカードが使用される場合は、複数のトピックとして解決されることがあります。
- パブリッシュされるメッセージに適用される任意指定の選択ストリング。
- 選択されたパブリケーションを置くキュー (サブスクライバー・キュー と呼ばれる) のハンドル、および任意指定の CorrelId。

ローカル・キュー・マネージャーはサブスクリプション情報を保管し、パブリケーションを受信すると、情報をスキャンして、パブリケーションのトピックと選択ストリングが一致するサブスクリプションがあるかどうかを判別します。 一致するサブスクリプションごとに、キュー・マネージャーはサブスクライバーのサブスクライバー・キューにパブリケーションを送信します。 キュー・マネージャーが保管しているサブスクリプション情報は、DIS SUB コマンドおよび DIS SBSTATUS コマンドを使用することによって表示できます。

サブスクリプションが削除されるのは、以下のいずれかのイベントが発生したときだけです。

- サブスクライバーが MQCLOSE 呼び出しを使用してアンサブスクライブした (サブスクリプションが非永続になっていた場合)。
- サブスクリプションの有効期限が切れた。
- システム管理者が DELETE SUB コマンドを使用してサブスクリプションを削除した。
- サブスクライバー・アプリケーションが終了した (サブスクリプションが非永続になっていた場合)。
- キュー・マネージャーが停止または再始動した (サブスクリプションが非永続になっていた場合)。

メッセージを入手する際には、MQGET 呼び出しで適切なオプションを使用します。 アプリケーションが 1 つのサブスクリプションのメッセージのみを処理する場合は、少なくとも、C サンプル・プログラム `amqssbxa.c` および 非管理 MQ サブスクライバーで示されているように、 `get-by-correlid` を使用する必要があります。 使用する **CorrelId** は、MQSD 内の MQSUB から返されます。**SubCorrelId** フィールド。

関連概念

複製サブスクリプションおよび共用サブスクリプション

関連資料

sharedSubscription プロパティーの定義方法を示す例

管理対象キューおよびパブリッシュ/サブスクライブ

サブスクリプションを作成する際、管理キューイングを使用するよう選択できます。 管理キューイングを使用する場合、サブスクリプションの作成時にサブスクリプション・キューが自動的に作成されます。 管理対象キューは、サブスクリプションの永続性に従って、自動的にタイディアップが行われます。 管理対象キューを使用すると、パブリケーションを受け取るキューの作成に関して心配する必要がなくなり、非永続のサブスクリプション接続が閉じられると、コンシュームされていないパブリケーションがサブスクライバー・キューから自動的に除去されます。

アプリケーションが特定のキューをサブスクライバー・キュー (受け取るパブリケーションの宛先) として使用する必要がない場合、MQSO_MANAGED サブスクリプション・オプションを使用して、管理対象サブスクリプション を使用できます。 管理対象サブスクリプションを作成する場合、キュー・マネージャーは、サブスクライバー・キュー用のサブスクライバーにオブジェクト・ハンドルを返します。このサブスクライバー・キューは、キュー・マネージャーによって作成され、そこでパブリケーションを受け取ります。 これは、管理対象サブスクリプションが、IBM MQ がサブスクリプションを処理する場所であるためです。 キュー上での参照、取得、または問い合わせを許可して、キューのオブジェクト・ハンドルが返されます (一時的な動的キューへのアクセス権限を明示的に与えられない限り、管理対象キューの属性の書き込みまたは設定を行うことはできません)。

サブスクリプションの永続性によって、キュー・マネージャーへのサブスクライブ・アプリケーションの接続が中断されたときに、管理対象キューが残るかどうかが決まります。

非永続サブスクリプションで使用される場合、管理対象サブスクリプションは特に便利です。これ以外の方法では、アプリケーションの接続が終了しても、コンシュームされていないメッセージはサブスクライバー・キューに残り、いつまでもキュー・マネージャー内のスペースを占めてしまうからです。 管理対象サブスクリプションを使用する場合、管理対象キューは一時的な動的キューになります。そのため、以下のいずれかの原因で接続が中断した場合、コンシュームされていないメッセージとともに削除されます。

- MQCO_REMOVE_SUB が指定された MQCLOSE が使用され、管理対象 Hobj が閉じられた。
- 非永続サブスクリプション (MQSO_NON_DURABLE) を使用しているアプリケーションへの接続が失われた。
- サブスクリプションの有効期限が切れ、管理対象 Hobj が閉じられたため、サブスクリプションが削除された。

管理対象サブスクリプションは永続サブスクリプションとともに使用できますが、接続が再オープンされたときに、コンシュームされていないメッセージを取得できるように、それらをサブスクライバー・キューに入れたままにするという場合も考えられます。 そのため、永続サブスクリプション用の管理対象キューは永続的な動的キューの形をとり、キュー・マネージャーへのサブスクライブ・アプリケーションの接続が中断されたときにも残ります。

永続的な動的管理対象キューを使用する場合にサブスクリプションに有効期限を設定して、接続が中断された後もそのキューを存続させるものの、無期限には存続させないようにすることができます。

管理対象キューを削除すると、エラー・メッセージを受け取ります。

作成される管理対象キューの名前の末尾には数値 (タイム・スタンプ) が付けられるため、それぞれが固有になります。

サブスクリプション永続性

サブスクリプションを永続的または非永続的として構成できます。 サブスクライブ・アプリケーションがキュー・マネージャーから切断された場合にサブスクリプションで行われる処理は、サブスクリプション永続性で決まります。

永続サブスクリプション

永続サブスクリプションは、キュー・マネージャーへのサブスクライブ・アプリケーションの接続が閉じられても存続します。 サブスクリプションが永続的である場合は、サブスクライブ・アプリケーションが切断されてもサブスクリプションは依然として有効であり、サブスクライブ・アプリケーションは、サブスクリプションを要求して改めて再接続すると使用することができます。このときサブスクライブ・アプリケーションは、サブスクリプションが作成されたときに返された **SubName** を使用します。

永続的にサブスクライブするときは、サブスクリプション名 (**SubName**) が必要です。 サブスクリプション名は、サブスクリプションの識別に使用できるように、キュー・マネージャー内で固有でなければなりません。 つまり、サブスクリプションに対する接続を意図的に閉じていても (MQCO_KEEP_SUB オプションを使用)、キュー・マネージャーから切断されていても、再開するサブスクリプションを指定するときは ID が必要であるということです。 MQSO_RESUME オプションを指定した MQSUB 呼び出しを使用することによって、既存のサブスクリプションを再開できます。 サブスクリプション名は、 SUBTYPE ALL または ADMIN を指定して DISPLAY SBSTATUS コマンドを使用した場合にも表示されます。

アプリケーションが必要としなくなった永続サブスクリプションは、MQCO_REMOVE_SUB オプションを指定した MQCLOSE 関数呼び出しを使用して削除するか、MQSC コマンド DELETE SUB を使用して手動で削除できます。

DURSUB トピック属性を使用して、トピックに対する永続サブスクリプションが可能かどうかを指定できます。

MQSO_RESUME オプションを使用した MQSUB 呼び出しから戻るときにサブスクリプション有効期限が設定されますが、サブスクリプションの残りの有効期限時間ではなく、元の有効期限に設定されます。

キュー・マネージャーは、永続サブスクリプションに対応するためにパブリケーションの送信を、そのサブスクライバー・アプリケーションが接続されていなくても、続行します。 そのため、サブスクライバー・キューにメッセージがたまることになります。 この問題を回避する最も簡単な方法は、適切なところでは非永続サブスクリプションを使用することです。 一方、永続サブスクリプションを使用する必要がある場合、サブスクライバーが保存パブリケーション・オプションを使用してサブスクライブすることで、メッセージが溜まるのを回避できます。 その場合、サブスクライバーは、パブリケーションをいつ受け取るかを MQSUBRQ 呼び出しを使用して制御できます。

非永続サブスクリプション

非永続サブスクリプションは、キュー・マネージャーへのサブスクライブ・アプリケーションの接続が開いている間だけ存在します。 サブスクライブ・アプリケーションが、意図的に、あるいは接続の損失により、キュー・マネージャーから切断されると、サブスクリプションは除去されます。 接続が閉じられると、キュー・マネージャーからサブスクリプションに関する情報が削除され、DISPLAY SBSTATUS コマンドを使用してサブスクリプションを表示しようとしても現れなくなります。 サブスクライバー・キューにはメッセージが書き込まれなくなります。

非永続サブスクリプションの場合にサブスクライバー・キュー上の未コンシューム・パブリケーションがどうなるかは、以下のように決まります。

- サブスクライブしているアプリケーションが 管理対象宛先を使用している場合、コンシュームされていないパブリケーションは自動的に削除されます。
- サブスクライブ・アプリケーションがサブスクライブ時に専用サブスクライバー・キューのハンドルを提供する場合は、未コンシューム・メッセージの自動削除は行われません。 適切な場合にキューをクリアするのは、アプリケーションが行います。 キューが複数のサブスクライバーまたは他の Point-to-Point アプリケーションによって共有されている場合は、キューを完全にクリアするのは適切でない可能性があります。

非永続サブスクリプションの必須ではありませんが、サブスクリプション名が定義されればキュー・マネージャーによって使用されます。 サブスクリプション名は、サブスクリプションの識別に使用できるように、キュー・マネージャー内で固有でなければなりません。

関連概念

複製サブスクリプションおよび共用サブスクリプション

関連タスク

JMS 2.0 共用サブスクリプションの使用

関連資料

sharedSubscription プロパティーの定義方法を示す例

選択ストリング

選択ストリングとは、サブスクリプションと一致するかどうかを判別するため、パブリケーションに適用される式のことです。 選択ストリングにはワイルドカード文字を含めることができます。

サブスクライブするときには、トピックを指定することに加えて、選択ストリングを指定して、メッセージ・プロパティーに従ってパブリケーションを選択することができます。

パブリッシャーによってメッセージが書き込まれると、各サブスクライバーに送達するために変更される前に、選択ストリングがそのメッセージに対して評価されます。 パブリッシュ操作の一部として変更される可能性のある、選択ストリング内のフィールドを使用する際には注意してください。 例えば、MQMD フィールドの `UserIdentifier`、`MsgId`、および `CorrelId` が該当します。

選択ストリングは、パブリッシュ操作の一部としてキュー・マネージャーが追加するメッセージ・プロパティー・フィールドを参照すべきではありません (パブリッシュ/サブスクライブのメッセージ・プロパティーを参照)。 ただし、パブリケーションのトピック・ストリングを含むメッセージ・プロパティー `MQTopicString` は例外です。

関連概念

選択ストリングの規則と制約事項

トピック

トピックは、パブリッシュ/サブスクライブ・メッセージでパブリッシュされる情報の主題です。

Point-to-Point システムでは、メッセージは特定の宛先アドレスに送信されます。 主題ベースのパブリッシュ/サブスクライブ・システムでは、メッセージの内容を表す主題に基づいて、サブスクライバーにメッセージが送信されます。 内容ベースのシステムでは、メッセージ自体の内容に基づいてサブスクライバーにメッセージが送信されます。

IBM MQ パブリッシュ/サブスクライブ・システムは、主題ベースのパブリッシュ/サブスクライブ・システムです。 パブリッシャーはメッセージを作成し、パブリケーションの主題に最適なトピック・ストリング

と共に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。 70 ページの『トピック・ストリング』の文書では、パブリケーションの主題を識別するためのトピック・ストリングの構文について説明しています。 サブスクライバーも、受信するトピックを選択するためにトピック・ストリングを作成します。 サブスクライバーが作成するトピック・ストリングには、パブリケーションのトピック・ストリングとのパターン・マッチングのための 2 つの代替ワイルドカード・スキームのうちのどちらかを含めることができます。 パターン・マッチングについては、71 ページの『ワイルドカード・スキーマ』で説明しています。

主題ベースのパブリッシュ/サブスクライブでは、パブリッシャー (管理者) が主題をトピックに分類する必要があります。 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・ツリーの例については、77 ページの『トピック・ツリー』を参照してください。 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。

主題を階層型トピック・ツリーに編成すると同時に、トピックを管理トピック・オブジェクトに関連付けることができます。 トピックを管理トピック・オブジェクトに関連付けることにより、トピックをクラスター内に配布するかどうかなどの属性をトピックに割り当てます。 関連付けは、管理トピック・オブジェクトの TOPICSTR 属性を使用してトピックを指定することによって行います。 管理トピック・オブジェクトをトピックに明示的に関連付けない場合、トピックは、管理トピック・オブジェクトに既に 関連付けられている、トピック・ツリー内の最も近い上位トピックの属性を継承します。 親トピックをまったく定義していない場合は、SYSTEM.BASE.TOPIC から継承します。 管理トピック・オブジェクトについては、78 ページの『管理トピック・オブジェクト』で説明しています。

注 : トピックのすべての属性を SYSTEM.BASE.TOPIC から継承する場合でも、SYSTEM.BASE.TOPIC から直接継承するトピックのルート・トピックを定義してください。 例えば、米国の州のトピック・スペース (USA/Alabama や USA/Alaska など) では、USA がルート・トピックです。 ルート・トピックの主な目的は、誤ったサブスクリプションにパブリケーションが一致することがないように、重複しない離散的トピック・スペースを作成することにあります。 ルート・トピックの属性を変更すれば、トピック・スペース全体に影響を及ぼせることにもなります。 例えば、**CLUSTER** 属性の名前を設定することができます。

パブリッシャーまたはサブスクライバーとしてトピックを表すときに、トピック・ストリングを指定するか、トピック・オブジェクトを参照することができます。 あるいはその両方を行うことができ、その場合、指定するトピック・ストリングはトピック・オブジェクトのサブトピックを定義します。 キュー・マネージャーは、トピック・オブジェクトで指定されたトピック・ストリング接頭部にトピック・ストリングを追加し、2 つのトピック・ストリングの間に追加の '/' を挿入することによって、トピックを識別します (例えば、 トピック・ストリング/オブジェクト・ストリング)。 これについては、75 ページの『トピック・ストリングの結合』で詳しく説明しています。 結果のトピック・ストリングを使用してトピックが識別され、管理トピック・オブジェクトに関連付けられます。 管理トピック・オブジェクトは、マスター・トピックに対応するトピック・オブジェクトと同じトピック・オブジェクトであるとは限りません。

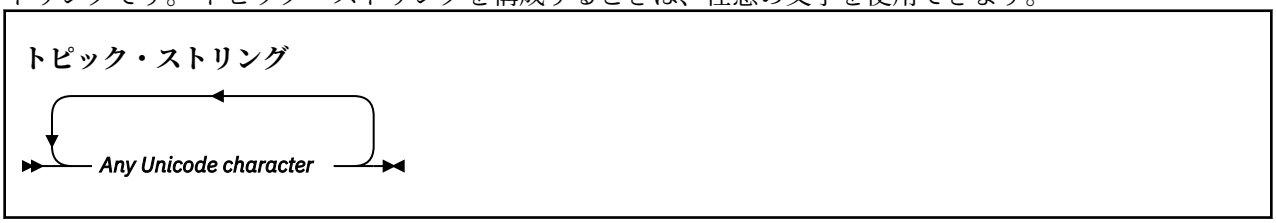
内容ベースのパブリッシュ/サブスクライブでは、それぞれのメッセージの内容を検索する選択ストリングを指定することによって、受信するメッセージを定義します。 IBM MQ は、中間形式の内容ベース・パブリッシュ/サブスクライブを提供します。これは、メッセージの全内容ではなくメッセージ・プロパティーをスキャンするメッセージ・セレクターを使用します (セレクターを参照)。 メッセージ・セレクターの典型的な用途は、トピックをサブスクライブし、次に数値プロパティーで選択を限定することです。 セレクターを使用すると、特定の範囲に限定した値に関心があるということを指定できます。これは、文字ベースまたはトピック・ベースのワイルドカードを使用する場合にはできないことです。 メッセージの全内容に基づいてどうしてもフィルタリングを行う必要がある場合は、IBM Integration Bus を使用する必要があります。

トピック・ストリング

トピック・ストリングを使用してトピックとしてパブリッシュする情報にラベルを付けます。 文字ベースまたはトピック・ベースのワイルドカードのどちらかのトピック・ストリングを使用してトピックのグループにサブスクライブします。

トピック

トピック・ストリング は、パブリッシュ/サブスクライブ・メッセージのトピックを識別するための文字ストリングです。 トピック・ストリングを構成するときは、任意の文字を使用できます。



IBM WebSphere® MQ 7 のパブリッシュ/サブスクライブでは、3 文字は特別な意味を持ちます。 これらはトピック・ストリング内のどこにあってもかまいませんが、使用にあたって注意が必要です。 特殊文字の使用については、72 ページの『トピック・ベース・ワイルドカード・スキーム』で説明しています。

スラッシュ (/)

トピック・レベル分離文字。 トピックをトピック・ツリーとして構成するには、'/' 文字を使用します。

できれば、空のトピック・レベル ('//') は避けてください。 これは、トピック階層内のトピック・ストリングのないノードに相当します。 トピック・ストリング内の先頭または末尾の '/' は、先頭または末尾の空ノードに相当します。これも避けてください。

ハッシュ記号 (#)

サブスクリプションでマルチレベル・ワイルドカードを構成する場合に '/' と組み合わせて使用します。 パブリッシュされるトピックの指定に使用するトピック・ストリングで '/' に隣接して '#' を使用する場合は、注意が必要です。 71 ページの『トピック・ストリングの例』で、'#' の理にかなった使い方を示しています。

ストリング '.../#/...'、'#/...'、および '.../#' は、サブスクリプション・トピック・ストリングでは特別な意味を持ちます。 これらのストリングは、トピック階層の 1 つ以上のレベルのすべてのトピックに一致します。 したがって、これらのシーケンスのいずれかを使用してトピックを作成した場合は、同様にトピック階層の複数レベルのすべてのトピックのサブスクライブがなければ、そのトピックをサブスクライブできないことになります。

正符号 (+)

サブスクリプションで単一レベル・ワイルドカードを構成する場合に '/' と組み合わせて使用。 パブリッシュされるトピックの指定に使用するトピック・ストリングで '/' に隣接して '+' を使用する場合は、注意が必要です。

ストリング '.../+/...'、'+/...'、および '.../+' は、サブスクリプション・トピック・ストリングでは特別な意味を持ちます。 これらのストリングは、トピック階層の 1 つのレベルのすべてのトピックに一致します。 したがって、これらのシーケンスのいずれかを使用してトピックを作成した場合は、同様にトピック階層の 1 つのレベルのすべてのトピックのサブスクライブがなければ、そのトピックをサブスクライブできないことになります。

トピック・ストリングの例

```
IBM/Business Area#/Results
IBM/Diversity/%African American
```

関連資料

トピック

ワイルドカード・スキーマ
複数のトピックへのサブスクライブに使用される 2 つのワイルドカード方式が存在します。 方式の選択はサブスクリプション・オプションです。

MQSO_WILDCARD_TOPIC
トピック・ベースのワイルドカード方式を使用して、サブスクライブするトピックを選択します。

ワイルドカード・スキーマを明示的に選択しない場合は、これがデフォルトです。

MQSO_WILDCARD_CHAR
文字ベースのワイルドカード方式を使用して、サブスクライブするトピックを選択します。

DEFINE SUB コマンドで **wschema** パラメーターを指定して、いずれかのスキームを設定してください。 詳細については、DEFINE SUB を参照してください。

注 : IBM WebSphere MQ 7.0 より前に作成されたサブスクリプションは、常に文字ベースのワイルドカード方式を使用します。

例

```
IBM/+/Results
#/Results
IBM/Software/Results
IBM/*ware/Results
```

トピック・ベース・ワイルドカード・スキーム
トピック・ベースのワイルドカードを使用すると、サブスクライバーは同時に複数のトピックをサブスクライブできます。

トピック・ベースのワイルドカードは、IBM MQ パブリッシュ/サブスクライブのトピック・システムの強力なフィーチャーです。 マルチレベル・ワイルドカードと単一レベル・ワイルドカードはサブスクリプションに使用できますが、メッセージのパブリッシャーがトピック内で使用することはできません。

トピック・ベース・ワイルドカード・スキームでは、トピック・レベル別にグループ化されたパブリケーションを選択できます。 サブスクリプションのトピック・レベルのストリングがパブリケーションのストリングと完全に一致しなければならないかどうかを、トピック階層のレベルごとに選択できます。 例えば、サブスクリプションが `IBM/+/Results` の場合は、次のトピックがすべて選択されます。

```
IBM/Software/Results
IBM/Services/Results
IBM/Hardware/Results
```

ワイルドカードには 2 つのタイプがあります。

マルチレベル・ワイルドカード

- マルチレベル・ワイルドカードは、サブスクリプションで使用されます。 パブリケーションで使用すると、リテラルとして扱われます。
- マルチレベル・ワイルドカード文字 '#' は、トピック内のレベルをいくつでも一致させる場合に使用します。 例えば、トピック・ツリー例を使用すると、'USA/Alaska/#' をサブスクライブした場合は、'USA/Alaska' トピックと 'USA/Alaska/Juneau' トピックに関するメッセージを受け取ります。
- マルチレベル・ワイルドカードはゼロ個以上のレベルを表すことができます。 したがって、'USA/#' は 'USA' 単独とも一致します。この場合、'#' はゼロ個のレベルを表しています。 このコンテキストでは、トピック・レベル分離文字は意味を持ちません。分離するレベルがないからです。
- マルチレベル・ワイルドカードは、単独で指定された場合、またはトピック・レベル分離文字に続いて指定された場合のみ有効です。 したがって、'#' と 'USA/#' は有効なトピックです。この場合、'#' 文字はワイルドカードとして扱われます。 一方、'USA#' も有効なトピック・ストリングではありますが、'#' 文字はワイルドカードと見なされず、特別な意味を持ちません。 詳しくは、74 ページの『トピック・ベースのワイルドカードが無効な場合』 を参照してください。

単一レベル・ワイルドカード

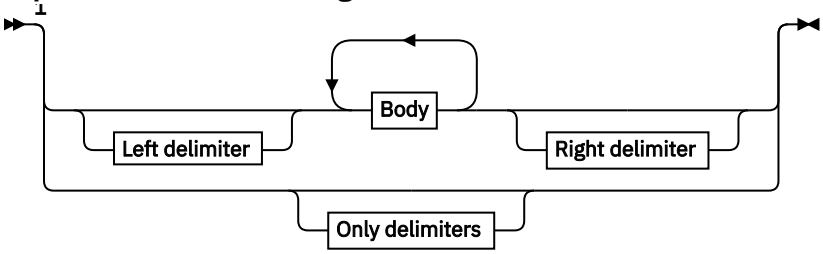
- 単一レベル・ワイルドカードは、サブスクリプションで使用されます。 パブリケーションで使用すると、リテラルとして扱われます。

- 単一レベル・ワイルドカード文字 '+' は、トピック・レベルが 1 つだけ一致します。 例えば、'USA/+' は 'USA/Alabama' と一致しますが、'USA/Alabama/Auburn' とは一致しません。 単一レベル・ワイルドカードは単一レベルのみと一致するので、'USA/+' は 'USA' と一致しません。
- 単一レベル・ワイルドカードはトピック・ツリーのどのレベルでも使用でき、マルチレベル・ワイルドカードとの併用が可能です。 単一レベル・ワイルドカードは、単独で指定する場合を除いて、トピック・レベル分離文字に続いて指定する必要があります。 したがって、'+' と 'USA/+' は有効なトピックです。この場合、'+' 文字はワイルドカードとして扱われます。 一方、'USA+' も有効なトピック・ストリングではありますが、'+' 文字はワイルドカードと見なされず、特別な意味を持ちません。 詳しくは、74 ページの『トピック・ベースのワイルドカードが無効な場合』 を参照してください。

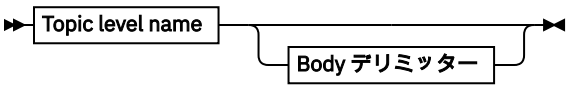
トピック・ベース・ワイルドカード・スキームの構文には、エスケープ文字がありません。 '#' および '+' がワイルドカードとして扱われるかどうかは、それらのコンテキストによります。 詳しくは、74 ページの『トピック・ベースのワイルドカードが無効な場合』 を参照してください。

注 : トピック・ストリングの先頭と末尾は、特別に扱われます。 '$' を使用してストリングの終わりを示すと、 '$#/...' はマルチレベル・ワイルドカードで、 '$/#/..'になります。 ルートにある空のノードで、その後にマルチレベル・ワイルドカードが続きます。

Topic-based wildcard string

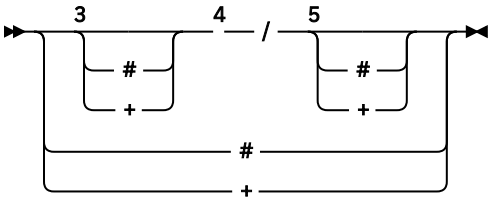


Body



Topic level name

Only delimiters



注:

[1] ヌル・トピック・ストリングもゼロ長トピック・ストリングも無効です。

[2] char ベースのワイルドカード・スキームとトピック・ベースのワイルドカード・スキームの間の互換性のために、レベル名ストリングでは *, ?, % のいずれも使用しないことをお勧めします。

[3] これらのケースは *left delimiter* パターンと同等です。

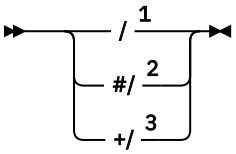
[4] ワイルドカードなしの / は、単一の空トピックに相当します。

[5] これらのケースは *right delimiter* パターンと同等です。

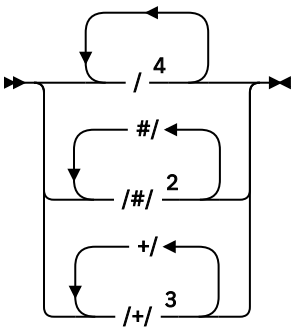
[6] すべてのトピックと一致します。

[7] レベルが 1 つだけのすべてのトピックと一致します。

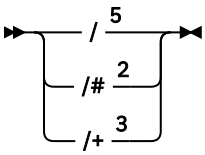
Left delimiter



Body delimiter



Right delimiter



注:

[1] トピック・ストリングは空トピックで始まります。
[2] ゼロ個以上のレベルと一致します。 複数のマルチレベル・マッチング・ストリングは、1 つのマルチレベル・マッチング・ストリングと同じ効果があります。
[3] 1 レベルだけ一致します。
[4] // は空トピック (トピック・ストリングのないトピック・オブジェクト) です。
[5] トピック・ストリングは空トピックで終わります。

トピック・ベースのワイルドカードが無効な場合

ワイルドカード文字の '+' と '#' は、あるトピック・レベル内で他の文字 (それら自身を含む) と混用されると、特別な意味を持たなくなります。

つまり、あるトピック・レベルに '+' または '#' が他の文字と一緒に含まれるトピックをパブリッシュできることになります。

例えば、次の 2 つのトピックを考えてみましょう。

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

最初の例では、'+' および '#' 文字はワイルドカードとして扱われるので、パブリッシュのトピック・ストリングでは無効ですが、サブスクリプションでは有効です。

2 番目の例では、'+' および '#' 文字はワイルドカードとして扱われないので、パブリッシュとサブスクライブの両方のトピック・ストリングにすることができます。

例

```
IBM/+/Results
#/Results
IBM/Software/Results
```

文字ベースのワイルドカード・スキーム
文字ベースのワイルドカード・スキームでは、 従来どおりの文字のマッチングに基づいてトピックを選択できます。

ストリング '*' を使用すると、トピック階層の複数レベルにあるすべてのトピックを選択できます。 文字ベースのワイルドカード・スキームで '*' を使用することは、トピック・ベース・ワイルドカード・ストリング '#' を使用することと同等です。

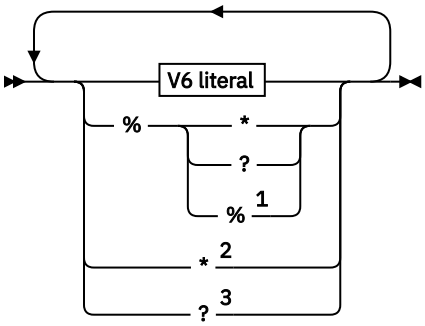
' *x*/*/*y* ' は、トピック・ベースのスキームでは ' *x*/#/*y* ' と同等であり、レベル ' *x* と *y* ' の間のトピック階層内のすべてのトピックを選択します。ここで、 ' *x* ' と ' *y* ' は、ワイルドカードによって返される一連のレベルに含まれていないトピック名です。

トピック・ベースのスキーマの '/+/' には、文字ベースのスキーマで完全に同等のものはありません。 'IBM/*/Results' は、'IBM/Patents/Software/Results'も選択します。 階層の各レベルにおいてトピック名セットが固有である場合にのみ、 2 つのスキーマを使用して、完全な一致をもたらす照会を常に構成できます。

一般的な使用法において、文字ベースのスキーマの '*' と '?' には、トピック・ベースのスキームで同等のものはありません。 トピック・ベースのスキーマでは、ワイルドカードを使用した部分マッチングを実行しません。 文字ベースのワイルドカード・サブスクリプション 'IBM/*ware/Results' には、トピック・ベースのスキームで同等のものはありません。

注 : 文字のワイルドカード・サブスクリプションを使用したマッチングは、 トピック・ベースのサブスクリプションを使用したマッチングよりも遅くなります。

Character-based wildcard string



V6 literal

▶▶ *?* を除く任意の Unicode 文字 *および* % ▶◀

注:

1 「次の文字をエスケープ」を意味するため、リテラルとして扱われます。 '%' の後には '*'、'?' または '%' が続く必要があります。 71 ページの『トピック・ストリングの例』を参照してください。
2 サブスクリプションで「0 文字以上一致」を意味します。
3 サブスクリプションで「1 文字だけ一致」を意味します。

例

```
IBM/*/Results
IBM/*ware/Results
```

トピック・ストリングの結合

サブスクリプションを作成するとき、またはトピックを開いてメッセージをパブリッシュできるようにする場合、2 つの別個のサブトピック・ストリングまたは "サブトピック"を結合することにより、トピック・ストリングを形成することができます。 1 つのサブトピックは、アプリケーションまたは管理コマンドに

よってトピック・ストリングとして提供され、もう 1 つはトピック・オブジェクトに関連付けられたトピック・ストリングです。 サブトピック自体をトピック・ストリングとして使用することも、複数を組み合わせて新しいトピック名を形成することもできます。

例えば、MQSC コマンド **DEFINE SUB** を使用してサブスクリプションを定義する場合、コマンドは **TOPICSTR** (トピック・ストリング) または **TOPICOBJ** (トピック・オブジェクト) のいずれかまたは両方を属性として受け入れることができます。 **TOPICOBJ** のみが提供された場合、そのトピック・オブジェクトに関連付けられたトピック・ストリングがトピック・ストリングとして使用されます。 **TOPICSTR** のみが提供された場合、それがトピック・ストリングとして使用されます。 両方が提供されると、**TOPICOBJ** / **TOPICSTR** 形式でそれらが連結されて単一のトピック・ストリングになります。ここで、**TOPICOBJ** で構成されるトピック・ストリングが常に最初であり、ストリングの 2 つの部分は常に "/" 文字で区切られます。

同様に、MQI プログラムでは、MQOPEN によってフル・トピック名が作成されます。 これは、パブリッシュ/サブスクライブ MQI 呼び出しで使用される 2 つのフィールドにより、次にリストする順序で構成されます。

1. **ObjectName** フィールドで指定されたトピック・オブジェクトの **TOPICSTR** 属性。
2. アプリケーションが提供するサブトピックを定義する **ObjectString** パラメーター。

結果のトピック・ストリングは **ResObjectString** パラメーターで戻されます。

各フィールドの最初の文字がブランクや NULL 文字ではなく、フィールド長がゼロより大きい場合に、これらのフィールドは存在すると見なされます。 1 つのフィールドだけが存在する場合は、未変更のままトピック名として使用されます。 どちらのフィールドにも値が設定されていない場合、呼び出しは理由コード MQRC_UNKNOWN_OBJECT_NAME または MQRC_TOPIC_STRING_ERROR (フル・トピック名が無効な場合) により失敗します。

両方のフィールドが存在する場合、結果の結合されたトピック名の 2 つのエレメントの間に "/" 文字が挿入されます。

以下の表は、トピック・ストリング連結の例を示しています。

表 2. トピック・ストリング連結の例

トピック・オブジェクトの **TOPICSTR**	アプリケーションまたは **DEFINE SUB** コマンドにより提供されるトピック・ストリング	フル・トピック名	コメント
Football/Scores	''	Football/Scores	トピック・オブジェクトの TOPICSTR は単独で使用されます。
''	Football/Scores	Football/Scores	ObjectString/TOPICSTR は単独で使用されます。
Football	Scores	Football/Scores	"/" 文字が連結点に追加されます。
Football	/Scores	Football//Scores	2 つのストリングの間に「空ノード」が生成されます。 これは、"Football/Scores" とは異なります。
/Football	Scores	/Football/Scores	トピックが「空ノード」で始まります。 これは、"Football/Scores" とは異なります。

"/" 文字は特殊文字と見なされ、77 ページの『トピック・ツリー』 内の完全なトピック名に構造を提供します。 "/" 文字は、トピック・ツリーの構造が影響を受けるため、他の理由で使用することはできません。 トピック "/Football" は、トピック "Football"と同じではありません。

注 : サブスクリプションの作成時にトピック・オブジェクトを使用する場合、トピック・オブジェクトのトピック・ストリングの値は、定義時にサブスクリプションで固定されます。 それ以降、トピック・オブジェクトが変更されても、サブスクリプションが定義されているトピック・ストリングには影響しません。

トピック・ストリングのワイルドカード文字

以下のワイルドカード文字が特殊文字です。

- 正符号 (+)
- 番号記号 (#)
- アスタリスク (*)
- 疑問符 (?)

ワイルドカード文字は、サブスクリプションで使用される場合にのみ特別な意味を持ちます。 これらの文字は、他の場所で使用した場合に無効とは見なされませんが、使用方法を確実に理解しておく必要があります。 トピック・オブジェクトをパブリッシュまたは定義するときには、トピック・ストリングでこれらの文字を使用しない方が良い場合があります。

1 つのトピック・レベル内で # または + が他の文字 (それらの文字自体を含む) と混在するようなトピック・ストリングでパブリッシュする場合、どちらかのワイルドカード体系を使用してトピック・ストリングをサブスクライブできます。

2 つの / 文字間の唯一の文字として # または + があるトピック・ストリングでパブリッシュする場合、ワイルドカード体系 MQSO_WILDCARD_TOPIC を使用するアプリケーションによってトピック・ストリングを明示的にサブスクライブできません。 この状態の結果、アプリケーションは予期していたよりも多くのパブリケーションを受け取ることになります。

定義されたトピック・オブジェクトのトピック・ストリングでワイルドカード文字を使用しないでください。 使用すると、オブジェクトがパブリッシャーによって使用されるときに、文字がリテラル文字として処理され、サブスクリプションによって使用されるときにワイルドカード文字として処理されます。 これにより混乱が発生する可能性があります。

コード・スニペットの例

このコード・スニペットは、サンプル・プログラムの例 2: 可変トピックへのパブリッシャーから抽出したもので、トピック・オブジェクトと可変トピック・ストリングを結合しています。

```
MQOD   td = {MQOD_DEFAULT}; /* Object Descriptor      */
td.ObjectType = MQOT_TOPIC;  /* Object is a topic      */
td.Version = MQOD_VERSION_4;  /* Descriptor needs to be V4  */
strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

トピック・ツリー

定義する各トピックは、トピック・ツリー内の要素、つまりノードです。 トピック・ツリーは、最初は空にしておくことも、MQSC または PCF コマンドを使用して既に定義されたトピックを含めることも可能です。 トピック作成コマンドを使用するか、パブリケーションまたはサブスクリプションで初めてトピックを指定することによって、新規トピックを定義できます。

トピックのトピック・ストリングを定義する際に任意の文字ストリングを使用できますが、階層ツリー構造に適合するトピック・ストリングにすることをお勧めします。 トピック・ストリングとトピック・ツリーを注意深く設計すると、次の操作が容易になります。

- 複数のトピックのサブスクライブ。
- セキュリティー・ポリシーの確立。

トピック・ツリーをフラットな線形構造で構成することは可能ですが、ルート・トピックが 1 つ以上ある階層構造のトピック・ツリーを構築したほうがより効果的です。 セキュリティーの計画およびトピックについて詳しくは、パブリッシュ/サブスクライブのセキュリティーを参照してください。

78 ページの図 18 は、ルート・トピックが 1 つあるトピック・ツリーの例を示しています。

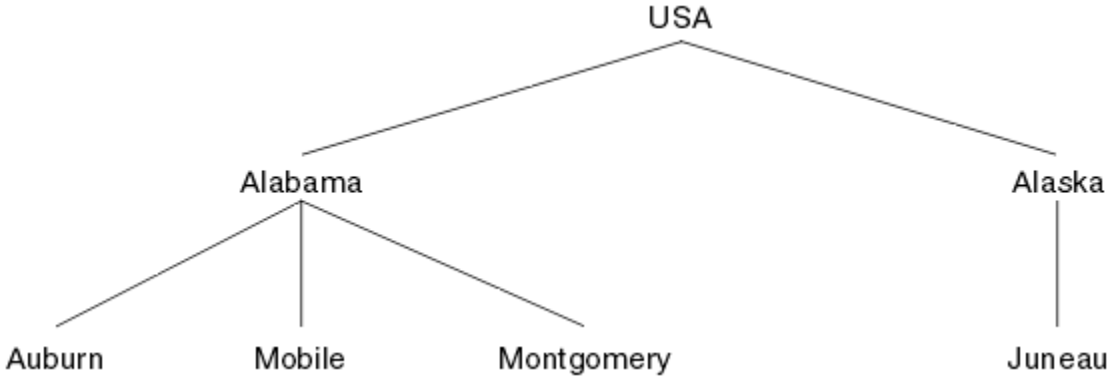


図 *18.* トピック・ツリーの例

図の中の各文字ストリングは、トピック・ツリー内のノードを表しています。 完全なトピック・ストリングは、トピック・ツリー内の 1 つ以上のレベルからノードを集約することによって作成されます。 レベルは「/」文字で分離されます。 完全指定トピック・ストリングの形式は、「root/level2/level3」になります。

78 ページの図 18 に示すトピック・ツリー内の有効なトピックは、以下のとおりです。

「USA」
「USA/Alabama」
「USA/Alaska」
「USA/Alabama/Auburn」
「USA/Alabama/Mobile」
「USA/Alabama/Montgomery」
「USA/Alaska/Juneau」

トピック・ストリングおよびトピック・ツリーを設計するときは、キュー・マネージャーがトピック・ストリングそのものを解釈したり、トピック・ストリングそのものから意味を引き出したりはしないということを覚えておいてください。 選択されたメッセージをそのトピックのサブスクライバーに送信するのにトピック・ストリングが使用されるだけです。

トピック・ツリーの構造と内容には、次の原則が適用されます。

- トピック・ツリー内のレベル数には制限がありません。
- トピック・ツリー内のレベルの名前の長さには制限がありません。
- 「ルート」ノードがいくつあってもかまいません。つまり、トピック・ツリーがいくつあってもかまいません。

関連タスク

トピック・ツリー内の不要なトピック数の削減

管理トピック・オブジェクト

管理トピック・オブジェクトを使用すると、デフォルトではない特定の属性をトピックに割り当てることができます。

79 ページの図 19 は、 さまざまなスポーツを扱う別個のトピックに分割された `Sport` というハイレベル・トピックを トピック・ツリーとして視覚化できる方法を示しています。

図 *19.* 視覚化したトピック・ツリー

79 ページの図 20 は、 トピック・ツリーをさらに細かく、各スポーツに関するさまざまなタイプの情報に分割できる方法を示しています。

図 *20.* 拡張したトピック・ツリー

この図のようなトピック・ツリーを作成するために、管理トピック・オブジェクトを定義する必要はありません。 このツリーの各ノードは、パブリッシュまたはサブスクライブ操作で作成されたトピック・ストリングによって定義されます。 このツリーの各トピックは、親から属性を継承します。 デフォルトではすべての属性が ASPARENT に設定されているため、属性は親トピック・オブジェクトから継承されます。 この例では、すべてのトピックは Sport トピックと同じ属性を持っています。 Sport トピックには管理トピック・オブジェクトがなく、 SYSTEM.BASE.TOPIC。

トピック・ツリーのルート・ノード (SYSTEM.BASE.TOPIC) で mqm 以外のユーザーに権限を付与することはお勧めできません。権限は継承されますが、制限はできないためです。 したがって、このレベルで権限を付与することは、ツリー全体に権限を付与することを意味します。 階層の下層にあるトピック・レベルで権限を付与してください。

トピック・ツリー内の特定のノードに特定の属性を定義するために、管理トピック・オブジェクトを使用できます。 次の例で、管理トピック・オブジェクトは「Soccer」トピックの 永続サブスクリプション・プロパティー DURSUB を値 NO に設定するように定義されています。

```
DEFINE TOPIC(FOOTBALL.EUROPEAN)
TOPICSTR('Sport/Soccer')
DURSUB(NO)
DESCR('Administrative topic object to disallow durable subscriptions')
```

このトピック・ツリーは、以下のように視覚化できます。

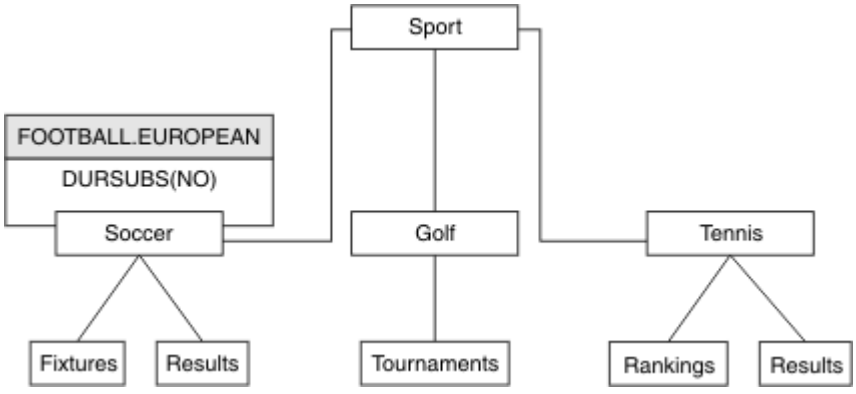


図 *21.* 「*Sport/Soccer*」トピックに管理トピック・オブジェクトを関連付けたトピック・ツリーを視覚化した図

ツリー内の「Soccer」の下にあるトピックにサブスクライブするアプリケーションは、管理トピック・オブジェクトを追加する前に使用していたトピック・ストリングを引き続き使用できます。 ただし、ストリング /Sport/Soccer の代わりにオブジェクト名 FOOTBALL.EUROPEAN を使用してサブスクライブするようにアプリケーションを作成できるようになりました。 例えば、/Sport/Soccer/Results にサブスクライブするアプリケーションでは、MQSD.ObjectName として FOOTBALL.EUROPEAN、MQSD.ObjectString として Results を指定できます。

この機能を使用すると、トピック・ツリーの一部をアプリケーション開発者から隠すことができます。 トピック・ツリーの特定のノードで管理トピック・オブジェクトが定義された後、 アプリケーション開発者は独自のトピックをそのノードの子として定義できます。 開発者は親トピックについて知る必要がありますが、親ツリーの他のノードについて知る必要はありません。

属性の継承

トピック・ツリーに多数の管理トピック・オブジェクトがある場合、 各管理トピック・オブジェクトは、デフォルトで直近の親管理トピックから属性を継承します。 80 ページの図 22 は、前の例を拡張したものです。

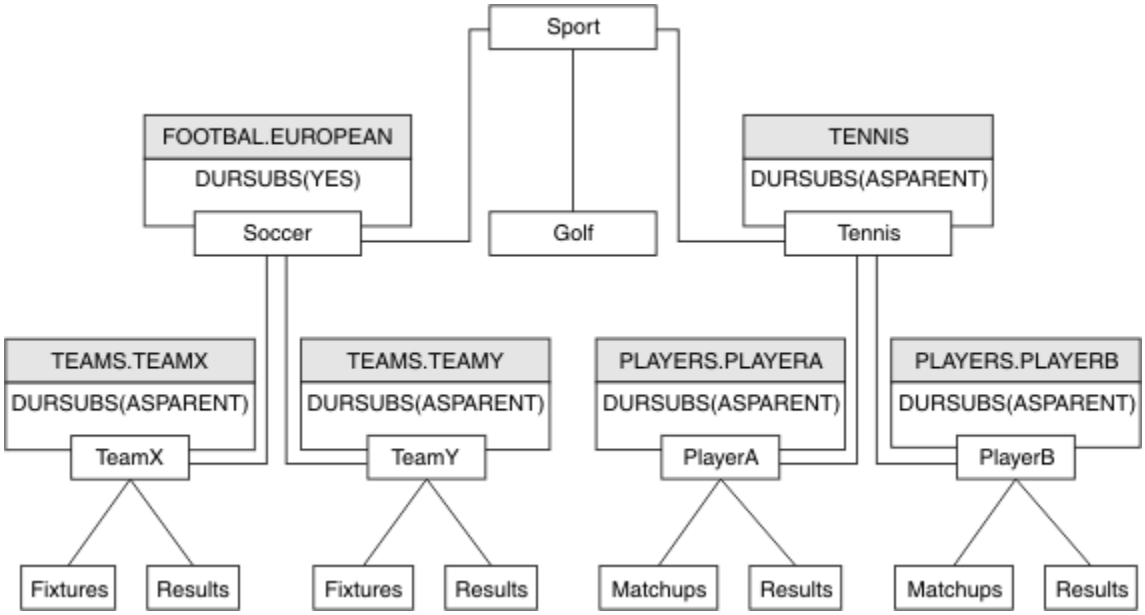


図 *22.* 複数の管理トピック・オブジェクトを定義したトピック・ツリー

例えば、継承を使用して、 /Sport/Soccer のすべての子トピックに、 サブスクリプションが非永続であることを示すプロパティーを設定するとします。 これには、FOOTBALL.EUROPEAN の DURSUB 属性を NO に変更します。

その属性を設定するには、以下のコマンドを使用できます。

```
ALTER TOPIC(FOOTBALL.EUROPEAN) DURSUB(NO)
```

Sport/Soccer の子トピックのすべての管理トピック・オブジェクトは、 プロパティー DURSUB がデフォルト値 ASPARENT に設定されています。 FOOTBALL.EUROPEAN の DURSUB プロパティー値を NO に変更すると、 Sport/Soccer の子トピックは DURSUB プロパティー値 NO を継承します。 Sport/Tennis のすべての子トピックは、 SYSTEM.BASE.TOPIC オブジェクトから DURSUB の値を継承します。 SYSTEM.BASE.TOPIC の値は、YES です。

この状態でトピック Sport/Soccer/TeamX/Results に対する永続サブスクリプションを作成しようとすると失敗しますが、Sport/Tennis/PlayerB/Results に対する永続サブスクリプションを作成しようとする操作は成功します。

WILDCARD プロパティーによるワイルドカードの使用の制御

MQSC **Topic** WILDCARD プロパティーまたは 同等の PCF Topic `WildcardOperation` プロパティーを使用すると、 ワイルドカード・トピック・ストリング名を使用するサブスクライバー・アプリケーションへのパブリケーションの送達を制御できます。 WILDCARD プロパティーには、以下の 2 つの値のいずれかを指定できます。

WILDCARD
このトピックに対するワイルドカード・サブスクリプションの動作。

PASSTHRU
このトピック・オブジェクトのトピック・ストリングよりも具体的でないワイルドカード・トピックに対するサブスクリプションは、そのトピックまたはそのトピックよりも具体的なトピック・ストリングに対するパブリケーションを受信できるようになります。

BLOCK
このトピック・オブジェクトのトピック・ストリングよりも具体的でないワイルドカード・トピックに対するサブスクリプションは、このトピックまたはこのトピックよりも具体的なトピック・ストリングに対するパブリケーションを受信できなくなります。

サブスクリプションが定義されている場合に、この属性の値が使用されます。 この属性を変更しても、既存のサブスクリプションによってカバーされているトピック・セット は、変更による影響を受けません。 このシナリオは、トピック・オブジェクトが作成または削除されてトポロジーが変更された場合にも当てはまります。WILDCARD 属性の変更後に作成されたサブスクリプションに一致するトピックのセットは、変更後のトポロジーを使用して作成されます。 既存のサブスクリプションについて、一致するトピック・セットを強制的に再評価する場合は、キュー・マネージャーを再開する必要があります。

85 ページの『例: Sport パブリッシュ/サブスクライブ・クラスターを作成する』 の例では、 81 ページの図 23 で示されるトピック・ツリー構造を作成するステップに従うことができます。

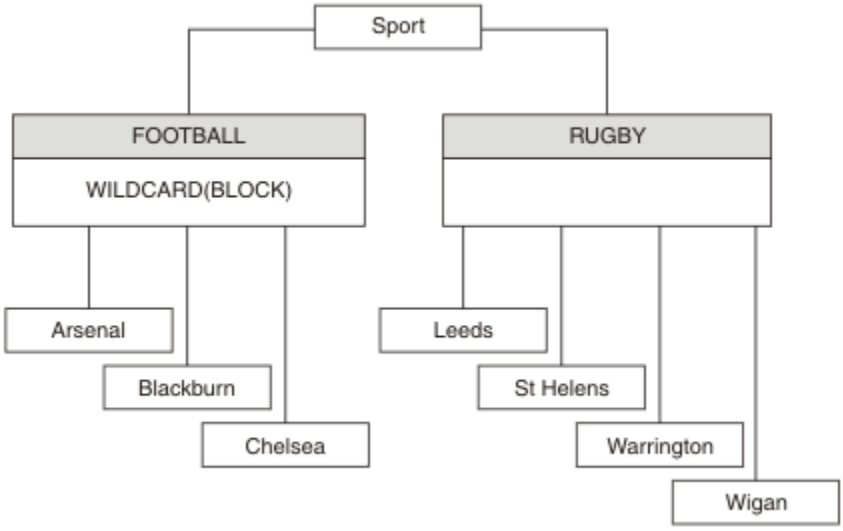


図 *23. WILDCARD* プロパティー *BLOCK* を使用するトピック・ツリー

ワイルドカード・トピック・ストリング `#` を使用するサブスクライバーは、 `Sport` トピックと `Sport/Rugby` サブツリーへのすべてのパブリケーションを受け取ります。 `Sport/Football` トピックの `WILDCARD` プロパティー値が `BLOCK` であるため、このサブスクライバーは `Sport/Football` サブツリーへのパブリケーションは受け取りません。

`PASSTHRU` は、デフォルトの設定値です。 `Sport` ツリーのノードには、`WILDCARD` プロパティー値 `PASSTHRU` を設定できます。 ノードで `WILDCARD` プロパティー値 `BLOCK` が設定されていない場合、`PASSTHRU` を設定しても、`Sports` ツリーのノードのサブスクライバーによって 観測される動作が変化することはありません。

この例では、サブスクリプションを作成して、 送達されるパブリケーションにワイルドカード設定が与える影響を確認します。 86 ページの図 27 を参照してください。 87 ページの図 30 でパブリッシュ・コマンドを実行して、 パブリケーションをいくつか作成します。

```
pub QMA
```

図 *24. QMA* へのパブリッシュ

82 ページの表 3 では結果が示されています。 WILDCARD プロパティー値 BLOCK を設定すると、 ワイルドカードを含むサブスクリプションは、 そのワイルドカードの有効範囲内にあるトピックへのパブリケーションを受信しなくなることに注意してください。

表 *3. QMA* で受信されたパブリケーション

サブスクリプション	トピック・ストリング	受信されたパブリケーション	注
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	「Football」サブツリーへのすべてのパブリケーションは、 Sports/Football の WILDCARD(BLOCK) によってブロックされます。
SARSENAL	Sports/#/Arsenal	-	Sports/Football の WILDCARD(BLOCK) により、 Arsenal でのワイルドカード・サブスクリプションは防止されます。
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	「Sports/Rugby」のデフォルトである WILDCARD は、Leeds でのワイルドカード・サブスクリプションを防止しません。

注 :

あるサブスクリプションに、WILDCARD プロパティー値 BLOCK を持つトピック・オブジェクトに一致するワイルドカードがあるとします。 このサブスクリプションで、一致するワイルドカードの右側にトピック・ストリングもある場合、 サブスクリプションがパブリケーションを受信することはありません。 ブロックされないパブリケーションのセットは、 ブロックされたワイルドカードの親であるトピックへのパブリケーションです。 BLOCK プロパティー値を持つトピックの子であるトピックへのパブリケーションは、 ワイルドカードによってブロックされます。 したがって、ワイルドカードの右側にトピックが含まれるサブスクリプション・トピック・ストリングは、 一致するパブリケーションを受信することがありません。

WILDCARD プロパティー値を BLOCK に設定しても、ワイルドカードが含まれるトピック・ストリングを使用したサブスクライブが実行できなくなるわけではありません。 このようなサブスクリプションは正常に行われます。 このサブスクリプションには、 WILDCARD プロパティー値 BLOCK を持つトピック・オブジェクトを含むトピックに一致する 明示的なトピックが含まれます。 これは、WILDCARD プロパティー値 BLOCK を持つトピックの親または子であるトピック用に、 ワイルドカードを使用します。 81 ページの図 23 の例では、 Sports/Football/# のようなサブスクリプションがパブリケーションを受信できます。

ワイルドカードとクラスター・トピック

クラスター・トピック定義はクラスター内のすべてのキュー・マネージャーに伝搬されます。 クラスター内のキュー・マネージャーでクラスター・トピックへのサブスクリプションを行うと、 そのキュー・マネージャーで複数のプロキシー・サブスクリプションが作成されます。 クラスター内の他のすべてのキュー・マネージャーで 1 つのプロキシー・サブスクリプションが作成されます。 ワイルドカードを含むトピック・ストリングを使用したサブスクリプションをクラスター・トピックと組み合わせると、 動作の予測が難しくなる可能性があります。 次の例は、この動作について説明しています。

85 ページの『例: Sport パブリッシュ/サブスクライブ・クラスターを作成する』 の例にあるクラスター・セットアップでは、QMB が QMA と同じサブスクリプションのセットを持っているにもかかわらず、パブリッシャーが QMA にパブリッシュした後で QMB はパブリケーションを 受信しませんでした (82 ページの図 24 を参照)。 Sports/Football と Sports/Rugby のトピックは クラスター・トピックですが、fullsubs.tst で定義されているサブスクリプションは クラスター・トピックを参照しません。 QMB から QMA に伝搬されるプロキシー・サブスクリプションはありません。 プロキシー・サブスクリプションがないと、QMA へのパブリケーション は QMB に転送されません。

Sports/#/Leeds などの一部のサブスクリプションは、クラスター・トピック (このケースでは Sports/Rugby) を参照するように見えることがあります。 実際には、Sports/#/Leeds サブスクリプション はトピック・オブジェクト SYSTEM.BASE.TOPIC に解決されます。

Sports/#/Leeds などのサブスクリプションによって参照されるトピック・オブジェクトを解決するための規則は、 以下のとおりです。 トピック・ストリングが最初のワイルドカードの位置まで切り捨てられます。 トピック・ストリングを左方向にスキャンし、 関連付けられた管理トピック・オブジェクトを持つ最初のトピックを探します。 そのトピック・オブジェクトがクラスター名を指定するか、ローカル・トピック・オブジェクトを定義している可能性があります。 Sports/#/Leeds の例では、切り捨て後のトピック・ストリングは Sports であり、 トピック・オブジェクトを持たないため、Sports/#/Leeds は ローカル・トピック・オブジェクトである SYSTEM.BASE.TOPIC から継承します。

クラスター化されたトピックのサブスクライブによってワイルドカード伝搬の動作が変更される仕方を確認するには、 バッチ・スクリプト upsubs.bat を実行します。 このスクリプトはサブスクリプション・キューをクリアし、 fullsubs.tst 内にクラスター・トピック・サブスクリプションを追加します。 パブリケーションのバッチを作成するには、 puba.bat を再び実行します (82 ページの図 24 を参照してください)。

83 ページの表 4 は、パブリケーションがパブリッシュされたのと同じキュー・マネージャーに 2 つの新規サブスクリプションを追加した結果を示しています。 結果は予測どおりであり、新規サブスクリプションはそれぞれ 1 つのパブリケーションを受信し、 他のサブスクリプションによって受信されるパブリケーションの数は変更されません。 他のクラスター・キュー・マネージャーでは予測しない結果が発生します。84 ページの表 5 を参照してください。

表 *4. QMA* で受信されたパブリケーション

サブスクリプション	トピック・ストリング	受信されたパブリケーション	注
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	「Football」サブツリーへのすべてのパブリケーションは、 Sports/Football の WILDCARD(BLOCK) によってブロックされます。
SARSENAL	Sports/#/Arsenal	-	Sports/Football の WILDCARD(BLOCK) により、Arsenal でのワイルドカード・サブスクリプションは防止されます。
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	「Sports/Rugby」のデフォルトである WILDCARD は、Leeds でのワイルドカード・サブスクリプションを防止しません。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	サブスクリプションにワイルドカードがないため、 Arsenal はパブリケーションを受信します。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds は、いかなるイベントでもパブリケーションを受信します。

84 ページの表 5 は、 QMB で 2 つの新規サブスクリプションを追加して QMA でパブリッシュした結果を示しています。 これら 2 つの新規サブスクリプションがないときに QMB はパブリケーションを受信しませんでした。 Sports/FootBall と Sports/Rugby はどちらもクラスター・トピックであるため、 これら 2 つの新規サブスクリプションは予測どおりパブリケーションを受信します。 QMB から Sports/Football/Arsenal と Sports/Rugby/Leeds 用のプロキシー・サブスクリプション が QMA に転送された後、 そこからパブリケーションが QMB に送信されました。

予測しなかった結果として、 以前はパブリケーションを受信しなかった 2 つのサブスクリプション (Sports/# と Sports/#/Leeds) が、 パブリケーションを受信するようになりました。 これは、他のサブスクリプション用に QMB に転送されたパブリケーションである Sports/Football/Arsenal と Sports/Rugby/Leeds が、QMB に接続された任意のサブスクライバーから使用可能になったためです。 その結果、ローカル・トピックの Sports/# と Sports/#/Leeds へのサブスクリプションは、 Sports/Rugby/Leeds パブリケーションを受信します。 「Sports/Football」は独自の WILDCARD プロパティー値が BLOCK に設定されているため、Sports/#/Arsenal は引き続きパブリケーションを受信しません。

表 *5. QMB* で受信されたパブリケーション

サブスクリプション	**トピック・ストリング**	**受信されたパブリケーション**	**注**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*WILDCARD(BLOCK) on Sports/Football* によってブロックされる、フットボール・サブツリーへのすべてのパブリケーション
SARSENAL	Sports/#/Arsenal	-	Sports/Football の WILDCARD(BLOCK) により、 Arsenal でのワイルドカード・サブスクリプションは防止されます。
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	「*Sports/Rugby*」のデフォルトである *WILDCARD* は、*Leeds* でのワイルドカード・サブスクリプションを防止しません。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	サブスクリプションにワイルドカードがないため、 Arsenal はパブリケーションを受信します。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds は、いかなるイベントでもパブリケーションを受信します。

ほとんどのアプリケーションにおいて、 あるサブスクリプションが別のサブスクリプションの動作に影響することは望ましくありません。 WILDCARD プロパティーの値 BLOCK の重要な使用法の 1 つは、 ワイルドカードを含む同じトピック・ストリングへのサブスクリプションをすべて同じように動作させることです。 サブスクリプションがパブリッシャーと同じキュー・マネージャー上にあるか別のキュー・マネージャー上にあるかにかかわらず、 サブスクリプションの結果は同じです。

ワイルドカードとストリーム

パブリッシュ/サブスクライブ API に書き込まれた新規アプリケーションの場合、 結果として、* へのサブスクリプションがパブリケーションを受信しなくなります。 「Sports」へのすべてのパブリケーションを受信するには、Sports/* または Sports/# にサブスクライブするとともに、Business パブリケーションについても同様の処理を行う必要があります。

既存のキュー型パブリッシュ/サブスクライブ・アプリケーションの動作は、パブリッシュ/サブスクライブ・ブローカーを新しいバージョンの IBM MQ に移行しても変更されません。 **Publish**、**Register Publisher**、または **Subscriber** コマンドの **StreamName** プロパティーは、ストリームの移行先のトピックの名前にマップされます。

ワイルドカードとサブスクリプション・ポイント

パブリッシュ/サブスクライブ API に書き込まれた新規アプリケーションの場合、 移行の結果、* へのサブスクリプションがパブリケーションを受信しなくなります。 「Sports」へのすべてのパブリケーションを受信するには、Sports/* または Sports/# にサブスクライブするとともに、Business パブリケーションについても同様の処理を行う必要があります。

既存のキュー型パブリッシュ/サブスクライブ・アプリケーションの動作は、パブリッシュ/サブスクライブ・ブローカーを新しいバージョンの IBM MQ に移行しても変更されません。 **Publish**、**Register Publisher**、または **Subscriber** コマンドの **SubPoint** プロパティーは、サブスクリプションの移行先のトピックの名前にマップされます。

例: Sport パブリッシュ/サブスクライブ・クラスターを作成する

以降のステップでは、クラスター CL1 と ともに 4 つのキュー・マネージャー (CL1A および CL1B という 2 つの全リポジトリーと、 QMA および QMB という 2 つの部分リポジトリー) を作成します。 全リポジトリーは、クラスター定義を保持するためにのみ使用されます。 QMA は、クラスター・トピック・ホストに指定されます。 永続サブスクリプションは QMA と QMB の両方で定義されます。

注 : この例は、Windows 用にコーディングされています。 他のプラットフォームでこの例を構成してテストするには、Create qmgrs.bat と create pub.bat を再コーディングする必要があります。

1. 以下のスクリプト・ファイルを作成します。
    a. Create topics.tst
    b. Create wildsubs.tst
    c. Create fullsubs.tst
    d. Create qmgrs.bat
    e. create pub.bat
2. Create qmgrs.bat を実行して構成を作成します。

```
qmgrs
```

81 ページの図 23 でトピックを作成します。 図 5 のスクリプトは、クラスター・トピック Sports/Football および Sports/Rugby を作成します。

注 : REPLACE オプションは、トピックの TOPICSTR プロパティーを置き換えません。 TOPICSTR は、この例でさまざまなトピック・ツリーをテストするために役立つプロパティーです。 トピックを変更するには、最初にトピックを削除します。

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

図 *25.* トピックの削除と作成*: topics.tst*

注 : REPLACE によってトピック・ストリングが置き換えられることはないため、トピックを削除します。

ワイルドカードが含まれるサブスクリプションを作成します。 ワイルドカードは、81 ページの図 23 で示されているトピック・オブジェクトを持つトピックに対応します。 各サブスクリプション用のキューを作成します。 スクリプトが実行または再実行されると、キューがクリアされてサブスクリプションは削除されます。

注 : REPLACE オプションによって サブスクリプションの TOPICOBJ または TOPICSTR プロパティーが置き換えられることはありません。 TOPICOBJ または TOPICSTR は、さまざまなサブスクリプションをテストするために変更される便利なプロパティーです。 これらを変更するには、最初にサブスクリプションを削除します。

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

図 *26.* ワイルドカード・サブスクリプションの作成*: wildsubs.tst*

クラスター・トピック・オブジェクトを 参照するサブスクリプションを作成します。

注 :

TOPICOBJ によって参照されるトピック・ストリング と TOPICSTR によって定義されるトピック・ストリングの間には、 デリミッター / が自動的に挿入されます。

定義 DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) は、同じサブスクリプションを作成します。 TOPICOBJ は、すでに定義したトピック・ストリングを迅速に参照する方法として使用されます。 サブスクリプションは作成された後、トピック・オブジェクトを参照しなくなります。

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

図 *27.* サブスクリプションの削除と作成*: fullsubs.tst*

2 つのリポジトリーが含まれる クラスターを作成します。 パブリッシュとサブスクライブ用に 2 つの部分リポジトリーを作成します。 すべてを削除してやり直すには、スクリプトを再実行します。 このスクリプトでは、トピック階層と初期ワイルドカード・サブスクリプションも作成されます。

注 :

他のプラットフォームでは、同様のスクリプトを作成するか、すべてのコマンドを入力します。 スクリプトを使用すると、迅速にすべてを削除して同一の構成でやり直すことができます。

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

図 *28.* キュー・マネージャーの作成*: qmgrs.bat*

サブスクリプションをクラスター・トピックに追加して、構成を更新します。

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

図 *29.* サブスクリプションの更新*: upsubs.bat*

パブリケーション・トピック・ストリングが含まれる メッセージをパブリッシュするには、キュー・マネージャーをパラメーターとして pub.bat を実行します。 Pub.bat は、サンプル・プログラム **amqspub** を使用します。

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

図 *30.* パブリッシュ*: pub.bat*

ストリームおよびトピック

キュー型パブリッシュ/サブスクライブには、統合パブリッシュ/サブスクライブ・モデルには存在しないパブリケーション・ストリームの概念があります。 キュー型パブリッシュ/サブスクライブにおいてストリームとは、異なるトピックの情報の流れを分離する手段を提供するものです。 ストリームは、管理上別のトピック ID にマップできる最上位トピックとして実装されます。

ネットワーク上のすべてのブローカーおよびキュー・マネージャーに対して、デフォルトのストリーム SYSTEM.BROKER.DEFAULT.STREAM が自動的にセットアップされます。 このデフォルトのストリームを使用するために追加の構成を行う必要はありません。 デフォルトのストリームを、名前の付けられていないデフォルトのトピック・スペースであると考えてみます。 デフォルト・ストリームにパブリッシュされたトピックは、キューに入れられたパブリッシュ/サブスクライブを有効にすると、接続されているすべてのキュー・マネージャーですぐに使用可能になります。 名前付きのストリームは、名前付きの別個のトピ

ック・スペースに類似しています。 名前付きのストリームは、それを使用するブローカーごとに定義する必要があります。

パブリッシャーとサブスクライバーが別々のキュー・マネージャー上にあり、それらのブローカーが同じブローカー階層内で接続した場合、追加で構成を行う必要なしに、パブリケーションおよびサブスクリプションはブローカー間を流れます。 逆の場合にも同じインターオペラビリティーが働きます。

名前付きストリーム

キュー型パブリッシュ/サブスクライブ・プログラミング・モデルを使用するソリューション・デザイナーが、すべてのスポーツ・パブリケーションを Sport という 1 つの名前付きストリーム内に入れることにしたとします。 キューに入れられたパブリッシュ/サブスクライブが有効になっている IBM MQ 上で実行されるキュー・マネージャーがストリームを使用できるようにするには、ストリームを手動で追加する必要があります。

ストリーム Sport の Soccer/Results にサブスクライブするキュー型パブリッシュ/サブスクライブ・アプリケーションは、変更なしで機能します。 MQSUB を使用してトピック Sport にサブスクライブし、トピック・ストリング Soccer/Results を提供する統合パブリッシュ/サブスクライブ・アプリケーションも、同じパブリケーションを受け取ります。

ストリームを追加する作業の説明は、ストリームの追加のトピックに記載されています。 ストリームを手動で追加しなければならない場合、それには以下の 2 つの理由があります。

1. 後のバージョンのキュー・マネージャー上で作動するキュー型パブリッシュ/サブスクライブ・アプリケーションを引き続き開発し、それらのアプリケーションを統合パブリッシュ/サブスクライブ MQI インターフェースには移行しない場合。
2. トピックに対するストリームのデフォルトのマッピングによりトピック・スペース内で「衝突」が発生し、ストリーム上のパブリケーションに、他の場所にあるパブリケーションと同じトピック・ストリングがある場合。

権限

デフォルトでは、トピック・ツリーのルートに複数のトピック・オブジェクト SYSTEM.BASE.TOPIC, SYSTEM.BROKER.DEFAULT.STREAM および SYSTEM.BROKER.DEFAULT.SUBPOINT があります。 権限 (例えば、パブリッシュやサブスクライブ) は、SYSTEM.BASE.TOPIC 上の権限によって決定されます。 SYSTEM.BROKER.DEFAULT.STREAM または SYSTEM.BROKER.DEFAULT.SUBPOINT 上の権限はどれも無視されます。 SYSTEM.BROKER.DEFAULT.STREAM または SYSTEM.BROKER.DEFAULT.SUBPOINT のいずれかが削除され、空ではないトピック・ストリングで再作成された場合、これらのオブジェクトに定義された権限は通常のトピック・オブジェクトと同じ方法で使用されます。

ストリームとトピックの間のマッピング

キューに入れられたパブリッシュ/サブスクライブ・ストリームは、キューを作成し、ストリームと同じ名前を付けることによって、 IBM MQ で模倣されます。 このキューはストリーム・キューと呼ばれることがあります。 キュー型パブリッシュ/サブスクライブ・アプリケーションではそのように見えるためです。 このキューを SYSTEM.QPUBSUB.QUEUE.NAMELIST という特別な名前リストに追加すると、パブリッシュ/サブスクライブ・エンジンがこのキューを識別します。 この名前リストに特別なキューを追加することによって、ストリームを必要な数だけ追加できます。 最後に、トピックにパブリッシュおよびサブスクライブできるように、ストリームと同じ名前、およびストリーム名と同じトピック・ストリングを持つトピックを追加する必要があります。

ただし、例外的な状況では、トピックを定義する際に、ストリームに対応するトピックに対して、任意のトピック・ストリングを指定できます。 トピック・ストリングの目的は、トピックに、そのトピック・スペース内で固有の名前を付けることです。 通常、その目的は、ストリーム名によって完全に果たされます。 時折、ストリーム名と既存のトピック名が衝突する場合があります。 この問題を解決するには、ストリームに関連付けられたトピックに対して別のトピック・ストリングを選択します。 いずれかのトピック・ストリングを、固有であることを確認して選択してください。

トピック定義で定義されたトピック・ストリングが、パブリッシャーおよびサブスクライバーが MQOPEN MQI 呼び出しまたは MQSUB MQI 呼び出しを使用して指定したトピック・ストリングに通常の方法で接頭部として付加されます。 トピック・オブジェクトを使用してトピックを参照するアプリケーションは、接頭部のトピック・ストリングの選択によって影響を受けることはありません。そのため、トピック・スペース内でパブリケーションが固有になるような任意のトピック・ストリングを選択できます。

各ストリームの各トピックへの再マップは、トピック・ストリングを固有にするために使用されている接頭部によって、あるトピック・セットが他のトピックと完全に区別されることを前提としています。 マッピングを機能させるために厳密に順守する、共通のトピック命名規則を定義する必要があります。

IBM MQ では、接頭部を付けるメカニズムを使用して、トピック・ストリングをトピック・スペース内の別の場所に再マップします。

注 : ストリームを削除するときには、最初に、そのストリームでのすべてのサブスクリプションを削除してください。 サブスクリプションのいずれかが、ブローカー階層内の他のブローカー由来のものである場合には、このアクションが最も重要です。

サブスクリプション・ポイントおよびトピック

名前付きサブスクリプション・ポイントはトピックとトピック・オブジェクトでエミュレートされます。

サブスクリプション・ポイントを手動で追加する場合は、サブスクリプション・ポイントの追加を参照してください。

IBM MQ におけるサブスクリプション・ポイント

IBM MQ は、IBM MQ トピック・ツリー内のさまざまなトピック・スペースにサブスクリプション・ポイントをマップします。 サブスクリプション・ポイントのないコマンド・メッセージのトピックは、未変更のまま IBM MQ トピック・ツリーのルートにマップされ、SYSTEM.BASE.TOPIC からプロパティーを継承します。

サブスクリプション・ポイントを持つコマンド・メッセージは、SYSTEM.QPUBSUB.SUBPOINT.NAMELIST のトピック・オブジェクトのリストを使用して処理されます。 コマンド・メッセージ内のサブスクリプション・ポイント名は、リスト内のトピック・オブジェクトそれぞれのトピック・ストリングと突き合わされます。 一致が検出されると、サブスクリプション・ポイント名がトピック・ノードとしてトピック・ストリングの前に付加されます。 このトピックは、SYSTEM.QPUBSUB.SUBPOINT.NAMELIST で検出された関連トピック・オブジェクトからそのプロパティーを継承します。

サブスクリプション・ポイントを使用することの効果は、サブスクリプション・ポイントごとに別々のトピック・スペースを作成する点にあります。 サブスクリプション・ポイントと名前が同じトピックが、トピック・スペースのルートになります。 各トピック・スペース内のトピックは、サブスクリプション・ポイントと名前が同じトピック・オブジェクトからそれぞれのプロパティーを継承します。

一致するトピック・オブジェクトに設定されていないプロパティーは、SYSTEM.BASE.TOPIC からの通常の方法で継承されます。

MQRFH2 メッセージ・ヘッダーを使用する、キューに入れられた既存のパブリッシュ/サブスクライブ・アプリケーションは、Publish または Register subscriber コマンド・メッセージで **SubPoint** プロパティーを設定することによって処理を続行します。 サブスクリプション・ポイントがコマンド・メッセージ内のトピック・ストリングと結合され、結果のトピックは他の場合と同様に処理されます。

IBM MQ アプリケーションは、サブスクリプション・ポイントの影響を受けません。 アプリケーションが、一致するトピック・オブジェクトのいずれかから情報を継承するトピックを使用する場合、そのアプリケーションは、一致するサブスクリプション・ポイントを使用するキュー型アプリケーションと相互運用できます。

例

既存の WebSphere Message Broker (現在は IBM Integration Bus と呼ばれる) IBM MQ に移行されたパブリッシュ/サブスクライブ・アプリケーションは、 GBP と USD の 2 つのトピック・オブジェクトを、対応するトピック・ストリング 'GBP' と 'USD'を使用して作成しました。

サブスクリプション・ポイント NYSE/IBM/SPOT を使用する IBM MQ 上で実行されるように移行された、トピック「USD」に対する既存のパブリッシャーは、トピック USD/NYSE/IBM/SPOT 上にパブリケーションを作成します。 同様に、サブスクリプション・ポイント NYSE/IBM/SPOT を使用して、USD への既存のサブスクライバーを USD/NYSE/IBM/SPOT に作成します。

MQSUB を呼び出して、IBM MQ パブリッシュ/サブスクライブ・プログラムでドル・スポット価格をサブスクライブします。「C」コード・フラグメントで説明されているように、USD トピック・オブジェクトおよびトピック・ストリング 'NYSE/IBM/SPOT' を使用してサブスクリプションを作成します。

```
strncpy(sd.ObjectName, "USD", MQ_TOPIC_NAME_LENGTH);
sd.ObjectString.VSPtr = "NYSE/IBM/SPOT";
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

1. クラスター・トピック・ホスト上の USD および GBP トピック・オブジェクトの CLUSTER 属性を設定します。
2. クラスター内の他のキュー・マネージャー上の USD および GBP トピック・オブジェクトのコピーをすべて削除します。
3. USD および GBP が、クラスター内のすべてのキュー・マネージャーで SYSTEM.QPUBSUB.SUBPOINT.NAMELIST に定義されていることを確認してください。

単一キュー・マネージャーのパブリッシュ/サブスクライブ構成の例

91 ページの図 31 は、基本的な単一キュー・マネージャーのパブリッシュ/サブスクライブ構成を表しています。 この例はニュース・サービスの構成を示すもので、ここではパブリッシャーからいくつかのトピックについての情報が提供されています。

- パブリッシャー 1 は、トピック「Sport」を使用してスポーツの試合結果の情報をパブリッシュしています。
- パブリッシャー 2 は、トピック「Stock」を使用して株価の情報をパブリッシュしています。
- パブリッシャー 3 はトピック「Films」を使用して映画のレビュー情報をパブリッシュし、トピック「TV」を使用してテレビ番組表をパブリッシュしています。

3 人のサブスクライバーは、それぞれ関心のある別々のトピックに登録しているため、キュー・マネージャーがそれぞれが興味を持っている情報を送信します。

- サブスクライバー 1 はスポーツの試合結果と株価を受信します。
- サブスクライバー 2 は映画のレビューを受信します。
- サブスクライバー 3 はスポーツの試合結果を受信します。

テレビ番組表に登録しているサブスクライバーはいないので、テレビ番組表は配布されません。

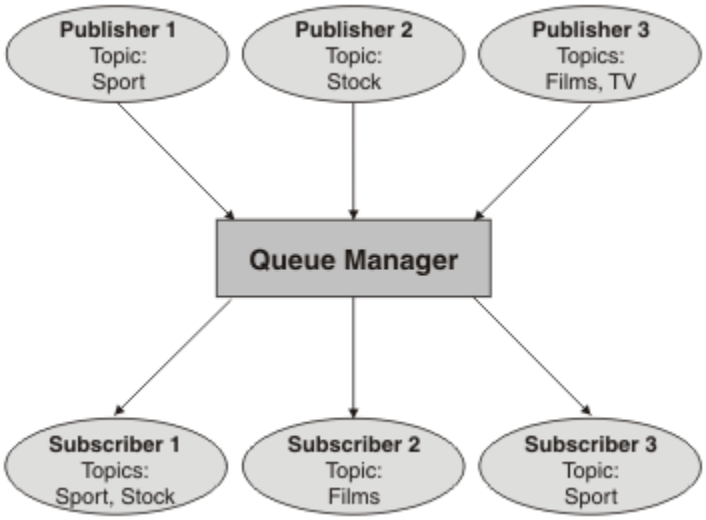


図 *31.* 単一キュー・マネージャーのパブリッシュ/サブスクライブの例

分散パブリッシュ/サブスクライブのネットワーク

各キュー・マネージャーは、トピックにパブリッシュされたメッセージと、そのトピックをサブスクライブしている、ローカルに作成されたサブスクリプションのマッチングを行います。 あるキュー・マネージャーに接続したアプリケーションからパブリッシュされたメッセージが、ネットワーク内の他のキュー・マネージャー上に作成された一致するサブスクリプションに配信されるように、キュー・マネージャーのネットワークを構成することができます。 このためには、キュー・マネージャー間のシンプルなチャネルを介する追加の構成が必要になります。

分散パブリッシュ/サブスクライブ構成とは、一連のキュー・マネージャーを接続したものです。 それらのキュー・マネージャーすべてを同じ物理システム上に配置することもできますし、いくつかの物理システムに分散することも可能です。 キュー・マネージャーどうしを接続すると、サブスクライバーは、1 つのキュー・マネージャーにサブスクライブした状態で、最初は別のキュー・マネージャーにパブリッシュされていたメッセージを受信することができます。 これを例示するため、次の図では、90 ページの『単一キュー・マネージャーのパブリッシュ/サブスクライブ構成の例』で説明した構成にキュー・マネージャーをもう 1 つ追加しています。

- キュー・マネージャー 2 は、パブリッシャー 4 が天気予報情報 (トピック「Weather」を使用) と主要道路の交通情報 (トピック「Traffic」を使用) をパブリッシュするために使用されます。
- サブスクライバー 4 もこのキュー・マネージャーを使用して、トピック「Traffic」を使用した交通情報にサブスクライブします。
- サブスクライバー 3 は、パブリッシャーからの別のキュー・マネージャーを使用しますが、やはり天気情報にサブスクライブします。 キュー・マネージャーが互いにリンクしているので、このようなことが可能になります。

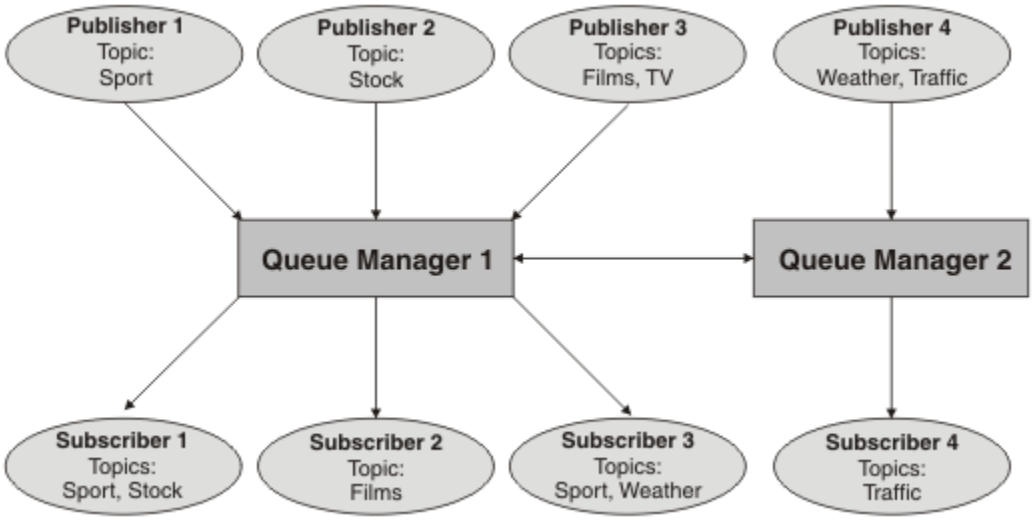


図 *32.* キュー・マネージャーが *2* つあるパブリッシュ/サブスクライブの例

手動によりキュー・マネージャーを親子階層で接続するか、パブリッシュ/サブスクライブ・クラスターを作成して IBM MQ に接続の詳細の大部分を自動定義させることができます。 両方のトポロジーを組み合わせて使用することもできます。 例えば、複数のクラスターを階層にして結合できます。

パブリッシュ/サブスクライブ・クラスターの概要

パブリッシュ/サブスクライブ・クラスターは、標準的なクラスターに、1 つ以上のトピック・オブジェクトが追加されたものです。 クラスター内のいずれかのキュー・マネージャーに管理トピック・オブジェクトを定義し、クラスター名を指定してそのトピック・オブジェクトをクラスター化すると、そのトピックのパブリッシャーとサブスクライバーは、クラスター内の任意のキュー・マネージャーに接続でき、パブリッシュされたメッセージは、キュー・マネージャー間のクラスター・チャネルを介してサブスクライバーにルーティングされます。

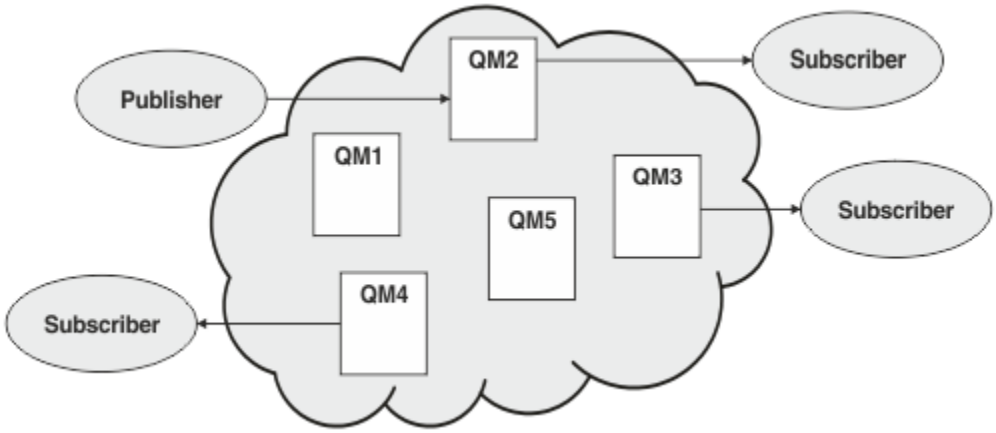


図 *33.* パブリッシュ/サブスクライブ・クラスター

クラスター内でどのようにパブリッシュ/サブスクライブ・メッセージをルーティングするかは、次の 2 つの方法で構成できます。

- 直接ルーティング (direct routing)
- トピック・ホスト・ルーティング (topic host routing)

直接ルーティング型のクラスター・トピックを構成すると、あるキュー・マネージャーでパブリッシュされたメッセージは、そのキュー・マネージャーからクラスター内の他のすべてのキュー・マネージャーのすべてのサブスクリプションに直接送信されます。 こうすると、最短パスによるパブリケーションが可能ですが、クラスター内のすべてのキュー・マネージャーが、そのキュー・マネージャー以外のすべてのキ

ュー・マネージャーを認識するようになるため、それぞれにクラスター・チャネルをキュー・マネージャー間に確立する可能性があります。

トピック・ホスト・ルーティングを使用する場合、あるキュー・マネージャーでパブリッシュされたメッセージは、そこから、管理対象トピック・オブジェクトの定義をホストするキュー・マネージャーに送信されます。 そのトピック・ホスト・キュー・マネージャー が、クラスター内の他のすべてのキュー・マネージャーのすべてのサブスクリプションにメッセージをルーティングします。 トピック・ホスト・キュー・マネージャー上にパブリッシャーまたはサブスクライバーが存在しないと、パブリケーションの経路が長くなります。 しかし、トピック・ホスト・キュー・マネージャーのみが、クラスター内の他のすべてのキュー・マネージャーを認識してキュー・マネージャー間にクラスター・チャネルを確立する可能性があるという利点があります。

詳しくは、94 ページの『パブリッシュ/サブスクライブ・クラスター』 を参照してください。

パブリッシュ/サブスクライブ階層の概要

パブリッシュ/サブスクライブ階層とは、一連のキュー・マネージャーをチャネルで接続して階層構造にしたものです。 パブリッシュ/サブスクライブ階層へのキュー・マネージャーの接続で説明されているように、各キュー・マネージャーはその 親 キュー・マネージャーを識別します。

トピックのパブリッシャーとサブスクライバーは、階層内の任意のキュー・マネージャーに接続できます。また、メッセージは、階層のキュー・マネージャー接続を使用してキュー・マネージャー間を流れます。

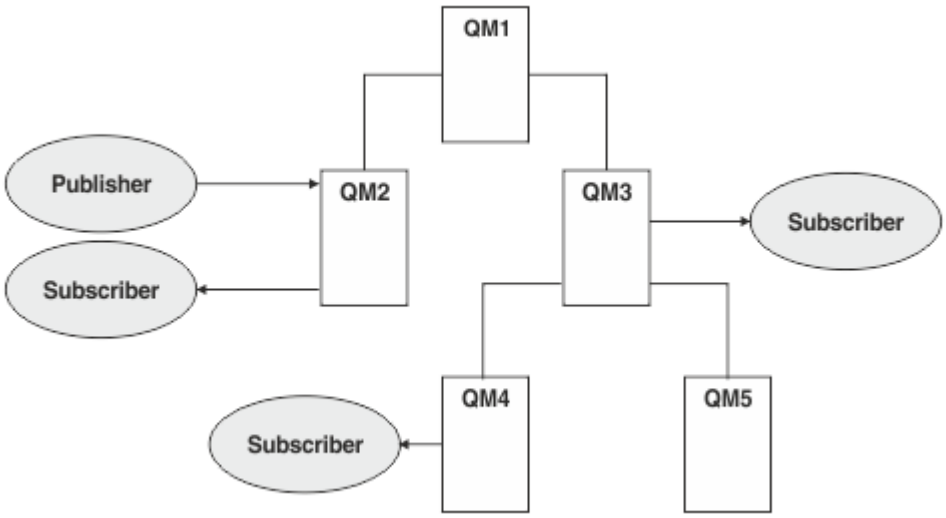


図 *34.* パブリッシュ/サブスクライブ階層

前の図では、QM3 および QM4 のサブスクライバーに配信されたパブリケーションは、QM2 から QM1 にルーティングされた後、QM3 に到達し、最後に QM4 に到達します。

階層を使用すると、階層内のすべてのキュー・マネージャーの関係を直接制御できます。 これは、パブリッシャーからサブスクライバーへのメッセージのルーティングをきめ細かく制御できるため、特に、接続が制限されたキュー・マネージャー・ネットワーク間でルーティングする場合に便利です。 パブリッシャーからサブスクライバーまでのメッセージのルーティング経路にあるすべてのキュー・マネージャーについて、その可用性および能力を慎重に考慮する必要があります。

詳しくは、97 ページの『パブリッシュ/サブスクライブの階層』 を参照してください。

キュー・マネージャー間のパブリケーションの分散

ルーティングの選択に加えて、次の 2 つのアプローチにより、キュー・マネージャーのネットワーク内にパブリケーションを分散させることができます。

- あるキュー・マネージャーからのパブリケーションは、そのパブリケーションのサブスクリプションを現在ホストしているキュー・マネージャーにのみ送信する。

- 各パブリケーションをすべてのキュー・マネージャーに送信し、各キュー・マネージャーにパブリケーションとサブスクリプションのマッチングを行わせる。

前者の場合、パブリケーション・メッセージは必要な場所にのみ送信されますが、サブスクリプションに関するある程度の情報をキュー・マネージャー間で共有する必要があります。 後者の場合、サブスクリプション情報を共有する必要はありませんが、不要なパブリケーション・メッセージがキュー・マネージャー間で送信される可能性があります。

デフォルトでは、IBM MQ は前者の方法を使用するため、パブリケーションは、該当するサブスクリプションを持つキュー・マネージャーにのみ送信されます。 サブスクリプション情報は、プロキシー・サブスクリプション の形でキュー・マネージャー間に伝搬します。 分散パブリッシュ/サブスクライブ・トポロジーにおいて最も使用効率が高い方法は、サブスクリプションの分散状況と存続期間、およびパブリケーションの頻度によって異なります。 『パブリッシュ/サブスクライブ・ネットワークでのサブスクリプションのパフォーマンス』を参照してください。

関連概念

77 ページの『トピック・ツリー』
定義する各トピックは、トピック・ツリー内の要素、つまりノードです。 トピック・ツリーは、最初は空にしておくことも、MQSC または PCF コマンドを使用して既に定義されたトピックを含めることも可能です。 トピック作成コマンドを使用するか、パブリケーションまたはサブスクリプションで初めてトピックを指定することによって、新規トピックを定義できます。

パブリッシュ/サブスクライブ階層のシナリオ

関連タスク

パブリッシュ/サブスクライブ・クラスターの設計

パブリッシュ/サブスクライブ・クラスター

パブリッシュ/サブスクライブ・クラスターとは、キュー・マネージャーを相互接続した標準的なクラスターであり、パブリケーションは、パブリッシュ元のアプリケーションから、クラスター内のキュー・マネージャー上に存在するサブスクリプションに自動的に移動します。 パブリッシュ/サブスクライブ・クラスター内のパブリケーションのルーティングには、直接ルーティング とトピック・ホスト・ルーティング という 2 つの選択肢があります。 どちらのルーティングを選択するかは、クラスターの規模および予測されるアクティビティーのパターンによって決まります。

パブリッシュ/サブスクライブ・メッセージングに使用されるクラスターは、標準 IBM MQ クラスターとまったく変わりはありません。 そのため、パブリッシュ/サブスクライブ・クラスター内のキュー・マネージャーは物理的に別々のコンピューター上に存在でき、キュー・マネージャーの各ペアは、必要に応じてクラスター・チャネルによって相互に接続されます。 詳しくは、クラスターを参照してください。

パブリッシュ/サブスクライブ・メッセージング用に標準的なキュー・マネージャー・クラスターを構成するには、クラスター内の 1 つのキュー・マネージャーに 1 つ以上の管理対象トピック・オブジェクトを定義します。 そのトピックをクラスター・トピックにするには、クラスターの名前を指定して **CLUSTER** プロパティーを構成します。 この構成を行うと、トピック・ツリー内のそのトピック以下の、パブリッシャーまたはサブスクライバーによって使用されるすべてのトピックが、クラスター内のすべてのキュー・マネージャー間で共用されます。また、そのトピック・ツリーのクラスター・ブランチにパブリッシュされたメッセージは、そのクラスター内の他のキュー・マネージャー上のサブスクリプションに自動的にルーティングされます。

ターゲット・キュー・マネージャー上に存在するメッセージのサブスクライバーの数にかかわらず、パブリッシャー・キュー・マネージャーと他の各キュー・マネージャーの間では、メッセージ 1 つにつきコピーが 1 つのみ送信されます。 1 つのキュー・マネージャーに 1 つ以上のサブスクリプションがある場合、メッセージが到達すると、すべてのサブスクリプションに複製されます。

このクラスターに属しているキュー・マネージャーはクラスター・トピックを自動的に認識するようになり、そのキュー・マネージャー上のパブリッシャーおよびサブスクライバーも自動的にクラスターに参加します。

クラスター・トピック・オブジェクト下にないトピック・ストリングで作業することで、非クラスター・パブリッシュ/サブスクライブ・アクティビティーをパブリッシュ/サブスクライブ・クラスターで行うこともできます。

パブリッシュ/サブスクライブ・クラスター内のパブリケーションのルーティングには、直接ルーティングとトピック・ホスト・ルーティング という 2 つの選択肢があります。 クラスター内で使用するメッセージ・ルーティングを選択するには、管理対象トピック・オブジェクトの **CLROUTE** プロパティーを以下のいずれかの値に設定します。

- **DIRECT**
- **TOPICHOST**

デフォルトでは、トピック・ルーティングは **DIRECT** です。 直接経路指定されたクラスター・トピックをキュー・マネージャーで構成すると、クラスター内のすべてのキュー・マネージャーがクラスター内の他のすべてのキュー・マネージャーを認識するようになります。 各キュー・マネージャーは、パブリッシュ操作およびサブスクライブ操作を実行するときに、クラスター内の他のすべてのキュー・マネージャーに直接接続できます。

IBM MQ 8.0 以降、代わりにトピック・ルーティングを **TOPICHOST** として構成できるようになりました。トピック・ホスト経路指定を使用すると、クラスター内のすべてのキュー・マネージャーは、経路指定されたトピック定義をホストするクラスター・キュー・マネージャー (つまり、トピック・オブジェクトを定義したキュー・マネージャー) を認識するようになります。 パブリッシュ操作およびサブスクライブ操作を行うとき、クラスター内のキュー・マネージャーは、それらのトピック・ホスト・キュー・マネージャーにのみ接続し、相互に直接接続されることはありません。 トピック・ホスト・キュー・マネージャーは、パブリケーションがパブリッシュされるキュー・マネージャーから、一致するサブスクリプションがあるキュー・マネージャーへのパブリケーションの経路指定を担当します。

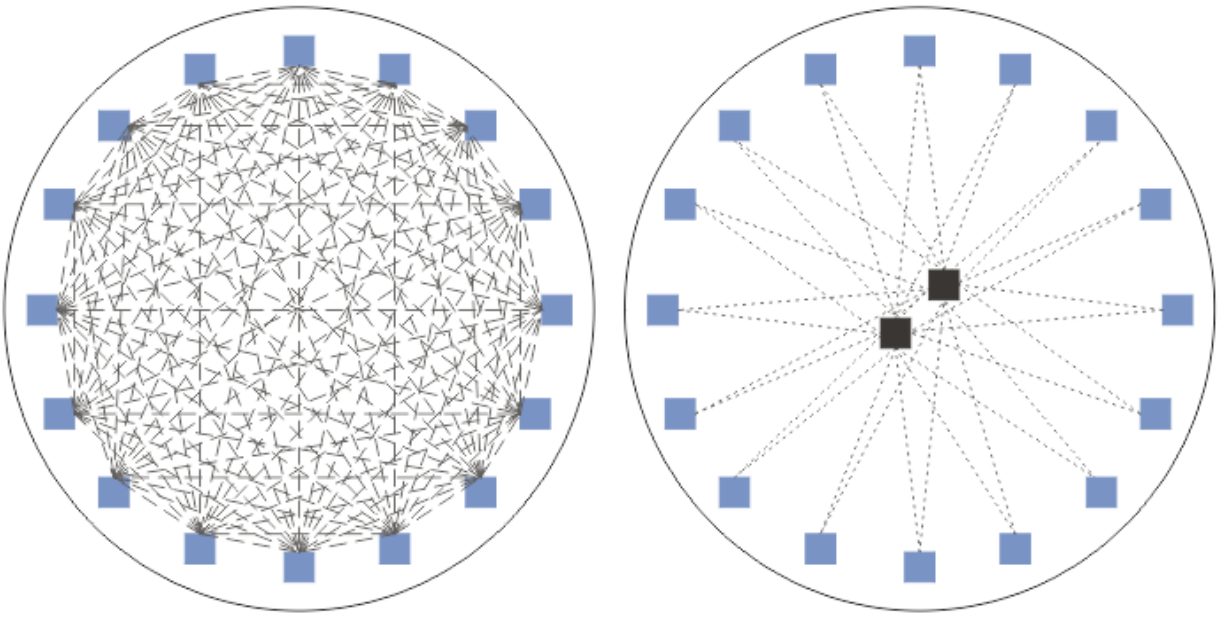


図 *35.* 直接ルーティングとトピック・ホスト・ルーティング

直接ルーティングの概要

管理対象トピック・オブジェクトに直接ルーティングを構成する場合、トピック・オブジェクトは、クラスター内のいずれかのキュー・マネージャーに定義するだけで、すべてのキュー・マネージャーから認識されるようになります。 どのキュー・マネージャーにトピックを定義しようと、そのトピックのパブリッシュ/サブスクライブ・メッセージングの動作には影響しません。

各メッセージは、中間キュー・マネージャーを経由して渡されるのではなく、パブリッシャー・キュー・マネージャーからクラスター内の他のキュー・マネージャー上の各サブスクリプションに直接流れます。

デフォルトでは、クラスター内でサブスクリプションを 1 つ以上ホストする他のキュー・マネージャーにのみ、メッセージが送信されます。

- これは、各キュー・マネージャーが、そこへの 1 つ以上のサブスクリプションが現在存在するすべてのトピックの情報を、クラスター内の他のすべてのキュー・マネージャーに直接通知することによって成り

立っています。 その結果、クラスター内のすべてのキュー・マネージャーが、サブスクライブされているすべてのトピックを認識するようになり、サブスクリプションをホストするすべてのキュー・マネージャーが、他のすべてのキュー・マネージャーに対してチャネルを確立します。 これは、各キュー・マネージャーにパブリッシャーが存在しているかどうかに関係なく行われます。

- サブスクライブされている個々のトピックについての情報をすべてのキュー・マネージャーで認識するという動作が発生しないようにするには、サブスクリプションが存在するかどうかにかかわらず、クラスター内のすべてのキュー・マネージャーにすべてのパブリケーションを送信するという方式に変更します。こうすると、サブスクリプションに関する情報のトラフィックは削減されますが、パブリケーションのトラフィック、および各キュー・マネージャーが確立するチャネルの数は増加する可能性があります。『パブリッシュ/サブスクライブ・ネットワークでのサブスクリプションのパフォーマンス』を参照してください。

直接ルーティングのクラスター・トピックを使用するパブリッシュ/サブスクライブのメッセージ・フローは、複数のパブリッシュ/サブスクライブ・クラスターをまたぐように設定することもできます。 これを行うには、各クラスターの中からそれぞれ 1 つのキュー・マネージャーをパブリッシュ/サブスクライブ階層に追加します。 複数のクラスターのトピック・スペースの結合を参照してください。

直接ルーティングの詳細な説明については、パブリッシュ/サブスクライブ・クラスターの直接ルーティングを参照してください。

トピック・ホスト・ルーティングの概要

管理対象トピック・オブジェクトにトピック・ホスト・ルーティングを構成すると、クラスター内のキュー・マネージャーのパブリケーションは、そのトピック・オブジェクトが構成されているキュー・マネージャー (トピック・ホスト) を経由し、そこからサブスクリプションが存在するキュー・マネージャーにルーティングされます。

- これは、各キュー・マネージャーが、そこへの 1 つ以上のサブスクリプションが現在存在するすべてのトピックの情報を、すべてのトピック・ホストに通知することによって成り立っています。 サブスクリプションをホストするすべてのキュー・マネージャーは、そのサブスクリプションに関係するトピックのすべてのトピック・ホストに対してチャネルを確立します。
- 非トピック・ホスト・キュー・マネージャーは、パブリッシュ/サブスクライブの目的でクラスター内の他の非トピック・ホスト・キュー・マネージャーを認識することはなく、それらのキュー・マネージャーとの間でこの目的のためのチャネルが確立されることもありません。
- パブリッシュ元のアプリケーションが、そのトピックをホストするキュー・マネージャーに接続されている場合、パブリッシュされたメッセージは、マッチングするサブスクリプションが作成されているキュー・マネージャーに直接ルーティングされるため、余分な「ホップ」は発生しません。 同様に、マッチングするサブスクリプションが、そのトピックをホストする 1 つのキュー・マネージャー上のみで作成されている場合、そのトピックにパブリッシュされるメッセージはそのキュー・マネージャーに直接ルーティングされるため、余分なホップは発生しません。
- パブリッシャーと同じキュー・マネージャー上にサブスクリプションが存在する場合は、最初にパブリケーションをトピック・オブジェクトのホストにルーティングする必要はありません。

クラスター・キューについては、複数のキュー・マネージャーが同じ管理トピック・オブジェクトを構成することができます。 これにより、メッセージ・ルーティングの可用性が高くなり、ワークロード・バランスを維持した水平方向のスケーラビリティーが得られます。 トピック・ホスト・ルーティングを使用するトピック・オブジェクトでは、複数のキュー・マネージャーで、同じ名前のトピックをトピック・ツリーの同じブランチに構成すると、各トピック・ホストは、サブスクリプションをホストするそれぞれのキュー・マネージャーでサブスクライブされるトピックを認識します。

- パブリッシュされたメッセージは、いずれかのトピック・ホスト・キュー・マネージャーに送信されてから、サブスクリプションをホストするキュー・マネージャーに転送されます。 どのトピック・ホスト・キュー・マネージャーが選択されるかは、Point-to-Point クラスター・キューの場合と同じデフォルトのワークロード・バランシング・ルールに従います。
- 1 つ以上のトピック・ホスト・キュー・マネージャーにパブリッシュ元のキュー・マネージャーから通信できない場合、他の使用可能なトピック・ホスト・キュー・マネージャーにメッセージがルーティングされます。

トピック・ツリーのルーティング対象のブランチに含まれているトピックへのすべてのパブリケーションは、そのトピックへのサブスクリプションがクラスター内のどこにも存在しない場合であっても、トピック・ホストの 1 つに転送されます。 デフォルトでは、クラスター内でサブスクリプションを 1 つ以上ホストする他のキュー・マネージャーにのみ、ここからメッセージが送信されます。

- これは、クラスター内の各キュー・マネージャー上のサブスクライブされているすべてのトピック・ストリングが、各トピック・ホスト・キュー・マネージャーに通知されることによって成り立っています。
- サブスクライブされている個々のトピックについての情報を認識するという動作が発生しないようにするには、サブスクリプションが存在するかどうかにかかわらず、クラスター内のすべてのキュー・マネージャーに、トピック・ホストにルーティングされたすべてのパブリケーションを送信するという方式に変更します。 こうすると、サブスクリプションに関する情報のトラフィックは削減されますが、パブリケーションのトラフィック、および各トピック・ホスト・キュー・マネージャーとの間に確立されるチャネルの数は増加する可能性があります。 『パブリッシュ/サブスクライブ・ネットワークでのサブスクリプションのパフォーマンス』を参照してください。

トピック・ホスト・ルーティングのクラスター・トピックを使用するパブリッシュ/サブスクライブのメッセージ・フローを、パブリッシュ/サブスクライブ階層を使用して複数のパブリッシュ/サブスクライブ・クラスターをまたぐフローにすることは**できません**。

トピック・ホスト・ルーティングの詳細な説明については、パブリッシュ/サブスクライブ・クラスターのトピック・ホスト・ルーティングを参照してください。

パブリッシュ/サブスクライブの階層

パブリッシュ/サブスクライブ階層を構築するには、チャネルを使用してキュー・マネージャーどうしをリンクした後に、キュー・マネージャーのペアの間に親子関係を定義します。 メッセージは、パブリッシャーから、階層内の直接的な関係を介してサブスクリプションまで流れます。 これは、複数の"ホップ"を意味する場合があることに注意してください。

ターゲット・キュー・マネージャー上に存在するメッセージのサブスクライバーの数にかかわらず、1 組のキュー・マネージャーの間で送信されるメッセージのコピーは 1 つのみです。 1 つのキュー・マネージャーに 1 つ以上のサブスクリプションがある場合、メッセージが到達すると、すべてのサブスクリプションに複製されます。

デフォルトでは、別のキュー・マネージャー上のサブスクリプションまでのルート上にある、階層内の他のキュー・マネージャーにのみメッセージが送信されます。

- これは、各キュー・マネージャーが、このキュー・マネージャー上またはこれに関係があるいずれかのキュー・マネージャー上に存在するトピックの情報のうち、1 つ以上のサブスクリプションが現在存在するものすべてを、直接関係する他のそれぞれのキュー・マネージャーに通知することによって成り立っています。 この結果、階層内のすべてのキュー・マネージャーが、サブスクライブされているすべてのトピックを認識するようになります。
- この動作を変更して、サブスクリプションの存在にかかわらず、常に階層内のすべてのキュー・マネージャーにパブリケーションを送信することもできます。 このようにした場合、サブスクリプション情報を階層の中で伝搬させる必要はなくなりますが、パブリケーションのトラフィックが増加する可能性があります。

クラスターを作成する場合は、ネットワーク内をメッセージが無限に循環するループを生成しないように注意する必要があります。 階層では、このようなループは生成されません。

キュー・マネージャーの名前はすべて固有でなければなりません。

パブリッシュ/サブスクライブのメッセージ・フローは、複数のパブリッシュ/サブスクライブ・クラスターをまたぐことができます。 これを行うには、各クラスターの中からそれぞれ 1 つのキュー・マネージャーをパブリッシュ/サブスクライブ階層に追加します。

詳細な説明については、パブリッシュ/サブスクライブ階層のルーティングを参照してください。

パブリッシュ/サブスクライブ・ネットワークでのプロキシー・サブスクリプション

プロキシー・サブスクリプションは、あるキュー・マネージャーによってパブリッシュされるトピックに関する、別のキュー・マネージャーによって行われるサブスクリプションです。 プロキシー・サブスクリプションは、サブスクリプションによってサブスクライブされている各トピック・ストリングのキュー・マネージャーの間で流れます。 ユーザーはプロキシー・サブスクリプションを明示的に作成しません。キュー・マネージャーが代わりにそれを行います。

キュー・マネージャーをまとめて、パブリッシュ/サブスクライブ・クラスターまたはパブリッシュ/サブスクライブ階層に接続できます。 プロキシー・サブスクリプションは、接続されているキュー・マネージャーの間で流れます。 プロキシー・サブスクリプションによって、あるキュー・マネージャーに接続されているパブリッシャーが作成したトピックへのパブリケーションを、他のキュー・マネージャーに接続されているそのトピックへのサブスクライバーが受け取ります。 91 ページの『分散パブリッシュ/サブスクライブのネットワーク』を参照。

パブリッシュ/サブスクライブ・トポロジーが個々のトピック・ストリングへの何千ものサブスクリプションを処理する場合、またはそのようなサブスクリプションの存在が急激に変化している可能性がある場合、プロキシー・サブスクリプション伝搬のオーバーヘッドについて検討する必要があります。 このトピックの残りの部分で説明する自動集約に加えて、手動の構成変更を行うことで、接続されたキュー・マネージャー間のプロキシー・サブスクリプションとパブリケーションのフローをさらに制限し、接続されたすべてのキュー・マネージャーにプロキシー・サブスクリプションが伝搬されるまでの待機遅延を縮小できます。 『パブリッシュ/サブスクライブ・ネットワークでのサブスクリプションのパフォーマンス』を参照してください。

プロキシー・サブスクリプションには、ローカル・サブスクリプションで使用されるセレクターは含まれないため、ワイルドカードを指定したサブスクリプション・トピック・ストリングが単純化される可能性があります。 その結果、実際のサブスクリプションにはマッチングしないパブリケーションがプロキシー・サブスクリプションとマッチングし、キュー・マネージャー間に余分のパブリケーション・フローが発生する可能性があります。 このような矛盾は、サブスクリプションをホストするキュー・マネージャーがフィルタリングにより除外するため、余分のパブリケーションはサブスクリプションには返されません。

プロキシー・サブスクリプションの集約

プロキシー・サブスクリプションは、重複除去システムを使用して集約されます。 特定の解決されたトピック・ストリングで、最初のローカル・サブスクリプションまたは最初に受け取ったプロキシー・サブスクリプションに関して、プロキシー・サブスクリプションが送信されます。 同じトピック・ストリングに対する後続のサブスクリプションは、その既存のプロキシー・サブスクリプションを利用します。

このプロキシー・サブスクリプションは、最後のローカル・サブスクリプションまたは最後に受け取ったプロキシー・サブスクリプションが取り消された後に取り消されます。

パブリケーションの集約

あるキュー・マネージャーに同じトピック・ストリングへの複数のサブスクリプションが存在する場合、そのトピック・ストリングに一致する各パブリケーションの単一コピーのみが、パブリッシュ/サブスクライブ・トポロジーのその他のキュー・マネージャーから送信されます。 メッセージが到着すると、ローカル・キュー・マネージャーはメッセージのコピーを一致する各サブスクリプションに送信します。

プロキシー・サブスクリプションにワイルドカードが含まれている場合、1 つのパブリケーションのトピック・ストリングに複数のプロキシー・サブスクリプションが一致する可能性があります。 接続されている 1 つのキュー・マネージャーによって作成された複数のプロキシー・サブスクリプションに一致するメッセージが、キュー・マネージャーでパブリッシュされた場合、複数のプロキシー・サブスクリプションを満たすために、パブリケーションの 1 つのコピーだけがリモート・キュー・マネージャーに転送されます。

関連概念

分散パブリッシュ/サブスクライブ・ネットワークでのループ検出

プロキシー・サブスクリプション内のワイルドカード

サブスクリプションでは、トピック・ストリングにワイルドカードを使用してパブリケーション内の複数のトピック・ストリングに突き合わせることができます。

サブスクリプションで使用できるワイルドカードのスキーマには、トピック・ベース と文字ベース の 2 つがあります。 『71 ページの『ワイルドカード・スキーマ』』を参照してください。

IBM MQ では、ワイルドカード・サブスクリプションのすべてのプロキシー・サブスクリプションは、トピック・ベースのワイルドカードを使用するように変換されます。 文字ベースのワイルドカードが検出されると、そのワイルドカードは、直前の / までさかのぼって # 文字に置き換えられます。 例えば、/aaa/bbb/c*d は /aaa/bbb/# に変換されます。 この変換の結果、リモート・キュー・マネージャーは、明示的にサブスクライブされた場合よりも少し多めにパブリケーションを送信することになります。追加のパブリケーションは、ローカル・キュー・マネージャーがそのローカル・サブスクライバーにパブリケーションを送信するときに、ローカル・キュー・マネージャーによってフィルターに掛けられます。

WILDCARD プロパティーによるワイルドカードの使用の制御

MQSC **Topic** WILDCARD プロパティーまたは 同等の PCF Topic `WildcardOperation` プロパティーを使用すると、 ワイルドカード・トピック・ストリング名を使用するサブスクライバー・アプリケーションへのパブリケーションの送達を制御できます。 WILDCARD プロパティーには、以下の 2 つの値のいずれかを指定できます。

WILDCARD
このトピックに対するワイルドカード・サブスクリプションの動作。

PASSTHRU
このトピック・オブジェクトのトピック・ストリングよりも具体的でないワイルドカード・トピックに対するサブスクリプションは、そのトピックまたはそのトピックよりも具体的なトピック・ストリングに対するパブリケーションを受信できるようになります。

BLOCK
このトピック・オブジェクトのトピック・ストリングよりも具体的でないワイルドカード・トピックに対するサブスクリプションは、このトピックまたはこのトピックよりも具体的なトピック・ストリングに対するパブリケーションを受信できなくなります。

サブスクリプションが定義されている場合に、この属性の値が使用されます。 この属性を変更しても、既存のサブスクリプションによってカバーされているトピック・セット は、変更による影響を受けません。 このシナリオは、トピック・オブジェクトが作成または削除されてトポロジーが変更された場合にも当てはまります。WILDCARD 属性の変更後に作成されたサブスクリプションに一致するトピックのセットは、変更後のトポロジーを使用して作成されます。 既存のサブスクリプションについて、一致するトピック・セットを強制的に再評価する場合は、キュー・マネージャーを再開する必要があります。

85 ページの『例: Sport パブリッシュ/サブスクライブ・クラスターを作成する』 の例では、 81 ページの図 23 で示されるトピック・ツリー構造を作成するステップに従うことができます。

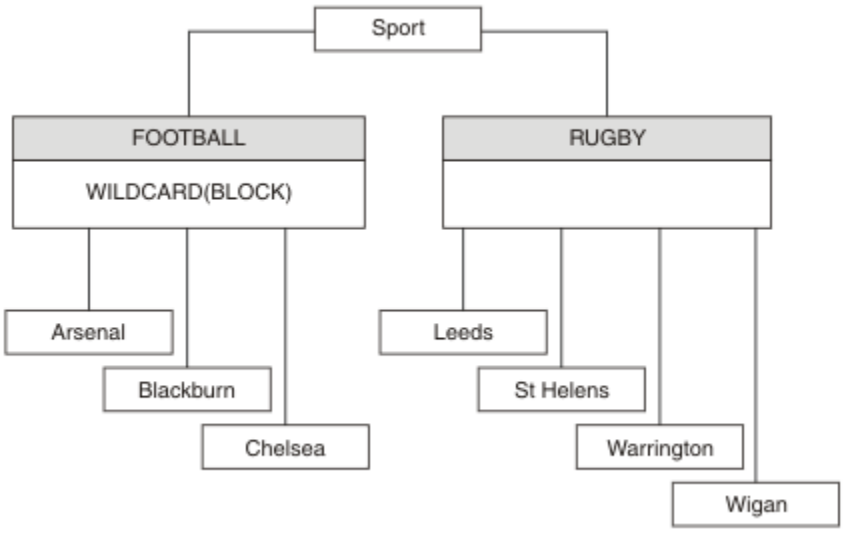


図 *36. WILDCARD* プロパティー *BLOCK* を使用するトピック・ツリー

ワイルドカード・トピック・ストリング *#* を使用するサブスクライバーは、 Sport トピックと Sport/Rugby サブツリーへのすべてのパブリケーションを受け取ります。 Sport/Football トピックの WILDCARD プロパティー値が BLOCK であるため、このサブスクライバーは Sport/Football サブツリーへのパブリケーションは受け取りません。

PASSTHRU は、デフォルトの設定値です。 Sport ツリーのノードには、WILDCARD プロパティー値 PASSTHRU を設定できます。 ノードで WILDCARD プロパティー値 BLOCK が設定されていない場合、PASSTHRU を設定しても、Sports ツリーのノードのサブスクライバーによって 観測される動作が変化することはありません。

この例では、サブスクリプションを作成して、 送達されるパブリケーションにワイルドカード設定が与える影響を確認します。 86 ページの図 27 を参照してください。 87 ページの図 30 でパブリッシュ・コマンドを実行して、 パブリケーションをいくつか作成します。

```
pub QMA
```

図 *37. QMA* へのパブリッシュ

82 ページの表 3 では結果が示されています。 WILDCARD プロパティー値 BLOCK を設定すると、ワイルドカードを含むサブスクリプションは、 そのワイルドカードの有効範囲内にあるトピックへのパブリケーションを受信しなくなることに注意してください。

表 *6. QMA* で受信されたパブリケーション

サブスクリプション	**トピック・ストリング**	**受信されたパブリケーション**	**注**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	「Football」サブツリーへのすべてのパブリケーションは、 Sports/Football の WILDCARD(BLOCK) によってブロックされます。
SARSENAL	Sports/#/Arsenal	-	Sports/Football の WILDCARD(BLOCK) により、Arsenal でのワイルドカード・サブスクリプションは防止されます。

表 6. QMA で受信されたパブリケーション (続き)			
サブスクリプション	**トピック・ストリング**	**受信されたパブリケーション**	**注**
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	「Sports/Rugby」のデフォルトである WILDCARD は、Leeds でのワイルドカード・サブスクリプションを防止しません。

注:

あるサブスクリプションに、WILDCARD プロパティー値 BLOCK を持つトピック・オブジェクトに一致するワイルドカードがあるとします。 このサブスクリプションで、一致するワイルドカードの右側にトピック・ストリングもある場合、 サブスクリプションがパブリケーションを受信することはありません。 ブロックされないパブリケーションのセットは、 ブロックされたワイルドカードの親であるトピックへのパブリケーションです。 BLOCK プロパティー値を持つトピックの子であるトピックへのパブリケーションは、ワイルドカードによってブロックされます。 したがって、ワイルドカードの右側にトピックが含まれるサブスクリプション・トピック・ストリングは、 一致するパブリケーションを受信することがありません。

WILDCARD プロパティー値を BLOCK に設定しても、ワイルドカードが含まれるトピック・ストリングを使用したサブスクライブが実行できなくなるわけではありません。 このようなサブスクリプションは正常に行われます。 このサブスクリプションには、 WILDCARD プロパティー値 BLOCK を持つトピック・オブジェクトを含むトピックに一致する 明示的なトピックが含まれます。 これは、WILDCARD プロパティー値 BLOCK を持つトピックの親または子であるトピック用に、 ワイルドカードを使用します。 81 ページの図 23 の例では、 Sports/Football/# のようなサブスクリプションがパブリケーションを受信できます。

ワイルドカードとクラスター・トピック

クラスター・トピック定義はクラスター内のすべてのキュー・マネージャーに伝搬されます。 クラスター内のキュー・マネージャーでクラスター・トピックへのサブスクリプションを行うと、 そのキュー・マネージャーで複数のプロキシー・サブスクリプションが作成されます。 クラスター内の他のすべてのキュー・マネージャーで 1 つのプロキシー・サブスクリプションが作成されます。 ワイルドカードを含むトピック・ストリングを使用したサブスクリプションをクラスター・トピックと組み合わせると、 動作の予測が難しくなる可能性があります。 次の例は、この動作について説明しています。

85 ページの『例: Sport パブリッシュ/サブスクライブ・クラスターを作成する』 の例にあるクラスター・セットアップでは、QMB が QMA と同じサブスクリプションのセットを持っているにもかかわらず、パブリッシャーが QMA にパブリッシュした後で QMB はパブリケーションを 受信しませんでした (82 ページの図 24 を参照)。 Sports/Football と Sports/Rugby のトピックは クラスター・トピックですが、fullsubs.tst で定義されているサブスクリプションは クラスター・トピックを参照しません。 QMB から QMA に伝搬されるプロキシー・サブスクリプションはありません。 プロキシー・サブスクリプションがないと、QMA へのパブリケーション は QMB に転送されません。

Sports/#/Leeds などの一部のサブスクリプションは、クラスター・トピック (このケースでは Sports/Rugby) を参照するように見えることがあります。 実際には、Sports/#/Leeds サブスクリプション はトピック・オブジェクト SYSTEM.BASE.TOPIC に解決されます。

Sports/#/Leeds などのサブスクリプションによって参照されるトピック・オブジェクトを解決するための規則は、 以下のとおりです。 トピック・ストリングが最初のワイルドカードの位置まで切り捨てられます。 トピック・ストリングを左方向にスキャンし、 関連付けられた管理トピック・オブジェクトを持つ最初のトピックを探します。 そのトピック・オブジェクトがクラスター名を指定するか、ローカル・トピック・オブジェクトを定義している可能性があります。 Sports/#/Leeds の例では、切り捨て後のトピック・ストリングは Sports であり、 トピック・オブジェクトを持たないため、Sports/#/Leeds は ローカル・トピック・オブジェクトである SYSTEM.BASE.TOPIC から継承します。

クラスター化されたトピックのサブスクライブによってワイルドカード伝搬の動作が変更される仕方を確認するには、 バッチ・スクリプト upsubs.bat を実行します。 このスクリプトはサブスクリプション・キ

ューをクリアし、 fullsubs.tst 内にクラスター・トピック・サブスクリプションを追加します。 パブリケーションのバッチを作成するには、 puba.bat を再び実行します (82 ページの図 24 を参照してください)。

83 ページの表 4 は、パブリケーションがパブリッシュされたのと同じキュー・マネージャーに 2 つの新規サブスクリプションを追加した結果を示しています。 結果は予測どおりであり、新規サブスクリプションはそれぞれ 1 つのパブリケーションを受信し、 他のサブスクリプションによって受信されるパブリケーションの数は変更されません。 他のクラスター・キュー・マネージャーでは予測しない結果が発生します。 84 ページの表 5 を参照してください。

表 7. *QMA* で受信されたパブリケーション

サブスクリプション	トピック・ストリング	受信されたパブリケーション	注
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	「Football」サブツリーへのすべてのパブリケーションは、 Sports/Football の WILDCARD(BLOCK) によってブロックされます。
SARSENAL	Sports/#/Arsenal	-	Sports/Football の WILDCARD(BLOCK) により、 Arsenal でのワイルドカード・サブスクリプションは防止されます。
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	「Sports/Rugby」のデフォルトである WILDCARD は、Leeds でのワイルドカード・サブスクリプションを防止しません。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	サブスクリプションにワイルドカードがないため、 Arsenal はパブリケーションを受信します。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds は、いかなるイベントでもパブリケーションを受信します。

84 ページの表 5 は、 QMB で 2 つの新規サブスクリプションを追加して QMA でパブリッシュした結果を示しています。 これら 2 つの新規サブスクリプションがないときに QMB はパブリケーションを受信しませんでした。 Sports/FootBall と Sports/Rugby はどちらもクラスター・トピックであるため、 これら 2 つの新規サブスクリプションは予測どおりパブリケーションを受信します。 QMB から Sports/Football/Arsenal と Sports/Rugby/Leeds 用のプロキシー・サブスクリプション が QMA に転送された後、 そこからパブリケーションが QMB に送信されました。

予測しなかった結果として、 以前はパブリケーションを受信しなかった 2 つのサブスクリプション (Sports/# と Sports/#/Leeds) が、 パブリケーションを受信するようになりました。 これは、他のサブスクリプション用に QMB に転送されたパブリケーションである Sports/Football/Arsenal と Sports/Rugby/Leeds が、QMB に接続された任意のサブスクライバーから使用可能になったためです。 その結果、ローカル・トピックの Sports/# と Sports/#/Leeds へのサブスクリプションは、 Sports/Rugby/Leeds パブリケーションを受信します。 「Sports/Football」は独自の WILDCARD プロパティー値が BLOCK に設定されているため、Sports/#/Arsenal は引き続きパブリケーションを受信しません。

表 *8. QMB* で受信されたパブリケーション

サブスクリプション	トピック・ストリング	受信されたパブリケーション	注
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*WILDCARD(BLOCK) on Sports/Football* によってブロックされる、フットボール・サブツリーへのすべてのパブリケーション
SARSENAL	Sports/#/Arsenal	-	Sports/Football の WILDCARD(BLOCK) により、Arsenal でのワイルドカード・サブスクリプションは防止されます。
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	「*Sports/Rugby*」のデフォルトである *WILDCARD* は、*Leeds* でのワイルドカード・サブスクリプションを防止しません。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	サブスクリプションにワイルドカードがないため、 Arsenal はパブリケーションを受信します。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds は、いかなるイベントでもパブリケーションを受信します。

ほとんどのアプリケーションにおいて、 あるサブスクリプションが別のサブスクリプションの動作に影響することは望ましくありません。 WILDCARD プロパティーの値 BLOCK の重要な使用法の 1 つは、 ワイルドカードを含む同じトピック・ストリングへのサブスクリプションをすべて同じように動作させることです。 サブスクリプションがパブリッシャーと同じキュー・マネージャー上にあるか別のキュー・マネージャー上にあるかにかかわらず、 サブスクリプションの結果は同じです。

ワイルドカードとストリーム

パブリッシュ/サブスクライブ API に書き込まれた新規アプリケーションの場合、結果として、* へのサブスクリプションがパブリケーションを受信しなくなります。「Sports」へのすべてのパブリケーションを受信するには、Sports/* または Sports/# にサブスクライブするとともに、Business パブリケーションについても同様の処理を行う必要があります。

既存のキュー型パブリッシュ/サブスクライブ・アプリケーションの動作は、パブリッシュ/サブスクライブ・ブローカーを新しいバージョンの IBM MQ に移行しても変更されません。 **Publish**、**Register Publisher**、または **Subscriber** コマンドの **StreamName** プロパティーは、ストリームの移行先のトピックの名前にマップされます。

ワイルドカードとサブスクリプション・ポイント

パブリッシュ/サブスクライブ API に書き込まれた新規アプリケーションの場合、 移行の結果、* へのサブスクリプションがパブリケーションを受信しなくなります。「Sports」へのすべてのパブリケーションを受信するには、Sports/* または Sports/# にサブスクライブするとともに、Business パブリケーションについても同様の処理を行う必要があります。

既存のキュー型パブリッシュ/サブスクライブ・アプリケーションの動作は、パブリッシュ/サブスクライブ・ブローカーを新しいバージョンの IBM MQ に移行しても変更されません。 **Publish**、**Register Publisher**、または **Subscriber** コマンドの **SubPoint** プロパティーは、サブスクリプションの移行先のトピックの名前にマップされます。

例: Sport パブリッシュ/サブスクライブ・クラスターを作成する

以降のステップでは、クラスター CL1 とともに 4 つのキュー・マネージャー (CL1A および CL1B という 2 つの全リポジトリーと、 QMA および QMB という 2 つの部分リポジトリー) を作成します。 全リポジトリーは、クラスター定義を保持するためにのみ使用されます。 QMA は、クラスター・トピック・ホストに指定されます。 永続サブスクリプションは QMA と QMB の両方で定義されます。

注 : この例は、Windows 用にコーディングされています。 他のプラットフォームでこの例を構成してテストするには、Create qmgrs.bat と create pub.bat を再コーディングする必要があります。

1. 以下のスクリプト・ファイルを作成します。
   a. Create topics.tst
   b. Create wildsubs.tst
   c. Create fullsubs.tst
   d. Create qmgrs.bat
   e. create pub.bat
2. Create qmgrs.bat を実行して構成を作成します。

```
qmgrs
```

81 ページの図 23 でトピックを作成します。 図 5 のスクリプトは、クラスター・トピック Sports/Football および Sports/Rugby を作成します。

注 : REPLACE オプションは、トピックの TOPICSTR プロパティーを置き換えません。 TOPICSTR は、この例でさまざまなトピック・ツリーをテストするために役立つプロパティーです。 トピックを変更するには、最初にトピックを削除します。

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

図 *38.* トピックの削除と作成*: topics.tst*

注 : REPLACE によってトピック・ストリングが置き換えられることはないため、トピックを削除します。

ワイルドカードが含まれるサブスクリプションを作成します。 ワイルドカードは、81 ページの図 23 で示されているトピック・オブジェクトを持つトピックに対応します。 各サブスクリプション用のキューを作成します。 スクリプトが実行または再実行されると、キューがクリアされてサブスクリプションは削除されます。

注 : REPLACE オプションによって サブスクリプションの TOPICOBJ または TOPICSTR プロパティーが置き換えられることはありません。 TOPICOBJ または TOPICSTR は、さまざまなサブスクリプションをテストするために変更される便利なプロパティーです。 これらを変更するには、最初にサブスクリプションを削除します。

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

図 *39.* ワイルドカード・サブスクリプションの作成*: wildsubs.tst*

クラスター・トピック・オブジェクトを 参照するサブスクリプションを作成します。

注 :

TOPICOBJ によって参照されるトピック・ストリング と TOPICSTR によって定義されるトピック・ストリングの間には、 デリミッター / が自動的に挿入されます。

定義 DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) は、同じサブスクリプションを作成します。 TOPICOBJ は、すでに定義したトピック・ストリングを迅速に参照する方法として使用されます。 サブスクリプションは作成された後、トピック・オブジェクトを参照しなくなります。

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

図 *40.* サブスクリプションの削除と作成*: fullsubs.tst*

2 つのリポジトリーが含まれる クラスターを作成します。 パブリッシュとサブスクライブ用に 2 つの部分リポジトリーを作成します。 すべてを削除してやり直すには、スクリプトを再実行します。 このスクリプトでは、トピック階層と初期ワイルドカード・サブスクリプションも作成されます。

注 :

他のプラットフォームでは、同様のスクリプトを作成するか、すべてのコマンドを入力します。 スクリプトを使用すると、迅速にすべてを削除して同一の構成でやり直すことができます。

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

図 *41.* キュー・マネージャーの作成*: qmgrs.bat*

サブスクリプションをクラスター・トピックに追加して、構成を更新します。

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

図 *42.* サブスクリプションの更新*: upsubs.bat*

パブリケーション・トピック・ストリングが含まれる メッセージをパブリッシュするには、キュー・マネージャーをパラメーターとして pub.bat を実行します。 Pub.bat は、サンプル・プログラム **amqspub** を使用します。

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

図 *43.* パブリッシュ*: pub.bat*

関連概念

ワイルドカード・サブスクリプションと保存パブリケーション

パブリケーション有効範囲

パブリッシュ/サブスクライブ・クラスターまたはパブリッシュ/サブスクライブ階層を構成する場合、パブリケーションの有効範囲により、キュー・マネージャーがリモート・キュー・マネージャーにパブリッシュを転送するかどうかをさらに制御することができます。 パブリケーションの有効範囲の管理には、**PUBSCOPE** トピック属性を使用します。

パブリケーションがリモート・キュー・マネージャーに転送されない場合、ローカル・サブスクライバーだけがそのパブリケーションを受け取ります。

パブリッシュ/サブスクライブ・クラスターを使用する場合、パブリケーションの有効範囲は、主に、トピック・ツリーの特定の場所にあるクラスター・トピック・オブジェクトの定義によって制御されます。 パブリケーションの有効範囲を設定するときには、パブリケーションがクラスター内の他のキュー・マネージャーに流れるようにしなければなりません。 クラスター・トピックのパブリケーションの有効範囲を制限するのは、特定のキュー・マネージャー上の特定のトピックを細かく制御する必要がある場合だけにしてください。

パブリッシュ/サブスクライブ階層を使用する場合、パブリケーションの有効範囲は、主に、この属性とサブスクリプションの有効範囲属性の組み合わせによって制御されます。

PUBSCOPE 属性は、特定のトピックに対して行われるパブリケーションの有効範囲を判別するために使用されます。 この属性には以下のいずれかの値を設定できます。

QMGR
: パブリケーションは、ローカル・サブスクライバーだけに送信されます。 これらのパブリケーションはローカル・パブリケーション と呼ばれます。 ローカル・パブリケーションはリモート・キュー・マネージャーに転送されないため、リモート・キュー・マネージャーに接続されたサブスクライバーはそれを受信しません。

ALL
: パブリケーションの送信先は、パブリッシュ/サブスクライブ・クラスターまたはパブリッシュ/サブスクライブ階層内のリモート・キュー・マネージャーに接続されたサブスクライバーと、ローカル・サブスクライバーになります。 これらのパブリケーションはグローバル・パブリケーション と呼ばれます。

ASPARENT
: トピック・ツリー内の親トピックの **PUBSCOPE** 設定を使用します。

MQPMO_SCOPE_QMGR メッセージ書き込みオプションを使用して、パブリッシャーはパブリケーションをローカルに行うかグローバルに行うかを指定することもできます。 このオプションを使用する場合には、**PUBSCOPE** トピック属性を使用して設定されているすべての動作がオーバーライドされます。

関連概念

78 ページの『管理トピック・オブジェクト』
管理トピック・オブジェクトを使用すると、デフォルトではない特定の属性をトピックに割り当てることができます。

関連タスク

分散パブリッシュ/サブスクライブ・ネットワークの構成

サブスクリプション有効範囲

サブスクリプションの有効範囲は、1 つのキュー・マネージャーのサブスクリプションが、パブリッシュ/サブスクライブ・クラスターまたは階層の別のキュー・マネージャーでパブリッシュされるパブリケーションを受け取るか、あるいはローカル・パブリッシャーからのみのパブリケーションを受け取るかを制御します。

サブスクリプションの有効範囲をキュー・マネージャーに制限することで、パブリッシュ/サブスクライブ・トポロジー内の別のキュー・マネージャーにプロキシー・サブスクリプションが転送されないようにします。 これにより、キュー・マネージャー間のパブリッシュ/サブスクライブ・メッセージ・トラフィックが削減されます。

パブリッシュ/サブスクライブ・クラスターを使用する場合、サブスクリプションの有効範囲は、主に、トピック・ツリーの特定の場所にあるクラスター・トピック・オブジェクトの定義によって制御されます。 サブスクリプションの有効範囲を設定するときには、プロキシー・サブスクリプションがクラスター内の他のキュー・マネージャーに流れるようにしなければなりません。 クラスター・トピックのサブスクリプションの有効範囲を制限するのは、特定のキュー・マネージャー上の特定のトピックを細かく制御する必要がある場合だけにしてください。

パブリッシュ/サブスクライブ階層を使用する場合、サブスクリプションの有効範囲は、主に、この属性とパブリケーションの有効範囲属性の組み合わせによって制御されます。

SUBSCOPE トピック属性は、特定のトピックに対して行われるサブスクリプションの有効範囲を決定するために使用されます。 この属性には以下のいずれかの値を設定できます。

QMGR
サブスクリプションはローカル・パブリケーションのみを受け取り、プロキシー・サブスクリプションはリモート・キュー・マネージャーに伝搬されません。

ALL
プロキシー・サブスクリプションは、パブリッシュ/サブスクライブ・クラスターまたはパブリッシュ/サブスクライブ階層内のリモート・キュー・マネージャーに伝搬され、サブスクライバーは、ローカルおよびリモートのパブリケーションを受け取ります。

ASPARENT
トピック・ツリー内の親トピックの **SUBSCOPE** 設定を使用します。

トピックのサブスクリプション有効範囲がすべてに設定されている場合、直接または ASPARENT によって解決される場合、サブスクリプションの作成時に MQSO_SCOPE_QMGR を指定することにより、そのトピックの個々のサブスクリプションの有効範囲を QMGR に制限できます。 有効範囲が QMGR であるトピックのサブスクリプションの有効範囲を ALL に広げることはできません。

関連概念
78 ページの『管理トピック・オブジェクト』
管理トピック・オブジェクトを使用すると、デフォルトではない特定の属性をトピックに割り当てることができます。

関連タスク
分散パブリッシュ/サブスクライブ・ネットワークの構成

トピック・スペース

トピック・スペースとは、サブスクライブとパブリッシュを実行できるトピックのセットです。 分散パブリッシュ/サブスクライブ・トポロジー内のキュー・マネージャーにはトピック・スペースがあり、そのトピック・スペースには、そのトポロジー内の接続されたキュー・マネージャーでサブスクライブおよびパブリッシュされたトピックが含まれている可能性があります。

注 : 管理トピック・オブジェクト、トピック・ストリング、トピック・ツリーなどの、キュー・マネージャー内のトピックの概要については、69 ページの『トピック』を参照してください。 特に指定がない限り、この記事におけるトピック という記述は、トピック・ストリング を指すものとします。

トピックは、以下のいずれかの方法によって初期作成されます。

- トピック・オブジェクトまたは永続サブスクリプションを定義するときに、管理者が作成する。
- アプリケーションが新しいトピックに対するパブリケーションまたはサブスクリプションを動的に作成するときに、動的に作成される。

トピックは、プロキシー・サブスクリプションを介して、および管理クラスター・トピック・オブジェクトを作成することによって、他のキュー・マネージャーに伝搬されます。 プロキシー・サブスクリプションの場合、パブリッシャーの接続先であるキュー・マネージャーからサブスクライバーのキュー・マネージャーにパブリケーションが転送されます。

プロキシー・サブスクリプションは、キュー・マネージャー階層で親子関係によって互いに接続されたすべてのキュー・マネージャーの間で伝搬されます。 このため、1 つのキュー・マネージャーにおいて、階層内の他の任意のキュー・マネージャーで定義されたトピックにサブスクライブすることができます。 キュー・マネージャー間に接続パスが存在する限り、それらのキュー・マネージャーの接続方法は問題となりません。

パブリッシュ/サブスクライブ・クラスター内のクラスター・トピックへのサブスクリプションについては、プロキシー・サブスクリプションも伝搬されます。 クラスター・トピックとは、**CLUSTER** 属性を持つ (または親から属性を継承する) トピック・オブジェクトに付加されたトピックです。 クラスター・トピックではないトピックをローカル・トピックといい、これらはクラスターに複製されません。 ローカル・トピックへのサブスクリプションからクラスターにプロキシー・サブスクリプションが伝搬することはありません。

要約すると、2 つの環境でプロキシー・サブスクリプションがサブスクライバー用に作成されます。

1. キュー・マネージャーが階層のメンバーであり、キュー・マネージャーの親と子にプロキシー・サブスクリプションが転送される。

2. キュー・マネージャーがクラスターのメンバーであり、クラスター・トピック・オブジェクトに関連したトピックにサブスクリプション・トピック・ストリングが解決される。 トピックが直接ルーティングのクラスター・トピックである場合、プロキシー・サブスクリプションは、クラスターのすべてのメンバーに転送されます。 トピックがトピック・ホスト・ルーティング のクラスター・トピックである場合、プロキシー・サブスクリプションは、クラスター・トピック・オブジェクトを定義したクラスター内のキュー・マネージャーにのみ転送されます。 詳しくは、94 ページの『パブリッシュ/サブスクライブ・クラスター』を参照してください。

キュー・マネージャーがクラスターと階層のメンバーである場合、プロキシー・サブスクリプションは両方のメカニズムによって伝搬されます。重複するパブリケーションはサブスクライバーに送信されません。

以下のリストでは、3 つのパブリッシュ/サブスクライブ・トポロジーのトピック・スペースについて説明しています。

- 109 ページの『ケース 1. パブリッシュ/サブスクライブ・クラスター』.
- 110 ページの『ケース 2. パブリッシュ/サブスクライブの階層』.

各トピックで、以下の構成タスクはトピック・スペースを結合する方法を説明しています。

- パブリッシュ/サブスクライブ・クラスターでの単一のトピック・スペースの作成。
- 複数のクラスターのトピック・スペースの結合。
- 複数のクラスター内でのトピック・スペースの結合および分離。
- 複数のクラスター内のトピック・スペースに対するパブリッシュおよびサブスクライブ。

ケース 1. パブリッシュ/サブスクライブ・クラスター

この例では、キュー・マネージャーがパブリッシュ/サブスクライブ階層に接続されていないと仮定します。

キュー・マネージャーがパブリッシュ/サブスクライブ・クラスターのメンバーである場合、そのトピック・スペースはローカル・トピックとクラスター・トピックで構成されます。 ローカル・トピックは **CLUSTER** 属性を持たないトピック・オブジェクトに関連付けられています。 あるキュー・マネージャーにローカル・トピック・オブジェクト定義が存在する場合、そのトピック・スペースは、ローカルに定義された独自のトピック・オブジェクトを持つ、クラスター内の別のキュー・マネージャーとは異なります。

パブリッシュ/サブスクライブ・クラスターでは、サブスクライブ先のトピックがクラスター・トピック・オブジェクトに解決される場合を除いて、別のキュー・マネージャー上に定義されたトピックにサブスクライブすることはできません。

トピック・ホスト・ルーティング を使用する場合のように、複数のキュー・マネージャー上に同じ名前のクラスター・トピック・オブジェクトの定義が必要になる場合は、必要な場所ですべての定義が一致することが重要です。 詳しくは、パブリッシュ/サブスクライブ・クラスターでの単一のトピック・スペースの作成を参照してください。

トピック・オブジェクトのローカル定義は、それがクラスター・トピックの定義かローカル・トピックの定義かにかかわらず、クラスター内の別の場所で定義された同じトピック・オブジェクトよりも優先されます。 別の場所で定義されたオブジェクトの方が新しい場合でも、ローカルに定義されたトピックが使用されます。

クラスター内のすべての場所で、同じトピック・ストリングにクラスター・トピック・オブジェクトを関連付けることが重要です。 トピック・オブジェクトが関連付けられたトピック・ストリングを変更することはできません。 同じトピック・オブジェクトを別のトピック・ストリングに関連付けるには、トピック・オブジェクトを削除し、新しいトピック・ストリングを使って再作成する必要があります。 トピックをクラスター化した場合、クラスターの他のメンバーに保管されているトピック・オブジェクトのコピーが削除された後、クラスター内のすべての場所で新しいトピック・オブジェクトのコピーが作成されます。 トピック・オブジェクトのコピーはすべて同じトピック・ストリングを参照します。

クラスター内の異なるキュー・マネージャーで、異なるトピック・ストリングを使って、同じ名前のトピック・オブジェクトの定義を誤って作成してしまうことがあります。 異なるトピック・ストリングを使って同じトピック・オブジェクトを多重定義した場合、トピックが参照される方法と場所に応じて異なる結果が生じる可能性があるので、動作が混乱する場合があります。 この重要なポイントについて詳しくは、同じ名前の複数のクラスター・トピック定義を参照してください。

ケース 2. パブリッシュ/サブスクライブの階層

この例では、キュー・マネージャーがパブリッシュ/サブスクライブ・クラスターのメンバーではないと仮定します。

IBM MQ では、キュー・マネージャーがパブリッシュ/サブスクライブ階層のメンバーである場合、そのトピック・スペースは、ローカルに定義されたすべてのトピックと、接続されているキュー・マネージャーで定義されたすべてのトピックで構成されます。 1 つの階層内のすべてのキュー・マネージャーのトピック・スペースは同じです。 トピックはローカル・トピックとグローバル・トピックに分かれていません。

階層内の別のキュー・マネージャーに接続されたサブスクライバーにトピックのパブリケーションがパブリッシャーから送られることを防ぐには、**PUBSCOPE** オプションおよび **SUBSCOPE** オプションのいずれかを QMGR に設定してください。

キュー・マネージャー QMA 上のトピック・ストリング Alabama を使用して、トピック・オブジェクト USA/Alabama を定義するとします。 結果は次のとおりです。

1. QMA のトピック・スペースには、トピック・オブジェクト Alabama とトピック・ストリング USA/Alabama が含まれるようになりました。
2. アプリケーションまたは管理者は、QMA でトピック・オブジェクト名 Alabama を使用してサブスクリプションを作成できます。
3. アプリケーションは、階層内の任意のキュー・マネージャーに、USA/Alabama を含む任意のトピックへのサブスクリプションを作成できます。 QMA がローカルに定義されていない場合、トピック「USA/Alabama」はトピック・オブジェクト SYSTEM.BASE.TOPIC に分解されます。

関連概念

106 ページの『パブリケーション有効範囲』
パブリッシュ/サブスクライブ・クラスターまたはパブリッシュ/サブスクライブ階層を構成する場合、パブリケーションの有効範囲により、キュー・マネージャーがリモート・キュー・マネージャーにパブリッシュを転送するかどうかをさらに制御することができます。 パブリケーションの有効範囲の管理には、**PUBSCOPE** トピック属性を使用します。

107 ページの『サブスクリプション有効範囲』
サブスクリプションの有効範囲は、1 つのキュー・マネージャーのサブスクリプションが、パブリッシュ/サブスクライブ・クラスターまたは階層の別のキュー・マネージャーでパブリッシュされるパブリケーションを受け取るか、あるいはローカル・パブリッシャーからのみのパブリケーションを受け取るかを制御します。

関連タスク

分散パブリッシュ/サブスクライブ・ネットワークの構成

IBM MQ マルチキャスト

IBM MQ Multicast は、待ち時間が短く、ファンアウトが大きい、高信頼性マルチキャスト・メッセージングです。

マルチキャストは、パフォーマンスに悪影響を与えずに非常に多数のサブスクライバーに拡張することが可能な、効率の良いパブリッシュ/サブスクライブ・メッセージング方式です。 IBM MQ は、肯定応答、否定応答、およびシーケンス番号を使用して、大きなファンアウトで待ち時間の短いメッセージングを実現することにより、信頼性の高いマルチキャスト・メッセージングを可能にします。

IBM MQ Multicast のフェア・デリバリーにより、ほぼ同時の配信が可能になり、特定の受信側が有利になることはありません。 IBM MQ Multicast ではメッセージの配信にネットワークを使用するため、データをファンアウトするためのパブリッシュ/サブスクライブ・エンジンは必要ありません。 トピックがグループ・アドレスにマップされたら、パブリッシャーとサブスクライバーはピアツーピア・モードで作動できるため、キュー・マネージャーは必要ありません。 これにより、キュー・マネージャー・サーバーの負荷が軽減され、キュー・マネージャー・サーバーが潜在的な障害点となることはなくなります。

初期マルチキャストの概念

通信情報 (COMMINFO) オブジェクトを使用して、IBM MQ Multicast を既存のシステムやアプリケーションに簡単に統合できます。 2 つの TOPIC オブジェクト・フィールドを使用して、マルチキャスト・トラフィックをサポートしたり無視したりするように既存の TOPIC オブジェクトを短時間で構成できます。

マルチキャストに必要なオブジェクト

以下の情報は、IBM MQ Multicast に必要な 2 つのオブジェクトの概要です。

COMMINFO オブジェクト
COMMINFO オブジェクトには、マルチキャスト伝送に関連付けられた属性が含まれます。 COMMINFO オブジェクト・パラメーターの詳細については、DEFINE COMMINFO を参照してください。

必ず設定しなければならない COMMINFO フィールドは、COMMINFO オブジェクトの名前だけです。 設定後、この名前はトピックに対して COMMINFO オブジェクトを識別するために使用されます。 COMMINFO オブジェクトの **GRPADDR** フィールドを調べて、値が有効なマルチキャスト・グループ・アドレスであることを確認しなければなりません。

TOPIC オブジェクト
トピックとは、パブリッシュ/サブスクライブ・メッセージでパブリッシュされる情報のサブジェクトのことで、トピックを定義するには TOPIC オブジェクトを作成します。 TOPIC オブジェクト・パラメーターの詳細については、DEFINE TOPIC を参照してください。

TOPIC オブジェクト・パラメーター **COMMINFO** および **MCAST** の値を変更すると、既存のトピックをマルチキャストで使用できます。

- **COMMINFO** パラメーターは、マルチキャスト通信情報オブジェクトの名前を指定します。
- **MCAST** パラメーターは、トピック・ツリー内のこの位置でマルチキャストを許容するかどうかを指定します。 デフォルトでは、**MCAST** は ASPARENT に設定されます。これは、トピックのマルチキャスト属性が親から継承されることを意味します。 **MCAST** を ENABLED に設定すると、このノードでマルチキャスト・トラフィックが許可されます。

マルチキャスト・ネットワークとトピック

以下の情報は、様々なタイプのサブスクリプションとトピック定義を持つサブスクリプションがどうなるかに関する概要です。 これらの例はすべて、TOPIC オブジェクトの **COMMINFO** パラメーターが有効な COMMINFO オブジェクトの名前に設定されていることを前提にしています。

マルチキャスト有効に設定されているトピック
トピック・ストリング **MCAST** パラメーターが ENABLED に設定されていると、以下の場合以外は、マルチキャスト可能なクライアントからのサブスクリプションが許可され、マルチキャスト・サブスクリプションが行われます。

- マルチキャスト可能なクライアントからの永続サブスクリプションである。
- マルチキャスト可能なクライアントからの非管理対象サブスクリプションである。
- マルチキャスト不能なクライアントからのサブスクリプションである。

これらの場合は、非マルチキャスト・サブスクリプションが行われ、サブスクリプションが通常のパブリッシュ/サブスクライブにダウングレードされます。

マルチキャスト無効に設定されているトピック
トピック・ストリング **MCAST** パラメーターが DISABLED に設定されていると、常に非マルチキャスト・サブスクリプションが行われ、サブスクリプションが通常のパブリッシュ/サブスクライブにダウングレードされます。

マルチキャスト専用に設定されているトピック
トピック・ストリング **MCAST** パラメーターが ONLY に設定されていると、以下の場合以外は、マルチキャスト可能なクライアントからのサブスクリプションが許可され、マルチキャスト・サブスクリプションが行われます。

- 永続サブスクリプションです。永続サブスクリプションは拒否され、理由コード 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED が出されます。
- これは非管理サブスクリプションです。管理対象外サブスクリプションは拒否され、理由コード 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR が出されます。
- これは、非マルチキャスト対応クライアントからのサブスクリプションです。これらのサブスクリプションは拒否され、理由コード 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY が出されます。
- これは、ローカルにバインドされたアプリケーションからのサブスクリプションです。これらのサブスクリプションは拒否され、理由コードが 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY が出されます。

Windows Linux AIX MQ Telemetry 概要

MQ Telemetry は、キュー・マネージャーの一部である遠隔測定 (MQXR) サービス、テレメトリー・クライアント (自分で作成することも、無償でダウンロードすることも可能)、コマンド・ラインインターフェース、およびエクスプローラー形式の管理インターフェースで構成されます。 テレメトリーとは、幅広い種類のリモート・デバイスからデータを収集し、それらのデバイスを管理することです。 MQ Telemetry を利用すると、データの収集と、デバイスの制御を Web アプリケーションに統合できます。

MQ Telemetry は IBM MQ のコンポーネントです。 これらのバージョンのアップグレードは、基本的に新しいバージョンの IBM MQ のインストールです。

サンプル・アプリケーションは、引き続き Eclipse Paho および MQTT.org から無料で入手できます。 IBM MQ Telemetry Transport サンプル・プログラムを参照してください。

MQ Telemetry は IBM MQ のコンポーネントであるため、 MQ Telemetry はメイン製品と一緒にインストールすることも、メイン製品のインストール後にインストールすることもできます。 移行情報ついては、「MQ Telemetry 上で移行 Windows」および「MQ Telemetry 上で移行 Linux」を参照してください。

MQ Telemetry には、以下のコンポーネントが含まれています。

遠隔測定チャネル
MQTT クライアントと IBM MQ との接続を管理するには、遠隔測定チャネルを使用します。 テレメトリー・チャネルは、SYSTEM.MQTT.TRANSMIT.QUEUE などの新しい IBM MQ オブジェクトを使用して IBM MQ と対話します。

遠隔測定 (MQXR) サービス
MQTT クライアントは、 SYSTEM.MQXR.SERVICE テレメトリー・サービスを使用してテレメトリー・チャネルに接続します。

MQ Telemetry の IBM MQ Explorer サポート
MQ Telemetry は IBM MQ Explorer を使用して管理されます。

資料
MQ Telemetry 資料は、標準の IBM MQ 製品資料に含まれています。 Java および C クライアント用の SDK 資料は、Javadoc および HTML 形式で製品資料の中に用意されています。

遠隔測定の概念

何をすべきかを決める際は、自分を取り巻く環境から情報を収集するものです。 例えば消費者の立場では、どの食料を買うかを決める前に、蓄えとして何があるかを確認します。 乗り継ぎを予約する前に、今出発すると移動にどれだけ時間がかかるかを知りたいでしょう。 医者に診てもらうかどうかを決める前に、自分の症状を確認します。 バスを待つかどうかを決める前に、バスの到着予定時刻を確認します。 こうした意思決定のための情報は、メーターと装置から、または紙に書かれた言葉または画面から、あるいは自分自身から、直接もたらされます。 どこにいようと、いつやらなければならないとしても、情報を収集してまとめ、分析し、それに基づいて行動します。

情報源が広く分散していたりアクセスできなかったりする場合は、最も正確な情報を収集することは難しくなり、コストもかかるようになります。 加える変更が多い場合や変更することが難しい場合は、変更がなされないか、あまり効果的でないときに変更が行われます。

広く分散した装置からの情報収集とそれらの装置の制御のコストが、デジタル・テクノロジーを搭載した装置をインターネットに接続することによって大幅に削減されるとしたら、どうなるでしょう。 インター

ネットと企業のリソースを使って情報を分析することができます。 情報で裏付けられた判断をし、それに基づいて行動する機会が増えます。

テクノロジー・トレンドと環境面、経済面のプレッシャーが、以下のような変化をもたらしています。

1. 標準化と低価格デジタル・プロセッサーへの接続により、センサーとアクチュエーターの接続と制御のコストが削減されています。
2. 装置を接続するのにインターネットとインターネット・テクノロジーがますます使われています。 国によっては、インターネット・アプリケーションへの接続数の点で、携帯電話がパーソナル・コンピューターを上回っています。 他の装置も確実に後を追っています。
3. インターネットとインターネット・テクノロジーは、アプリケーションによるデータの取得を格段に容易にします。 データに簡単にアクセスできるので、センサーからのデータをさらに多くのソリューションで役立つ情報に変えるためのデータ分析論が使われるようになっています。
4. リソースを賢く使うことは、多くの場合に、炭酸ガス放出とコストを削減するための、より手っ取り早く安上がりな方法です。 別の方法としては、新しいリソースを見つけるか、既存のリソースを利用するための新しいテクノロジーを開発することになりますが、この場合は長期的ソリューションになる可能性があります。 短期間のうちに新しいテクノロジーを開発したり新しいリソースを見つけたりするのは、たいていは既存のソリューションを改良するよりもリスクがあり、時間とコストもかかります。

例

このような傾向が、環境とインテリジェントに連携する新しい機会をいかに生み出すかを、1 つの例が示しています。

「海上における人命の安全のための国際条約」(SOLAS) は、多くの船舶に船舶自動識別装置 (AIS) を配備することを義務付けています。 AIS を義務付けられているのは、300 トンを超える商船と、客船です。 AIS は本来、沿岸運航を対象とした衝突回避システムです。 管海官庁による沿岸水域の監視と管理に使われます。

世界中の熱心な人たちが、低価格 AIS 追跡局を配備して、沿岸運航情報をインターネット上に流しています。 その一方で、AIS からの情報をインターネットからの他の情報と結合するアプリケーションを作成している熱心な人たちもいます。 結果はウェブサイトに掲載され、Twitter や SMS を使って公開される。

あるアプリケーションでは、サウサンプトン近辺の AIS 局からの情報が、船舶所有権および地理情報と結合されます。 このアプリケーションは、フェリーの到着と Twitter への逸脱に関する情報をフィードします。 サウサンプトンとワイト島を結ぶフェリーを使用している定期利用者は、Twitter や SMS を使ってニュース・フィードをサブスクライブしています。 フェリーが遅れている場合は、定期便の到着時間が予定されている時間より遅く、出発を遅らせ、フェリーに乗船することができます。

さらに別の例については、115 ページの『Telemetry のユースケース』を参照してください。

関連タスク

MQ Telemetry のインストール
MQ Telemetry の管理
Windows 上の MQ Telemetry のマイグレーション
Linux 上の MQ Telemetry のマイグレーション
MQ Telemetry 用アプリケーションの開発
MQ Telemetry の問題のトラブルシューティング

関連資料

MQ Telemetry リファレンス

Windows Linux AIX MQ Telemetry の概要

世間一般や企業、行政機関の間で、私たちが生活し、働いている環境との連携を、MQ Telemetry を使ってもっとスマートなものにしたいという願望がますます強くなっています。 MQ Telemetry は、あらゆる種類の装置をインターネットや企業に接続し、スマート・デバイス用アプリケーションの作成コストを削減します。

MQ Telemetry とは何ですか？

- これは IBM MQ の機能で、IBM MQ が提供する汎用のメッセージング・バックボーンを、広範なリモート・センサー、アクチュエーター、および遠隔測定装置に拡張します。 MQ Telemetry による拡張により IBM MQ は、ネットワークや機能の備わったデバイスを使用して、インテリジェントなエンタープライズ・アプリケーション、サービス、および意思決定者と相互接続できるようになります。
- MQ Telemetry のコア部分は、以下のとおりです。

  MQ Telemetry (MQXR) サービス。
  このサービスは、IBM MQ サーバー内部で実行され、IBM MQ Telemetry Transport (MQTT) プロトコルを使用してテレメトリー・デバイスと通信します。

  ユーザーが作成した MQTT アプリケーション。
  これらのアプリケーションは、遠隔測定装置と IBM MQ キュー・マネージャーの間で送受信される情報、およびその情報への応答として行われるアクションを制御します。 このようなアプリケーションを作成する場合は、MQTT クライアント・ライブラリーを使用すると役に立ちます。

利点

- MQTT は、幅広い種類のデバイス用に MQTT 実装環境を作成できるようにするオープン・メッセージング・トランスポートです。
- MQTT クライアントは、リソースが制限されている、フットプリントの小さいデバイスで実行できます。
- MQTT は、処理能力の低いネットワークや、データ送信のコストが高いネットワーク、または不安定なネットワークで効率的に機能します。
- メッセージ・デリバリーは確実であり、アプリケーションからは分離されています。
- アプリケーション・プログラマーには通信プログラミングの知識は 必要ありません。
- メッセージを、他のメッセージング・アプリケーションと交換することができます。 他のメッセージング・アプリケーションとは別のテレメトリー・アプリケーション、 MQI、JMS、または企業のメッセージング・アプリケーションにすることができます。

使用方法

- サンプル・スクリプトは、サンプルの IBM MQ Telemetry Transport v3 クライアント・アプリケーション (`mqttv3app.jar`) を処理する場合に提供されます。IBM MQ Telemetry Transport サンプル・プログラムを参照。
- IBM MQ Explorer とその関連ツールを使用して、IBM MQ のテレメトリー機能を管理します。
- クライアント・ライブラリーを使用すると、キュー・マネージャーに接続してパブリッシュ/サブスクライブ・メッセージングを使用する MQTT アプリケーションの作成に役立ちます。
- アプリケーションを実行するデバイスに、アプリケーションとクライアント・ライブラリーを配布します。

動作方法

- MQTT は、パブリッシュ/サブスクライブ・プロトコルです。 MQTT クライアント・アプリケーションは、MQTT サーバーにメッセージをパブリッシュしたり、MQTT サーバーに接続しているアプリケーションから送信されたメッセージをサブスクライブしたりできます。
- MQTT クライアント・アプリケーションは、MQTT メッセージ・トランスポートを実装するクライアント・ライブラリーを使用します。
- 基本的な MQTT クライアント・アプリケーションは、標準的な MQ クライアントと同じように動作しますが、より多様なプラットフォームおよびネットワークで実行できます。
- MQ Telemetry (MQXR) サービスが、IBM MQ キュー・マネージャーを MQTT サーバーに変えます。
- IBM MQ キュー・マネージャーが MQTT サーバーとして機能する場合、キュー・マネージャーに接続する他のアプリケーションは、MQTT クライアントからのメッセージのサブスクライブおよび受信を行うことができます。

- キュー・マネージャーは、パブリッシュするアプリケーションからサブスクライブするアプリケーションへメッセージを配布するルーターとして動作します。
- メッセージは、異なるタイプのクライアント・アプリケーション間で配布できます。 例えば、Telemetry クライアントと JMS クライアント間などです。

注 : MQ Telemetry は、WebSphere Message Broker のバージョン 7 で廃止された SCADA ノード (現在は IBM Integration Bus と呼ばれている) を置き換えます。そして、Windows、Linux、および AIX 上で実行されます。

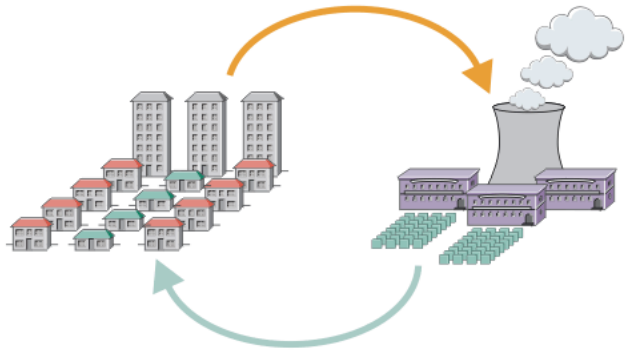
Windows Linux AIX Telemetry のユースケース

テレメトリーは、自動化されたセンシング、データの測定、およびリモート装置の制御を行うものです。特に重要なのは、装置から中央制御点へのデータ伝送です。 遠隔測定には、構成情報と制御情報を装置に送信することも含まれます。

MQ Telemetry は、MQTT protocol を使用して小型装置を接続し、IBM MQ を使用してそれらの装置を他のアプリケーションに接続します。 MQ Telemetry は、装置とインターネットの間のギャップを埋めて、「スマート・ソリューション」の作成を容易にします。 スマート・ソリューションは、装置をモニターして制御するアプリケーションのために、インターネットおよびエンタープライズ・アプリケーションで使用可能な情報の宝庫の錠を開けます。

以下の各図は、MQ Telemetry の代表的な使用例を示したものです。

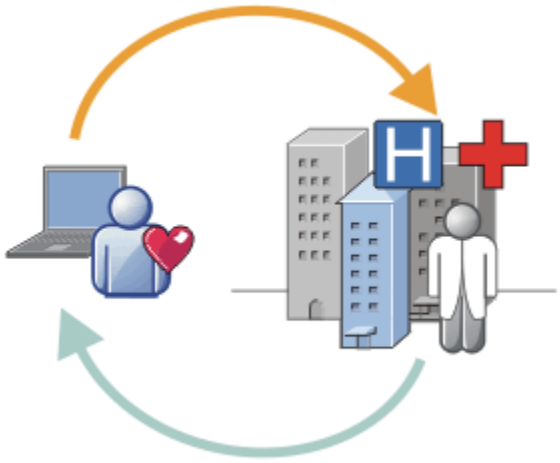
Telemetry: スマート電力

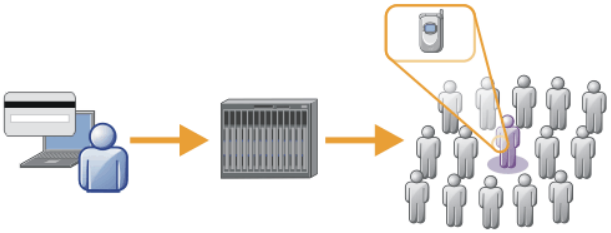


- エネルギー使用量データを含んだ MQTT メッセージがサービス・プロバイダーに送られます。
- MQ Telemetry は、エネルギー使用量データの分析に基づいて制御コマンドを送ります。
- 詳しくは、117 ページの『遠隔測定ユース・ケース: 家庭エネルギーのモニターと制御』のユース・ケースを参照してください。

Telemetry: スマート医療サービス

- MQ Telemetry は、ヘルス・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、116 ページの『遠隔測定ユース・ケース: 在宅患者モニター』のユース・ケースを参照してください。



Telemetry: 大勢の中の一人	
	• シンプルなカード・トランザクションが銀行のサーバーに送られます。 • MQ Telemetry は、数千名の中から一人の個人を特定して、その顧客のカードが使用されているという警報を返します。 • MQ Telemetry は、最もシンプルな入力情報に基づいてその個人を特定します。

サブトピックで説明されているユース・ケースは、実例に基づいたものです。 これらのユース・ケースは、遠隔測定のいくつかの使用法と、遠隔測定テクノロジーで解決されるはずの一般的な問題のいくつかを示しています。

Windows Linux AIX 遠隔測定ユース・ケース: 在宅患者モニター

心臓病患者ケア・システムに関する IBM と医療機関のコラボレーションでは、埋め込み除細動器が病院とコミュニケーションを取ります。 患者と埋め込みデバイスに関するデータが、RF 遠隔測定を使用して患者の自宅内の MQTT 装置に転送されます。

転送は通常、ベッドのそばに置かれた送信機に対して毎晩行われます。 送信機は電話システムを介してデータを病院に安全に転送し、病院でデータが分析されます。

このシステムにより、患者が医師に診てもらう回数が少なくなります。 患者またはデバイスが要注意であることが検出され、緊急の場合には待機医師に通報されます。

IBM と医療機関のこのコラボレーションは、さまざまな遠隔測定ユース・ケースに共通の以下の特性を持っています。

不可視性

電源と電話回線をつなぐことと一日の一部の時間に装置のすぐそばにいること以外に、装置はユーザー介入を必要としません。 確実に操作でき、使い方も簡単です。

患者が装置をセットアップしなくてもよいように、装置サプライヤーが装置を事前構成します。 患者は装置のプラグを差し込むだけでよいのです。 患者による構成がないので、装置の操作が単純になり、装置が誤って構成される可能性が低くなります。

MQTT クライアントが装置の一部として組み込まれます。 装置デベロッパーは装置に MQTT クライアント実装を組み込み、デベロッパーまたはサプライヤーが事前構成の一部として MQTT クライアントを構成します。

MQTT クライアントは、Java SE JAR ファイルとして出荷されます。この JAR ファイルは、開発者が Java アプリケーションに組み込んでいます。 このシナリオのような非 Java 環境の場合、装置デベロッパーは MQTT の公開フォーマットと公開プロトコルを使用して、異なる言語でクライアントを実装できます。 また、開発者は、Windows、Linux、および ARM プラットフォームの共有ライブラリーとして出荷された C クライアントの 1 つを使用することもできます。

不均等な接続性

除細動器と病院のコミュニケーションは、不均等なネットワーク特性を持ちます。 患者からデータを収集することとそのデータを病院に送信することにまつわるさまざまな問題を解決するために、2 つの異なるネットワークが使用されます。 患者と MQTT 装置の間には、近距離低出力 RF ネットワークが使用されます。 送信機は、低帯域幅電話回線を介する VPN TCP/IP 接続を使用して病院に接続します。

何とか工夫してすべての装置をインターネット・プロトコル・ネットワークに直接接続することは、多くの場合、実際的ではありません。 ハブによって接続された 2 つのネットワークを使用するのが、一般的なソリューションです。 MQTT 装置は単純なハブであり、患者からの情報を保管し、それを病院に転送します。

セキュリティー

医師は患者データの確実性を信用できなければならず、患者は自分のデータのプライバシーが尊重されることを望んでいます。

VPN または TLS を使用して接続を暗号化するだけで十分な状況もあります。 一方、保管されたデータもセキュリティー保護しておくことが望ましい状況もあります。

遠隔測定装置がセキュリティー保護されていないこともあります。 例えば、共用住居の中に置かれる可能性があります。 装置のユーザーは、正しい患者からのデータであることを確認するために認証されなければなりません。 装置自体は TLS を使用してサーバーに対して認証することができ、またそのサーバーを装置に対して認証できます。

装置とキュー・マネージャーの間の遠隔測定チャネルは、ユーザー認証のための JAAS と通信暗号化のための TLS、および装置認証をサポートします。 パブリケーションに対するアクセス権限は、IBM MQ のオブジェクト権限マネージャーによって制御されます。

ユーザーを認証するために使用される ID は、共通患者 ID などの異なる ID にマップできます。 共通 ID を使用すると、IBM MQ でパブリケーション・トピックに対する許可を構成するのが簡単になります。

接続性

MQTT 装置と病院の間の接続にはダイヤルアップが使用され、300 ボーという低帯域幅で動作します。

300 ボーで効果的に作動するように、MQTT protocol では、TCP/IP ヘッダーの他にさらにメッセージに追加されるのは数バイトだけです。

MQTT protocol には、待ち時間が少ない単一伝送の「応答不要送信」メッセージングが用意されています。 応答時間よりも送達が保証されることの方が重要である場合は、「最低 *1* 回」および「正確に *1* 回」の送達を保証する複数伝送も使用できます。 送達を保証するために、メッセージは正常に送達されるまで装置で保管されます。 装置がワイヤレス接続される場合に、保証送達は特に有用です。

拡張容易性

遠隔測定装置は通常、数万から数百万まで、大量に配備されます。

多数の装置をシステムに接続するとなると、大きな要求がソリューションに突きつけられます。 装置とそのソフトウェアのコストなどのビジネス上の要求と、ライセンスや装置、ユーザーの管理という管理上の要求があります。 技術上の要求としては、ネットワークおよびサーバーの負荷があります。

接続を開く際に、オープン接続を維持するよりも多くのサーバー・リソースを使用します。 しかし、このような電話回線を使用するユース・ケースでは、接続の費用は、必要な時間しか接続が開かれていないことが重要になります。 データ転送は、大体はバッチ処理される性質のものです。 就寝時間前の突然の接続ピークを回避するために、接続のスケジュールを夜間に分散させることができます。

クライアントに関しては、必要とするクライアント構成が最低限のものであることが、クライアントの拡張容易性に寄与します。 MQTT クライアントは装置に組み込まれます。 構成ステップや MQTT クライアント・ライセンス承諾ステップを患者への装置の配備に組み込まなければならないという要件はありません。

サーバーに関しては、MQ Telemetry の初期ターゲットは、キュー・マネージャー当たり 50,000 個のオープン接続です。

接続は IBM MQ Explorer を使用して管理されます。 IBM MQ Explorer は、表示する接続をフィルターに掛けて、管理しやすい数に絞り込みます。 ID をクライアントに割り振るためのスキームを適切に選択すれば、地理に基づいて、または患者の名前のアルファベット順に、接続をフィルターに掛けることができます。

Windows Linux AIX 遠隔測定ユース・ケース: 家庭エネルギーのモニターと制御

スマート・メーターは、従来型メーターよりも詳細にエネルギー使用量について収集します。

スマート・メーターはしばしば、家庭内の個々の電気製品をモニターおよび制御するために、ローカル遠隔測定ネットワークと結合されます。 ある距離を置いてモニターと制御を行うために、リモート接続されるものもあります。

リモート接続のセットアップを行う場合は、個人または電力事業者 (つまり中央制御点) が行うことになります。 リモート制御点は電力使用量を読み取り、使用量データを提供できます。 また、継続料金設定や気象情報など、使用量に影響するデータを提供できます。 全体発電効率を上げるために、負荷を制限することもできます。

スマート・メーターは、広く配備され始めています。 例えば英国政府は、2020 年までに英国のすべての家庭にスマート・メーターを配備することについて協議しています。

家庭メーターのユース・ケースは、以下のようないくつかの共通特性を持ちます。

不可視性

ユーザーがメーターを利用してエネルギー節約に踏み込むつもりでない限り、メーターはユーザー介入を必要としません。 メーターが個々の電気製品へのエネルギー供給の信頼性を落としてはなりません。

MQTT クライアントはメーターと一体で配備されたソフトウェアに組み込むことができ、別個のインストールや構成を必要としません。

不均等な接続性

電気製品とスマート・メーターの間の通信は、メーターとリモート接続ポイントの間とは異なる接続標準を必要とします。

スマート・メーターから電気製品への接続は可用性が高くなければならず、Home Area Network のネットワーク標準に準拠していなければなりません。

リモート・ネットワークでは、さまざまな物理接続が使用されるでしょう。 その中には、セルラーのように、伝送コストが高く、途切れることがあるものもあります。 MQTT v3 仕様は、リモート接続、およびローカル・アダプターとスマート・メーターの間の接続を対象としています。

電源コンセントと電気製品の間の接続とメーターでは、Zigbee のような Home Area Network が使用されます。 センサー・ネットワーク用 MQTT (MQTT-S) は、Zigbee などの低帯域幅ネットワーク・プロトコルと連動するように設計されています。 MQ Telemetry は、MQTT-S を直接サポートしていません。 MQTT-S を MQTT v3 に接続するためのゲートウェイが必要です。

在宅患者モニターのように、家庭エネルギーのモニターと制御のソリューションも複数のネットワークを必要とし、これらのネットワークはスマート・メーターをハブとして使用して接続されます。

セキュリティー

スマート・メーターに関連するいくつかのセキュリティー問題があります。 トランザクションの否認防止、開始された制御アクションの許可、電力使用量データのプライバシーが問題として挙げられます。

プライバシーを確保するために、メーターとリモート制御点の間で MQTT により転送されるデータを、TLS を使用して暗号化できます。 制御アクションの許可を確かなものにするために、メーターとリモート制御点の間の MQTT 接続を、TLS を使用して相互に認証できます。

接続性

リモート・ネットワークの物理的性質は、大きく異なる可能性があります。 既存のブロードバンド接続を使用する場合もあれば、通話コストが高く使用可能状態が断続的なモバイル・ネットワークを使用する場合もあります。 高コストで途切れることがある接続にとって、MQTT は効率的で信頼性の高いプロトコルです (116 ページの『遠隔測定ユース・ケース: 在宅患者モニター』を参照)。

拡張容易性

電力会社 (つまり中央制御点) は、最終的に数千万台のスマート・メーターを配備することを計画しています。 当初の配備ごとのメーター台数は、数万台から数十万台の規模です。 この数は、MQTT の初期ターゲットであるキュー・マネージャー当たりのオープン・クライアント接続数 50,000 に匹敵します。

家庭エネルギーのモニターと制御のアーキテクチャーの重大な側面は、スマート・メーターをネットワーク・コンセントレーターとして使用する点です。 電気製品用アダプターは、それぞれが別個のセンサーです。 それらを MQTT を使用するローカル・ハブに接続することにより、ハブは中央制御点との

単一 TCP/IP セッションにデータ・フローを集信することができます。また、セッション障害を乗り越えられるように、しばらくの間メッセージを保管できます。

家庭エネルギーのユース・ケースでは、2 つの理由により、リモート接続を開いたままにしておく必要があります。 第 1 に、接続を開くための時間が、要求を送信することと比較して長くかかるためです。 短い間隔で"「負荷制限」"要求を送信するために接続をいくつも開いていては、時間がかかりすぎてしまいます。 第 2 に、電力会社から負荷制限要求を受信するためには、まずクライアントが接続を開かなければなりません。 MQTT の場合、接続は常にクライアントによって開始されます。電力会社から負荷制限要求を受信するには、接続を開いたままにしておく必要があります。

接続を開く速度が重要である場合や、時間が重要な要求をサーバーが開始する場合、解決策としては通常、オープン接続を多数維持しておきます。

Windows Linux AIX 遠隔測定ユース・ケース: Radio Frequency Identification (RFID)

RFID とは、対象をワイヤレスで識別および追跡するために組み込み RFID タグを使用することです。 RFID タグは、RFID リーダーの照準に入っていれば、最大で数メートルの距離から読み取ることができます。 パッシブ・タグは RFID リーダーによってアクティブ化されます。 アクティブ・タグは、外部からの活動化がなくても送信します。 アクティブ・タグには給電部がなければなりません。 パッシブ・タグにも、距離を広げるために給電部を組み込むことができます。

RFID は多くのアプリケーションで使用されており、ユース・ケースのタイプは実にさまざまです。 RFID のユース・ケース、在宅患者モニターのユース・ケース、家庭エネルギーのモニターおよび制御のユース・ケースには、類似点と相違点がいくつかあります。

不可視性

多くのユース・ケースでは、RFID リーダーは大量に配備されるため、ユーザー介入なしで動作しなければなりません。 リーダーには、中央制御点と通信するための組み込み MQTT クライアントが搭載されます。

例えば、流通倉庫では、リーダーはモーション・センサーを使用してパレットを検出します。 パレットに積まれたアイテムの RFID タグをアクティブ化し、中央アプリケーションにデータと要求を送信します。 データは、在庫の場所を更新するために使用されます。 要求は、特定のベイに移動するなどの、パレットに対する次のアクションを制御します。 航空会社と空港のバゲージ (荷物) システムでは、RFID がこのように使用されています。

一部の RFID ユース・ケースでは、リーダーには Java Platform, Micro Edition (Java ME) などの標準的なコンピューティング環境があります。これらのケースでは、MQTT クライアントは、製造後に別個の構成ステップにデプロイされる場合があります。

不均等な接続性

RFID リーダーが MQTT クライアント搭載のローカル制御装置とは別になっている場合もあれば、各リーダーに MQTT クライアントが組み込まれている場合もあります。 通常は、地理的要因や通信上の要因から、トポロジーを選択することになります。

セキュリティー

RFID タグの添付におけるセキュリティー上の懸念は、プライバシーと認証性です。 RFID タグは人目に付かないため、モニターやスプーフ、改ざんがひそかに行われるおそれがあります。

RFID のセキュリティー問題のソリューションにより、新しい RFID ソリューションを配備する機会が増えています。 RFID タグとローカル・リーダーにセキュリティー上の脆弱性があるものの、中央情報処理を使用することが、さまざまな脅威に対抗するためのアプローチになります。 例えば、タグの改ざんは、デリバリーとディスパッチの在庫基準を動的に相互に関連付けることにより、検出される可能性があります。

接続性

RFID アプリケーションには通常、RFID リーダーと即時照会から集められた情報のバッチ型ストア・アンド・フォワードが含まれます。 流通倉庫ユース・ケースでは、RFID リーダーは常に接続されています。 タグが読み取られると、リーダーに関する情報と共にパブリッシュされます。 倉庫アプリケーションが、リーダーに応答をパブリッシュして戻します。

倉庫アプリケーションでは、ネットワークは一般的に信頼性が高いので、即時要求ではパフォーマンス面で待ち時間の少ない「応答不要送信」メッセージが使用されることがあります。 データのバッチ型ストア・アンド・フォワードでは、データ脱落に伴う管理コストを最小にするために、「正確に *1* 回」メッセージングが使用されることがあります。

拡張容易性

RFID アプリケーションが 1 秒か 2 秒程度の即時応答を必要とする場合、RFID リーダーは接続されたままでなければなりません。

Windows Linux AIX 遠隔測定ユース・ケース: 環境センシング

環境センシングでは、遠隔測定を使用して、河川の水位や質、大気汚染物質などの環境データに関する情報を収集します。

センサーはしばしば、有線通信への到達経路のない遠隔地に置かれます。 無線帯域幅はコストがかかり、信頼性が低い場合があります。 通常は、地理上の小区域内のいくつかの環境センサーが、安全な場所に設置されたローカル・モニター装置に接続されます。 このローカル接続は、有線の場合もあれば、無線の場合もあります。

不可視性

センサー・デバイスは、中央モニター装置よりも、その場所に行きにくく、低電力駆動であり、多数配備される傾向があります。 センサーが「ダム」(無言) の場合もあり、ローカル・モニター装置には、センサー・データを変換して保管するためのアダプターが組み込まれます。 モニター・デバイスには、Java Platform, Standard Edition (Java SE) または Java Platform, Micro Edition (Java ME) をサポートする汎用コンピューターが組み込まれている可能性があります。MQTT クライアントを構成するときに、非可視性が主要な要件になる可能性は低いです。

不均等な接続性

センサーの能力と、リモート接続のコストおよび帯域幅は通常、中央サーバーに接続されたローカル・モニター・ハブに依存することになります。

セキュリティー

このソリューションが軍や防衛のユース・ケースで使用されない限り、セキュリティーは大きな要件ではありません。

接続性

多くの場合、連続してモニターすることも、データを直ちに使用できることも、必須ではありません。洪水水位警報などの例外データは、直ちに転送されなければなりません。 センサー・データは、接続コストと通信コストを削減するためにローカル・モニターで集約され、予定接続を使用して転送されます。 例外データは、モニターで検出されると直ちに転送されます。

拡張容易性

センサーはローカル・ハブの周囲に集められ、センサー・データはパケットに集約されて、スケジュールに従って送信されます。 これらの要素は両方とも、直接接続されたセンサーを使用した場合に中央サーバーにかかる負荷を軽減します。

Windows Linux AIX 遠隔測定ユース・ケース: モバイル・アプリケーション

モバイル・アプリケーションとは、ワイヤレス・デバイス上で動作するアプリケーションのことです。 デバイスは、汎用アプリケーション・プラットフォームかカスタム・デバイスのどちらかです。

汎用プラットフォームとしては、電話や携帯情報端末などのハンドヘルド・デバイスと、ノートブック・コンピューターなどのポータブル・デバイスがあります。 カスタム・デバイスでは、特定のアプリケーションに合わせた特殊目的のハードウェアが使用されます。 「受領署名」小荷物配達を記録するためのデバイスは、カスタム・モバイル・デバイスの 1 例です。 カスタム・モバイル・デバイス上のアプリケーションは、多くの場合、汎用ソフトウェア・プラットフォーム上で作成されます。

不可視性

カスタム・モバイル・アプリケーションのデプロイメントは管理されるので、MQTT クライアント・アプリケーションの構成を含めることができます。 不可視性は、MQTT クライアントを構成するときの大きな要件にはならないでしょう。

不均等な接続性

これまでのユース・ケースのローカル・ハブ・トポロジーとは異なり、モバイル・クライアントはリモートで接続します。 クライアント・アプリケーション層は、中央ハブでアプリケーションに直接接続します。

セキュリティー

物理的セキュリティーがほとんどないため、モバイル・デバイスとモバイル・ユーザーは認証されなければなりません。 デバイスの ID の確認には TLS、ユーザーの認証には JAAS が使用されます。

接続性

モバイル・アプリケーションは、無線可能区域に依存する場合は、オフラインで作動できなければならず、また中断接続に効率的に対処できなければなりません。 こうした環境では、接続されたままであることが目標とはいえ、アプリケーションはメッセージのストア・アンド・フォワードを実行できなければなりません。 メッセージは注文や配達確認であることが多く、重要なビジネス価値を持っています。 メッセージのストア・アンド・フォワードを確実に行う必要があります。

拡張容易性

拡張容易性は大きな課題ではありません。 アプリケーション・クライアントの数は、カスタム・モバイル・アプリケーションのユース・ケースの場合、数千あるいは数万を超えることはないと思われます。

Windows Linux AIX キュー・マネージャーへの遠隔測定装置の接続

遠隔測定装置は、MQTT v3 クライアントを使用してキュー・マネージャーに接続しています。 MQTT v3 クライアントは、遠隔測定 (MQXR) サービスと呼ばれる TCP/IP リスナーに、TCP/IP を使用して接続します。

遠隔測定装置をキュー・マネージャーに接続すると、MQTT クライアントが `MqttClient.connect` メソッドを使用して TCP/IP 接続を開始します。 IBM MQ クライアントのように、MQTT クライアントは、メッセージを送受信するにはキュー・マネージャーに接続されなければなりません。 接続は、遠隔測定 (MQXR) サービスと呼ばれる TCP/IP リスナー (MQ Telemetry と共にインストールされる) を使用してサーバーで行われます。 各キュー・マネージャーは、最大 1 つの遠隔測定 (MQXR) サービスを実行します。

遠隔測定 (MQXR) サービスは、各クライアントによって `MqttClient.connect` メソッドで設定されたリモート・ソケット・アドレスを使用して、接続を遠隔測定チャネルに割り振ります。 ソケット・アドレスは、TCP/IP ホスト名とポート番号の組み合わせです。 同じリモート・ソケット・アドレスを使用する複数のクライアントが、遠隔測定 (MQXR) サービスによって同一遠隔測定チャネルに接続されます。

サーバー上に複数のキュー・マネージャーがある場合は、キュー・マネージャー間で遠隔測定チャネルを分割します。 これらのキュー・マネージャー間で、リモート・ソケット・アドレスを割り振ります。 各遠隔測定チャネルに、固有のリモート・ソケット・アドレスを定義します。 2 つの遠隔測定チャネルが同じソケット・アドレスを使用してはなりません。

複数のキュー・マネージャーの遠隔測定チャネルに同じリモート・ソケット・アドレスを構成した場合は、最初に接続する遠隔測定チャネルが成功します。 それ以降に同じアドレスで接続するチャネルは失敗します。

サーバー上に複数のネットワーク・アダプターがある場合は、遠隔測定チャネル間でリモート・ソケット・アドレスを分割します。 特定のソケット・アドレスが 1 つの遠隔測定チャネルのみに構成されている限り、ソケット・アドレスの割り振りは完全に任意となります。

Configure IBM MQ 用の MQTT サプリメントで提供されたウィザードを使用して、MQ Telemetry クライアントへ接続するために IBM MQ Explorer を構成します。 あるいは、Telemetry 対応キュー・マネージャーの構成 (Linux および AIX) および Windows 上のテレメトリー用キュー・マネージャーの構成の説明に従って、手動で遠隔測定を構成します。

関連資料
MQXR プロパティー

Windows Linux AIX Telemetry 接続プロトコル

MQ Telemetry は、TCP/IP IPv4 と IPv6、および TLS をサポートします。

Windows Linux AIX 遠隔測定 (MQXR) サービス

遠隔測定 (MQXR) サービスは TCP/IP リスナーであり、IBM MQ サービスとして管理されます。 このサービスは、IBM MQ Explorer のウィザード、または **runmqsc** コマンドを使用して作成します。

MQ Telemetry (MQXR) サービスは SYSTEM.MQXR.SERVICE

IBM MQ Explorer の MQ Telemetry 機能に用意されているテレメトリー・サンプル構成ウィザードは、テレメトリー・サービスとサンプル・テレメトリー・チャネルを作成します。 IBM MQ Explorer を使用した MQ Telemetry のインストールの検査 を参照してください。

コマンド・ラインからサンプル構成を作成します (『コマンド・ラインを使用した MQ Telemetry のインストールの検査』 を参照)。

遠隔測定 (MQXR) サービスは、キュー・マネージャーと同時に自動的に開始および停止します。 IBM MQ Explorer でサービス・フォルダーを使用して、サービスを制御します。 サービスを表示するには、IBM MQ Explorer が表示から SYSTEM オブジェクトをフィルターで除外するのを停止するためのアイコンをクリックする必要があります。

サービスを手動で作成する方法の例については、以下を参照してください。

- Linux AIX Linux での SYSTEM.MQXR.SERVICE の作成。
- Windows Windows での SYSTEM.MQXR.SERVICE の作成。

これらのトピックでは、MQTT TLS チャネルを暗号化するためにパスフレーズを必要とするデフォルト鍵も指定します。 詳しくは、MQTT TLS チャネルのパスフレーズの暗号化を参照してください。

Windows Linux AIX 遠隔測定チャネル

遠隔測定チャネルを作成して、Java 認証・承認サービス (JAAS) または TLS 認証などのさまざまなプロパティーを持つ接続を作成したり、クライアントのグループを管理したりします。

IBM MQ Explorer の MQ Telemetry 関数で提供される **New Telemetry Channel** ウィザードを使用して、テレメトリー・チャネルを作成します。 このウィザードを使用して、特定の TCP/IP ポートで MQTT クライアントからの接続を受け入れるようにチャネルを構成します。 IBM WebSphere MQ 7.1 以降、コマンド行プログラム **runmqsc** を使用して MQ Telemetry を構成できます。

さまざまなポートについて複数の遠隔測定チャネルを作成し、クライアントをグループに分けることによって、多数のクライアント接続を管理しやすくします。 遠隔測定チャネルごとに別々の名前を持ちます。

セキュリティー属性が異なる遠隔測定チャネルを構成して、異なるタイプの接続を作成することができます。 複数のチャネルを作成して、さまざまな TCP/IP アドレスでクライアント接続を受け入れるようにします。 TLS を使用して、メッセージを暗号化し、遠隔測定チャネルとクライアントを認証するようにします (MQTT クライアントおよびテレメトリー・チャネルの TLS 構成を参照)。 ユーザー ID を指定して、IBM MQ オブジェクトへのアクセスの許可を単純化します。 JAAS 構成を指定して、MQTT ユーザーを JAAS で認証するようにします (MQTT クライアントの識別、許可、および認証を参照)。

Windows Linux AIX IBM MQ Telemetry Transport プロトコル

IBM MQ Telemetry Transport (MQTT) v3 プロトコルは、低帯域幅または高コスト接続を使用する小型デバイス間のメッセージ交換用に設計されており、メッセージを確実に送信することを目的としています。 TCP/IP を使用します。

MQTT protocol がパブリッシュされます。IBM 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。 プロトコルのバージョン 3 は、パブリッシュ/サブスクライブを使用し、「応答不要送信」、「最低 *1* 回」、「正確に *1* 回」の 3 種類のサービス品質をサポートします。

プロトコル・ヘッダーのサイズが小さいことと、メッセージ・ペイロードがバイト配列であることから、小さいメッセージになります。 ヘッダーは、2 バイト固定ヘッダーと、最大 12 バイトの追加可変ヘッダーから成ります。 プロトコルは 12 バイトの可変ヘッダーを使用してサブスクライブと接続を行い、ほとんどのパブリケーションには、わずか 2 バイトの可変ヘッダーを使用します。

3 種類のサービス品質がサポートされているので、待ち時間の少なさと信頼性の間でトレードオフが可能です (MQTT クライアントによって提供されるサービス品質を参照)。 「応答不要送信」は持続的デバイス・ストレージを使用せず、パブリケーションの送受信に伝送を 1 回だけ行います。 「最低 *1* 回」と「正確に *1* 回」は、プロトコル状態を維持して確認応答があるまでメッセージを保存するための持続的ストレージをデバイス上に必要とします。

Windows Linux AIX MQTT クライアント

MQTT クライアント・アプリは、遠隔測定装置からの情報収集、サーバーへの接続、およびサーバーへの情報のパブリッシュを担います。 さらに、トピックにサブスクライブする、パブリケーションを受信する、遠隔測定装置を制御するという処理も行えます。

IBM MQ クライアントアプリケーションとは異なり、MQTT クライアント・アプリケーションは IBM MQ アプリケーションではありません。 MQTT クライアントは接続先のキュー・マネージャーを指定しません。 特定の IBM MQ プログラミング・インターフェースを使用しなければならないという制限を受けません。 代わりに、MQTT クライアントに MQTT 3 プロトコルが実装されます。 MQTT protocol とのインターフェースを提供する独自のクライアント・ライブラリーを、選択したプログラミング言語で、選択したプラットフォーム上に作成することができます。 IBM 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。

MQTT クライアント・アプリケーションを簡単に作成できるように、さまざまなプラットフォームを対象とする Java をカプセル化した C、JavaScript、および MQTT protocol クライアント・ライブラリーを使用します。 これらのライブラリーを MQTT アプリに取り込むと、完全に機能する MQTT クライアントを、15 行ほどの短いコードで書くことができます。 MQTT クライアント・ライブラリーを Eclipse Paho および MQTT.org から無料で入手することができます。 IBM MQ Telemetry Transport サンプル・プログラムを参照してください。

MQTT クライアント・アプリは、遠隔測定チャネルとの接続の開始を常に担っています。 接続されると、MQTT クライアント・アプリか IBM MQ アプリケーションのどちらかがメッセージの交換を開始できます。

MQTT クライアント・アプリと IBM MQ アプリケーションは、同じトピック・セットにパブリッシュとサブスクライブを行います。 IBM MQ アプリケーションは、MQTT クライアント・アプリが最初にサブスクリプションを作成しなくても、クライアント・アプリに直接メッセージを送信することもできます。 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を参照してください。

MQTT クライアント・アプリは、遠隔測定チャネルを使用して IBM MQ に接続されます。 テレメトリー・チャネルは、MQTT および IBM MQ によって使用されるさまざまなタイプのメッセージとの間のブリッジとして機能します。 MQTT クライアント・アプリのためにキュー・マネージャーにパブリケーションとサブスクリプションを作成します。 遠隔測定チャネルは、キュー・マネージャーから MQTT クライアント・アプリへ、MQTT クライアント・アプリのサブスクリプションと一致するパブリケーションを送信します。

Windows Linux AIX MQTT クライアントへのメッセージの送信

IBM MQ アプリケーションは、クライアントによって作成されたサブスクリプションにパブリッシュするか、あるいはメッセージを直接送信して、MQTT v3 クライアントにメッセージを送信できます。 MQTT クライアントは、他のクライアントからサブスクライブされたトピックにパブリッシュすることで、互いにメッセージを送信できます。

MQTT クライアントがパブリケーションにサブスクライブする (このパブリケーションは IBM MQ から受信)

タスク 126 ページの『MQTT クライアント・ユーティリティーへのメッセージを IBM MQ Explorer からパブリッシュする』 を実行して、 IBM MQ から MQTT クライアントにパブリケーションを送信します。

MQTT v3 クライアントがメッセージを受信する標準的な方法は、次のように、クライアントがトピックまたはトピックのセットへのサブスクリプションを作成することです。 124 ページの図 44 のコード・スニペットの例で、MQTT クライアントは、トピック・ストリング "MQTT Examples" を使用してサブスクライブを行います。 IBM MQ C アプリケーション 125 ページの図 45 は、トピック・ストリング "MQTT Examples"を使用してトピックにパブリッシュします。 125 ページの図 46 のコード・スニペットでは、MQTT クライアントは、コールバック・メソッド messageArrived でパブリケーションを受け取ります。

IBM MQ クライアントからのサブスクリプションへの応答としてパブリケーションを送信するように WebSphere MQ を構成する方法 MQTT について詳しくは、MQTT クライアント・サブスクリプションへの応答としてのメッセージのパブリッシュを参照してください。

IBM MQ アプリケーションが MQTT クライアントにメッセージを直接送信する

IBM MQ から MQTT クライアントにメッセージを直接送信するには、130 ページの『MQTT を使用した IBM MQ Explorer クライアントへのメッセージの送信』 のタスクを実行します。

この方法で MQTT クライアントに送信されるメッセージは、非送信請求メッセージと呼ばれます。 MQTT v3 クライアントは、トピック名のセットを持つパブリケーションとして非送信請求メッセージを受け取ります。 遠隔測定 (MQXR) サービスは、トピック名をリモートのキュー名に設定します。

IBM MQ クライアントにメッセージを直接送信するように MQTT を構成する方法について詳しくは、クライアントへのメッセージの直接送信を参照してください。

MQTT クライアントがメッセージをパブリッシュする

MQTT v3 クライアントは、別の MQTT v3 クライアントが受信するメッセージをパブリッシュすることはできますが、非送信請求メッセージを送信することはできません。 コード・スニペットの 125 ページの図 47 は、 Java で作成された MQTT v3 クライアントがメッセージをパブリッシュする方法を示しています。

ある特定の MQTT v3 クライアントにメッセージを送信する場合、各クライアントが独自の ClientIdentifier へのサブスクリプションを作成するのが標準的なパターンです。 トピック・ストリングとして ClientIdentifier を使用して、ある MQTT クライアントから別の MQTT クライアントにメッセージをパブリッシュするには、タスク 132 ページの『特定の MQTT v3 クライアントへのメッセージのパブリッシュ』 を実行します。

コード・スニペットの例

124 ページの図 44 のコード・スニペットは、 Java で作成された MQTT クライアントがサブスクリプションを作成する方法を示しています。 これには、そのサブスクリプションに対するパブリケーションを受信するためのコールバック・メソッド、messageArrived も必要です。

```
String    clientId = String.format("%-23.23s",
                     System.getProperty("user.name") + "_" +
                     (UUID.randomUUID().toString())).trim()).replace('-', '_');
MqttClient  client = new MqttClient("localhost", clientId);
String topicString = "MQTT Examples";
int            QoS = 1;
client.subscribe(topicString, QoS);
```

図 *44. MQTT v3* クライアント・サブスクライバー

125 ページの図 45 のコード・スニペットは、C で書かれた IBM MQ アプリケーションがパブリケーションを送信する方法を示しています。 このコード・スニペットは、可変トピックへのパブリッシャーの作成のタスクから抜き出したものです。

```
/* Define and set variables to.defaults */
/* Omitted lines declaring variables    */
char * topicName   = ""
char * topicString = "MQTT Examples"
char * publication = "Hello world!";
do {
  MQCONN(qMgrName, &Hconn, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
  td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
  strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
  td.ObjectString.VSPtr = topicString;
  td.ObjectString.VSLength = (MQLONG)strlen(topicString);
  MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
  MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
```

図 *45. IBM MQ* パブリッシャー

パブリケーションが到着すると、 MQTT クライアントは MQTT アプリケーション・クライアント `MqttCallback` ・クラスの `messageArrived` メソッドを呼び出します。

```
public class CallBack implements MqttCallback {
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
// ... Other callback methods
}
```

図 *46. messageArrived* メソッド

125 ページの図 47 は、124 ページの図 44 で作成されたサブスクリプションにメッセージをパブリッシュする MQTT v3 を示しています。

```
      String          address = "localhost";
      String         clientId = String.format("%-23.23s",
                                System.getProperty("user.name") + "_" +
                                (UUID.randomUUID().toString())).trim()).replace('-', '_');
      MqttClient       client = new MqttClient(address, clientId);
      String      topicString = "MQTT Examples";
      MqttTopic         topic = client.getTopic(Example.topicString);
      String     publication  = "Hello world";
      MqttMessage     message = new MqttMessage(publication.getBytes());
      MqttDeliveryToken token = topic.publish(message);
```

図 *47. MQTT v3* クライアント・パブリッシャー

Windows Linux AIX MQTT クライアント・ユーティリティーへのメッセージを IBM MQ Explorer からパブリッシュする

この作業のステップに従って、IBM MQ Explorer を使用してメッセージをパブリッシュし、MQTT クライアント・ユーティリティーでそのメッセージにサブスクライブします。 追加作業では、デフォルトの伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定するのではなく、キュー・マネージャー別名を構成する方法を示します。

始める前に

この作業は、ユーザーが IBM MQ および IBM MQ Explorer に精通しており、IBM MQ および MQ Telemetry 機能がインストールされていることを前提としています。

この作業用にキュー・マネージャー・リソースを作成するユーザーには、そのための十分な権限が必要です。 デモンストレーションを目的として、IBM MQ Explorer ユーザー ID は、mqm グループのメンバーであると想定されます。

このタスクについて

この作業では、IBM MQ 内にトピックを作成し、MQTT クライアント・ユーティリティーを使用してそのトピックにサブスクライブします。 IBM MQ Explorer を使用してトピックへのパブリッシュを行うと、MQTT クライアントがパブリケーションを受信します。

手順

以下の作業のいずれかを実行します。

- MQ Telemetry は既にインストールされているが、まだ始動していない場合。 126 ページの『作業の開始時に遠隔測定 (MQXR) サービスが定義されていない場合』の作業を実行します。
- 以前は IBM MQ Telemetry を実行していたが、新しいキュー・マネージャーを使用してデモンストレーションを行う場合。 126 ページの『作業の開始時に遠隔測定 (MQXR) サービスが定義されていない場合』の作業を実行します。
- テレメトリー・リソースが定義されていない既存のキュー・マネージャーを使用して、この作業を実行する場合。 「**サンプル構成の定義**」ウィザードを実行しない場合。
  a. 以下の作業のいずれかを実行して、テレメトリーをセットアップします。
     - Telemetry 対応キュー・マネージャーの構成 (Linux および AIX)
     - Telemetry 対応キュー・マネージャーの構成 (Windows)
  b. 127 ページの『作業の開始時に遠隔測定 (MQXR) サービスが実行されている場合』の作業を実行します。
- テレメトリー・リソースが既に定義されている既存のキュー・マネージャーを使用してこの作業を行う場合には、127 ページの『作業の開始時に遠隔測定 (MQXR) サービスが実行されている場合』の作業を実行します。

次のタスク

130 ページの『MQTT を使用した IBM MQ Explorer クライアントへのメッセージの送信』を実行して、クライアント・ユーティリティーにメッセージを直接送信します。

作業の開始時に遠隔測定 *(MQXR)* サービスが定義されていない場合

キュー・マネージャーを作成し、「**サンプル構成の定義**」を実行して、キュー・マネージャー用にサンプルのテレメトリー・リソースを定義します。 IBM MQ Explorer を使用してメッセージをパブリッシュし、MQTT クライアント・ユーティリティーでそのメッセージにサブスクライブします。

このタスクについて

「**サンプル構成の定義**」を使用してサンプルのテレメトリー・リソースをセットアップすると、このウィザードによってゲスト・ユーザー ID のアクセス権が設定されます。 この方法でゲスト・ユーザー ID に権限

を与えることについては、注意深く検討してください。 Windows 上の guest 、および Linux 上の nobody には、トピック・ツリーのルートへのパブリッシュとサブスクライブ、および SYSTEM.MQTT.TRANSMIT.QUEUE へのメッセージの書き込みを行う権限が付与されています。

またこのウィザードは、デフォルトの伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定します。これを行うと、既存のキュー・マネージャー上で実行されているアプリケーションに干渉する可能性があります。 難しい作業になりますが、デフォルトの伝送キューを使用せずにテレメトリーを構成することも可能です。これを行う場合は、129 ページの『キュー・マネージャー別名の使用』の作業を実行してください。 この作業では、既存のデフォルト伝送キューに干渉する可能性を回避するように、キュー・マネージャーを作成します。

手順

1. IBM MQ Explorer を使用して、新しいキュー・マネージャーを作成および開始します。
   a) Queue Managers「フォルダー」 >「**新規**」 >「**キュー・マネージャー ...**」を右クリックします。 キュー・マネージャーの名前を入力し、>「**完了**」を選択します。

      キュー・マネージャー名 (例えば、MQTTQMGR) を作成します。
2. 遠隔測定 (MQXR) サービスを作成および開始し、サンプルの遠隔測定チャネルを作成します。
   a) Queue Managers*QmgrName*\Telemetry フォルダーを開きます。
   b)「**サンプル構成の定義...**」>「**完了**」をクリックします。

      「**MQTT クライアント・ユーティリティーを起動**」チェック・ボックスは、チェック・マークを付けたままにしておきます。
3. MQTT クライアント・ユーティリティーを使用して、 MQTT Example のサブスクリプションを作成します。
   a)「**接続**」 をクリックします。

      「**クライアント・ヒストリー**」 には、 Connected イベントが記録されます。
   b)「**サブスクリプション \ トピック**」 フィールド>「**サブスクライブ**」に MQTT Example と入力します。

   「**クライアント・ヒストリー**」 には、 Subscribed イベントが記録されます。
4. MQTTExampleTopic 内で IBM MQ を作成します。
   a) **MQ エクスプローラー** >「**新規**」 >「**トピック**」内の Queue Managers*QmgrName*\Topics フォルダーを右クリックします。
   b)「**名前**」に MQTTExampleTopic と入力し、>「**次へ**」を選択します。
   c)「**トピック・ストリング**」 >「**終了**」として MQTT Example と入力します。
   d)「**OK**」をクリックして、確認応答ウィンドウを閉じます。
5. IBM MQ Explorer を使用して、 Hello World! をトピック MQTT Example にパブリッシュします。
   a) Queue Managers*QmgrName*\Topics 内の IBM MQ Explorer フォルダーをクリックします。
   b)「右クリック」MQTTExampleTopic >「**パブリケーションのテスト ...**」
   c)「**メッセージ・データ**」 フィールド>「**メッセージのパブリッシュ**」 >「 MQTT クライアント・ユーティリティーへの切り替え」ウィンドウに Hello World! と入力します。

   「**クライアント・ヒストリー**」 には、 Received イベントが記録されます。

作業の開始時に遠隔測定 *(MQXR)* サービスが実行されている場合

テレメトリー・チャネルおよびトピックを作成します。 トピックおよびテレメトリー伝送キューを使用する権限をユーザーに与えます。 IBM MQ Explorer を使用してメッセージをパブリッシュし、MQTT クライアント・ユーティリティーでそのメッセージにサブスクライブします。

始める前に

この作業バージョンでは、キュー・マネージャー *QmgrName* が定義および実行されています。 遠隔測定 (MQXR) サービスは定義および実行されています。 遠隔測定 (MQXR) サービスは、手動で作成されたか、あるいは「**サンプル構成の定義**」ウィザードを実行して作成された可能性があります。

このタスクについて

この作業では、既存のキュー・マネージャーを構成して、MQTT クライアント・ユーティリティーにパブリケーションを送信します。

この作業のステップ 128 ページの『1』では、デフォルトの伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定します。これを行うと、既存のキュー・マネージャー上で実行されているアプリケーションに干渉する可能性があります。 難しい作業になりますが、デフォルトの伝送キューを使用せずにテレメトリーを構成することも可能です。これを行う場合は、129 ページの『キュー・マネージャー別名の使用』の作業を実行してください。

手順

1. SYSTEM.MQTT.TRANSMIT.QUEUE をデフォルトの伝送キューとして設定します。
   a) Queue Managers*QmgrName* folder > 「**プロパティー...**」 を右クリック
   b) ナビゲーターで「**通信**」をクリックします。
   c)「**選択...**」をクリックし、> SYSTEM.MQTT.TRANSMIT.QUEUE を選択して、>「**OK**」>「**OK**」を選択します。
2. テレメトリー・チャネル MQTTExampleChannel を作成して、 MQTT クライアント・ユーティリティーを IBM MQ に接続し、 MQTT クライアント・ユーティリティーを開始します。
   a)「「**MQ エクスプローラー**」 >「**新規**」 >「**テレメトリー・チャネル...**」の中の Queue Managers*QmgrName* \Telemetry\Channels フォルダーを右クリックします。
   b)「**チャネル名**」フィールドに MQTTExampleChannel と入力し、>「**次へ**」>「**次へ**」を選択します。
   c) クライアント認証パネル上の「**固定ユーザー ID** 」を、「MQTTExample」 >「 **次へ**」にパブリッシュおよびサブスクライブする予定のユーザー ID に変更します。
   d)「**クライアント・ユーティリティーを起動する**」にチェック・マークを付けたままにして、>「**完了**」を選択します。
3. MQTT クライアント・ユーティリティーを使用して、 MQTT Example のサブスクリプションを作成します。
   a)「**接続**」 をクリックします。

   「**クライアント・ヒストリー**」 には、 Connected イベントが記録されます。
   b)「**サブスクリプション \ トピック**」 フィールド>「**サブスクライブ**」に MQTT Example と入力します。

   「**クライアント・ヒストリー**」 には、 Subscribed イベントが記録されます。
4. MQTTExampleTopic 内で IBM MQ を作成します。
   a) **MQ エクスプローラー** >「**新規**」 >「**トピック**」内の Queue Managers*QmgrName*\Topics フォルダーを右クリックします。
   b)「**名前**」に MQTTExampleTopic と入力し、>「**次へ**」を選択します。
   c)「**トピック・ストリング**」 >「**終了**」として MQTT Example と入力します。
   d)「**OK**」をクリックして、確認応答ウィンドウを閉じます。
5. mqm グループ内ではなく、ユーザーが MQTTExample トピックをパブリッシュおよびサブスクライブする場合は、以下のようにします。
   a) 以下のようにして、ユーザーにトピック MQTTExampleTopic のパブリッシュおよびサブスクライブを許可します。

```
setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
```

b) メッセージを SYSTEM.MQTT.TRANSMIT.QUEUE に入れる権限をユーザーに与えます。

```
setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
```

6. IBM MQ Explorer を使用して、 Hello World! をトピック MQTT Example にパブリッシュします。
   a) Queue Managers*QmgrName*\Topics 内の IBM MQ Explorer フォルダーをクリックします。
   b)「右クリック」MQTTExampleTopic >「**パブリケーションのテスト ...**」
   c)「**メッセージ・データ**」 フィールド>「**メッセージのパブリッシュ**」 >「 MQTT クライアント・ユーティリティーへの切り替え」ウィンドウに Hello World! と入力します。

「**クライアント・ヒストリー**」 には、 Received イベントが記録されます。

キュー・マネージャー別名の使用

デフォルト伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定せずに、 IBM MQ Explorer を使用して MQTT クライアント・ユーティリティーにメッセージをパブリッシュします。

この作業は前の作業の続きであり、キュー・マネージャー別名を使用することで、デフォルト伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定することを回避します。

始める前に

126 ページの『作業の開始時に遠隔測定 (MQXR) サービスが定義されていない場合』または 127 ページの『作業の開始時に遠隔測定 (MQXR) サービスが実行されている場合』のいずれかの作業を実行します。

このタスクについて

MQTT クライアントがサブスクリプションを作成すると、IBM MQ は ClientIdentifier をリモート・キュー・マネージャー名として使用して、応答を送信します。 このタスクでは、クライアント ID、MyClient を使用します。

MyClient という名前の伝送キューまたはキュー・マネージャー別名がない場合、応答はデフォルト伝送キューに入れられます。 デフォルト伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定すると、 MQTT クライアントは応答を受け取ります。

キュー・マネージャー別名を使用することで、デフォルト伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定することを回避できます。 キュー・マネージャー別名は、ClientIdentifier ごとにセットアップする必要があります。 通常は、多数のクライアントが存在しており、キュー・マネージャー別名を使用するのは現実的ではありません。 ClientIdentifier が予測不能な場合も多く、この方法でテレメトリーを構成することはできません。

それでもなお、デフォルトの伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE 以外に構成しなければならない状況は存在します。 手順 のステップでは、デフォルト伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に設定する代わりに、キュー・マネージャーの別名を構成します。

手順

1. デフォルトの伝送キューとしての SYSTEM.MQTT.TRANSMIT.QUEUE を削除します。
   a) Queue Managers*QmgrName* folder >「**プロパティー...**」 を右クリック
   b) ナビゲーターで「**通信**」をクリックします。
   c)「**デフォルト伝送キュー**」フィールドから SYSTEM.MQTT.TRANSMIT.QUEUE を削除し、>「**OK**」を選択します。
2. MQTT クライアント・ユーティリティーを使用して、サブスクリプションを作成できなくなったことを確認します。

a)「**接続**」 をクリックします。

「**クライアント・ヒストリー**」 には、 Connected イベントが記録されます。

b)「**サブスクリプション \ トピック**」 フィールド>「**サブスクライブ**」に MQTT Example と入力します。

「**クライアント・ヒストリー**」 には、 Subscribe failed イベントと Connection lost イベントが記録されます。

3. ClientIdentifier のキュー・マネージャー別名、MyClient を作成します。
   a)「Queue Managers*QmgrName*\Queues フォルダー」 >「**新規**」 >「**リモート・キュー定義**」を右クリックします。
   b) 定義に MyClient という名前を付け、>「**次へ**」を選択します。
   c)「**リモート・キュー・マネージャー**」フィールドに MyClient と入力します。
   d)「**伝送キュー**」フィールドに SYSTEM.MQTT.TRANSMIT.QUEUE と入力し、>「**完了**」を選択します。
4. MQTT クライアント・ユーティリティーに再度接続します。
   a)「**クライアント ID**」が MyClient に設定されていることを確認します。
   b) **接続**

「**クライアント・ヒストリー**」 には、 Connected イベントが記録されます。

5. MQTT クライアント・ユーティリティーを使用して、 MQTT Example のサブスクリプションを作成します。
   a)「**接続**」 をクリックします。

「**クライアント・ヒストリー**」 には、 Connected イベントが記録されます。

b)「**サブスクリプション \ トピック**」 フィールド>「**サブスクライブ**」に MQTT Example と入力します。

「**クライアント・ヒストリー**」 には、 Subscribed イベントが記録されます。

6. IBM MQ Explorer を使用して、 Hello World! をトピック MQTT Example にパブリッシュします。
   a) Queue Managers*QmgrName*\Topics 内の IBM MQ Explorer フォルダーをクリックします。
   b)「右クリック」MQTTExampleTopic >「**パブリケーションのテスト ...**」
   c)「**メッセージ・データ**」 フィールド>「**メッセージのパブリッシュ**」 >「 MQTT クライアント・ユーティリティーへの切り替え」ウィンドウに Hello World! と入力します。

「**クライアント・ヒストリー**」 には、 Received イベントが記録されます。

Windows Linux AIX MQTT を使用した IBM MQ Explorer クライアントへのメッセージの送信

MQTT を使用して IBM MQ キューにメッセージを入れることで、IBM MQ Explorer クライアント・ユーティリティーにメッセージを送信します。 この作業では、MQTT クライアントにメッセージを直接送信するように、リモート・キュー定義を構成する方法を示します。

始める前に

126 ページの『MQTT クライアント・ユーティリティーへのメッセージを IBM MQ Explorer からパブリッシュする』のタスクを実行します。 MQTT クライアント・ユーティリティーは、接続したままにします。

このタスクについて

この作業では、トピックへのパブリッシュを使用するのではなく、キューを使用して MQTT クライアントへメッセージを送信する方法について説明します。 クライアントでのサブスクリプションの作成は行いません。 この作業のステップ 131 ページの『2』では、以前のサブスクリプションが削除されていることを確認します。

手順

1. MQTT クライアント・ユーティリティーへの接続を切断してから再接続することで、既存のサブスクリプションをすべて廃棄します。

   デフォルトが変更されていなければ、MQTT クライアント・ユーティリティーは新規セッションで接続するため、サブスクリプションは廃棄されます。クリーン・セッションを参照してください。

   タスクを簡単に実行できるようにするには、 MQTT クライアント・ユーティリティーによって作成された生成済み ClientIdentifier を使用するのではなく、独自の ClientIdentifier を入力します。

   a)「**切断**」をクリックして、テレメトリー・チャネルから MQTT クライアント・ユーティリティーを切断します。

   「**クライアント履歴**」 には、 Disconnected イベントが記録されます。

   b)「**クライアント ID**」を MyClient に変更します。

   c)「**接続**」 をクリックします。

   「**クライアント履歴**」 には、 Connected イベントが記録されます。
2. MQTT クライアント・ユーティリティーが MQTTExampleTopic 用のパブリケーションを受信しなくなっていることを確認します。

   a) Queue Managers*QmgrName*\Topics 内の IBM MQ Explorer フォルダーをクリックします。

   b)「右クリック」MQTTExampleTopic > 「**パブリケーションのテスト ...**」

   c)「**メッセージ・データ**」 フィールド>「**メッセージのパブリッシュ**」 >「 MQTT クライアント・ユーティリティーへの切り替え」ウィンドウに Hello World! と入力します。

   「**クライアント・ヒストリー**」にはイベントは何も記録されません。
3. クライアントのリモート・キュー定義を作成します。

   リモート・キュー定義で、リモート・キュー・マネージャー名として、ClientIdentifier である MyClient を設定します。 リモート・キュー名には任意の名前を使用できます。 リモート・キュー名は、トピック名として MQTT クライアントに渡されます。

   a)「Queue Managers*QmgrName*\Queues フォルダー」 >「**新規**」 >「**リモート・キュー定義**」を右クリックします。

   b) 定義に MyClientRemoteQueue という名前を付け、>「**次へ**」を選択します。

   c)「**リモート・キュー**」フィールドに MQTTExampleQueue と入力します。

   d)「**リモート・キュー・マネージャー**」フィールドに MyClient と入力します。

   e)「**伝送キュー**」フィールドに SYSTEM.MQTT.TRANSMIT.QUEUE と入力し、>「**完了**」を選択します。
4. テスト・メッセージを MyClientRemoteQueue に書き込みます。

   a) **MyClientRemoteQueue** を右クリックし、>「**テスト・メッセージの書き込み...**」を選択します。

   b)「メッセージ・データ」フィールドに Hello queue! と入力し、「**メッセージの書き込み**」>「**閉じる**」を選択します。

   「**クライアント・ヒストリー**」 には、 Received イベントが記録されます。
5. デフォルトの伝送キューとしての SYSTEM.MQTT.TRANSMIT.QUEUE を削除します。

   a) Queue Managers*QmgrName* folder > 「**プロパティー...**」 を右クリック

   b) ナビゲーターで「**通信**」をクリックします。

   c)「**デフォルト伝送キュー**」フィールドから SYSTEM.MQTT.TRANSMIT.QUEUE を削除し、>「**OK**」を選択します。
6. ステップ 131 ページの『4』をやり直します。

   MyClientRemoteQueue は、伝送キューの名前を明示的に指定するリモート・キュー定義です。 MyClient にメッセージを送信するためにデフォルト伝送キューを定義する必要はありません。

次のタスク

デフォルト伝送キューが SYSTEM.MQTT.TRANSMIT.QUEUE に設定されなくなったため、 MQTT クライアント・ユーティリティーは、 ClientIdentifier、 MyClient にキュー・マネージャー別名が定義されていない限り、新しいサブスクリプションを作成できません。 デフォルト伝送キューを SYSTEM.MQTT.TRANSMIT.QUEUE に復元します。

Windows Linux AIX 特定の MQTT v3 クライアントへのメッセージのパブリッシュ

トピック名として ClientIdentifier を使用し、パブリッシュ/サブスクライブ・ブローカーとして IBM MQ を使用して、ある MQTT v3 クライアントから別のクライアントにメッセージをパブリッシュします。

始める前に

126 ページの『MQTT クライアント・ユーティリティーへのメッセージを IBM MQ Explorer からパブリッシュする』のタスクを実行します。 MQTT クライアント・ユーティリティーは、接続したままにします。

このタスクについて

この作業では、以下の 2 つについて説明します。

1. 1 つの MQTT クライアントでトピックへのサブスクライブを行い、別の MQTT クライアントからのパブリケーションを受け取ります。
2. トピック・ストリングとして ClientIdentifier を使用して、「Point-to-Point」サブスクリプションをセットアップします。

手順

1. MQTT クライアント・ユーティリティーへの接続を切断してから再接続することで、既存のサブスクリプションをすべて廃棄します。

   デフォルトが変更されていなければ、MQTT クライアント・ユーティリティーは新規セッションで接続するため、サブスクリプションは廃棄されます。クリーン・セッションを参照してください。

   タスクを簡単に実行できるようにするには、 MQTT クライアント・ユーティリティーによって作成された生成済み ClientIdentifier を使用するのではなく、独自の ClientIdentifier を入力します。

   a)「**切断**」をクリックして、テレメトリー・チャネルから MQTT クライアント・ユーティリティーを切断します。

   「**クライアント履歴**」 には、 Disconnected イベントが記録されます。

   b)「**クライアント ID**」を MyClient に変更します。

   c)「**接続**」 をクリックします。

   「**クライアント履歴**」 には、 Connected イベントが記録されます。
2. トピック MyClient へのサブスクリプションを作成します。

   MyClient は、このクライアントの ClientIdentifier です。

   a)「**サブスクリプション¥トピック**」フィールドに MyClient と入力し、>「**サブスクライブ**」を選択します。

   「**クライアント・ヒストリー**」 には、 Subscribed イベントが記録されます。
3. 別の MQTT クライアント・ユーティリティーを開始します。

   a) Queue Managers*QmgrName*\Telemetry\channels フォルダーを開きます。

   b)「**PlainText**」チャネルを右クリックし、>「**MQTT クライアント・ユーティリティーを実行...**」を選択します。

   c)「**接続**」 をクリックします。

「**クライアント履歴**」 には、 Connected イベントが記録されます。

4. トピック MyClient に Hello MyClient! をパブリッシュします。
   a) クライアント ID、MyClient で実行されている MQTT クライアント・ユーティリティーから、定期購読トピック MyClient をコピーします。
   b) MyClient クライアント・ユーティリティー・インスタンスのそれぞれの **パブリケーション\トピック** フィールドに MQTT を貼り付けます。
   c)「**パブリケーション¥メッセージ**」フィールドに Hello MyClient! と入力します。
   d) 両方のインスタンスで「**パブリッシュ**」をクリックします。

タスクの結果

クライアント ID、MQTT を使用する MyClient クライアント・ユーティリティーのクライアント・ヒストリーは、2 つの**受信済み**イベントおよび 1 つの**公開済み**イベントを記録します。 もう一方の MQTT クライアント・ユーティリティー・インスタンスには、1 つの「**パブリッシュ済み**」イベントが記録されます。

1 つの「**受信済み**」イベントが表示されている場合は、以下の原因が考えられますので確認してください。

1. キュー・マネージャーのデフォルト伝送キューが SYSTEM.MQTT.TRANSMIT.QUEUE に設定されている。
2. 他の演習を実行する際に、MyClient を参照するキュー・マネージャー別名またはリモート・キュー定義を作成しましたか？構成上の問題が発生した場合は、MyClient を参照するリソース (キュー・マネージャー別名や伝送キューなど) を削除します。 クライアント・ユーティリティーへの接続を切断し、遠隔測定 (MQXR) サービスを停止して再開します。

Windows Linux AIX IBM MQ クライアントから MQTT アプリケーションへのメッセージの送信

IBM MQ アプリケーションは、トピックにサブスクライブすることにより、MQTT v3 クライアントからメッセージを受信できます。 MQTT クライアントは、遠隔測定チャネルを使用して IBM MQ に接続し、同じトピックにパブリッシュすることにより、メッセージを IBM MQ アプリケーションに送信します。

133 ページの『IBM MQ クライアントからの MQTT へのメッセージのパブリッシュ』のタスクを実行して、MQTT クライアントから IBM MQ で定義されているサブスクリプションにパブリケーションを送信する方法を学習してください。

トピックがクラスター化されている場合、またはパブリッシュ/サブスクライブ階層を使用して分散されている場合は、サブスクリプションを、MQTT クライアントが接続されているキュー・マネージャーとは異なるキュー・マネージャー上に置くことができます。

Windows Linux AIX IBM MQ クライアントからの MQTT へのメッセージのパブリッシュ

IBM MQ Explorer を使用して、トピックへのサブスクリプションを作成し、MQTT クライアント・ユーティリティーを使用して、トピックにパブリッシュします。

始める前に

126 ページの『MQTT クライアント・ユーティリティーへのメッセージを IBM MQ Explorer からパブリッシュする』のタスクを実行します。 MQTT クライアント・ユーティリティーは、接続したままにします。

このタスクについて

その作業は MQTT クライアントを使用してメッセージのパブリッシングと IBM MQ Explorer を使用して作成された非管理永続サブスクリプションを使用してパブリケーションの受入を実演します。

手順

1. トピック・ストリング MQTT Example に対する永続サブスクリプションを作成します。

   IBM MQ Explorer を使用して、以下のステップを実行し、キューおよびサブスクリプションを作成します。

   a) Queue Managers*QmgrName*\Queues フォルダーから「IBM MQ Explorer」>「**新規**」>「**ローカル・キュー...**」をクリックします。

   b) キュー名として MQTTExampleQueue と入力し、「**終了**」をクリックします。

   c) Queue Managers*QmgrName*\Subscriptions フォルダーから「IBM MQ Explorer」>「**新規**」>「**サブスクリプション...**」を右クリックします。

   d) キュー名として MQTTExampleSubscription と入力し、「**次へ**」をクリックします。

   e)「**選択...**」>「MQTTExampleTopic」>「**OK**」をクリックします。

   MQTTExampleTopic のステップ 127 ページの『4』 でトピック「126 ページの『MQTT クライアント・ユーティリティーへのメッセージを IBM MQ Explorer からパブリッシュする』」が既に作成されています。

   f) 宛先名として MQTTExampleQueue と入力し、「**終了**」をクリックします。

2. 任意指定のステップとして、mqm 権限を持たずに、異なるユーザーが使用できるようにキューをセットアップします。

   mqm 以外の少ない権限を有するユーザーのために設定をセットアップしている場合、put と get 権限を MQTTExampleQueue に付与すべきです。 トピックと伝送キューへのアクセス権限は 126 ページの『MQTT クライアント・ユーティリティーへのメッセージを IBM MQ Explorer からパブリッシュする』 で構成されています。

   a) 以下のようにして、ユーザーにキュー MQTTExampleQueue の書き込みと取得を認可します。

   ```
   setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
   ```

3. MQTT クライアント・ユーティリティーを使用して、 Hello IBM MQ! をトピック MQTT Example にパブリッシュします。

   MQTT クライアント・ユーティリティーが接続されたままの状態でない場合は、「**PlainText**」チャネルを右クリックして、「**MQTT クライアント・ユーティリティーを実行...**」>「**接続**」をクリックします。

   a)「**パブリケーション¥トピック**」フィールドに MQTT Example と入力します。

   b)「**パブリケーション \ メッセージ**」 フィールド>「**パブリッシュ**」に Hello IBM MQ! と入力します。

4. Queue Managers*QmgrName*\Queues フォルダーを開き、MQTTExampleQueue を見つけます。

   「**現行キュー項目数**」フィールド値は 1 です。

5. MQTTExampleQueue >「**メッセージの参照 ...**」 を右クリックします。 資料を調べてください。

Windows Linux AIX MQTT パブリッシュ/サブスクライブ・アプリケーション

トピック・ベースのパブリッシュ/サブスクライブを使用して MQTT アプリケーションを作成します。

MQTT クライアントが接続されると、クライアントとサーバーの間でどちらかの方向にパブリケーションがフローされます。 クライアントで情報がパブリッシュされる場合、パブリケーションはクライアントから送信されます。 クライアントによって作成されたサブスクリプションと一致するトピックにメッセージがパブリッシュされた場合、パブリケーションはクライアントで受信されます。

IBM MQ パブリッシュ/サブスクライブ・ブローカーは、MQTT クライアントによって作成されたトピックとサブスクリプションを管理します。 MQTT クライアントによって作成されたトピックは、IBM MQ アプリケーションによって作成されたトピックと同じトピック・スペースを共有します。

MQTT クライアント・サブスクリプションのトピック・ストリングと一致するパブリケーションは、リモート・キュー・マネージャー名がクライアントの ClientIdentifier に設定された SYSTEM.MQTT.TRANSMIT.QUEUE に置かれます。 遠隔測定 (MQXR) サービスは、サブスクリプションを作成したクライアントにパブリケーションを転送します。 その場合、クライアントを識別するリモート・キュー・マネージャー名として設定されている ClientIdentifier が使用されます。

通常は、SYSTEM.MQTT.TRANSMIT.QUEUE をデフォルトの伝送キューとして定義する必要があります。 デフォルトの伝送キューを使用しないように MQTT を構成することはできますが、煩わしい作業です (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を参照)。

MQTT クライアントは持続セッションを作成できます (138 ページの『MQTT のステートレス・セッションとステートフル・セッション』を参照)。 持続セッションで作成されたサブスクリプションは永続的です。 持続セッションを持つクライアント宛に到着したパブリケーションは、SYSTEM.MQTT.TRANSMIT.QUEUE に保管され、クライアントが再接続した時点でクライアントに転送されます。

MQTT クライアントは、保存パブリケーションにパブリッシュとサブスクライブを行うこともできます (保存パブリケーションおよび MQTT クライアントを参照)。 保存パブリケーション・トピックのサブスクライバーは、トピックへの最新のパブリケーションを受信します。 サブスクライバーは、サブスクリプションの作成時、または以前のセッションへの再接続時に保存パブリケーションを受信します。

Windows Linux AIX 遠隔測定アプリケーション

IBM MQ または IBM Integration Bus メッセージ・フローを使用してテレメトリー・アプリケーションを作成します。

JMS や MQI などの IBM MQ プログラミング・インターフェースを使用して、IBM MQ の遠隔測定アプリケーションをプログラムします。

遠隔測定 (MQXR) サービスは、MQTT v3 メッセージと IBM MQ メッセージの間の変換を行います。 MQTT クライアントのためにサブスクリプションとパブリケーションを作成し、パブリケーションを MQTT クライアントに転送します。 パブリケーションは MQTT v3 メッセージのペイロードです。 ペイロードは、メッセージ・ヘッダーと jms-bytes フォーマットのバイト配列から成ります。 遠隔測定サーバーは、MQTT v3 メッセージのヘッダーと IBM MQ メッセージのヘッダーをマップします (135 ページの『MQ Telemetry とキュー・マネージャーの統合』を参照)。

Publication、MQInput、および JMSInput ノードを使用して、IBM Integration Bus と MQTT クライアントの間でパブリケーションを送受信します。

メッセージ・フローを使用して、テレメトリーを HTTP を使用して Web サイトと統合したり、 IBM MQ および WebSphere Adapters を使用して他のアプリケーションと統合したりすることができます。

Windows Linux AIX MQ Telemetry とキュー・マネージャーの統合

MQTT クライアントは、パブリッシュ/サブスクライブ・アプリケーションとして IBM MQ と統合されます。 新しいトピックを作成するか、既存のトピックを使用して、IBM MQ 内のトピックに対して、パブリッシュとサブスクライブのどちらも行えます。 自身のサブスクリプションのトピックにパブリッシュする IBM MQ クライアント (自身を含む) や他の MQTT アプリケーションの結果として、IBM MQ からパブリケーションを受信します。 パブリケーションの属性を決定するために規則が適用されます。

トピック、パブリケーション、サブスクリプション、メッセージに関連付けられる属性のうち、IBM MQ で定義された属性の多くはサポートされません。 136 ページの『MQTT クライアントから IBM MQ パブリッシュ/サブスクライブ・ブローカーへ』と 137 ページの『IBM MQ クライアントへの MQTT』は、パブリケーションの属性がどのように設定されるかを示しています。 設定は、パブリケーションが IBM MQ パブリッシュ/サブスクライブ・ブローカーに向かうものであるか、そこからのものであるかによって異なります。

IBM MQ 内のパブリッシュ/サブスクライブ・トピックは、管理可能トピック・オブジェクトに関連付けられます。 MQTT クライアントによって作成されるトピック間に違いはありません。 MQTT クライアントがパブリケーション用のトピック・ストリングを作成すると、IBM MQ パブリッシュ/サブスクライブ・ブローカーはそれを管理トピック・オブジェクトに関連付けます。 ブローカーはパブリケーションに含まれるトピック・ストリングを、最も近い管理可能トピック・オブジェクトの親にマップします。 このマッピングは、IBM MQ アプリケーションの場合と同じです。 ユーザーが作成したトピックがない場合、パブリケ

ーション・トピックは SYSTEM.BASE.TOPIC にマップされます。 パブリケーションに適用される属性は、トピック・オブジェクトに由来します。

IBM MQ アプリケーションまたは管理者がサブスクリプションを作成すると、そのサブスクリプションに名前が付けられます。サブスクリプションの一覧 IBM MQ Explorer、または **runmqsc** または PCF コマンド。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。 これらには次の形式の名前が付けられます。*ClientIdentifier*:*Topic name*

MQTT クライアントから IBM MQ パブリッシュ/サブスクライブ・ブローカーへ

MQTT クライアントが IBM MQ にパブリケーションを送信しました。 遠隔測定 (MQXR) サービスが、パブリケーションを IBM MQ メッセージに変換します。 IBM MQ メッセージは、以下の 3 つの部分で構成されます。

1. MQMD
2. RFH2
3. メッセージ

MQMD の各プロパティーは、136 ページの表 9 に示すものを除いて、それぞれのデフォルト値に設定されます。

表 *9. MQMD* プロパティーの設定

MQMD フィールド	**タイプ**	**値**
Format	MQCHAR8	MQFMT_RF_HEADER_2
UserIdentifier	MQCHAR12	以下のいずれかに設定されます。 MqttClient.ClientIdentifier MqttConnectOptions.UserName IBM MQ 管理者によって設定された遠隔測定チャネルのユーザー ID。
Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (4 のデフォルトを有する IBM MQ とは異なる JMS のデフォルト)
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

RFH2 ヘッダーには、 JMS メッセージのタイプを定義するための <msd> フォルダーは含まれていません。遠隔測定 (MQXR) サービスによって、IBM MQ メッセージがデフォルトの JMS メッセージとして作成されます。 デフォルトの JMS メッセージ・タイプは jms-bytes メッセージです。 アプリケーションからは、メッセージ・プロパティーとして追加されたヘッダー情報にアクセス可能です (メッセージ・プロパティーを参照)。

RFH2 の値は、136 ページの表 10 に示すように設定されます。 Format プロパティーは RFH2 固定ヘッダーに設定され、他の値は RFH2 フォルダーに設定されます。

表 *10. RFH2* プロパティーの設定

RFH2 プロパティー	**タイプ/フォルダー**	**ヘッダー**
形式	MQCHAR8	MQFMT_NONE

表 *10. RFH2* プロパティーの設定 (続き)		
RFH2 プロパティー	**タイプ/フォルダー**	**ヘッダー**
ClientIdentifier	mqtt/clientId	長さが 1...23 バイトの MqttClient.ClientIdentifier をコピーします。
QoS	mqtt/qos	QoS を着信 MQTT メッセージからコピーします。
メッセージ ID	mqtt/msgid	QoS が 1 または 2 の場合は、着信 MQTT メッセージから 「メッセージ ID」 をコピーします。
MQIsRetained	mqps/Ret	元の MQTT パブリケーションが RETAIN プロパティー・セットが設定されて送信された場合に設定し、メッセージは保存パブリケーションとして受信されます。
MQTopicString	mqps/Top	MQTT メッセージがパブリッシュされたトピック。

MQTT パブリケーションに含まれるペイロードは、IBM MQ メッセージの内容に次のようにマップされます。

表 *11. MQTT* パブリケーションのペイロードと *IBM MQ* メッセージの内容のマッピング		
メッセージの内容	**タイプ**	**IBM MQ メッセージの内容**
BUFFER	MQBYTE *n*	着信 MQTT メッセージに含まれるバイトのコピー。 長さがゼロの場合もあります。

IBM MQ クライアントへの MQTT

クライアントがパブリケーション・トピックにサブスクライブしました。 IBM MQ アプリケーションがトピックにパブリッシュした結果、MQTT パブリッシュ/サブスクライブ・ブローカーによってパブリケーションが IBM MQ サブスクライバーに送信されました。 あるいは、IBM MQ アプリケーションが非送信請求メッセージを直接 MQTT クライアントに送信しました。 137 ページの表 12 は、MQTT クライアントに送信されるメッセージで固定メッセージ・ヘッダーがどのように設定されるかを示しています。 IBM MQ メッセージ・ヘッダーの他のデータ、または他のヘッダーは破棄されます。 IBM MQ メッセージのメッセージ・データは、変更されることなく、MQTT メッセージのメッセージ・ペイロードとして送信されます。 MQTT メッセージは、遠隔測定 (MQXR) サービスによって MQTT クライアントに送信されます。

表 *12. MQTT* クライアントに送信される *IBM MQ* メッセージでの固定メッセージ・ヘッダーの設定方法		
MQTT フィールド	**タイプ**	**値**
DUP	ブール値	QoS = 1 または 2 であり、前の伝送でこのクライアントにメッセージが送信されており、一定の時間が経過してもメッセージの確認応答がない場合に設定されます。

表 12. *MQTT* クライアントに送信される *IBM MQ* メッセージでの固定メッセージ・ヘッダーの設定方法 (続き)

MQTT フィールド	タイプ	値
QoS	int	IBM MQ のパブリッシュ/サブスクライブ・ブローカーからの発信パブリケーションにおける QoS の値の設定方法は、着信パブリケーションによって異なります。 それは、着信パブリケーションが MQTT クライアントから送信されたものであるか IBM MQ アプリケーションから送信されたものであるかによって異なります。 **MQTT** 着信パブリケーションでの QoS とサブスクライバーによって要求された QoS のうち、低い方の値。 **IBM MQ** 着信パブリケーションから派生した QoS の低い方の値 MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2 およびサブスクライバーにより要求された QoS。 サブスクリプションなしでクライアントにメッセージが送信される場合、QoS はデフォルトで 2 に設定されます。 クライアントはこの値を、異なる QoS を指定して DEFAULT.QoS にサブスクライブすることによって変更できます。
RETAIN	ブール値	着信パブリケーションに保存プロパティー・セットがある場合に設定されます。

138 ページの表 13 は、MQTT クライアントに送信される MQTT メッセージで可変メッセージ・ヘッダーがどのように設定されるかを示しています。

表 13. *MQTT* クライアントに送信される *MQTT* メッセージで *MQTT* 可変ヘッダー・プロパティーが設定される方法

MQTT フィールド	タイプ	値
Topic name	文字列	パブリッシュされたメッセージに含まれるトピック・ストリング。
Message ID	文字列	パブリケーションが SYSTEM.MQTT.TRANSMIT.QUEUE に入れられたときのパブリケーションの MQMD.MsgId プロパティーの最後の 2 バイト。
Payload	byte[]	着信パブリケーションからパブリッシュ/サブスクライブ・ブローカーへのバイトの直接コピー。 長さがゼロの場合もあります。

Windows Linux AIX MQTT のステートレス・セッションとステートフル・セッション

MQTT クライアントは、キュー・マネージャーを使用してステートフル・セッションを作成できます。 ステートフル MQTT クライアントが切断した場合、キュー・マネージャーは、クライアントによって作成されたサブスクリプションと未完了メッセージを維持します。 クライアントが再接続されると、未完了メッセージを解決します。 配信用のキューに入れられているすべてのメッセージを送信し、切断中にそのサブスクリプションに対してパブリッシュされたすべてのメッセージを受信します。

MQTT クライアントは、遠隔測定チャネルに接続すると、新しいセッションを開始するか、前のセッションを再開します。 新しいセッションには、確認応答のない未解決メッセージも、サブスクリプションも、

配信を待機しているパブリケーションもありません。 クライアントは接続するときに、クリーン・セッションから開始するか、既存のセッションを再開するかを指定します (クリーン・セッションを参照)。

クライアントが既存のセッションを再開すると、接続の切断はなかったかのようにセッションが続行されます。 配信を待っているパブリケーションはクライアントに送信され、コミットされていないメッセージの転送はすべて完了されます。 クライアントが持続セッションで遠隔測定 (MQXR) サービスから切断しても、クライアントが作成したサブスクリプションは残ります。 サブスクリプションに対応するパブリケーションが、クライアントが再接続した時点でクライアントに送信されます。 クライアントが前のセッションを再開せずに再接続した場合、パブリケーションは遠隔測定 (MQXR) サービスによって破棄されます。

セッション状態情報は、キュー・マネージャーによって SYSTEM.MQTT.PERSISTENT.STATE キューに保存されます。

IBM MQ 管理者は、セッションを切断してパージすることができます。

Windows Linux AIX MQTT クライアントが接続されていないとき

クライアントが接続されていないときも、キュー・マネージャーはクライアントのためにパブリケーションの受信を続行できます。 こうしたパブリケーションは、クライアントが再接続した時点でクライアントに転送されます。 クライアントが予期せず切断した場合、キュー・マネージャーはクライアントに代わって、この"「遺言」"をパブリッシュします。

クライアントが予期せず切断したときに通知されるようにする場合は、遺言パブリケーションを登録できます (遺言パブリケーションを参照)。 この「遺言」は、クライアントの要求なしにクライアントへの接続が切断されていることを遠隔測定 (MQXR) サービスが検出した場合に、遠隔測定サービスによって送信されます。

クライアントは保存パブリケーションをいつでもパブリッシュできます (保存パブリケーションおよび MQTT クライアントを参照)。 トピックへの新規サブスクリプションは、トピックに関連付けられた保存パブリケーションを送信するよう要求できます。 「遺言」を保存パブリケーションとして作成すると、それを使用してクライアントの状況をモニターすることができます。

例えば、クライアントは接続時に保存パブリケーションをパブリッシュして、自身が使用可能であることを広告します。 同時に、自身が使用不可であることを通告するための「遺言」保存パブリケーションも作成します。 さらに、計画的な切断を行う直前に、自身が保存パブリケーションとして使用不可であることをパブリッシュします。 クライアントが使用可能かどうかを知るには、その保存パブリケーションのトピックにサブスクライブすることになります。 常に、3 つのパブリケーションのうちの 1 つを受信することになります。

クライアントがその切断中にパブリッシュされたメッセージを受信するようにする場合は、クライアントを前のセッションに再接続します (138 ページの『MQTT のステートレス・セッションとステートフル・セッション』を参照)。 前のセッションのサブスクリプションは、削除されるかクライアントがクリーン・セッションを作成するまではアクティブです。

Windows Linux AIX MQTT クライアントと IBM MQ アプリケーションの疎結合

MQTT クライアントと IBM MQ アプリケーション間のパブリケーションのフローは疎結合されています。 パブリケーションは、MQTT クライアントまたは IBM MQ アプリケーションのいずれかから、順不同で発信されます。 パブリッシャーとサブスクライバーは疎結合です。 これらは、パブリケーションとサブスクリプションを介して間接的に対話します。 MQTT アプリケーションから IBM MQ クライアントにメッセージを直接送信することもできます。

MQTT クライアントと IBM MQ アプリケーションは、以下の 2 つの理由により、疎結合していると言えます。

1. パブリッシャーとサブスクライバーは、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との関連により疎結合しています。 パブリッシャーとサブスクライバーは、通常、パブリケーションまたはサブスクリプションの他のソースのアドレスまたは ID を認識しません。
2. MQTT クライアントは、パブリケーションのパブリッシュ、サブスクライブ、受信、および送達確認の処理を、別々のスレッドで行います。

MQTT クライアント・アプリケーションは、パブリケーションの送達を待機しません。 アプリケーションは、メッセージを MQTT クライアントに渡してから、自身のスレッドの実行を続けます。 アプリケーションとパブリケーションの送達の同期を取るのに送達トークンが使用されます (送達トークンを参照)。

MQTT クライアントにメッセージを渡した後、アプリケーションには送達トークンを待つという選択肢があります。 クライアントは、待機するのではなく、パブリケーションが IBM MQ に送達されると呼び出されるコールバック・メソッドを定義できます。 このコールバック・メソッドは、送達トークンを無視することもできます。

メソッドに関連付けられたサービス品質によって、送達トークンはコールバック・メソッドに直ちに返されるか、場合によってはかなり時間がたってから返されます。 送達トークンは、クライアントが切断して再接続した後でも返されることがあります。 サービス品質が「応答不要送信」である場合、送達トークンは直ちに返されます。 これ以外の 2 つの場合は、サブスクライバーにパブリケーションが送信されたという確認応答をクライアントが受信したときのみ、送達トークンが返されます。

クライアント・サブスクリプションの結果として MQTT クライアントに送信されたパブリケーションは、`messageArrived` コールバック・メソッドに送達されます。 `messageArrived` は、メインアプリケーションとは別スレッドで動作します。

MQTT クライアントへのメッセージの直接送信

2 つの方法のいずれかを使用して、特定の MQTT クライアントにメッセージを送信することができます。

1. IBM MQ アプリケーションは、サブスクリプションなしで MQTT クライアントにメッセージを直接送信できます (クライアントへのメッセージの直接送信を参照)。
2. 別の方法は、`ClientIdentifier` 命名規則を使用します。 すべての MQTT サブスクライバーに、それぞれの固有の `ClientIdentifier` をトピックとして使用して、サブスクリプションを作成させます。 *ClientIdentifier* にパブリッシュします。 トピック *ClientIdentifier* にサブスクライブしたクライアントに、パブリケーションが送信されます。 この手法を使用すれば、パブリケーションを特定の MQTT サブスクライバーに送信できます。

Windows Linux AIX MQ Telemetry のセキュリティー

遠隔測定装置をセキュリティー保護することが重要になる場合があります。 装置がポータブルであることと、綿密に管理できない場所で使用されることが予想されるからです。 VPN を使用して、MQTT 装置から遠隔測定 (MQXR) サービスへの接続をセキュリティー保護できます。 MQ Telemetry では、これとは別の 2 つのセキュリティー・メカニズム (TLS と JAAS) を使用できます。

TLS は主に、装置と遠隔測定チャネルの間の通信を暗号化するため、および正しいサーバーに装置が接続しようとしていることを認証するために使用されます (TLS を使用する遠隔測定チャネルの認証を参照)。 TLS を使用して、クライアント装置がサーバーに接続することを許可されているかどうかを検査することもできます (TLS を使用する MQTT クライアントの認証を参照)。

JAAS は主に、装置のユーザーがサーバー・アプリケーションを使用することを許可されているかどうかを検査するために使用されます (パスワードを使用する MQTT クライアントの認証を参照)。 JAAS を LDAP と一緒に使用することにより、シングル・サインオン・ディレクトリーを使用してパスワードを検査できます。

TLS と JAAS を併用することにより、2 因子認証を行えます。 TLS によって使用される暗号を FIPS 標準に適合する暗号に制限できます。

少なくとも何万というユーザーがいれば、個人のセキュリティー・プロファイルを用意することは必ずしも現実的とは限りません。 そしてまた、個別ユーザーが IBM MQ オブジェクトにアクセスするのをプロファイルを使用して許可するのも、必ずしも現実的ではありません。 代わりに、トピックへのパブリケーションとサブスクリプションを許可するためのクラス、およびクライアントにパブリケーションを送信するためのクラスに、ユーザーをグループ化します。

各遠隔測定チャネルを構成して、クライアントをクライアント共通ユーザー ID にマップします。 特定のチャネルで接続するすべてのクライアントに、共通ユーザー ID を使用します (MQTT クライアントの ID および許可を参照)。

ユーザーのグループを許可することが、各個人の認証を阻害するわけではありません。 各個別ユーザーをクライアントまたはサーバーでそれぞれのユーザー名とパスワードで認証した後、サーバーで共通ユーザー ID を使用して許可することができます。

Windows Linux AIX MQ Telemetry グローバリゼーション

MQTT v3 プロトコルでは、メッセージ・ペイロードはバイト配列としてエンコードされます。 一般に、テキストを扱うアプリケーションは、UTF-8 でメッセージ・ペイロードを作成します。 遠隔測定チャネルは、メッセージ・ペイロードを UTF-8 であると示しますが、コード・ページ変換はまったく行いません。 パブリケーション・トピックのストリングは UTF-8 でなければなりません。

アプリケーションは、英字データを正しいコード・ページに変換し、数値データを正しい数値エンコード方式に変換する必要があります。

MQTT Java クライアントには、便利なメソッド `MqttMessage.toString` があります。 このメソッドは、メッセージ・ペイロードを、ローカル・プラットフォームのデフォルト文字セット (一般に UTF-8) でエンコードされているものとして扱います。 これはペイロードを Java String に変換します。 Java には、ストリングをローカル・プラットフォームのデフォルト文字セットを使用してエンコードされたバイト配列に変換する String メソッド `getBytes` があります。 2 つの MQTT Java プログラムが、同じデフォルト文字セットを持つプラットフォーム間で、メッセージ・ペイロード内のテキストを UTF-8 で非常に簡単かつ効率的に交換できます。

いずれかのプラットフォームのデフォルト文字セットが UTF-8 でない場合、アプリケーションはメッセージの交換のためのきまりを確立しなければなりません。 例えば、パブリッシャーは、`getBytes("UTF8")` メソッドを使用して、ストリングから UTF-8 への変換を指定します。 メッセージのテキストを受け取るには、サブスクライバーはメッセージが UTF-8 文字セットでエンコードされているものと見なします。

遠隔測定 (MQXR) サービスは、MQTT クライアントのメッセージからのすべての着信パブリケーションのエンコード方式を、UTF-8 であると示します。 `MQMD.CodedCharSetId` を UTF-8 に、`RFH2.CodedCharSetId` を `MQCCSI_INHERIT` に設定します。 135 ページの『MQ Telemetry とキュー・マネージャーの統合』を参照。 パブリケーションのフォーマットが `MQFMT_NONE` に設定されるので、チャネルによる、つまり MQGET による変換は行えません。

Windows Linux AIX MQ Telemetry のパフォーマンスと拡張容易性

多数のクライアントを管理するときや MQ Telemetry の拡張容易性を高めるときには、以下の要因を考慮に入れてください。

キャパシティー・プランニング

MQ Telemetry のパフォーマンス・レポートについては、MQ Performance documents を参照してください。

接続

接続に関係するコストには、以下の要素が含まれます。

- プロセッサー使用率とプロセッサー時間の点から見た接続そのもののセットアップ・コスト。
- ネットワーク・コスト。
- 開いたままの接続を使用しないときの使用メモリー。

クライアントが接続されたままのときは、さらに負荷が発生します。 接続が開いたままの場合、TCP/IP フローと MQTT メッセージではネットワークを使用して、まだ接続されているかどうかが確認されます。 加えて、開いたままのクライアント接続ごとに、サーバーでメモリーが使用されます。

メッセージを毎分複数回の頻度で送信する場合は、新たに接続を開始するときのコストを回避するために、接続を開いたままにしておきます。 メッセージを 10 分から 15 分ごとに 1 回未満の頻度で送信する場合は、接続を開いたままにしておくときのコストを回避するために、切断することを検討してください。 TLS

接続は、セットアップにより多くの費用がかかるため、使用されない状態で長時間開いたままにしておいてもかまいません。

また、クライアントの能力も考慮に入れてください。 クライアントにストア・アンド・フォワード機能がある場合は、メッセージをバッチにまとめて、バッチの送信と送信の間は接続を切断することができます。ただし、クライアントが切断されると、クライアントはサーバーからのメッセージを受信できなくなります。 したがって、判断にはアプリケーションの目的が関係します。

システムのクライアントが 1 つで、それが多数のメッセージを送信する場合は (例えばファイル転送)、メッセージごとにサーバーの応答を待つことはしないでください。 代わりに、すべてのメッセージを送信し、最後にそれらがすべて受信されたことを検査します。 あるいは、サービス品質 (QoS) を使用します。

QoS をメッセージによって変更することができます。QoS 0 を使用する重要でないメッセージと、QoS 2 を使用する重要なメッセージを送達することができます。 メッセージのスループットは、QoS が 2 の場合よりも、QoS が 0 の 2 倍程度になることがあります。

命名規則

多数のクライアントに対応するアプリケーションを設計する場合は、実効的な命名規則を実施してください。 各クライアントを正しい `ClientIdentifier` にマップするために、`ClientIdentifier` を意味のある名前にしてください。 適切な命名規則により、管理者が実行中のクライアントを判別しやすくなります。 命名規則は、管理者が IBM MQ エクスプローラーで多数のクライアントのリストをフィルタリングする際に役立ち、また、問題の判別にも役立ちます (クライアント ID を参照)。

スループット

トピック名の長さは、ネットワーク上を流れるバイト数に影響します。 パブリッシュまたはサブスクライブ時に、メッセージに含まれるバイト数が重要になることがあります。 したがって、トピック名の文字数を制限してください。 MQTT クライアントがトピックをサブスクライブすると、IBM MQ はそれに次の形式の名前を付けます。

```
ClientIdentifier: TopicName
```

MQTT クライアントのサブスクリプションをすべて表示するには、IBM MQ MQSC **DISPLAY** コマンドを次のように使用します。

```
DISPLAY SUB(' ClientID1:*')
```

IBM MQ クライアントが使用するための MQTT の定義リソース

MQTT クライアントは、IBM MQ (リモート・キュー・マネージャー) に接続します。 IBM MQ アプリケーションが MQTT クライアントにメッセージを送信するには、2 つの基本的な方法があります。デフォルト伝送キューを `SYSTEM.MQTT.TRANSMIT.QUEUE` に設定する方法と、キュー・マネージャー別名を使用する方法です。 MQTT クライアントが多数ある場合は、キュー・マネージャーのデフォルト伝送キューを定義します。 伝送キューのデフォルト設定を使用すると、管理作業が簡単になります (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を参照)。

サブスクリプションを回避することによる拡張容易性の向上

MQTT V3 クライアントがトピックにサブスクライブすると、遠隔測定 (MQXR) サービスによってサブスクリプションが IBM MQ で作成されます。 サブスクリプションにより、クライアントのパブリケーションが `SYSTEM.MQTT.TRANSMIT.QUEUE` に送信されます。 各パブリケーションの伝送ヘッダーにあるリモート・キュー・マネージャー名は、サブスクリプションを作成した MQTT クライアントの `ClientIdentifier` に設定されます。 多数のクライアントがそれぞれ独自のサブスクリプションを行う場合は、IBM MQ パブリッシュ/サブスクライブのクラスターまたは階層の全体にわたって、多数のプロキシー・サブスクリプションが維持されることになります。 パブリッシュ/サブスクライブの代わりに、Point-

to-Point ベースのソリューションを使用する方法については、クライアントへのメッセージの直接送信を参照してください。

多数のクライアントの管理

同時に接続される多数のクライアントをサポートするには、JVM パラメーターの **-Xms** と **-Xmx** を設定して、遠隔測定 (MQXR) サービスで使用可能なメモリーを増やします。 以下のステップに従ってください: 。

1. テレメトリー・サービス構成ディレクトリー内の java.properties ファイルを見つけます。 Windows 上のテレメトリー (MQXR) サービス構成ディレクトリー または Linux 上のテレメトリー・サービス構成ディレクトリー を参照してください。
2. ファイルの指示に従います。同時に接続されるクライアントの数が 50,000 の場合は、1 GB のヒープで十分です。

```
# Heap sizing options - uncomment the following lines to set the heap to 1G
#-Xmx1024m
#-Xms1024m
```

3. java.properties ファイル内でテレメトリー (MQXR) サービスを実行している JVM に渡すために、他のコマンド行引数を追加します。テレメトリー (MQXR) サービスへの JVM パラメーターの引き渡し を参照してください。

Linux 上のオープン・ファイル記述子の数を増やすには、以下の行を /etc/security/limits.conf/に追加し、再度ログインしてください。

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

ソケットごとに 1 つのファイル記述子を必要とします。 遠隔測定サービスでさらにファイル記述子がいくつか必要になるため、この数は、必要なオープン・ソケットの数よりも多くする必要があります。

キュー・マネージャーは、非永続サブスクリプションごとに 1 つのオブジェクト処理を使用します。 多数のアクティブな非永続サブスクリプションをサポートするには、キュー・マネージャーで使用できるアクティブ・ハンドルの最大数を増やします。例えば、以下のようにします。

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

図 *48. Windows* でのハンドルの最大数の変更

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

図 *49. Linux* でのハンドルの最大数の変更

その他の考慮事項

システム要件を計画する際は、システムの再始動に要する時間の長さを考慮に入れてください。 計画したダウン時間が、処理待ちのキューに入れられるメッセージの数に影響する可能性があります。 許容時間内にメッセージを正常に処理できるように、システムを構成してください。 ディスク・ストレージ、メモリー、および処理能力を確認してください。 一部のクライアント・アプリケーションでは、クライアントの再接続時にメッセージを破棄できる場合があります。 メッセージを破棄するには、クライアント接続パラメーターで CleanSession を設定します (クリーン・セッションを参照)。 あるいは、 MQTT クライアントでベスト・エフォートのサービス品質 0 を使用してパブリッシュおよびサブスクライブを行います。 サービス品質を参照してください。 IBM MQ からメッセージを送信する場合は、非持続 メッセージを使用します。 これらのサービス品質を使用したメッセージは、システムまたは接続の再始動時には回復されません。

Windows Linux AIX MQ Telemetry によってサポートされる装置

MQTT クライアントは、センサーとアクチュエーターからハンドヘルド・デバイス、車両システムまで、さまざまな装置の上で実行できます。

MQTT クライアントは小さいので、小容量メモリーや低処理能力に制約される装置上で動作します。 MQTT protocol は信頼性が高くヘッダーが小さいので、低帯域幅、高コスト、および使用可能状態が断続的であることに制約されるネットワークに適しています。

MQ Telemetry は、MQTT クライアント・アプリケーションを介して遠隔測定装置と通信します。 このようなアプリケーションでは以下のリソースを使用します。どのリソースも MQTT v3 プロトコルを実装しています。

- 以下のクライアント・ライブラリー
  - *MQTT client for Java*。例えば、Android、OS X、Linux、Windows デバイスなどのネイティブ・アプリケーションの作成に使用されます。 このクライアント・ライブラリーを使用するアプリケーションは、最小の CLDC (Connected Limited Device Configuration)/MIDP (Mobile Information Device Profile) から、CDC (Connected Device Configuration)/Foundation、J2SE (Java Platform Standard Edition)、J2EE (Java Platform Enterprise Edition) に至るまで、あらゆるバリエーションの Java 上で実行可能です。 IBM jclRM カスタマイズ・クラス・ライブラリーもサポートされています。 Java ME プラットフォームは一般に、アクチュエーター、センサー、携帯電話などの組み込みデバイスのような小型装置上で使用されます。 Java SE プラットフォームは一般に、デスクトップ・コンピューターやサーバーなどのより高性能な組み込み装置にインストールされます。
  - *MQTT client for C*。これは、ネイティブ・アプリケーション (例えば、iOS、OS X、Linux、または Windows 装置) の作成に使用されます。 このクライアント・ライブラリーは、Windows および Linux システム用の事前ビルドされたネイティブ・クライアントと一緒に C 参照インプリメンテーションを提供します。 この C リファレンス実装に基づいて、広範囲の装置やプラットフォームに MQTT を移植できます。 WindowsIntel 7、RedHat、Ubuntu、および ARM プラットフォーム上のいくつかの Windows システム (Eurotech Viper など) の一部の Linux システムは、C クライアントを実行する Linux のバージョンを実装していますが、IBM ではプラットフォームに対するサービス・サポートは提供されません。 IBM サポート・センターに連絡を取る場合は、サポートされているプラットフォーム上でクライアントの問題を再現しておく必要があります。
  - *MQTT client for Java*。 ブラウザー・ベースの Web アプリケーションの作成に使用します。

MQTT クライアント・ライブラリーを Eclipse Paho および MQTT.org から無料で入手することができます。 IBM MQ Telemetry Transport サンプル・プログラムを参照してください。

IBM MQ のセキュリティー

IBM MQ では、セキュリティーを提供するいくつかの方法があります。許可サービス・インターフェース、ユーザー作成またはサード・パーティーのチャネル出口、Transport Layer Security (TLS) を使用したチャネル・セキュリティー、チャネル認証レコード、およびメッセージ・セキュリティーです。

許可サービス・インターフェース

MQI 呼び出しの使用、コマンドの発行、およびオブジェクトへのアクセスは、**オブジェクト権限マネージャー** (OAM) によって提供されます。OAM はデフォルトでは使用可能です。 IBM MQ エンティティーへのアクセスは、IBM MQ ユーザー・グループおよび OAM を通して制御されます。 管理者はコマンド行インターフェースを使用して、必要に応じて許可を与えたり取り消したりすることができます。

許可サービス・コンポーネントの作成の詳細については、AIX, Linux, and Windows システムでのセキュリティーのセットアップを参照してください。

ユーザー作成またはサード・パーティーのチャネル出口

チャネルでは、ユーザー作成出口またはサード・パーティー出口を使用できます。 詳細については、メッセージング・チャネルのためのチャネル出口プログラムを参照してください。

TLS を使用したチャネル・セキュリティー

Transport Layer Security (TLS) プロトコルは、業界標準のチャネル・セキュリティーを提供し、盗聴、改ざん、偽名の使用に対して保護します。

TLS は、公開鍵とシンメトリック手法を使用して、メッセージの機密性と保全性、および相互認証を提供します。

TLS の詳細情報を含む IBM MQ のセキュリティーに関する総合的なレビューについては、セキュリティーを参照してください。 このセクションで説明したコマンドのポインターを含めた TLS の概要については、暗号セキュリティー・プロトコル: TLS を参照してください。

チャネル認証レコード

チャネル認証レコードを使用して、チャネル・レベルで接続システムに許可されているアクセスに対して正確な制御を行います。 詳しくは、チャネル認証レコードを参照してください。

メッセージ・セキュリティー

Advanced Message Security (別個にインストールおよびライセンス交付される IBM MQ のコンポーネント) を使用し、IBM MQ を使用して送受信されるメッセージに対する暗号保護を提供します。 Advanced Message Security を参照してください。

関連タスク

セキュリティー

セキュリティー要件の計画

IBM MQ.NET 管理対象クライアントの TLS サポート

IBM MQ.NET の完全管理クライアントは、Microsoft.NET SSLStreams キットに基づく Transport Layer Security (TLS) サポートを提供します。 これは、IBM Global Security Kit (GSKit)に基づく他の IBM MQ クライアントとは異なります。

IBM MQ.NET アプリケーションを開発して、管理モードまたは非管理モードで実行できます。

- 管理モードでは、.NET アプリケーションは .NET CLR (共通言語ランタイム) 内で、C MQI 呼び出しなどのクロス・プラットフォーム呼び出しなしで動作します。
- 非管理モードでは、基礎となる MQI 操作のために C MQI が呼び出されます。 基本的には、非管理モード・インターフェースは、C MQI の上にある .NET ラッパー・クラスで構成されます。

管理 IBM MQ.NET クライアントは、Microsoft.NET Framework ライブラリーを使用して、TLS セキュア・ソケット・プロトコルを実装します。 Microsoft の System.NET.Security.SSLStream クラスは、 IBM MQ.NET でセキュリティー (TLS) を実装するために使用されます。

非管理対象 IBM MQ.NET クライアント・モードは、C MQI (および GSKit) に基づく TLS 機能を既にサポートしています。 つまり、TLS 操作は C MQI によって処理されます。 この場合、 GSKit は TLS セキュア・ソケット・プロトコルを実装します。

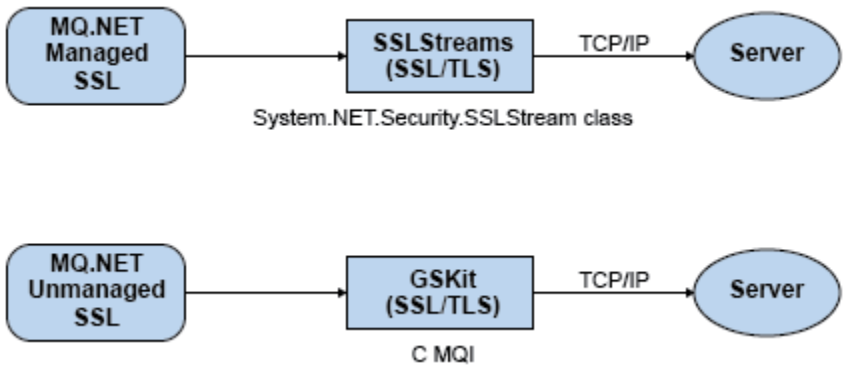


図 *50. IBM MQ.NET* 管理 *TLS* と非管理 *TLS* の比較

次の表に、管理実装と非管理実装の違いについてまとめます。

表 *14.* 管理実装と非管理実装の違い			
モード (Mode)	**プロトコル**	**実装**	**コメント**
IBM MQ.NET 管理 SSL	TLS	System.NET.Security.SSLStream クラス SSLStream クラスは、接続済み TCP ソケット上のストリームとして作動します	TLS 1.0 TLS 1.2 (Microsoft.NET Framework v4.5 でのみ)
IBM MQ.NET 非管理 SSL	TLS	GSKit および C-MQI	TLS セキュア・ソケット・プロトコル

関連概念

.NET のための Secure Sockets Layer (SSL) および Transport Layer Security (TLS) のサポート

IBM MQ MQI clients

IBM MQ MQI client は、キュー・マネージャーが実行されていないシステム上にインストールできる、IBM MQ 製品のコンポーネントです。

IBM MQ MQI クライアント とは、あるシステム上で稼働しているアプリケーションが、別のシステム上で稼働しているキュー・マネージャーに MQI 呼び出しを発行できるようにするコンポーネントのことです。 呼び出しからの出力は クライアントに返送され、さらにクライアントからアプリケーション に戻されます。

IBM MQ MQI client を使用すると、クライアントと同じシステム上で実行されているアプリケーションが、別のシステム上で実行されているキュー・マネージャーに接続することができます。 アプリケーションは、そのキュー・マネージャーに対して MQI 呼び出しを発行できます。 このようなアプリケーションは、IBM MQ MQI client・アプリケーションと呼ばれ、キュー・マネージャーはサーバー・キュー・マネージャーと呼ばれます。

IBM MQ サーバー とは、キューイング・サービスを 1 つ以上のクライアントに提供するキュー・マネージャーのことです。 キューなどのすべての IBM MQ オブジェクトは、キュー・マネージャーのマシン上 (IBM MQ サーバー・マシン) にのみ存在し、クライアント上には存在 しません。 IBM MQ サーバーは、ローカルの IBM MQ アプリケーションもサポートすることができます。

IBM MQ サーバーと通常のキュー・マネージャーとの相違点は、サーバーには、各クライアントとの専用通信リンクが備わっているという点です。 クライアントとサーバーにチャネルを作成する方法の詳細については、分散キューイングの構成を参照してください。

IBM MQ MQI client・アプリケーションとサーバー・キュー・マネージャーは、*MQI* チャネルを使用して相互に通信します。 MQI チャネルは、クライアント・アプリケーションが、キュー・マネージャーに接続するための **MQCONN** または **MQCONNX** 呼び出しを発行した時に開始されます。また、MQI チャネルは、クライアント・アプリケーションが、キュー・マネージャーとの接続を解除する **MQDISC** 呼び出しを発行した時に終了されます。 MQI 呼び出しの入力パラメーターは、MQI チャネル上で 1 つの方向に流れ、出力パラメーターは、それと反対の方向に流れます。

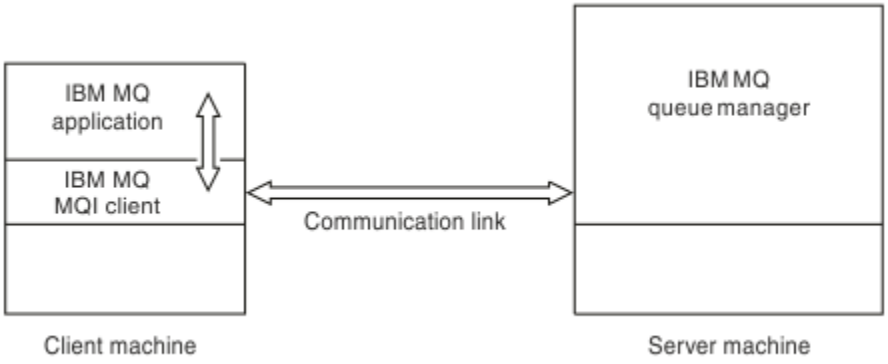


図 *51.* クライアントとサーバー間のリンク

次のプラットフォームを使用できます。 組み合わせは、使用する IBM MQ 製品によって異なります。 この点については、148 ページの『IBM MQ クライアントのプラットフォームのサポート』 で説明されています。

IBM MQ MQI client	**IBM MQ サーバー**
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

MQI は、クライアント・プラットフォーム上で稼働するアプリケーションで使用することができます。 キューおよびその他の IBM MQ オブジェクトは、サーバー上にインストールされたキュー・マネージャー上に保存されます。

IBM MQ MQI client 環境で実行するアプリケーションを、最初に、関連したクライアント・ライブラリーにリンクする必要があります。 アプリケーションが MQI 呼び出しを発行すると、IBM MQ MQI client は要求をキュー・マネージャーに送ります。キュー・マネージャーでは要求が処理され、応答が IBM MQ MQI client に返されます。

アプリケーションと IBM MQ MQI client の間のリンクは、実行時に動的に確立されます。

IBM MQ classes for .NET、IBM MQ classes for Java または IBM MQ classes for Java Message Service (JMS) を使って顧客のアプリケーションを開発することもできます。以下のプラットフォームで Java および JMS 顧客を使用することができます。

- IBM i
- AIX
- Linux
- Windows

Java および JMS の使用についてはここでは説明しません。 IBM MQ classes for Java および IBM MQ classes for JMS のインストール、構成および使用の方法に関する詳細については、IBM MQ classes for Java の使用および IBM MQ classes for JMS の使用をご参照ください。

クライアント/サーバー環境での IBM MQ アプリケーション

サーバーにリンクされていれば、クライアント IBM MQ アプリケーションは、ローカル・アプリケーションと同じやり方でたいていの MQI 呼び出しを発行することができます。 クライアント・アプリケーションは、MQCONN 呼び出しを発行して、指定のキュー・マネージャーに接続します。 接続要求から戻された接続ハンドルを指定した追加の MQI 呼び出しは、このキュー・マネージャーによって処理されます。

ユーザーのアプリケーションを、該当するクライアント・ライブラリーにリンクする必要があります。 IBM MQ MQI clients 用のアプリケーションの作成を参照してください。

関連概念

148 ページの『IBM MQ クライアントを使用する理由』
IBM MQ クライアントを使用することは、IBM MQ メッセージングおよびキューイングを効率的に実装する方法です。

150 ページの『拡張トランザクション・クライアントの概要』
IBM MQ 拡張トランザクション・クライアントは、外部トランザクション・マネージャーの制御の元で、別のリソース・マネージャーが管理するリソースを更新できます。

151 ページの『クライアントとサーバーの接続方法』
クライアントは MQCONN または MQCONNX を使ってサーバーに接続し、チャネルを通して通信します。

152 ページの『トランザクションの管理とサポート』
トランザクション管理、および IBM MQ がトランザクションをサポートをする方法に関する概要です。

154 ページの『キュー・マネージャーの機能の拡張』
ユーザー出口、API 出口、またはインストール可能サービスを使用して、キュー・マネージャーの機能を拡張できます。

関連情報

IBM MQ MQI client のセットアップ方法

IBM MQ クライアントを使用する理由

IBM MQ クライアントを使用することは、IBM MQ メッセージングおよびキューイングを効率的に実装する方法です。

MQI とキュー・マネージャーを別々のマシン上 (物理マシン、仮想マシンのどちらでも) で実行するようにアプリケーションを作成することもできます。 MQI とキュー・マネージャーを別のマシンで実行すると、次のような利点があります。

- クライアント・マシン上に IBM MQ を完全に実装する必要がなくなります。
- クライアント・システム側のハードウェア要件が減少します。
- システム管理要件が減少します。
- クライアントで実行する IBM MQ アプリケーションをさまざまなシステム上の複数のキュー・マネージャーに接続できます。
- 異なる伝送プロトコルを使用する代替チャネルが使用できます。

IBM MQ クライアントのプラットフォームのサポート

サポートされるすべてのサーバー・プラットフォーム上の IBM MQ は、複数のプラットフォーム上の IBM MQ MQI clients からのクライアント接続を受け入れます。

サポートされるすべてのサーバー・プラットフォーム上に *Base* 製品およびサーバー としてインストールされた IBM MQ は、以下のプラットフォーム上の IBM MQ MQI clients からの接続を受け入れることができます。

- IBM i
- AIX
- Linux
- Windows

クライアント接続によって、コード化文字セット ID (CCSID) および通信プロトコルが異なる場合があります。

IBM MQ MQI client 上で実行するアプリケーション

クライアント環境では MQI が完全にサポートされています。 これにより、IBM MQ MQI client 上のアプリケーションを MQI ライブラリーではなく MQIC ライブラリーにリンクすることで、ほとんどすべての IBM MQ アプリケーションを IBM MQ MQI client システム上で実行するように構成できます。 ただし、次の例外があります。

- 信号付きの MQGET
- 他のリソース管理プログラムとの同期点の調整を必要とするアプリケーションでは、拡張トランザクション・クライアントを使用する必要があります。

非持続メッセージングのパフォーマンスを良くするために先読みを使用可能にした場合、すべての MQGET オプションが使用可能な訳ではありません。 次の表は、使用できるオプションとそれらを MQGET 呼び出しの間に変更できるかどうかを示します。

表 *15.* 先読みが有効になっている場合に使用できる *MQGET* オプション

値	先読みが有効になっている場合に使用でき、**MQGET** 呼び出し間で変更できる	先読みが有効になっている場合に使用でき、**MQGET** 呼び出し間で変更できない [1]	先読みが有効になっている場合に使用できない **MQGET** オプション [2]
MQGET MD 値	MsgId [3] CorrelId [3]	Encoding CodedCharSetId	
MQGET MQGMO オプション	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF_QUIESCING MQGMO_BROWSE_FIRST [4] MQGMO_BROWSE_NEXT [4] MQGMO_BROWSE_MESSAGE_UNDER_CURSOR [4]	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP_BACKOUT MQGMO_MSG_UNDER_CURSOR [4] MQGMO_LOCK MQGMO_UNLOCK
MQGMO 値		MsgHandle	

1. これらのオプションが MQGET 呼び出し間で変更された場合、MQRC_OPTIONS_CHANGED 理由コードが戻されます。
2. これらのオプションが最初の MQGET 呼び出しで指定されると、先読みは使用不可になります。 これらのオプションを後続の MQGET 呼び出しで指定すると、理由コード MQRC_OPTIONS_ERROR が戻されます。
3. クライアント・アプリケーション側で以下の点に留意する必要があります。すなわち、MsgId および CorrelId の値が MQGET 呼び出し間で変更された場合、変更前の値によるメッセージがクライアントに送信済みの可能性があり、コンシューム (または自動的にパージ) されるまでクライアントの先読みバッファー内に残るということです。
4. 最初の MQGET 呼び出しは、先読みが有効である場合にメッセージをキューからブラウズするか取得するかを決定します。 アプリケーションがブラウズと取得の組み合わせを使用しようとすると、MQRC_OPTIONS_CHANGED 理由コードが戻されます。
5. MQGMO_MSG_UNDER_CURSOR は先読みでは使用できません。 先読みが有効な場合、メッセージのブラウズまたは取得が可能ですが、ブラウズと取得の組み合わせは指定できません。

IBM MQ MQI client 上で動作するアプリケーションは、同時に複数のキュー・マネージャーに接続できます。また、MQCONN や MQCONNX を呼び出すときに、キュー・マネージャー名にアスタリスク (*) を付けて使用することができます (キュー・マネージャーへの IBM MQ MQI client・アプリケーションの接続の例を参照)。

拡張トランザクション・クライアントの概要

IBM MQ 拡張トランザクション・クライアントは、外部トランザクション・マネージャーの制御の元で、別のリソース・マネージャーが管理するリソースを更新できます。

トランザクション管理の概念についてよくご存じでない場合は、152 ページの『トランザクションの管理とサポート』を参照してください。

XA トランザクション・クライアントは、IBM MQ の一部として提供されるようになったことに注意してください。

クライアント・アプリケーションは、接続先のキュー・マネージャーによって管理される作業単位に加わることができます。 その作業単位内で、クライアント・アプリケーションは、キュー・マネージャーが所有するキューにメッセージを書き込んだり、キューからメッセージを取得したりすることができます。 次に、クライアント・アプリケーションは、**MQCMIT** 呼び出しを使用して作業単位をコミットするか、**MQBACK** 呼び出しを使用して作業単位をバックアウトすることができます。 しかし、クライアント・アプリケーションは同じ作業単位内で、別のリソース・マネージャーのリソース、例えば Db2® データベースの表を更新することはできません。 IBM MQ 拡張トランザクション・クライアントを使用すると、この制限がなくなります。

IBM MQ 拡張トランザクション・クライアントは、いくつかの追加機能を持つ IBM MQ MQI client です。この機能を使用して、クライアント・アプリケーションは、同一の作業単位内で次のタスクを実行できます。

- 接続先のキュー・マネージャーが所有するキューにメッセージを書き込み、そのキューからメッセージを取得する
- IBM MQ キュー・マネージャー以外のリソース・マネージャーのリソースを更新する

この作業単位は、クライアント・アプリケーションと同じシステム上で実行されている外部トランザクション・マネージャーによって管理されなければなりません。 クライアント・アプリケーションの接続先のキュー・マネージャーが作業単位を管理することはできません。 つまり、キュー・マネージャーは、リソース・マネージャーの役目だけをすることができ、トランザクション・マネージャーの役目をすることはできません。 また、クライアント・アプリケーションは、作業単位のコミットまたはバックアウトを行うのに、外部トランザクション・マネージャーが提供するアプリケーション・プログラミング・インターフェース (API) しか使用できません。 したがって、クライアント・アプリケーションは、MQI 呼び出し **MQBEGIN**、**MQCMIT**、 および **MQBACK** を使用できません。

外部トランザクション・マネージャーは、キュー・マネージャーに接続されるクライアント・アプリケーションによって使用されるのと同じ MQI チャネルを使用して、リソース・マネージャーとしてのキュー・マネージャーと情報を交換できます。 しかし、障害後のリカバリー状態では、アプリケーションが実行していないので、トランザクション・マネージャーは、専用の MQI チャネルを使用して、障害の時点でキュー・マネージャーが参加していた未完了の作業単位をリカバリーすることができます。

このセクションでは、拡張トランザクション機能を持たない IBM MQ MQI client は、IBM MQ ベース・クライアントと呼ばれます。 したがって、IBM MQ 拡張トランザクション・クライアントは、拡張トランザクション機能が付加された IBM MQ ベース・クライアントであると見なすことができます。

注 : IBM i IBM i 上の IBM MQ MQI client は、IBM MQ 拡張トランザクション機能をサポートしません。

拡張トランザクション・クライアントのプラットフォームのサポート

Multi

拡張トランザクション・クライアントは、ベース・クライアントをサポートするすべての Multiplatforms で利用可能です。 クライアントは z/OS では利用できません。

拡張トランザクション・クライアントを使用するクライアント・アプリケーションは、以下の IBM MQ 9.0 以降の製品のキュー・マネージャーにのみ接続できます。

- 

IBM MQ for AIX
- 

IBM MQ for IBM i
- 

IBM MQ の Linux
- 

IBM MQ for Windows

z/OS で稼働する拡張トランザクション・クライアントはありませんが、拡張トランザクション・クライアント を使用しているクライアント・アプリケーションが z/OS で稼働するキュー・マネージャーに接続することはできます。

各プラットフォームでは、拡張トランザクション・クライアントのハードウェアおよびソフトウェアの要件は、IBM MQ ベース・クライアントの要件と同じです。 プログラム言語は、IBM MQ ベース・クライアントおよびご使用のトランザクション・マネージャーによってサポートされている場合は、拡張トランザクション・クライアントによってもサポートされます。

すべてのプラットフォームの外部トランザクション・マネージャーについては、 IBM MQ 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接続方法

クライアントは MQCONN または MQCONNX を使ってサーバーに接続し、チャネルを通して通信します。

IBM MQ クライアント環境で実行されるアプリケーションは、クライアント・マシンとサーバー・マシン間の接続をアクティブに保つ必要があります。

接続は、アプリケーションが MQCONN 呼び出しまたは MQCONNX 呼び出しを発行することによって、確立されます。 クライアントとサーバーは、*MQI* チャネル を介して通信します。ただし、共用会話の使用時には、それぞれの会話どうしが MQI チャネル・インスタンスを共有します。 その呼び出しが成功すると、アプリケーションが MQDISC 呼び出しを発行するまで、MQI チャネル・インスタンスまたは会話は接続されたままになります。 このことは、アプリケーションが接続する必要のあるどのキュー・マネージャーにも該当します。

同一マシン上のクライアントとキュー・マネージャー

また、ご使用のマシンにキュー・マネージャーがインストールされている場合は、 IBM MQ MQI client 環境でアプリケーションを実行することもできます。

この場合、キュー・マネージャーのライブラリー、クライアントのライブラリーのどちらにでも、リンクすることができます。ただし、クライアントのライブラリーにリンクする場合でも、チャネル接続を定義する必要があることに注意してください。 キュー・マネージャーがインストールされているマシンで、クライアント環境を実現できると、アプリケーションの開発段階で役に立ちます。 他のマシンに依存することなく、開発者自身のマシン上でプログラムをテストすることができ、独立した IBM MQ MQI client 環境に移動したときに、これまでどおり順調に稼働させることができます。

異なるプラットフォーム上のクライアント

この例では、サーバー・マシンは異なるプラットフォーム上で 3 つの IBM MQ MQI clients と通信します。

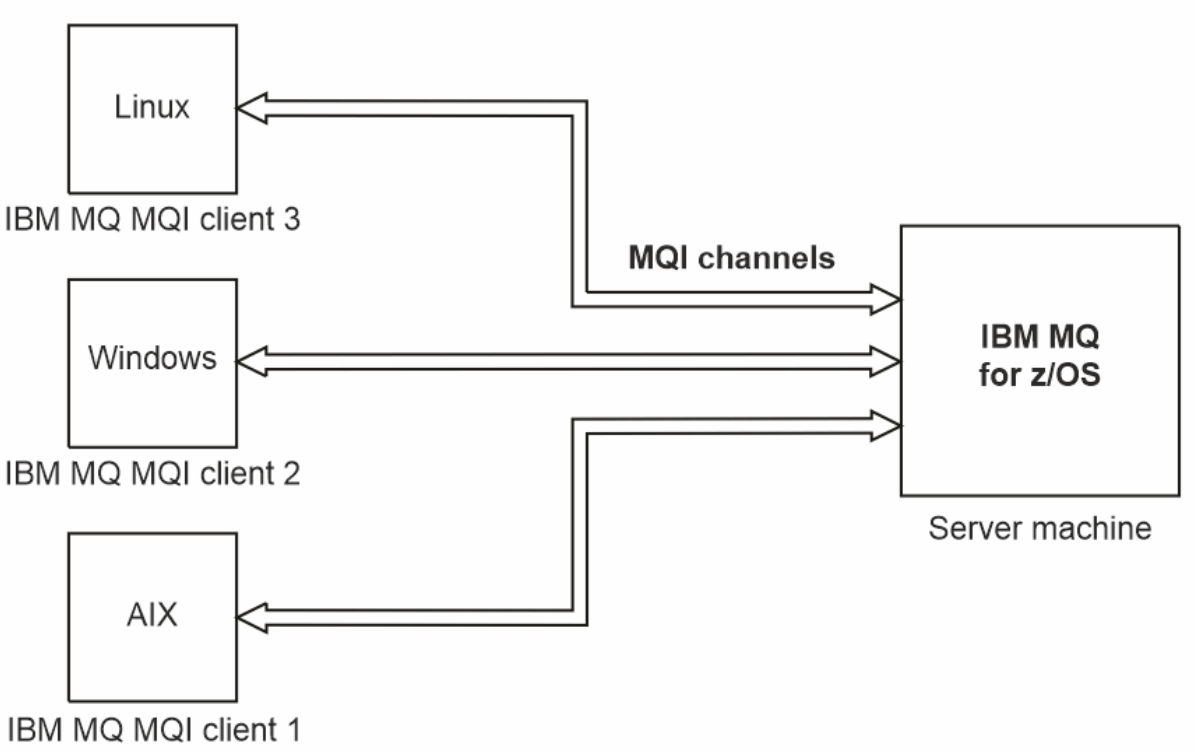


図 *52.* 異なるプラットフォーム上のクライアントに接続された *IBM MQ* サーバー

他のもっと複雑な環境も可能です。 例えば、1 つの IBM MQ クライアントが複数のキュー・マネージャーに接続する環境や、キュー共用グループの一部として接続されている任意の数のキュー・マネージャーに接続する環境も実現できます。

異なるバージョンのクライアントおよびサーバー・ソフトウェアの使用

旧バージョンの IBM MQ 製品を使用する場合は、クライアントの CCSID からのコード変換がサーバーでサポートされることを確認してください。

IBM MQ クライアントは、サポートされるすべてのバージョンのキュー・マネージャーに接続できます。 旧バージョンのキュー・マネージャーに接続する場合は、クライアント上の IBM MQ アプリケーションで、製品のより新しいバージョンの機能および構造体は使用できません。

IBM MQ キュー・マネージャーは、相互にサポートされている最も高いプロトコル・レベルまでネゴシエーションすることにより、異なるバージョンのクライアントとそれ自体と通信することができます。 これは、古いクライアントが新しいキュー・マネージャー・レベルで使用される可能性があることを意味します。 問題診断を容易にし、 IBM によるサポートを有効にするために、クライアントとサーバーの両方が、現在サポートされている IBM MQ のバージョンであることをお勧めします。

詳しくは、 アプリケーションの開発でサポートされているプログラミング言語を参照してください。

トランザクションの管理とサポート

トランザクション管理、および IBM MQ がトランザクションをサポートをする方法に関する概要です。

リソース・マネージャー は、アプリケーションによるアクセスと更新が可能なリソースを所有し管理する、コンピューター・サブシステムです。 以下にリソース・マネージャーの例を示します。

- IBM MQ キュー・マネージャー、キューがリソース
- Db2 データベース、テーブルがリソース

アプリケーションが 1 つ以上のリソース・マネージャーのリソースを更新する場合、所定の更新がすべて 1 つのグループとして 正常に必ず完了するようにする、またはすべて完了しないようにすることが、ビジ

ネス上必要になる場合があります。 こうした必要が生じる理由は、これらの更新の中で正常に完了しているものと、完了していないものがあると、ビジネス・データが不整合のままになるからです。

このように管理されるリソースの更新は、作業単位、またはトランザクション 内で行われます。 アプリケーション・プログラムは、更新のセットを作業単位にグループ化できます。

1 つの作業単位の中で、アプリケーションは、リソース・マネージャーのリソースを更新する要求をリソース・マネージャーに出します。 アプリケーションが、すべての更新をコミットする要求を出すと、この作業単位は終了します。 更新がコミットされるまで、それらの更新は、同じリソースにアクセスする他のアプリケーションから見えません。 または、アプリケーションは、なんらかの理由で作業単位を完了できないと判断した場合、その時点までに要求したすべての更新をバックアウトする要求を出すことができます。この場合、これらの更新はいずれも、他のアプリケーションから見えません。 作業単位としてグループ化される更新は相互に論理的に関連しているため、データ保全性を維持できるように正常に処理されなければなりません。 ある更新が正常に行われても、別の更新が失敗すれば、データ保全性は失われます。

作業単位が正常に完了したことを、コミット されたといいます。 コミットされると、作業単位内のすべての更新内容が永久的になり、これ以降は取り消し不可能になります。 しかし、作業単位が失敗した場合は、すべての更新がバックアウト されます。 このような、データの保全性を維持しながら作業単位をコミットまたはバックアウトするプロセスのことを 同期点調整 といいます。

作業単位内のすべての更新がコミットされるか、バックアウトされる時点は、同期点 と呼ばれます。 作業単位内の更新は、同期点制御内 で行われます。 アプリケーションが、同期点制御外 にある更新を要求する場合、リソース・マネージャーは、進行中の作業単位があっても、その更新をただちにコミットします。この更新を後でバックアウトすることはできません。

作業単位を管理するコンピューター・サブシステムは、トランザクション・マネージャー、またはポイント・コーディネーター と呼ばれます。

ローカル 作業単位とは、IBM MQ キュー・マネージャーのリソースのみが更新対象のリソースとなる作業単位をいいます。 この場合、キュー・マネージャー自体が単一フェーズ・コミットによって同期点を調整します。

グローバル 作業単位では、XA 準拠データベースなどの他のリソース・マネージャーにより管理されているリソースも更新されます。 この場合、必ず 2 フェーズ・コミットが使用され、作業単位がキュー・マネージャー自体によって調整されるか、あるいは IBM TXSeries® や BEA Tuxedo などの XA 準拠トランザクション・マネージャーによって外部から調整されます。

トランザクション・マネージャーは、作業単位内のリソースの更新がすべて正常に完了したか、完了しなかったかを確認します。 アプリケーションが、作業単位のコミットまたはバックアウトを要求するのは、トランザクション・マネージャーに対してです。 例えば、トランザクション・マネージャーの例として CICS や WebSphere Application Server がありますが、どちらも他の機能も持っています。

一部のリソース・マネージャーは、独自のトランザクション管理機能を備えています。 例えば、IBM MQ キュー・マネージャーは、独自のリソースの更新および Db2 表の更新を伴う作業単位を管理できます。 キュー・マネージャーは、この機能を実行するのに別個のトランザクション・マネージャーを必要としません。ただし、ユーザーの要求であれば、トランザクション・マネージャーを使用することはできます。 別個のトランザクション・マネージャーを使用する場合、外部トランザクション・マネージャー と呼ばれます。

外部トランザクション・マネージャーが作業単位を管理するには、トランザクション・マネージャーと、作業単位に加わっているすべてのリソース・マネージャーとの間に、標準インターフェースが存在している必要があります。 このインターフェースにより、トランザクション・マネージャーとリソース・マネージャーは互いに情報を交換することが可能になります。 これらのインターフェースの 1 つが *XA* インターフェース です。XA インターフェースは、複数のトランザクション・マネージャーとリソース・マネージャーによってサポートされる標準インターフェースです。 XA インターフェースは、The Open Group によって *Distributed Transaction Processing: The XA Specification* の中で公開されています。

複数のリソース・マネージャーが 1 つの作業単位に加わっている場合、トランザクション・マネージャーは、システム障害の場合であっても、*2* フェーズ・コミット ・プロトコルを使用して、その作業単位内のすべての更新が正常に完了するか、完了しないかを確認する必要があります。 アプリケーションが、作業単位をコミットする要求をトランザクション・マネージャーに出す場合、トランザクション・マネージャーは次のことを行います。

フェーズ 1 (コミットの準備)
トランザクション・マネージャーは、作業単位に加わっている各リソース・マネージャーに対して、そのリソースの対象の更新についてのすべての情報が、リカバリー可能な状態であることを確認するように求めます。 リソース・マネージャーは通常これを確認するために、情報をログに書き込み、その情報がハード・ディスクに書き込まれるようにします。 トランザクション・マネージャーが、そのリソースの対象の更新についての情報が、リカバリー可能な状態であるという通知を各リソース・マネージャーから受け取ると、フェーズ 1 が完了します。

フェーズ 2 (コミットの実行)
フェーズ 1 が完了すると、トランザクション・マネージャーは、作業単位をコミットするという、取り消すことができない決定を行います。 トランザクション・マネージャーは、その作業単位に加わっている各リソース・マネージャーに対して、そのリソースの更新をコミットするように求めます。 リソース・マネージャーは、この要求を受け取ると、更新をコミットする必要があります。 この段階で更新をバックアウトすることはできません。 トランザクション・マネージャーが、そのリソースの更新をコミットしたという通知を各リソース・マネージャーから受け取ると、フェーズ 2 が完了します。

XA インターフェースは、2 フェーズ・コミット・プロトコルを使用します。

詳細については、トランザクション・サポートのシナリオを参照してください。

IBM MQ は、Microsoft Transaction Server (COM+)のサポートも提供します。Microsoft トランザクション・サーバーの使用 (COM+) は、 COM+ サポートの利点を利用するために IBM MQ をセットアップする方法に関する情報を提供します。

キュー・マネージャーの機能の拡張

ユーザー出口、API 出口、またはインストール可能サービスを使用して、キュー・マネージャーの機能を拡張できます。

ユーザー出口

ユーザー出口は、独自のコードをキュー・マネージャー機能に 挿入するメカニズムを提供します。 サポートされるユーザー出口は、次のとおりです。

チャネル出口
この出口では、チャネルの動作方法を変更できます。 チャネル出口については、 メッセージング・チャネルのためのチャネル出口プログラムで説明されています。

データ変換出口
この出口では、アプリケーション・プログラムに書き込んでデータの形式を変換するための部分ソース・コードを作成できます。 データ変換出口については、データ変換出口の作成で説明されています。

クラスター・ワークロード出口
この出口により実行できる機能は、出口のプロバイダーにより定義されます。 呼び出し定義の情報は、MQ_CLUSTER_WORKLOAD_EXIT - 呼び出しの説明にあります。

API 出口

API 出口を使用すると、MQPUT および MQGET などの IBM MQ API 呼び出しの動作を変更するコードを作成して、これらの呼び出しの直前または直後に、そのコードを挿入することができます。 挿入は自動的に行われます。キュー・マネージャーは登録されているポイントで出口コードを駆動します。 API 出口の詳細は、API 出口の使用/作成方法を参照してください。

インストール可能サービス

インストール可能サービスには、複数のエントリー・ポイントを持つ形式化されたインターフェース (API) があります。

インストール可能サービスは、サービス・コンポーネント と呼ばれるもので実現されます。 IBM MQ で提供されているコンポーネントを使用することも、必要な機能を実行するコンポーネントを独自に作成することもできます。

現在、次のようなインストール可能サービスが用意されています。

許可サービス
これにより、独自のセキュリティー機能を構築することができます。

このサービスを実装するデフォルト・サービス・コンポーネントは、オブジェクト権限マネージャー (OAM) です。 デフォルトでは、OAM はアクティブになっています。つまり、OAM を構成する ための作業は一切必要ありません。 許可サービス・インターフェースを 使用して別のコンポーネントを作成し、OAM を置き換 えたり OAM を補強したりできます。 OAM の詳細については、AIX, Linux, and Windows システムでのセキュリティーのセットアップを参照してください。

ネーム・サービス
これにより、アプリケーションはリモート・キューをローカル・キューであるかのように認識することができ、キューを共用できます。

所有するネーム・サービス・コンポーネントを書き込むことができます。 これは、例えば、IBM MQ でネーム・サービスを使用する予定の場合に行う必要が生じることがあります。 ネーム・サービスを使用するには、ユーザー作成によるコンポーネント、または別のソフトウェア・ベンダーによって提供されたコンポーネントのどちらかが必要です。 デフォルトでは、ネーム・サービスは使用できない状態になっています。

関連概念
ユーザー出口、API 出口、および IBM MQ インストール可能サービス

IBM MQ Java 言語インターフェース

IBM MQ には、 Java アプリケーションで使用するための 3 つのアプリケーション・プログラミング・インターフェース (API) (IBM MQ classes for Jakarta Messaging、 IBM MQ classes for JMS、および IBM MQ classes for Java) が用意されています。

IBM は、オープン・スタンダードをサポートしており、オープン・スタンダードに積極的に参加しています。

- IBM MQ 8.0 以降、この製品は JMS 2.0 標準を実装しています。これにより、新しい簡素化された API と、共有サブスクリプションなどの機能が導入されました。
- IBM MQ 9.3.0 以降、 Jakarta Messaging 3.0 もサポートされます。
- さらに、 WebSphere Liberty は IBM MQ で JMS 2.0 および Jakarta Messaging 3.0 をサポートします。

IBM MQ 内には、 Java アプリケーションで使用するための以下の 3 つの API があります。

JM 3.0 **IBM MQ classes for Jakarta Messaging**
IBM MQ classes for Jakarta Messaging は、 IBM MQ 用の Jakarta Messaging インターフェースをメッセージング・システムとして実装する Jakarta Messaging プロバイダーです。 Jakarta Connectors Architecture は、 Jakarta EE 環境で実行されるアプリケーションを IBM MQ や Db2 などのエンタープライズ情報システム (EIS) に接続する標準的な方法を提供します。

JMS 2.0 **IBM MQ classes for JMS**
IBM MQ classes for JMS は、 IBM MQ 用の JMS インターフェースをメッセージング・システムとして実装する JMS プロバイダーです。 Java Platform, Enterprise Edition Connector Architecture (JCA) は、Java EE 環境内で実行されているアプリケーションを、IBM MQ や Db2 などのエンタープライズ情報システム (EIS) に接続する標準的な方法を提供します。

IBM MQ classes for Java
IBM MQ classes for Java を使用すると、 Java 環境で IBM MQ を使用できます。 IBM MQ classes for Java では、Java アプリケーションは IBM MQ に IBM MQ クライアントとして接続するか、または IBM MQ キュー・マネージャーに直接接続することができます。

注 :

- JMS 2.0 は Jakarta Messaging に置き換えられました。 IBM MQ classes for JMS は引き続き JMS 2.0 標準をサポートしますが、 Java メッセージングに対する将来の機能拡張は Jakarta Messaging でのみ行われるため、 IBM MQ classes for Jakarta Messaging で行われます。 IBM MQ classes for JMS は、既存の JMS 2.0 アプリケーションを保守および拡張する場合にのみ推奨されます。 IBM MQ classes for Jakarta Messaging は、新しい開発に推奨されるテクノロジーでなければなりません。

- Stabilized IBM MQ classes for Java は、IBM MQ 8.0 で出荷されたレベルで機能的に固定化されています。 IBM MQ classes for Java を使用する既存のアプリケーションは引き続き完全にサポートされますが、この API は固定化されているため、新機能は追加されず、機能拡張の要求も拒否されます。 完全なサポートとは、欠陥が見つかった場合、IBM MQ システム要件の変更によって必要とされる変更と一緒に修正されることを意味します。

JM 3.0 IBM MQ 9.3 以降、 IBM MQ classes for Java、 IBM MQ classes for JMS、および IBM MQ classes for Jakarta Messaging は、 Java 8 を使用してビルドされています。 これらのインターフェースを使用してアプリケーションを実行するには、これらのレベル以上の Java ランタイム環境を使用する必要があります。

関連概念

Java から IBM MQ へのアクセス-API の選択

IBM MQ classes for Jakarta Messaging を使用する理由

IBM MQ classes for JMS を使用する理由

IBM MQ classes for Java を使用する理由

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、IBM MQ で提供されるメッセージング・プロバイダーです。 これらの各プロバイダーは、メッセージング API に対する 2 セットの拡張機能も提供します。 これらのメッセージング・プロバイダーは、 Java Platform, Standard Edition (Java SE) アプリケーションと Java Platform, Enterprise Edition (Java EE) アプリケーションの両方で使用できます。

JM 3.0 IBM MQ 9.3.0 では、 Jakarta Messaging 3.0 のサポートが導入されました。 JMS 2.0 は引き続き完全にサポートされます。

JMS および Jakarta Messaging 仕様は、アプリケーションがメッセージング操作を実行するために使用できるインターフェースのセットを定義します。 この製品は、JMS 2.0 バージョンの JMS 標準をサポートしています。 この実装はクラシック API のすべての機能を提供しますが、要求されるインターフェースが少なくなり、より簡単に使用できます。 詳しくは、 159 ページの『JMS および Jakarta Messaging モデル』 および Java.net の JMS 2.0 仕様を参照してください。 JM 3.0 IBM MQ 9.3.0 以降、 Jakarta Messaging もサポートされています。

jakarta.jms (Jakarta Messaging 3.0) または javax.jms (JMS 2.0) パッケージは、メッセージング・インターフェースの詳細を指定し、メッセージング・プロバイダーは特定のメッセージング製品用にこれらのインターフェースを実装します。 以下に例を示します。

- IBM MQ classes for JMS は、 IBM MQ 用の JMS インターフェースを実装する JMS プロバイダーであり、JMS API に対して以下の 2 セットの拡張機能も提供します。
  - IBM MQ JMS 拡張
  - IBM JMS 拡張
- javax.dims、 jakarta.jms、インターフェース、またはいずれかの JMS 拡張セットを使用して作成された接続ファクトリー、キュー、またはトピック・オブジェクトは、これらの API のいずれかを使用してアドレス指定できます。つまり、任意のインターフェースにキャストできます。 最高レベルでアプリケーションの移植性を維持するには、要件に適した最も汎用的な API を使用してください。

JMS と Jakarta Messaging は多くの共通点を共有しているため、このトピックにおける JMS への以降の参照は、両方の参照と見なすことができます。 必要に応じて、差異が強調表示されます。

IBM MQ JMS 拡張

IBM MQ classes for JMS は、JMS API に対する拡張機能も提供します。 IBM MQ classes for JMS には、MQConnectionFactory、MQQueue、および MQTopic の各オブジェクトで実装された拡張機能が含まれています。 これらのオブジェクトには IBM MQ に固有のプロパティーやメソッドがあります。 オブジェクトは管理対象オブジェクトにすることができます。あるいは、アプリケーションはオブジェクトを実行時に動的に作成することができます。 これらの拡張機能は、 IBM MQ JMS 拡張機能と呼ばれます。 本書では、

実行時にアプリケーションによって動的に作成されるオブジェクトは、管理対象オブジェクトとは見なされないことに注意してください。

IBM JMS 拡張

IBM MQ JMS 拡張機能に加えて、 IBM MQ classes for JMS には、使用されるプログラミング言語として、JMS API または Java に対する拡張機能のより汎用的なセットが用意されています。 これらの拡張機能は、IBM JMS 拡張機能と呼ばれ、以下のような幅広い目的があります。

- IBM JMS プロバイダー間でより高いレベルの整合性を提供するため。
- 2 つの IBM メッセージング・システム間のブリッジ・アプリケーションの作成を容易にする。
- ある IBM JMS プロバイダーから別のプロバイダーへのアプリケーションの移植を容易にするため。

それらの拡張機能は主に、接続ファクトリーおよび宛先を実行時に動的に作成および構成することに重点が置かれていますが、メッセージングと直接的には関係のない機能 (例えば問題判別のための機能) も提供します。

関連タスク

IBM MQ classes for JMS/Jakarta Messaging の使用

JMS および Jakarta Messaging リソースの構成

JM 3.0 IBM MQ classes for Jakarta Messaging: 概要

IBM MQ 9.3.0 では、 Jakarta Messaging のサポートが導入されました。 Jakarta Messaging 3.0 の場合、JMS 仕様の制御が Oracle から Java Community Process に移動しました。 ただし、 Oracle は、他の Java テクノロジーで使用される「javax」名の制御を保持します。 したがって、 Jakarta Messaging 3.0 は JMS 2.0 と機能的には同等ですが、命名にはいくつかの違いがあります。 バージョン 3.0 の公式名は、 Java Message Service ではなく Jakarta Messaging であり、パッケージ名と定数名には、 `javax` ではなく `jakarta` という接頭部が付きます。

背景

長年にわたり、 Java プラットフォームには Standard Edition と Enterprise Edition の 2 つの形式があります。

Java Platform, Standard Edition (省略形は Java SE) は、スタンドアロン・コンテキストで実行できるコア言語およびクラス・ライブラリーです。 Java SE のほとんどの Java パッケージには、「java.」で始まる名前が付いています。

Java Platform, Enterprise Edition (Java EE) はこれを拡張し、メッセージング、各種 Bean、トランザクションなどの機能を追加します。 これらのテクノロジーの一部は、 Java SE コンテキストで使用することもできます。 Java EE のほとんどの Java パッケージは、歴史的に「javax」で始まる名前を持っています。ただし、クロスオーバーがいくつかあるため、一部の Java SE パッケージには「javax」があります。 名前の接頭部として使用されます。

Java Message Service (JMS) は、Java Platform, Enterprise Edition の一部です。 Java EE 7 には JMS 2.0 が組み込まれています。

Java EE 7 までは、テクノロジーは Oracle の管理下にありました。

Java EE テクノロジーは、最近、 Oracle のスチュワードシップから、 Eclipse Foundation が監督するコミュニティー・プロセスに移行しました。

"javax" として 名前を新しいプロジェクトに移動できませんでした。新しい名前が採用されました。すべてのパッケージ名とプロパティー名に「jakJakarta」という接頭部が付きます。 将来、 Java Platform, Enterprise Edition は「Jakarta EE」と呼ばれるようになります。 バージョン 8 は概して無視できる暫定バージョンであり、 Jakarta EE 9 は "jakJakarta" が存在するポイントである。 接頭部が有効になります。

IBM MQ コンテキストに適用される主な Jakarta EE テクノロジーは、 Jakarta Messaging 3.0 - Java Message Service (JMS) 2.0 の後継です。 そのため、 Jakarta EE 9 には Jakarta Messaging 3.0 が組み込まれています。

IBM MQ は、 Jakarta EE 9 および Jakarta Messaging 3.0 のサポートを導入する一方で、 Java EE 7 および JMS 2.0 を引き続きサポートします。

提供内容: Java SE

Java Platform, Standard Edition の場合、 IBM MQ classes for JMS (IBM MQ での JMS 2.0 操作をサポートします) に加えて、 IBM MQ 9.3.0 以降のバージョンでは IBM MQ classes for Jakarta Messaging が提供されます。 これらのクラスは、IBM MQ と統合する Jakarta Messaging 3.0 プロバイダーを提供します。これにより、 IBM MQ キュー・マネージャーを使用して Jakarta Messaging 操作を容易にすることができます。

これらは、 IBM MQ インストール済み環境の `java/lib` サブディレクトリーに、標準 JAR ファイル `com.ibm.mq.jakarta.client.jar` として提供されています。

Apache Felix や Eclipse Equinox などの OSGi コンテナーで使用するために、 IBM MQ には OSGi バンドルのペアも用意されています。

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

ここで、*V.R.M.F* は、IBM MQ のバージョンを表します (例: 9.3.0.0)。 これらのバンドルは、IBM MQ インストール済み環境の `java/lib/OSGi` サブディレクトリーにあります。

提供内容: Jakarta EE 9

Jakarta EE 9 互換アプリケーション・サーバーで IBM MQ ベースのメッセージングをサポートするために、IBM MQ は Jakarta EE 9 互換リソース・アダプター `wmq.jakarta.jmsra.rar` を提供しています。 これは、 IBM MQ インストール済み環境の `java/lib/jca` サブディレクトリーにあります。

IBM MQ は引き続き、 IBM MQ インストール済み環境の `java/lib/jca` サブディレクトリーに Java EE 7 互換リソース・アダプター `wmq.jmsra.rar` を提供します。

これらの成果物の提出方法

リソース・アダプター用のこれらの JAR および RAR ファイルは、通常の IBM MQ インストール・メディア (「`.rpm`」ファイルなどのプラットフォーム固有のインストール・メディアと、自己解凍型再配布可能クライアント JAR ファイルなどの再配布可能メディアの両方) に既存の成果物とともにパッケージされています。

JMS 2.0 と Jakarta Messaging 3.0 の間の変更点

Jakarta EE 9 および Jakarta Messaging 3.0 では、新機能は導入されていません。 変更されるのは名前のみです。 例えば、 JMS 2.0 で「javax.jms.Connection」を使用する場合は、 Jakarta Messaging 3.0 で「jakarta.jms.Connection」を使用します。

Eclipse Foundation は Jakarta EE プラットフォームを採用するため、このファウンデーションに基づいて構築されます。この命名規則は、将来導入される新機能に使用されます。

IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging の間の変更点

要約

JMS 2.0 のサポートを提供する IBM MQ classes for JMS は引き続き使用可能であり、主に既存のアプリケーションを保守および拡張するために推奨されます。 これらは完全にサポートされています。

Jakarta Messaging 3.0 のサポートを提供する IBM MQ classes for Jakarta Messaging は、新規開発に推奨されます。

IBM MQ 9.3.0 では、これら 2 つのオファリングは機能的に同等でした。 命名のみが異なります。 ただし、 IBM MQ classes for Jakarta Messaging では、 IBM MQ classes for JMS よりも新しいメッセージング機能が出現する可能性が高くなります。

この 2 つのオファリングは相互運用可能です。 IBM MQ classes for JMS によって生成されたメッセージは、 IBM MQ classes for Jakarta Messaging によってコンシュームされる可能性があります。また、その逆も同様です。 ただし、2 つのオファリングが単一のアプリケーション内で共存することはできません。

名前の変更

表 *16.* パッケージ名の変更

IBM MQ classes for JMS パッケージ名	**IBM MQ classes for Jakarta Messaging パッケージ名**
com.ibm.mq.jms[.*]	com.ibm.mq.jakarta.jms[.*]
com.ibm.jms	com.ibm.jakarta.jms
com.ibm.msg.client.jms.*	com.ibm.msg.client.jakarta.jms.*
com.ibm.msg.client.wmq.*	com.ibm.msg.client.jakarta.wmq.*

共通サービス (トレース、ロギング、各国語サポートなど) および JMQI 実装 (ローカルおよびリモート) に関連するパッケージは、IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging の両方に共通しているため、これらの領域での変更は必要ありません。

プロパティー名も変更されていることに注意してください。 例えば、 IBM MQ classes for Jakarta Messaging で IBM MQ 拡張機能を有効にするプロパティーは、 **com.ibm.mq.jakarta.jms.SupportMQExtensions** です。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging に依存しないプロパティー名 (さまざまな **com.ibm.msg.client.commonservices.trace.*** プロパティーなど) は、両方のオファリングに等しく適用されます。

管理ユーティリティー

crtmqenv および **setmqenv** ユーティリティーは、クラスパスを IBM MQ classes for JMS (`-j 2.0`) 用に構成するか IBM MQ classes for Jakarta Messaging (`-j 3.0`) 用に構成するかを指定するオプションを受け入れるようになりました。また、 **runjms30** と呼ばれる **runjms** ユーティリティーの IBM MQ classes for Jakarta Messaging バリアントおよび類似の名前があります。

dspmqver ユーティリティーは、 Java コンポーネントについて報告するよう要求された場合、その出力に IBM MQ classes for Jakarta Messaging を組み込みます。

JNDI を介して取得されるように IBM MQ classes for Jakarta Messaging オブジェクトを構成する場合、新しい **JMS30Admin** ユーティリティーは IBM MQ classes for JMS 用の **JMSAdmin** ユーティリティーと同等です。

基礎となるオブジェクトは異なるパッケージからのものであることに注意してください。 **JMSAdmin** によって作成された JNDI 定義を IBM MQ classes for Jakarta Messaging で使用することはできません。また、 **JMS30Admin** によって作成された JNDI 定義を IBM MQ classes for JMS で使用することもできません。

注 : IBM MQ Explorer によって提供される IBM MQ classes for Jakarta Messaging オブジェクトはサポートされません。その JNDI 統合は IBM MQ classes for JMS 専用です。

関連概念

IBM MQ classes for Jakarta Messaging を使用する理由

JMS および Jakarta Messaging モデル

JMS および Jakarta Messaging モデルは、 Java アプリケーションがメッセージング操作を実行するために使用できるインターフェースのセットを定義します。 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 Java メッセージング・オブジェクトが IBM MQ の概念にどのように関連するかを定義するメッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様は、特定のメッセージング・オブジェクトが管理対象オブジェクトであることを想定しています。

IBM MQ 8.0 以降、製品は JMS 標準の JMS 2.0 バージョンをサポートします。これにより、 JMS 1.1 のクラシック API も保持しながら、簡素化された API が導入されました。

JMS 3.0 IBM MQ 9.3.0 では、 Jakarta Messaging 3.0 のサポートが導入されました。 JMS 2.0 は引き続き完全にサポートされます。 JMS と Jakarta Messaging は多くの共通点を共有しているため、このトピックにおける JMS への以降の参照は、両方の参照と見なすことができます。 必要に応じて、差異が強調表示されます。

簡易 API

JMS 2.0 では、単純化された API が導入されました。また、 JMS 1.1 からのドメイン固有およびドメイン独立のインターフェースも保持されます。 簡易 API では、メッセージの送受信に必要なオブジェクトの数が減り、次のインターフェースで構成されます。

ConnectionFactory
ConnectionFactory は、接続を作成するために JMS クライアントが使用する管理オブジェクトです。このインターフェースはクラシック API でも使用されます。

JMSContext
このオブジェクトは、クラシック API の接続オブジェクトとセッション・オブジェクトを結合します。JMSContext オブジェクトは、他の JMSContext オブジェクトから作成でき、基礎接続が複製されます。

JMS プロデューサー
JMSProducer は、JMSContext によって作成され、キューまたはトピックにメッセージを送信するために使用されます。 JMSProducer オブジェクトによって、メッセージの送信に必要とされるオブジェクトの作成が発生します。

JMS コンシューマー
JMSConsumer は、JMSContext によって作成され、トピックまたはキューからのメッセージの受信に使用されます。

簡易 API には、様々な効果があります。

- JMSContext オブジェクトは、常に自動的に基礎接続を開始します。
- JMSProducer と JMSConsumer は、メッセージ・オブジェクト全体を取得しなくても、メッセージの getBody メソッドを使用してメッセージ本文を直接処理できるようになりました。
- メッセージのプロパティーは、「本文」、つまりメッセージのコンテンツを送信する前に、メソッド・チェーニングを使用して JMSProducer オブジェクトに設定できます。 JMSProducer は、メッセージの送信に必要なすべてのオブジェクトの作成を処理します。 JMS 2.0 を使用して、次のようにプロパティーを設定して、メッセージを送信できます。

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 では、メッセージを複数のコンシューマー間で共有できる共有サブスクリプションも導入されました。 すべての JMS 1.1 のサブスクリプションは、非共有サブスクリプションとして処理されます。

クラシック API

次のリストは、クラシック API の主な JMS インターフェースを要約しています。

Destination
Destination は、アプリケーションがメッセージを送信する場所、またはアプリケーションが受信するメッセージの送信元、あるいはその両方です。

ConnectionFactory
ConnectionFactory オブジェクトは、接続の構成プロパティーのセットをカプセル化します。 アプリケーションは、接続ファクトリーを使用して接続を作成します。

接続
Connection オブジェクトは、メッセージング・サーバーに対するアプリケーションのアクティブな接続をカプセル化します。 アプリケーションは、接続を使用してセッションを作成します。

Session
Session は、メッセージを送受信する単一スレッド化されたコンテキストです。 アプリケーションは、Session を使用してメッセージ、メッセージ・プロデューサー、およびメッセージ・コンシューマーを作成します。 セッションは、トランザクション化しても、トランザクション化しなくてもかまいません。

メッセージ
Message オブジェクトは、アプリケーションが送信または受信するメッセージをカプセル化します。

MessageProducer
アプリケーションがメッセージ・プロデューサーを使用して宛先にメッセージを送信します。

MessageConsumer
アプリケーションがメッセージ・コンシューマーを使用して宛先に送信されたメッセージを受信します。

161 ページの図 53 は、これらのオブジェクトとその関係を示します。

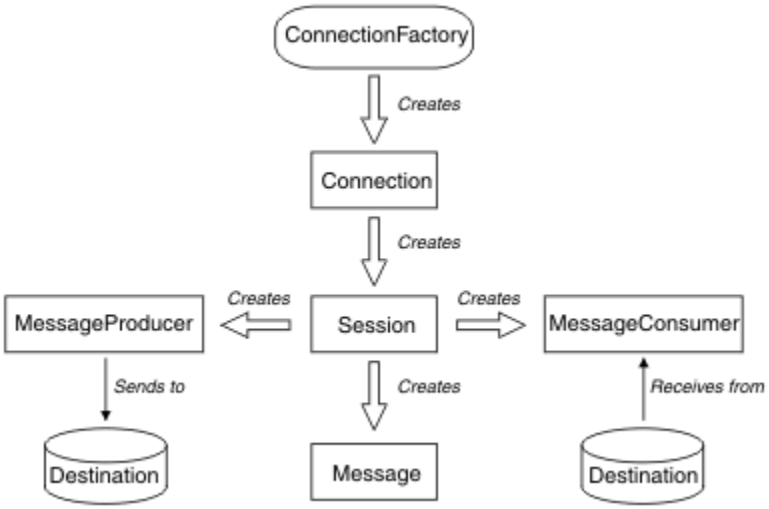


図 *53. JMS* オブジェクトとその関係

図には、メイン・インターフェースの ConnectionFactory、Connection、Session、MessageProducer、MessageConsumer、Message、および Destination が示されています。 アプリケーションは、接続ファクトリーを使用して接続を作成し、接続を使用してセッションを作成します。 アプリケーションは、次にセッションを使用してメッセージ、メッセージ・プロデューサー、およびメッセージ・コンシューマーを作成します。 アプリケーションはメッセージ・プロデューサーを使用してメッセージを宛先に送信し、メッセージ・コンシューマーを使用して宛先に送信されたメッセージを受信します。

Destination、ConnectionFactory、または Connection オブジェクトは、マルチスレッド・アプリケーションの異なるスレッドによって並行して使用できますが、Session、MessageProducer、または MessageConsumer オブジェクトは、異なるスレッドによって並行して使用することはできません。 Session、MessageProducer、または MessageConsumer オブジェクトが並行して使用されないようにする最も簡単な方法は、スレッドごとに別の Session オブジェクトを作成することです。

JMS は、2 つのスタイルのメッセージングをサポートします。

- Point-to-Point メッセージング
- パブリッシュ/サブスクライブ・メッセージング

メッセージングのこれらのスタイルは、メッセージ・ドメイン とも呼ばれ、1 つのアプリケーションで両方のスタイルのメッセージングを結合することができます。 Point-to-Point ドメインの場合、宛先はキューであり、パブリッシュ/サブスクライブ・ドメインの場合、宛先はトピックです。

JMS 1.1 の以前のバージョンの JMS では、Point-to-Point ドメインのプログラミングはインターフェースとメソッドの 1 つのセット使用され、パブリッシュ/サブスクライブ・ドメインのプログラミングは別のセットを使用します。 2 つのセットは似ていますが、別のものです。 JMS 1.1 以降、両方のメッセージ・ドメインをサポートするインターフェースとメソッドの共通のセットを使用できます。 共通のインターフェースは、メッセージ・ドメインごとにドメイン非依存のビューを提供します。 162 ページの表 17 に、JMS のドメイン非依存インターフェースと対応するドメイン固有インターフェースをリストします。

表 *17. JMS* のドメイン非依存インターフェースとその対応するドメイン固有インターフェース

ドメイン非依存インターフェース	Point-to-Point ドメインのドメイン固有インターフェース	パブリッシュ/サブスクライブ・ドメインのドメイン固有インターフェース
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
接続	QueueConnection	TopicConnection
Destination	キュー	トピック
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 は、以前の JMS 1.1 ドメイン固有インターフェースと、JMS 2.0 の単純化された API の両方をサポートします。 そのため、 IBM MQ classes for JMS 2.0 を使用して、既存のアプリケーションを保守することができます。これには、既存のアプリケーションでの新機能の開発も含まれます。

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 は、同じインターフェースの Jakarta Messaging バージョンをサポートしており、新しいアプリケーション開発に推奨されます。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では、 JMS オブジェクトは、以下の方法で IBM MQ の概念に関連しています。

- Connection オブジェクトには、接続の作成に使用された接続ファクトリーのプロパティーから派生したプロパティーがあります。 これらのプロパティーは、アプリケーションがキュー・マネージャーに接続する方法を制御します。 これらのプロパティーの例はキュー・マネージャーの名前であり、クライアント・モードのキュー・マネージャーに接続するアプリケーションでは、キュー・マネージャーが動作しているシステムのホスト名または IP アドレスです。
- Session オブジェクトは、IBM MQ 接続ハンドルをカプセル化するため、セッションのトランザクションの範囲を定義します。
- MessageProducer オブジェクトと MessageConsumer オブジェクトは、それぞれ IBM MQ オブジェクト・ハンドルをカプセル化します。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用する場合は、IBM MQ の通常の規則がすべて適用されます。 特に、アプリケーションはリモート・キューにメッセージを送信できますが、アプリケーションが接続しているキュー・マネージャーによって所有されるキューからしかメッセージを受信できないことに注意してください。

JMS 仕様は、ConnectionFactory オブジェクトと Destination オブジェクトが管理オブジェクトであると想定します。 管理者は管理オブジェクトを中央リポジトリーで作成して保守し、JMS アプリケーションは Java Naming and Directory Interface (JNDI) を使用してこれらのオブジェクトを取得します。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では、Destination インターフェースの実装は Queue および Topic の抽象スーパークラスであるため、Destination のインスタンスは Queue オブジェクトまたは Topic オブジェクトのいずれかになります。 ドメイン非依存インターフェースは、キューまたはトピックを宛先として処理します。 MessageProducer オブジェクトまたは MessageConsumer

オブジェクトのメッセージ・ドメインは、宛先がキューまたはトピックのいずれであるかによって決定されます。

したがって、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では、以下のタイプのオブジェクトを管理対象オブジェクトにすることができます。

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- キュー
- トピック
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

IBM MQ classes for JMS/Jakarta Messaging アーキテクチャー

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging には、階層化アーキテクチャーがあります。 コードの最上位レイヤーは、すべての IBM Java メッセージング・プロバイダーが使用できる共通のレイヤーです。

JM 3.0 IBM MQ 9.3.0 では、 Jakarta Messaging 3.0 のサポートが導入されました。 JMS 2.0 は引き続き完全にサポートされます。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging には、図 164 ページの図 54 に示す階層化アーキテクチャーがあります。 コードの最上位のレイヤーは、任意の IBM JMS または Jakarta Messaging プロバイダーが使用できる共通レイヤーです。 アプリケーションが JMS メソッドまたは Jakarta Messaging メソッドを呼び出すと、メッセージング・システムに固有でない呼び出しの処理は共通レイヤーによって実行されます。共通レイヤーは、呼び出しに対する一貫性のある応答も提供します。 メッセージング・システムに固有の呼び出しの処理は、より下の層に委任されます。 次の図で、IBM MQ メッセージング・プロバイダーが、他の 2 つのメッセージング・プロバイダー (メッセージング・プロバイダー A とメッセージング・プロバイダー B) と共に、下の層に示されています。

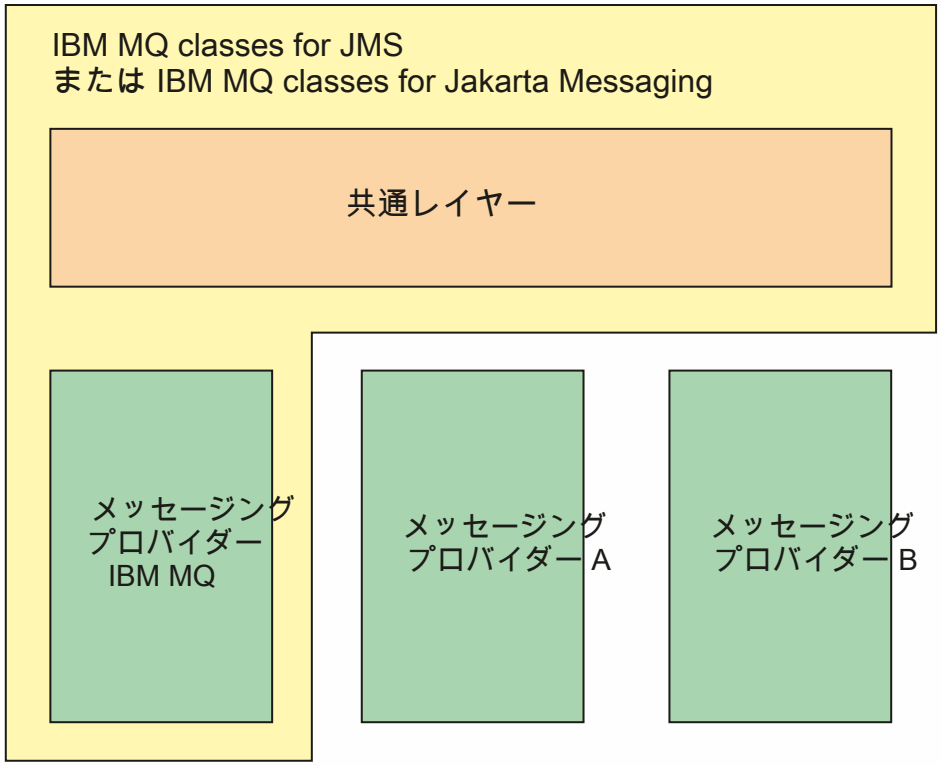


図 *54. IBM JMS および Jakarta Messaging プロバイダーの階層化アーキテクチャー*

階層化アーキテクチャーは、次の目的を達成します。

- さまざまな IBM JMS および Jakarta Messaging プロバイダーの動作の整合性を向上させる
- 2 つの IBM メッセージング・システムの間のブリッジ・アプリケーションの作成を簡単にする
- ある IBM JMS または Jakarta Messaging プロバイダーから別のプロバイダーへのアプリケーションの移植を容易にする

関連タスク

IBM MQ classes for JMS/Jakarta Messaging の使用

管理オブジェクトのサポート

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、管理対象オブジェクトの使用をサポートします。

JM 3.0 IBM MQ 9.3.0 以降、 Jakarta Messaging 3.0 は新規アプリケーションの開発用にサポートされています。 IBM MQ 9.3.0 以降では、既存のアプリケーションの JMS 2.0 が引き続きサポートされます。 Jakarta Messaging 3.0 API と JMS 2.0 API の両方を同じアプリケーションで使用することはサポートされていません。 詳しくは、 Using IBM MQ classes for JMS/Jakarta Messaging を参照してください。

JMS または IBM MQ classes for Jakarta Messaging アプリケーション内のロジックのフローは、ConnectionFactory オブジェクトと Destination オブジェクトで始まります。 アプリケーションは ConnectionFactory オブジェクトを使用して Connection オブジェクトを作成します。このオブジェクトは、メッセージング・サーバーへのアプリケーションのアクティブな接続を表します。 アプリケーションは Connection オブジェクトを使用して Session オブジェクトを作成します。このオブジェクトは、メッセージを作成およびコンシュームするための単一スレッド・コンテキストです。 さらに、アプリケーションは Session オブジェクトおよび Destination オブジェクトを使用して MessageProducer オブジェクトを作成できます。アプリケーションはこのオブジェクトを使用して、メッセージを指定された宛先に送信します。 宛先はメッセージング・システム内のキューまたはトピックのいずれかであり、Destination オブジェクトによってカプセル化されます。 また、アプリケーションは Session オブジェクトおよび Destination オ

ブジェクトを使用して MessageConsumer オブジェクトを作成できます。アプリケーションはこのオブジェクトを使用して、指定された宛先に送信されているメッセージを受信します。

JMS 仕様および Jakarta Messaging 仕様では、 ConnectionFactory オブジェクトと Destination オブジェクトが管理対象オブジェクトであることが想定されています。 管理者は、中央リポジトリーに管理対象オブジェクトを作成して保守します。 JMS または Jakarta Messaging アプリケーションは、 Java Naming Directory Interface (JNDI) を使用してこれらのオブジェクトを取得します。 管理対象オブジェクトのリポジトリーは、単純なファイルから Lightweight Directory Access Protocol (LDAP) ディレクトリーまでの範囲にすることができます。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、管理対象オブジェクトの使用をサポートします。 アプリケーションは、 IBM MQ 固有の情報をアプリケーション自体にハードコーディングすることなく、IBM MQ を介して公開される IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging のすべての機能を使用できます。 この方法で、アプリケーションは基盤となる IBM MQ の構成からある程度独立できます。

この独立性を実現するために、アプリケーションは JNDI を使用して管理対象オブジェクトとして保管されている接続ファクトリーおよび宛先を取得し、 `javax.jms` (JMS 2.0) または `jakarta.jms` (Jakarta Messaging 3.0) パッケージで定義されているインターフェースのみを使用してメッセージング操作を実行することができます。

JMS 2.0 JMS 2.0 の場合、管理者は IBM MQ JMS 管理ツール **JMSAdmin** または IBM MQ Explorer を使用して、中央リポジトリーで管理対象オブジェクトを作成および保守することができます。

JM 3.0 Jakarta Messaging 3.0 の場合、 IBM MQ Explorer を使用して JNDI を管理することはできません。 JNDI 管理は、 **JMSAdmin** の Jakarta Messaging 3.0 バリアント (**JMS30Admin**) によってサポートされます。

アプリケーション・サーバーは通常、管理対象オブジェクト用の独自のリポジトリーと、オブジェクトを作成および保守するための独自のツールを提供します。 したがって、Java EE JM 3.0 または Jakarta EE アプリケーションは、 JNDI を使用して、アプリケーション・サーバー・リポジトリーまたは中央リポジトリーから管理対象オブジェクトを取得することができます。

関連タスク

JMS および Jakarta Messaging リソースの構成

Java EE および Jakarta EE プラットフォームでサポートされる通信タイプ

Java EE および Jakarta EE プラットフォームでは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、アプリケーションのコンポーネントと IBM MQ キュー・マネージャーの間の 2 つのタイプの通信をサポートします。

JM 3.0 IBM MQ 9.3.0 では、 Jakarta Messaging 3.0 のサポートが導入されました。 JMS 2.0 は引き続き完全にサポートされます。 JMS と Jakarta Messaging は多くの共通点を共有しているため、このトピックにおける JMS への以降の参照は、両方の参照と見なすことができます。 必要に応じて、差異が強調表示されます。

アプリケーションのコンポーネントと IBM MQ キュー・マネージャーの間の次の 2 つのタイプの通信がサポートされます。

- アウトバウンド通信
- インバウンド通信

アウトバウンド通信

JMS API または Jakarta Messaging API を直接使用して、アプリケーション・コンポーネントはキュー・マネージャーへの接続を作成し、メッセージを送受信します。

例えば、アプリケーション・コンポーネントは、アプリケーション・クライアント、サーブレット、JavaServer Page (JSP)、エンタープライズ Java Bean (EJB)、またはメッセージ駆動型 Bean (MDB) である

ことが可能です。 このタイプの通信では、アプリケーション・サーバー・コンテナーは、接続のプールやスレッド管理など、メッセージング操作をサポートする低レベルの機能のみを提供します。

インバウンド通信

インバウンド通信の場合、宛先に到達するメッセージは MDB に配信されてから、そこでメッセージが処理されます。

Java EE JM 3.0 および Jakarta EE アプリケーションは、MDB を使用してメッセージを非同期で処理します。 MDB は JMS のメッセージ・リスナーとして機能し、メッセージの処理方法を定義する onMessage() メソッドによって実装されます。 MDB は、アプリケーション・サーバーの EJB コンテナーに実装されます。 MDB が構成される正確な方法は、使用しているアプリケーション・サーバーに依存しますが、構成情報で、接続先のキュー・マネージャー、キュー・マネージャーへの接続方法、メッセージをモニターする宛先、および MDB のトランザクションの動作を指定する必要があります。 この情報は EJB コンテナーによって使用されます。 MDB の選択基準を満たすメッセージが指定された宛先に到着すると、EJB コンテナーは IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用してキュー・マネージャーからメッセージを取得し、onMessage() メソッドを呼び出して MDB にメッセージを配信します。

IBM MQ classes for Java との関係

IBM MQ classes for Java、 IBM MQ classes for Jakarta Messaging 、および IBM MQ classes for JMS は、MQI への共通 Java インターフェースを使用するピアです。

166 ページの図 55 は、 IBM MQ classes for JMS、 IBM MQ classes for Jakarta Messaging 、および IBM MQ classes for Java の間の関係を示しています。

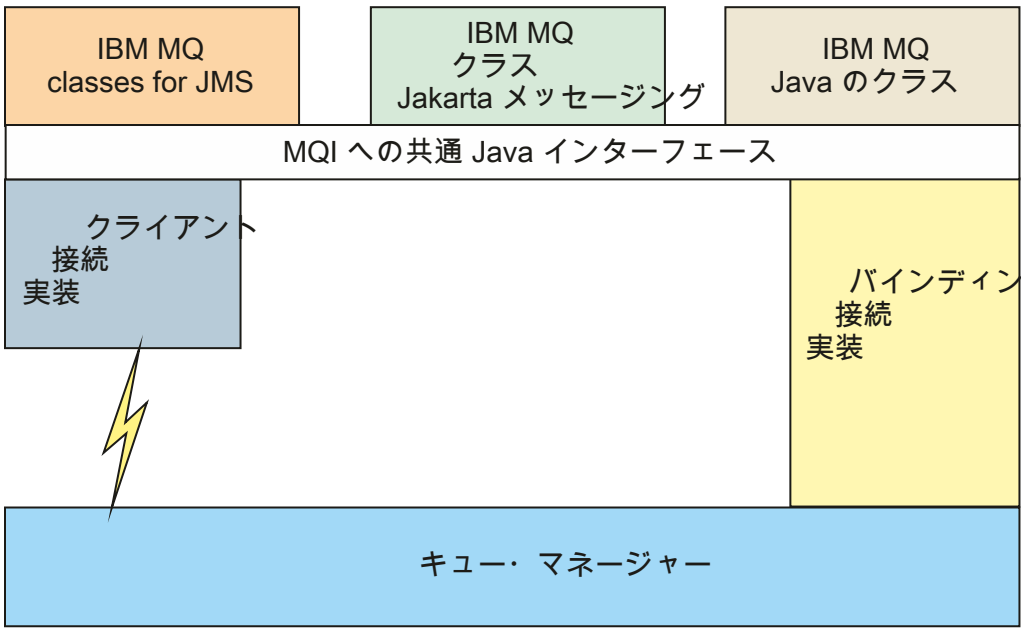


図 *55. IBM MQ classes for JMS*、 *IBM MQ classes for Jakarta Messaging* 、および *IBM MQ classes for Java* の間の関係

一般に、 Java プログラムは、 IBM MQ - IBM MQ classes for Java、 IBM MQ classes for Jakarta Messaging 、または IBM MQ classes for JMS とのインターフェースとして 1 つのインターフェースのみを使用する必要があります。 インターフェースの混合はサポートされていませんが、1 つの例外があります。 IBM WebSphere MQ 7.0 より前のリリースとの互換性を維持するために、Java で作成されたチャネル出口クラスは、チャネル出口クラスが IBM MQ classes for JMS から呼び出される場合でも、 IBM MQ classes for Java インターフェースを使用できます。 ただし、 IBM MQ classes for Java インターフェースを使用することは、アプリケーションが引き続き以下のいずれかに依存していることを意味します。

- JMS 2.0 IBM MQ classes for Java JAR ファイル、 `com.ibm.mq.jar`。 クラス・パスに `com.ibm.mq.jar` が不要な場合は、代わりに `com.ibm.mq.exits` パッケージのインターフェースのセットを使用できます。
- JM 3.0 IBM MQ classes for Jakarta Messaging と相互運用する場合の `com.ibm.mq.jakarta.client.jar` の使用。

関連概念

IBM MQ classes for Jakarta Messaging を使用する理由
JMS 用に IBM MQ クラスを使用する理由
IBM MQ classes for Java を使用する理由

IBM MQ メッセージング・プロバイダー

IBM MQ メッセージング・プロバイダーには、通常モード、制限付き通常モード、マイグレーション・モードという 3 つの操作モードがあります。

IBM MQ メッセージング・プロバイダーには、3 つの操作モードがあります。

- IBM MQ メッセージング・プロバイダーの通常モード
- IBM MQ メッセージング・プロバイダーの制限付き通常モード
- IBM MQ メッセージング・プロバイダーのマイグレーション・モード

IBM MQ メッセージング・プロバイダーの通常モードでは、IBM MQ キュー・マネージャーのすべての機能を使用して JMS が実装されます。 このモードは、 JMS 2.0 JM 3.0 または Jakarta Messaging 3.0 API および機能を使用するように最適化されています。

次の場合

- クライアントは、 **ConnectionFactory** 上でプロバイダー・バージョン 6 を指定します。クライアントは、 IBM WebSphere MQ 6.0 で提供されるクライアントと互換性のある方法で動作します。 JMS 1.1 および JMS 2 インターフェースのみがサポートされますが、一部の JMS 2 機能 (共有サブスクリプション、送達遅延、非同期送信など) は使用不可になります。 接続の共有はありません。
- クライアントは、 **ConnectionFactory** 上でプロバイダー・バージョン 7 を指定します。JMS 1.1 インターフェースと JMS 2 インターフェースの両方が完全にサポートされます。
- プロバイダー・バージョンが指定されていません。プロバイダー・バージョン 7 との接続が試行されます。 これが失敗すると、プロバイダー・バージョン 6 でさらに試行が行われます。

IBM MQ Enterprise Transport を使用して IBM Integration Bus に接続する場合は、マイグレーション・モードを使用します。 IBM MQ Real-Time Transport を使用する場合は、接続ファクトリー・オブジェクトでプロパティーを明示的に選択しているため、マイグレーション・モードが自動的に選択されます。 IBM MQ Enterprise Transport を使用した IBM Integration Bus への接続は、JMS **PROVIDERVERSION** プロパティーの構成で説明されているモード選択の一般規則に従います。

関連タスク

JMS リソースの構成

z/OS IBM MQ for z/OS concepts

Some of the concepts used by IBM MQ for z/OS are unique to the z/OS platform. For example, the logging mechanism, the storage management techniques, unit of recovery disposition, and queue sharing groups are provided only with IBM MQ for z/OS. Use this topic as an introduction to further information about these concepts.

The concepts include an overview of the objects that IBM MQ for z/OSuses, including:

- The queue manager
- The channel initiator
- Shared queues and queue sharing groups

- Intra-group queuing

The following topics also cover various procedures you need, including:

- System definitions on z/OS
- Storage management
- Recovery and restart
- Security concepts in IBM MQ for z/OS

Related concepts

"The queue manager on z/OS" on page 169
Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

"The channel initiator on z/OS" on page 170
The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

"Terms and tasks for managing IBM MQ for z/OS" on page 171
Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

"Shared queues and queue sharing groups" on page 174
You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

"Intra-group queuing" on page 218
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Storage management on z/OS" on page 231
IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

"Logging in IBM MQ for z/OS" on page 235
IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

"Recovery and restart on z/OS" on page 256
Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

"Security concepts in IBM MQ for z/OS" on page 272
Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

"Availability on z/OS" on page 278
IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

"Unit of recovery disposition on z/OS" on page 283
Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Related reference

"System definition on z/OS" on page 246
IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

"Monitoring and statistics on IBM MQ for z/OS" on page 282
IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

z/OS The queue manager on z/OS

Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

The queue manager

A *queue manager* is a program that provides messaging services to applications. Applications that use the Message Queue Interface (MQI) can put messages on queues and get messages from queues. The queue manager ensures that messages are sent to the correct queue or are routed to another queue manager. The queue manager processes both the MQI calls that are issued to it, and the commands that are submitted to it (from whatever source). The queue manager generates the appropriate completion codes for each call or command.

The resources managed by the queue manager include:

- Page sets that hold the IBM MQ object definitions and message data
- Logs that are used to recover messages and objects in the event of queue manager failure
- Processor storage
- Connections through which different application environments (CICS, IMS, and Batch) can access the IBM MQ API
- The IBM MQ channel initiator, which allows communication between IBM MQ on your z/OS system and other systems

The queue manager has a name, and applications can connect to it using this name.

Figure 56 on page 169 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

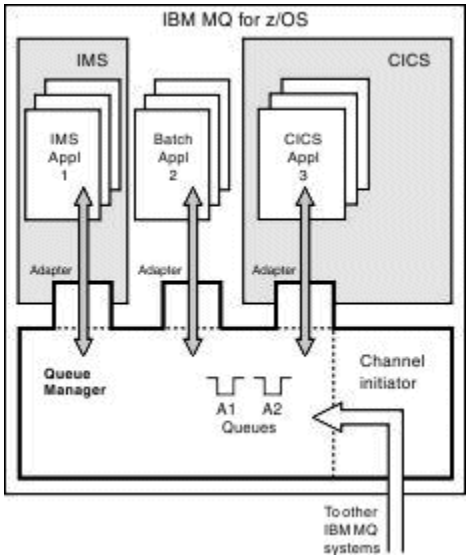


Figure 56. Overview of IBM MQ for z/OS

The queue manager subsystem on z/OS

On z/OS, IBM MQ runs as a z/OS subsystem that is started at IPL time. In the subsystem, the queue manager is started by executing a JCL procedure that specifies the z/OS data sets that contain information about the logs, and that hold object definitions and message data (the page sets). The subsystem and the queue manager have the same name, of up to four characters. All queue managers in your network must have unique names, even if they are on different systems, sysplexes, or platforms.

z/OS The channel initiator on z/OS

The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

To send messages from queue manager A to queue manager B, a *sending* MCA on queue manager A must set up a communications link to queue manager B. A *receiving* MCA must be started on queue manager B to receive messages from the communications link. This one-way path consisting of the sending MCA, the communications link, and the receiving MCA is known as a *channel*. The sending MCA takes messages from a transmission queue and sends them down a channel to the receiving MCA. The receiving MCA receives the messages and puts them on to the destination queues.

In IBM MQ for z/OS, the sending and receiving MCAs all run inside the channel initiator (the channel initiator is also known as the *mover*). The channel initiator runs as a z/OS address space under the control of the queue manager. There can be only a single channel initiator connected to a queue manager and it is run inside the same z/OS image as the queue manager. There can be thousands of MCA processes running inside the channel initiator concurrently.

Figure 57 on page 170 shows two queue managers within a sysplex. Each queue manager has a channel initiator and a local queue. Messages sent by queue managers on AIX and Windows are placed on the local queue, from where they are retrieved by an application. Reply messages are returned by a similar route.

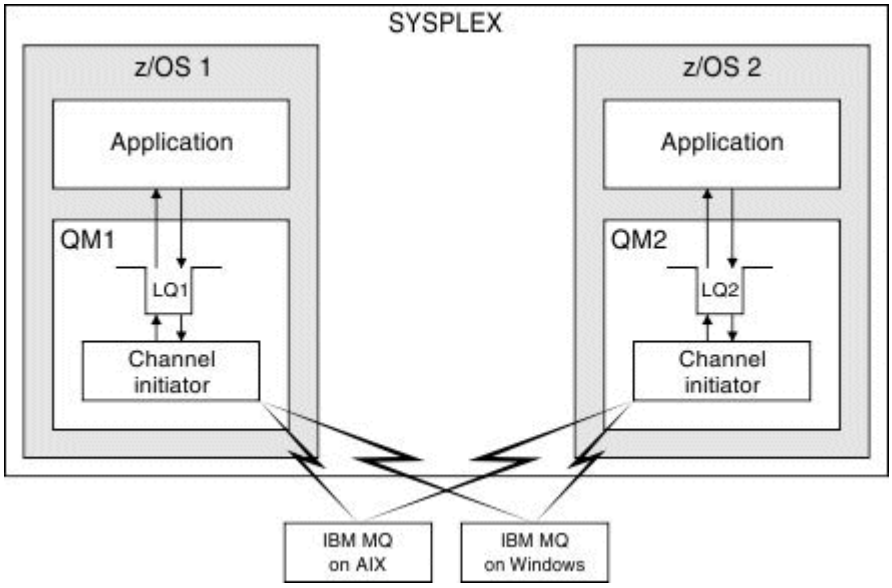


Figure 57. Communication between queue managers

The channel initiator also contains other processes concerned with the management of the channels. These processes include:

Listeners
These processes listen for inbound channel requests on a communications subsystem such as TCP, and start a named MCA when an inbound request is received.

Supervisor
This manages the channel initiator address space, for example it is responsible for restarting channels after a failure.

Name server
This is used to resolve TCP names into addresses.

TLS tasks
These are used to perform encryption and decryption and check certificate revocation lists.

z/OS SMF records for the channel initiator

The channel initiator (CHINIT) can produce SMF statistics records and accounting records with information on tasks and channels.

The CHINIT can produce SMF statistics records and accounting records with the following types of information:

- The tasks: dispatcher, adapter, Domain Name Server (DNS), and SSL. These tasks form what is called CHINIT statistics.
- Channels: provides accounting information similar to that available with the DIS CHSTATUS command. This is called channel accounting.

IBM MQ for Multiplatforms provides similar information by writing PCF messages to the SYSTEM.ADMIN.STATISTICS.QUEUE. See Channel statistics message data for further information on how statistics information is recorded on IBM MQ for Multiplatforms.

Statistics data

You can use this information to find out the following information:

- Whether you need more of the CHINIT tasks, such as number of SSL TCBs and how much CPU is used by these tasks.
- The average time for requests on these tasks.
- The longest duration request in the interval, and the time of day this occurred, for DNS and SSL tasks. You can correlate this time of day with problems you may experience with the channel.

Accounting data

You can use this information to monitor channel usage and find out the following information:

- The channels with the highest throughput.
- The rate at which messages were sent, and the rate of sending data in MB/second.
- The achieved batch size. If the achieved batch size is close to the batch size specified for the channel, the channel might be close to its limit for sending messages.

You use the START TRACE and STOP TRACE commands to control the collection of the accounting trace and the statistics trace. You can use the STATCHL and STATACLS options on the channel and queue manager to control whether channels produce SMF data.

z/OS Terms and tasks for managing IBM MQ for z/OS

Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

Some of the terms and tasks required for managing IBM MQ for z/OS are specific to the z/OS platform. The following list contains some of these terms and tasks.

- Shared queues
- Page sets and buffer pools
- Logging
- Tailoring the queue manager environment
- Restart and recovery
- Security
- Availability
- Manipulating objects
- Monitoring and statistics
- Application environments

Shared queues

Queues can be *non-shared*, owned by and accessible to only one queue manager, or *shared*, owned by a *queue sharing group*. A queue sharing group consists of a number of queue managers, running within a single z/OS sysplex, that can access the same IBM MQ object definitions and message data concurrently. Within a queue sharing group, the shareable object definitions are stored in a shared Db2 database. The shared queue messages are held inside one or more coupling facility structures (CF structures). If the message data is too large to store directly in the structure (more than 63 KB in size), or if the message is large enough that installation-defined rules select it for offloading, the message control information is still stored in the coupling facility entry, but the message data is offloaded to a shared message data set (SMDS) or to a shared Db2 database. The shared message data sets, the shared Db2 database, and the coupling facility structures are resources that are jointly managed by all of the queue managers in the group.

Pages sets and buffer pools

When a message is put on to a non-shared queue, the queue manager stores the data on a page set in such a way that it can be retrieved when a subsequent operation gets a message from the same queue. If the message is removed from the queue, space in the page set that holds the data is later freed for reuse. As the number of messages held on a queue increases, so the amount of space used in the page set increases, and as the number of messages on a queue reduces, the space used in the page set reduces.

To reduce the performance cost of writing data to and reading data from the page sets, the queue manager buffers the updates into processor storage. The amount of storage used to buffer the page set access is controlled through IBM MQ objects called *buffer pools*.

For more information about page sets and buffer pools, see Storage management.

Logging

Any changes to objects held on page sets, and operations on persistent messages, are recorded as log records. These log records are written to a log data set called the *active log*. The name and size of the active log data set is held in a data set called the *bootstrap data set* (BSDS).

When the active log data set fills up, the queue manager switches to another log data set so that logging can continue, and copies the content of the full active log data set to an *archive log* data set. Information about these actions, including the name of the archive log data set, is held in the bootstrap data set. Conceptually, there is a ring of active log data sets that the queue manager cycles through; when an active log is filled, the log data is offloaded to an archive log, and the active log data set is available for reuse.

For more information about the log and bootstrap data sets, see “Logging in IBM MQ for z/OS” on page 235.

Tailoring the queue manager environment

When the queue manager is started, a set of initialization parameters that control how the queue manager operates are read. In addition, data sets containing IBM MQ commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for IBM MQ to run, and you can tailor these to define or initialize the IBM MQ objects necessary for your operating environment. When these data sets have been read, any objects defined by them are stored, either on a page set or in Db2.

For more information about initialization parameters and system objects, see “System definition on z/OS” on page 246.

Recovery and restart

At any time during the operation of IBM MQ, there might be changes held in processor storage that have not yet been written to the page set. These changes are written out to the page set that is the least recently used by a background task within the queue manager.

If the queue manager terminates abnormally, the recovery phase of queue manager restart can recover the lost page set changes because persistent message data is held in log records. This means that IBM MQ can recover persistent message data and object changes right up to the point of failure.

If a queue manager that is a member of a queue sharing group encounters a coupling facility failure, the persistent messages on that queue can be recovered only if you have backed up your coupling facility structure.

For more information about recovery and restart, see “Recovery and restart on z/OS” on page 256.

Security

You can use an external security manager, such as Security Server (previously known as RACF) to protect the resources that IBM MQ owns and manages from access by unauthorized users. You can also use Transport Layer Security (TLS) for channel security. TLS is included as part of the IBM MQ product.

For more information about IBM MQ security, see “Security concepts in IBM MQ for z/OS” on page 272.

Availability

There are several features of IBM MQ that are designed to increase system availability in the event of queue manager or communications subsystem failure. For more information about these features, see “Availability on z/OS” on page 278.

Manipulating objects

When the queue manager is running, you can manipulate IBM MQ objects either through a z/OS console interface, or through an administration utility that uses ISPF services under TSO. Both mechanisms enable you to define, alter, or delete IBM MQ objects. You can also control and display the status of various IBM MQ and queue manager functions.

For more information about these facilities, see Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS.

You can also manipulate IBM MQ objects using the IBM MQ Explorer, a graphical user interface that provides a visual way of working with queues, queue managers, and other objects.

Monitoring and statistics

Several facilities are available to monitor your queue managers and channel initiators. You can also collect statistics for performance evaluation and accounting purposes.

For more information about these facilities, see “Monitoring and statistics on IBM MQ for z/OS” on page 282.

Application environments

When the queue manager has started, applications can connect to it and start using the IBM MQ API. These can be CICS, IMS, Batch, or WebSphere Application Server applications. IBM MQ applications can also access applications on CICS and IMS systems that are not aware of IBM MQ, using the CICS and IMS bridges.

For more information about these facilities, see “IBM MQ and other z/OS products” on page 285.

For information about writing IBM MQ applications, see the following documentation:

- Developing applications
- Using C++
- Using IBM MQ classes for Java

z/OS Shared queues and queue sharing groups

You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

This section describes the attributes and benefits, and offers information about how several queue managers can share the same queues and the messages on those queues.

What is a shared queue?

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

A queue sharing group

The queue managers that can access the same set of shared queues form a group called a *queue sharing group*.

Any queue manager can access messages

Any queue manager in the queue sharing group can access a shared queue. This means that you can put a message on to a shared queue on one queue manager, and get the same message from the queue from a different queue manager. This provides a rapid mechanism for communication within a queue sharing group that does not require channels to be active between queue managers.

IBM MQ supports the offloading of messages to Db2 or a shared message data set (SMDS). The offloading of messages of any size is configurable.

Figure 58 on page 175 shows three queue managers and a coupling facility, forming a queue sharing group. All three queue managers can access the shared queue in the coupling facility.

An application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access all the shared queues, the application does not depend on the availability of a specific queue manager; any queue manager in the queue sharing group can service the queue.

This gives greater availability because all the other queue managers in the queue sharing group can continue processing the queue if one of the queue managers has a problem.

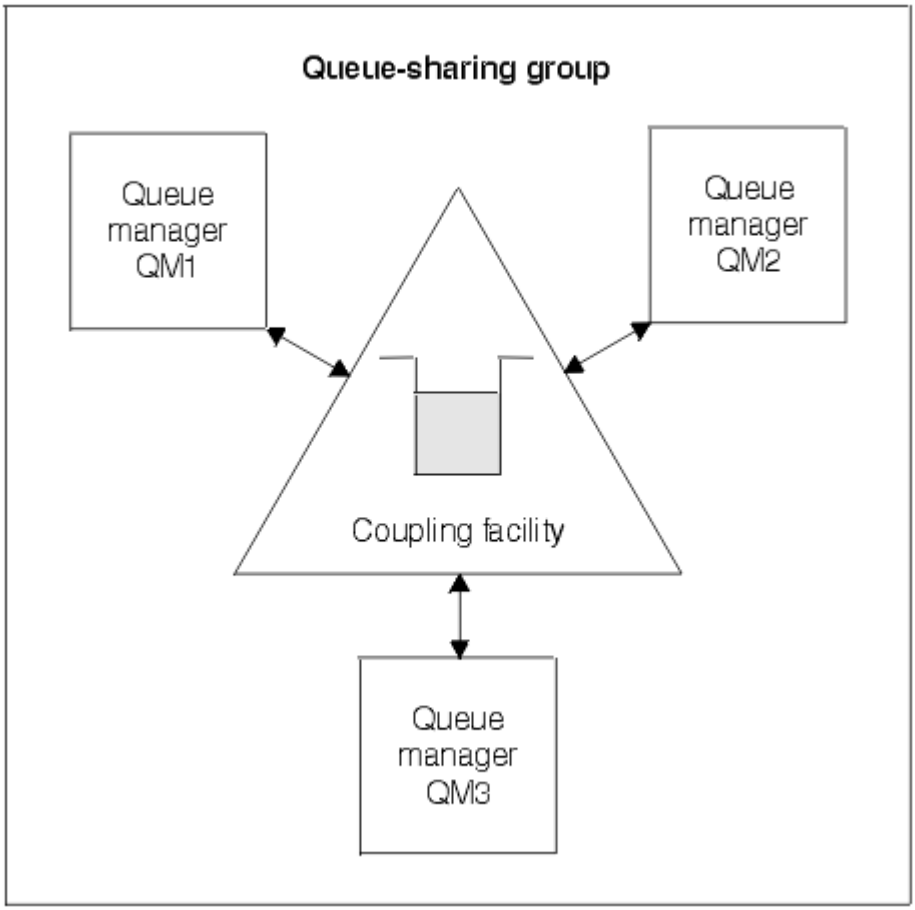


Figure 58. A queue sharing group

Queue definition is shared by all queue managers

Shared queue definitions are stored in the Db2 database table OBJ_B_QUEUE. Because of this, you need to define the queue only once and then it can be accessed by all the queue managers in the queue sharing group. This means that there are fewer definitions to make.

By contrast, the definition of a non-shared queue is stored on page set zero of the queue manager that owns the queue (as described in Page sets).

You cannot define a shared queue if a queue with that name has already been defined on the page sets of the defining queue manager. Likewise, you cannot define a local version of a queue on the queue manager page sets if a shared queue with the same name exists.

What is a queue sharing group?

A group of queue managers that can access the same shared queues is called a queue sharing group. Each member of the queue sharing group has access to the same set of shared queues.

Queue sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Figure 59 on page 176 illustrates a queue sharing group that contains two queue managers. Each queue manager has a channel initiator and its own local page sets and log data sets.

Each member of the queue sharing group must also connect to a Db2 system. The Db2 systems must all be in the same Db2 data-sharing group so that the queue managers can access the Db2 shared repository used to hold shared object definitions. These are definitions of any type of IBM MQ object (for example,

queues and channels) that are defined only once and then any queue manager in the group can use them. These are called *global* definitions and are described in Private and global definitions.

More than one queue sharing group can reference a particular data-sharing group. You specify the name of the Db2 subsystem and which data-sharing group a queue manager uses in the IBM MQ system parameters at startup.

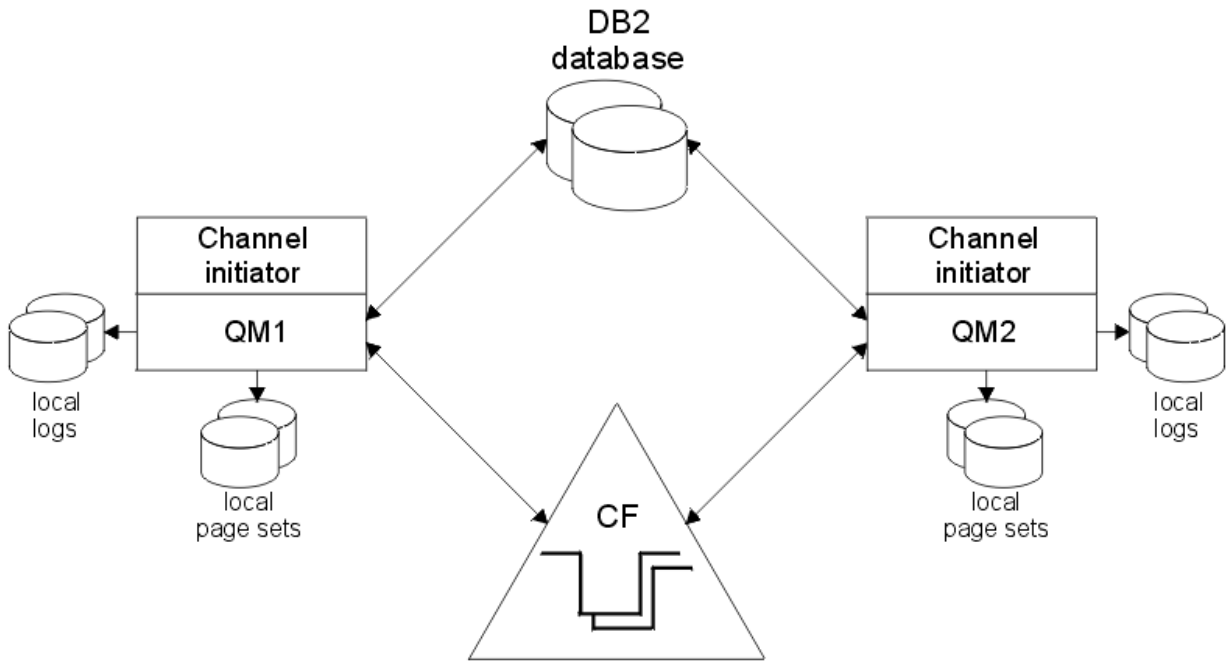


Figure 59. The components of queue managers in a queue sharing group

When a queue manager has joined a queue sharing group, it has access to the shared objects defined for that group, and you can use that queue manager to define new shared objects within the group. If shared queues are defined within the group, you can use this queue manager to put messages to and get messages from those shared queues. Any queue manager in the group can retrieve the messages held on a shared queue.

You can enter an MQSC command once, and have it executed on all queue managers within the queue sharing group as if it had been entered at each queue manager individually. The *command scope* attribute is used for this. This attribute is described in Directing commands to different queue managers.

When a queue manager runs as a member of a queue sharing group it must be possible to distinguish between IBM MQ objects defined privately to that queue manager and IBM MQ objects defined globally that are available to all queue managers in the queue sharing group. The *queue sharing group disposition* attribute is used for this. This attribute is described in Private and global definitions.

You can define a single set of security profiles that control access to IBM MQ objects anywhere within the group. This means that the number of profiles you have to define is greatly reduced.

A queue manager can belong to only one queue sharing group, and all queue managers in the group must be in the same sysplex. You specify which queue sharing group the queue manager belongs to in the system parameters at startup.

Related concepts

"Where are shared queue messages held?" on page 177
Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

"Advantages of using shared queues" on page 193

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

"Distributed queuing and queue sharing groups" on page 213
Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

"Influencing workload distribution with shared queues" on page 216
Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

Related reference

"Where to find more information about shared queues and queue sharing groups" on page 217
Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

z/OS Where are shared queue messages held?

Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

If the CF structure has been configured to use System Class Memory (SCM), IBM MQ can use this with no additional configuration.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Shared queue message storage

Messages that are put onto shared queues are not stored on page sets and do not use buffer pools.

The messages in shared queues have entries on list structures in the z/OS coupling facility (CF). Many queue managers in the same sysplex can access those messages using the CF list structure.

The message data for small shared queue messages is normally included in the coupling facility entry. For larger messages, the message data can be stored either in a shared message data set (SMDS), or as one or more binary large objects (BLOBs) in a Db2 table which is shared by a Db2 data sharing group. Message data exceeding 63 KB is always offloaded to SMDS or Db2. Smaller messages can also optionally be offloaded in the same way to save space in the coupling facility structure. See "Specifying offload options for shared messages" on page 179 for more details.

Messages put on a shared queue are referenced in a coupling facility structure until they are retrieved by an MQGET. Coupling facility operations are used to:

- Search for the next retrievable message
- Lock uncommitted messages on shared queues
- Notify interested queue managers about the arrival of committed messages

MQPUT and MQGET operations on persistent messages are recorded on the log of the queue manager performing that operation. This minimizes the risk of data loss in the event of a coupling facility failure.

The coupling facility

The messages held on shared queues are referenced inside a coupling facility. The coupling facility lies outside any of the z/OS images in the sysplex and is typically configured to run on a different power supply. The coupling facility is therefore resilient to software failures and you can configure it so that it is resilient to hardware failures or power-outages. This means that messages stored in the coupling facility are highly available.

Each coupling facility list structure used by IBM MQ is dedicated to a specific queue sharing group, but a coupling facility can hold structures for more than one queue sharing group. Queue managers in different queue sharing groups cannot share data. Up to 32 queue managers in a queue sharing group can connect to a coupling facility list structure at the same time.

A single coupling facility list structure can contain up to 512 shared queues. The total amount of message data stored in the structure is limited by the structure capacity. However, with **CFLEVEL(5)** you can use the offload parameters to offload data for messages less than 63 KB thereby increasing the number of messages which can be stored in the structure, although each message still requires at least a coupling facility entry plus at least 768 bytes of data, made up of 256 bytes for the entry and 512 bytes for the two elements of header and descriptor.

The size of the list structure is restricted by the following factors:

- It must lie within a single coupling facility.
- It might share the available coupling facility storage with other structures for IBM MQ and other products.

Coupling facility list structures can have storage class memory associated with them. In certain situations this storage class memory can be useful when used with shared queues. See "Use of storage class memory with shared queues" on page 194 for more information.

Planning the CF structure size

If you require guidance on the sizing of your CF structures you can use the MP16: IBM MQ for z/OS Capacity planning and tuning supportpac. You can also use the web-based tool CFSizer, which is provided by IBM to assist with CF sizes.

The CF structure object

The queue manager's use of a coupling facility structure is specified in a CF structure (CFSTRUCT) IBM MQ object.

These structure objects are stored in Db2.

When using z/OS commands or definitions relating to a coupling facility structure, the first four characters of the name of the queue sharing group are required. However, an IBM MQ CFSTRUCT object always exists within a single queue sharing group, and so its name does not include the first four characters of the name of the queue sharing group. For example, CFSTRUCT(MYDATA) defined in queue sharing group starting with SQ03 would use coupling facility list structure SQ03MYDATA.

CF structures have a CFLEVEL attribute that determines their functional capability:

- 1, 2 - can be used for nonpersistent messages less than 63 KB
- 3 - can be used for persistent and nonpersistent messages less than 63 KB
- 4 - can be used for persistent and nonpersistent messages up to 100 MB
- 5 - can be used for persistent and nonpersistent messages up to 100 MB and selectively offloaded to shared message data sets (SMDS) or Db2.

Note: When using IBM MQ you can encrypt a coupling facility structure. See Encrypting coupling facility structure data for more information.

Backup and recovery of the coupling facility

You can back up coupling facility list structures using the IBM MQ command BACKUP CFSTRUCT. This puts a copy of the persistent messages currently within the CF structure onto the active log data set of the queue manager making the backup, and writes a record of the backup to Db2.

If coupling facility fails, you can use the IBM MQ command RECOVER CFSTRUCT. This uses the backup record from Db2 to locate and restore persistent messages from the backup of the CF structure. Any activity since the last backup is replayed using the logs of all the queue managers in the queue sharing group, and the CF structure is then restored up to the point before the failure.

See the BACKUP CFSTRUCT and RECOVER CFSTRUCT commands for more details.

Related concepts

"Specifying offload options for shared messages" on page 179
You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

"Managing your shared message data set (SMDS) environment" on page 181
If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

z/OS *Specifying offload options for shared messages*

You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

The message data for shared queues can be offloaded from the coupling facility and stored in either a Db2 table or in an IBM MQ managed data set called a *shared message data set* (SMDS).

For messages larger than the coupling facility entry size of 63 KB, offloading message data to a SMDS can have a significant performance improvement compared with offloading to Db2.

Every shared queue message is still managed using a list entry in a coupling facility structure, but when the message data is offloaded to the SMDS, the coupling facility entry only contains some control information and a list of references to the relevant disk blocks where the message is stored. Using this mechanism means the amount of coupling facility element storage required for each message is only a fraction of the actual size of the message.

Selecting where the shared queue messages are stored

The selection of SMDS or Db2 shared message storage is controlled with the **OFFLOAD(SMDS|DB2)** parameter on the **CFSTRUCT** definition. **OFFLOAD(SMDS)** is the default value.

This parameter also requires the **CFSTRUCT** to use **CFLEVEL(5)** or greater.

The **OFFLOAD** parameter is only valid from **CFLEVEL(5)**. See DEFINE CFSTRUCT for more details.

OFFLOAD(DB2) is supported primarily for migration purposes.

Selecting which shared queue messages are offloaded

Message data is offloaded to SMDS or Db2 based on the size of the message data, and the current usage of the coupling facility structure. There are three rules, and each rule specifies a matching pair of parameters. These parameters are a corresponding coupling facility structure usage threshold percentage (**OFFLDnTH**) and a message size limit (**OFFLDnSZ**).

The current implementation of the three rules is specified using the following pairs of keywords:

- OFFLD1TH and OFFLD1SZ
- OFFLD2TH and OFFLD2SZ
- OFFLD3TH and OFFLD3SZ

Rule pair	Default value	Description
Rule pair 1	OFFLD1TH(70) and OFFLD1SZ(32K)	If the coupling facility structure is more than 70% full offload data for messages exceeding 32 KB
Rule pair 2	OFFLD2TH(80) and OFFLD2SZ(4K)	If the coupling facility structure is more than 80% full offload data for messages exceeding 4 KB
Rule pair 3	OFFLD3TH(90) and OFFLD3SZ(0K)	If the coupling facility structure is more than 90% full offload data for messages exceeding 0 KB (all messages)

If an offload rule has the OFFLD *x* SZ value of 64K this indicates that the rule is not in effect. In this case messages will only be offloaded if another offload rule is in effect, or if the message is greater than 63.75 KB and so, too large to store in the structure.

Each message which is offloaded still requires 0.75 KB of storage in the coupling facility.

The three offload rules which can be specified for each structure are intended to be used as follows.

- Performance
  - When there is plenty of space in the application structure, message data should only be offloaded if it is too large to store in the structure, or if it exceeds some lower message size threshold such that the performance value of storing it in the structure is not worth the amount of structure space that it would need.
  - If a specific message size threshold is required, it is conventionally specified using the first offload rule.
- Capacity
  - When there is very little space in the application structure, the maximum amount of message data should be offloaded so as to make the best use of the remaining space.
  - The third offload rule is conventionally used to indicate that when the structure is nearly full, most messages should be offloaded, so the entries in the application structure will be typically of the minimum size (requiring about 0.75K bytes).
  - The usage threshold parameter should be chosen based on the application structure size and the maximum anticipated backlog. For example, if the maximum anticipated backlog is 1M messages, then the amount of structure storage required for this number of messages is about 0.75G bytes. This means for example that if the structure is about 10G bytes, the usage threshold for offloading all messages must be set to 92% or lower.
  - Structure space is divided into elements and entries, and even though there may be enough space overall, one of these may run out before the other. The system provides AUTOALTER capabilities to adjust the ratio when necessary, but this is not very sensitive, so the amount of space actually available may be somewhat less. It may be better therefore to aim to use not more than 90% of the maximum structure space, so in the previous example, the usage threshold for offloading all messages would be better set around 80%.
- Cushioned transition:
  - As the amount of space left in the coupling facility structure decreases, it would be undesirable to have a large sudden change in the performance characteristics. It is also undesirable for coupling facility management to have a sudden threshold change in the typical ratio of entries to elements being used.
  - The second offload rule is conventionally used to provide some intermediate cushion between the performance and capacity biased offload rules. It can be set to cause a significant increase in

offload activity when the space used in the coupling facility structure exceeds an intermediate threshold. This means that the remaining space is used up more slowly, and gives the coupling facility automatic alter processing more time to adapt to the higher usage levels.

If the coupling facility structure cannot be expanded, and there is a need to store at least some predetermined number of messages, the third rule can be modified as necessary to ensure that offloading of data for all messages starts at an appropriate threshold to ensure space is reserved for that predetermined number of messages.

For example, if the coupling facility structure size is 4 GB, and the predetermined number of messages is 1 million, then 1,000,000 * 0.75 KB are needed, which is 768 MB, 18.75% of 4 GB. In this case the threshold for offloading all messages needs to be set around 80% rather than 90%. This gives parameters OFFLD3TH(80) and OFFLD3SZ(0K) . The other offload parameters would also need to be adjusted.

If it is found that offloading very small messages has a significant performance impact, but the relative impact is less for larger messages, then the usage thresholds for the other rules can be reduced to offload larger messages earlier, leaving more space in the structure for small messages before they need to be offloaded.

For example, if messages exceeding 32KB occur frequently but the elapsed time performance for offloading them (as determined from RMF statistics or application performance) is very similar to that for keeping them in the coupling facility, then the threshold for the first rule could be set to 0% to offload all such messages. This gives parameters OFFLD1TH(0) and OFFLD1SZ(32K). Again the other offload parameters would need to be adjusted.

If there are many messages around specific intermediate sizes, such as 16 KB and 6 KB, then it might be useful to change the message size option for the second rule so that the larger ones get offloaded at a fairly low usage threshold, saving a significant amount of space, but the smaller ones still get stored only in the coupling facility.

z/OS *Managing your shared message data set (SMDS) environment*

If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

SMDS objects

The properties and status of each shared message data set are tracked in a shared SMDS object which can be updated through any queue manager in the queue sharing group.

There is one shared message data set for each queue manager that can access each coupling facility application structure. The shared message data set is identified by the owning queue manager name, specified using the SMDS keyword, and by the application structure name, specified using the CFSTRUCT keyword.

Note: When defining SMDS data sets for a structure, you must have one for each queue manager.

The SMDS object is stored in an array (with one entry per queue manager in the group) which forms an extension of the corresponding CFSTRUCT object stored in Db2.

There is no command to DEFINE or DELETE the SMDS object because it is created or deleted as part of the CFSTRUCT object, but there is a command to ALTER it to change settings for an individual owning queue manager.

For further information on SMDS commands, see “SMDS related commands” on page 192

SMDSCONN information

It is possible for a shared message data set to be in a normal state, but for one or more queue managers to be unable to connect to it, for example because of a problem with a security definition or with direct access device connectivity. It is therefore necessary for each queue manager to keep track of connection

status, and availability information for each shared message data set, indicating for example whether it can currently connect to it, and if not why not.

The SMDSCONN information represents a queue manager connection to a shared message data set. As for the shared message data set itself, it is identified by the queue manager which owns the shared message data set (as specified on the SMDS keyword for the shared object itself) combined with the CFSTRUCT name.

There is no parameter to identify the connecting queue manager because commands addressed to a specific queue manager can only refer to SMDSCONN information for that same queue manager.

The SMDSCONN information entries are maintained in main storage in the owning queue manager, and are re-created when the queue manager is restarted. However, if a connection from an individual queue manager has been explicitly stopped, this information is also stored as a flag in a connection array in the corresponding CFSTRUCT or SMDS object, so that it persists across a queue manager restart.

Status and availability information

Status information indicates the state of a resource or connection (for example, whether it is not yet being used, is in normal use or is in need of recovery). It is usually described using the STATUS keyword. The possible values depend on the type of object.

Status information is normally updated automatically, for example when an error is detected while using the resource or connection. However, in some cases a command can also be used to update the status, to allow for cases when it is not possible for a queue manager to determine the correct status automatically.

Availability information indicates whether the resource or connection can be used, and is usually primarily determined by the status information. For the resource or connection types used in shared message data set support, three levels of availability are implemented:

Available
This means that the resource is available to be used normally. This does not necessarily mean that it is in use at present (which can be determined instead from the STATUS value). For a data set, if it requires restart processing, this allows the owning queue manager to open it, but other queue managers must wait until the data set is back in the ACTIVE state.

Unavailable because of error
This means that the resource has been made unavailable automatically because of an error and is not expected to be available again until some form of repair or recovery processing has been performed. However, attempts to make it available again are permitted without operator intervention. Such an attempt can also be triggered by a command to mark the resource as enabled, or a command which changes the status in such a way as to indicate that recovery processing has been completed.

The reason that the resource has been made unavailable is normally obvious from the related STATUS value, but in some cases there may be other reasons to make the resource unavailable, in which case a separate REASON value is provided to indicate the reason.

Unavailable because of operator command
This means that access to the resource has been explicitly disabled by a command. It can only be made available by using a command to enable it again.

SMDS availability

For the shared SMDS object, the availability is described by the ACCESS keyword, with the possible values ENABLED, SUSPENDED and DISABLED.

The availability can be updated using a **RESET SMDS** command for the relevant shared object from any queue manager in the group to set ACCESS(ENABLED) or ACCESS(DISABLED).

If the availability was previously ACCESS(SUSPENDED), changing it to ACCESS(ENABLED) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to ACCESS(SUSPENDED).

SMDSCONN availability

For a local SMDSCONN information entry, the availability is described by the AVAIL keyword, with the possible values NORMAL, ERROR or STOPPED. The availability can be updated using a **START SMDSCONN** or **STOP SMDSCONN** command addressed to a specific queue manager to enable or disable its connection.

If the availability was previously AVAIL(ERROR), changing it to AVAIL(NORMAL) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to AVAIL(ERROR).

Shared message data set shared status and availability

The availability of each shared message data set is managed within the group using shared status information, which can be displayed using the **DISPLAY CFSTATUS** command with TYPE(SMDS). This displays status information for each queue manager that has activated a data set for each structure. Each data set can be in one of the following states:

NOTFOUND
This means that the corresponding data set has not yet been activated. This status only appears when a specific queue manager is specified, as data sets which have not been activated are skipped when all queue managers are selected.

NEW
The data set is being opened and initialized for the first time, ready to be made active.

ACTIVE
This means that the data set is fully available and should be allocated and opened by all active queue managers for the structure.

FAILED
This means the data set is not available at all (except for recovery processing) and must be closed and deallocated by all queue managers.

INRECOVER
This means that media recovery (using RECOVER CFSTRUCT) is in progress for this data set.

RECOVERED
This indicates that a command has been issued to switch a failed data set back to the active state, but further restart processing is required which is not yet complete, so the data set can only be opened by the owning queue manager for restart processing.

EMPTY
The data set contains no messages. The data set is put into this state if it is closed normally by the owning queue manager, at a time when it does not contain any messages. It can also be put into EMPTY state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with TYPE PURGE or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to ACTIVE. As the previous data set contents are no longer required, a data set in this state can be replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

The command output includes the date and time at which recovery logging was enabled, if any, and the date and time at which the data set failed, if it is not currently active.

A shared message data set can be put into a FAILED state either by a **RESET SMDS** command or automatically when any of the following types of error are detected:

- The data set cannot be allocated or opened by the owning queue manager.
- Validation of the data set header fails after it has been successfully opened by any queue manager.
- A permanent I/O error occurs when the owning queue manager is reading or writing data.

- A permanent I/O error occurs when another queue manager is reading data from a data set which had successfully completed open processing and validation.

When a data set is in the FAILED or INRECOVER state, it not available for normal use, so if the availability state is ACCESS(ENABLED) it is changed to ACCESS(SUSPENDED).

If a data set has been put into the the FAILED state but no media recovery is required, for example because the data was still valid but the storage device was temporarily offline, then the **RESET SMDS** command can be used to request changing the status directly to the RECOVERED state.

When the data set enters the RECOVERED state, either on completion of recovery processing or as a result of the **RESET SMDS** command, then it is ready to be used again once restart processing has been completed. If it was in the ACCESS(SUSPENDED) state, it is automatically switched back to the ACCESS(ENABLED) state, which allows the owning queue manager to perform restart processing. When restart processing completes, the state is changed to ACTIVE and all other queue managers can then connect to the data set again.

Shared message data set connection status and availability

Each queue manager maintains local status and availability information for its connection to each shared message data set owned by itself and by other queue managers in the group. This information can be displayed using the **DISPLAY SMDSCONN** command.

If it is unable to access a shared message data set in the ACTIVE state which belongs to another queue manager it flags the connection as being unavailable from its own point of view.

If the error definitely indicates a problem with the data set itself, the queue manager also automatically changes the shared status to indicate that the data set is now in a FAILED state. However, if the error could be caused by an environmental problem, such as not being authorized to open the data set, the queue manager issues error messages and treats the data set as being unavailable, but it does not modify the shared data set status. If the environmental error turns out to be a problem with the data set anyway (for example it has been allocated on a device which cannot be accessed by some of the queue managers) then an operator can use the RESET SMDS command specifying STATUS(FAILED) to allow the data set to be recovered or repaired as necessary.

If a connection to a shared message data set could not be established but the data set appears to be valid, a new attempt to use it can be triggered by issuing a **START SMDSCONN** command for the owning queue manager.

If there is an operational need to terminate the connection between a specific queue manager and a data set temporarily, but the data set itself is not damaged, then the data set can be closed and deallocated using the **STOP SMDSCONN** command. If the data set is in use, the queue manager will close it normally (although any requests for data in that data set will be rejected with a return code). If it is the owned data set, the queue manager will save the space map during CLOSE processing, avoiding the need for restart processing.

If a data set needs to be taken out of service temporarily from all queue managers (for example to move it) but is not damaged, then it is best to use **STOP SMDSCONN** for the relevant data set with the option CMDSCOPE(*) to stop the queue managers using it first, as this will avoid the need for restart processing when the data set is brought back into service. In contrast, if the data set is marked as FAILED this tells queue managers that they must stop using it immediately, which means that the space map will not be saved and will need to be rebuilt by restart processing.

Access to any shared message data sets previously in the ACCESS(SUSPENDED) state will be retried if the queue manager is restarted.

Shared message data set recovery logging

Persistent shared messages are logged for media recovery purposes. This means that the messages can be recovered after any failure of coupling facility structures or shared message data sets, provided that the recovery logs are still intact. Persistent messages can also be re-created from the recovery logs at another site for disaster recovery purposes.

When the message data is written to a shared message data set, each block written to the data set is logged separately followed by the message entry (including the data map) as written to the coupling facility. The recovery process always recovers the coupling facility structure, but it does not need to recover individual shared message data sets except when the data set status is FAILED, or when the status is ACTIVE but the data set header record is no longer valid, indicating that the data set has been re-created. A data set is not selected for recovery if its status is ACTIVE and the data set header is still valid, nor if its status is EMPTY, indicating that no messages were stored in it at the time of the failure.

Shared message data set backups

When BACKUP CFSTRUCT is used to make a backup of the shared messages in an application structure, any data for persistent messages stored in shared message data sets is backed up at the same time, as for persistent shared messages previously stored in DB.

Shared message data set recovery

If a shared message data set is corrupted or lost, then it needs to be put into the FAILED state to stop the queue managers from using it until it has been repaired. This normally happens automatically, but can also be done using the **RESET SMDS** command specifying STATUS(FAILED).

If the shared message data set contained any persistent messages, these can be recovered using the RECOVER CFSTRUCT command. This command first restores any persistent message data for that shared message data set from the most recent BACKUP CFSTRUCT command, then applies all logged changes since that time. If no **BACKUP CFSTRUCT** command has been performed since the time that the data set was first activated, it is reset to empty then all changes since activation are applied.

If the CFSTRUCT contents and all of the shared message data sets are unavailable, for example in a disaster recovery situation, they can all be recovered in a single **RECOVER CFSTRUCT** command.

If a shared message data set is damaged but recovery was not active for the CFSTRUCT, or the log containing the latest BACKUP CFSTRUCT is unavailable or unusable, then the messages offloaded to that data set cannot be recovered. In this case, the **RECOVER CFSTRUCT** command with the parameter TYPE(PURGE) can be used to mark the shared message data set as empty and delete any messages from the structure which had data stored in that data set.

When the **RECOVER CFSTRUCT** command is issued, the shared message data set status is changed from FAILED to INRECOVER. If recovery completes successfully, the status is automatically changed to RECOVERED, otherwise it changes back to FAILED.

When the data set is changed to the RECOVERED state, this tells the owning queue manager that it can now try to open the data set and perform restart processing.

Shared message data set recovery and syncpoints

The shared message data set recovery process reapplies the changes for all complete log records up to the end of the log, regardless of syncpoints.

If changes were made within syncpoint, restart or recovery processing for the CFSTRUCT may result in backing out of uncommitted requests, so some of the recovered changes may not actually be used, but there is no harm in recovering them anyway.

It is also possible that an uncommitted MQPUT message may have been written to the structure but the corresponding data may not have been written to the data set or the log (as I/O completion is only forced at the start of syncpoint processing). This is harmless because restart processing will back out the message entry in the structure, so the fact that it refers to unrecovered data does not matter.

Shared message data set restart processing

If a queue manager connection to a CFSTRUCT terminates normally, the queue manager writes out the free block space map for each shared message data set to a checkpoint area within the data set, just before the data set is closed. The space map can then be read in again at connection restart time,

provided that neither the CFSTRUCT nor the shared message data set require any recovery processing before the next restart.

However, if a queue manager terminates abnormally, or the structure or data set require any recovery processing, then additional processing is required to rebuild the space map dynamically when the queue manager connection to the structure is restarted.

Provided that the data set itself did not need to be recovered, queue manager restart simply scans the current contents of the structure to locate references to message data owned by the current queue manager, and marks the relevant data blocks as owned in the space map. Other queue managers can continue to use the structure and read the data owned by the restarting queue manager while the space map is being rebuilt.

Shared message data set restart after recovery

If a shared message data set had to be recovered from a backup, then all nonpersistent messages stored in the data set will have been lost, and if the data set was recovered using TYPE(PURGE) then all messages stored in the data set will have been lost. Until recovery has completed, the data set will be marked as FAILED or INRECOVER so any attempt to read one of the affected messages from another queue manager returns an error code indicating that the data set is temporarily unavailable.

When the data set has been recovered, the status is changed to RECOVERED, which allows the owning queue manager to open it for restart processing, but the data set remains unavailable to other queue managers. Queue manager restart scans the structure to rebuild the space map for any remaining messages. The scan also checks for messages for which the data has been lost, and deletes them from the structure (or if necessary flags them as lost, to be deleted later).

The data set status is automatically changed from RECOVERED to ACTIVE when this restart scan completes, at which point other queue managers can start using it again.

Shared message data set usage information

The DISPLAY USAGE command now also shows information about shared message data set space and buffer pool usage for any currently open shared message data sets. This information is displayed if either the new option TYPE(SMDS) or the existing option TYPE(ALL) is specified.

Shared message data performance and capacity considerations

Monitoring data set usage

The current percentage full of each owned shared message data set can be displayed by the **DISPLAY USAGE** command with the option **TYPE(SMDS)**.

The queue manager will normally automatically expand a shared message data set when it reaches 90% full, provided that the option **DSEXPAND(YES)** is in effect for the SMDS definition. This applies when either the SMDS option is set to **DSEXPAND(YES)** or the SMDS option is set to **DSEXPAND(DEFAULT)** and the CFSTRUCT default option is set to **DSEXPAND(YES)**.

If the expansion attempt fails because no secondary allocation size was specified when the data set was created (giving message IEC070I with reason code 203) the queue manager repeats the expansion request using an override secondary allocation of approximately 20% of the current size.

When a data set is expanded, the new data set extents are formatted as part of the expansion processing, which can take tens of seconds, or even minutes for very large extents. The new space becomes available for use after formatting is complete and the catalog has been updated to show the new high used control interval.

If new messages are being created very rapidly, it is possible for the existing data set to become full before expansion processing completes. In this case, any request which could not allocate space is temporarily suspended until the expansion attempt completes and the new space becomes available for use. If the expansion was successful the request is retried automatically.

If an expansion attempt fails, because of a lack of available space or because the maximum extents have already been reached, a message is issued giving the reason for the failure, then the override option for the affected SMDS is automatically altered to **DSEXPAND(NO)** to prevent further expansion attempts. In this case, there is a risk that the data set may become full, in which case further action may be needed as described in Data set becomes full.

Monitoring application structure usage

The usage level of an application structure can be displayed using the MVS **DISPLAY XCF,STRUCTURE** command specifying the full name of the application structure (including the queue sharing group prefix). The IXC360I response message shows current usage of elements and entries.

When the structure usage exceeds the **FULLTHRESHOLD** value specified in the CFRM policy, the system issues message IXC585E and may perform automatic **ALTER** actions if specified, which may either alter the entry to element ratio or increase the structure size.

Optimising buffer pool sizes

Each buffer in a shared buffer pool is used to read or write a contiguous range of pages for one message of up to the logical block size. If the message spills over into further blocks, each range of pages in a separate block requires a separate buffer.

Buffers containing message data after a write or read operation are retained in storage and reused using a least-recently-used (LRU) cache scheme so that a request to read the same data again shortly afterwards will not need to go to disk. This provides a significant optimization when shared messages are written and then read back soon afterwards by applications running on the same system. If messages owned by another queue manager are browsed for selection purposes then retrieved, this also avoids the need to reread the message from disk.

This means that the number of buffers required for each application structure is one for each concurrent API request which reads or writes large messages for that application structure plus some number of additional buffers which will be used to save recently accessed data in order to optimize subsequent read accesses.

For shared buffer pools, if there are insufficient buffers, API requests will simply wait if a buffer is not immediately available. However, this situation should be avoided as it can cause significantly degraded performance.

The statistics from the **DISPLAY USAGE** command for shared buffer pools show whether there have been any buffer waits within the current statistics interval, and also shows the lowest number of free buffers (or a negative value indicating the maximum number of threads which waited for a buffer at any time), the number of buffers which have saved data, and the percentage of the times that a buffer request has successfully found saved data on the LRU chain ("LRU hits") instead of having to read it ("LRU misses") [1] .

- If there have been any waits, the number of buffers should be increased.
- If there are many unused buffers, the number of buffers may be reduced to make more storage available in the region for other purposes.
- If there are many buffers containing saved data but the proportion of reads which were hits against that saved data is very small, the number of buffers may be reduced if the storage could be better used for other purposes. The number of buffers should not however be reduced by more than the lowest number of free buffers, as that could trigger waits, and it should preferably be high enough that the lowest free buffer count is normally well above zero.

Deleting shared message data sets

The DELETE CFSTRUCT command (which is only allowed when all shared queues in the structure are empty and closed) does not delete the shared message data sets themselves, but they can be deleted in

[1] (Hits / (Hits+Misses))* 100

the usual way after this command has completed. If the same data set is to be reused as a shared message data set, it must be reformatted first to reset it to the empty state.

Exception situations for shared message data sets

There are a number of exception situations which can occur during normal use, even when no software or hardware error is present.

Data set becomes full

If a data set becomes full but cannot be expanded, or the expansion attempt fails, applications using the corresponding queue manager to write large messages to the corresponding application structure will receive error 2192, `MQRC_STORAGE_MEDIUM_FULL` (also known as `MQRC_PAGESET_FULL`).

A data set could become full because of a failure in the application which is supposed to process the data, causing a large backlog of messages to accumulate. If so, expanding the data set any further will only be a temporary solution, and it is important to get the processing application going again as soon as possible.

If more space can be made available the **ALTER SMDS** command can then be used to set **DSEXPAND(YES)** or **DSEXPAND(DEFAULT)** (assuming that YES has been set or assumed as the **DSEXPAND** default for the CFSTRUCT definition) to trigger a retry. If the reason for the failure was however that maximum extents had been reached, the new expansion attempt will be rejected with a message and **DSEXPAND(NO)** will be set again. In this case, the only way to expand it any further is to reallocate it, which involves making it temporarily unavailable, as described next.

Data set needs to be moved or reallocated

If a data set needs to be moved or expanded but is otherwise in normal use, it can be taken out of use temporarily to allow it to be moved or reallocated. Any API request which attempts to use the data set while it is unavailable will receive the reason code `MQRC_DATA_SET_NOT_AVAILABLE.`

1. Use the **RESET SMDS** command to mark the data set as **ACCESS(DISABLED)**. This will cause it to be closed normally and deallocated by all currently connected queue managers.
2. Move or reallocate the data set as necessary, copying the old contents to the newly allocated data set, for example using the Access Method Services (AMS) **REPRO** command.

   Do not attempt to preformat the new data set before copying the old data into it, as this would result in the copied data being appended to the end of the formatted data set.
3. Use the **RESET SMDS** command to mark the data set as **ACCESS(ENABLED)** again, to bring it back into use.

   If the old contents are smaller than the size of the new data set, the rest of the space will be preformatted automatically when the new data set is opened.

   If the old contents were larger than the size of the new data set then the queue manager has to scan the messages in the coupling facility structure and rebuild the space map to ensure that none of the active data has been lost. If any reference is found to a data block which is outside the new extents, the data set is marked as **STATUS(FAILED)** and must be repaired by replacing the data set with one of the correct size and either copying the old data set into it again or using **RECOVER CFSTRUCT** to recover any persistent messages.

Coupling facility structure is low on space

If the coupling facility structure is running out of space, causing message IXC585E, it is worth checking whether the offload rules have been set to ensure that the maximum amount of data is being offloaded in this case. If not, the offload rules can be modified using the **ALTER CFSTRUCT** command.

Error situations for shared message data sets

There are a number of problems to be aware of, which can only be caused by errors and not occur in normal operational situations.

Owned data set cannot be opened

If the queue manager which owns a shared message data set cannot allocate it or open it, or the data set attributes are not supported, the queue manager sets an appropriate **SMDSCONN** status value of **ALLOCFAIL** or **OPENFAIL** and sets the **SMDSCONN** availability to **AVAIL(ERROR)**. It also sets the SMDS availability to **ACCESS(SUSPENDED)**. When the error has been corrected, use the **RESET SMDS** command to set **ACCESS(ENABLED)** to trigger a retry, or issue the **START SMDSCONN** command to the owning queue manager.

Read-only data set cannot be opened

If a queue manager cannot allocate or open a shared message data set owned by another queue manager and marked as **STATUS(ACTIVE)**, it assumes that this is probably due to a specific problem with its connection to the data set (represented by the **SMDSCONN** object) rather than a problem with the data set itself.

It marks the **SMDSCONN** as **STATUS(ALLOCFAIL)** or **STATUS(OPENFAIL)** as appropriate and marks the **SMDSCONN** availability as AVAIL(ERROR) to prevent further attempts to use it.

If the problem can been corrected without affecting the status of the data set itself, use the **START SMDSCONN** command to trigger a retry.

If the problem turns out to be a problem with the data set itself, then the **RESET SMDS** command can be used to mark the data set as **STATUS(FAILED)** until it it has been recovered. When the data set has been recovered, the action of changing the status back to **STATUS(ACTIVE)** will cause other queue managers to be notified. If the **SMDSCONN** is marked as **AVAIL(ERROR)**, it will automatically be changed back to **AVAIL(NORMAL)** to trigger a new attempt to open the data set.

Data set header is corrupt

If the data set was successfully opened but the format of the header information is incorrect, the queue manager closes and deallocates the data set and sets the status set to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**. This allows **RECOVER CFSTRUCT** to be used to recover the contents.

If the error arose because the data set contained residual data from another use and had not been subsequently preformatted, then preformat the data set and use the **RESET SMDS** command to change the status to **STATUS(RECOVERED)**.

Otherwise, the data set must be recovered.

Data set is unexpectedly empty

If the queue manager opens a data set which is marked as **STATUS(ACTIVE)** but finds that it is uninitialized or newly preformatted but otherwise valid, the queue manager closes and deallocates the shared message data set then sets the status to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**.

Data set has permanent I/O errors

If a data set has permanent I/O errors after successful **OPEN** processing, it probably needs recovery. The queue manager will mark the data set as **STATUS(FAILED)** so that all currently connected queue managers will close and deallocate it.

Data set has recoverable I/O errors

If there are hardware problems with the data set, it is possible that this might result in recoverable I/O errors which are not reflected back to the queue manager but which cause significant performance degradation, and also indicate a risk of permanent I/O errors in the near future.

In this case, the data set may be taken off line for recovery by using the **RESET SMDS** command to mark it as **STATUS(FAILED)**. This will cause it to be closed and deallocated by all queue managers, so for example it could be moved to a new volume before being made available again.

When a data set is made unavailable in this way, the space map is not saved so the queue manager connection restart processing will need to scan the coupling facility structure to locate messages in the data set and rebuild the space map before the data set can be made available again. As an alternative, if the shared message data set is still usable, it set can be made unavailable more gently by using the **RESET SMDS** command to mark the data set **ACCESS(DISABLED)** until it is ready to be made available again.

Data set contents are incorrect

The queue manager cannot detect directly that a data set contains incorrect data or is not up to date, for example because a volume including that data set had to be restored from backups. However, it performs integrity checks which make it very unlikely that any such errors could result in incorrect message data being seen by application programs.

For integrity checking purposes, each message block in the data set is prefixed with a copy of the corresponding coupling facility entry ID, including a unique time stamp, which is checked whenever the message block is read, before the message data is passed to the user program. If the message block prefix does not match the entry ID (and the coupling facility entry was not deleted in the mean time) the message block is assumed to be damaged and unusable.

If the damaged message was persistent, the data set is marked as **STATUS(FAILED)** and the structure contents must be recovered using the **RECOVER CFSTRUCT** command. If the damaged message was non-persistent, there is no way to recover it, so a diagnostic message is issued and the corresponding coupling facility message entry is deleted.

If no saved space map is available when the data set is opened, it is rebuilt by scanning the coupling facility structure for references to data in the data set. During this scan, the queue manager performs a number of actions:

1. The queue manager determines the location of the most recent message (if any) currently remaining in the data set.
2. The queue manager then reads that message from the data set to ensure that the block prefix matches the message entry id

These actions ensure that the queue manager detects any case where the data set is down-level, and marks the data set as FAILED. This check does however tolerate the case where the data set was restored from a previous copy and either no new messages had been added since then or all messages added since that copy had been subsequently read and deleted.

To protect against down-level data in the case where the data set was closed normally, the queue manager performs a number of actions:

1. The queue manager saves a copy of the space map time stamp in the SMDS object within Db2 when the data set is closed normally.
2. The queue manager then checks the space map time stamp is the same, when the data set is opened again

If the time stamp does not match, this suggests that a down-level copy of the data set might have been used, so the queue manager ignores the existing space map and rebuilds it, which will succeed only if no message data was actually lost.

Note: These integrity checks do not guarantee to detect a down-level or damaged data set in all theoretically possible cases. For example, they will not detect a case where the start of a message block is valid but the rest of the data has been partly overwritten.

Recovery scenarios for shared message data sets

This section described shared message data set recovery scenarios.

Data set recovery where no data was lost

In some cases, the correct contents of a failed data set can be restored without needing actual recovery. One example is where a data set contains residual data from a previous use and has not been preformatted again, which can be fixed by preformatting it. Another case is when a data set has been moved, but there was an error in the process of copying the data across, which can be fixed by copying the data again correctly.

In such cases, the corrected data set can be made available again by using the **RESET SMDS** command to set **STATUS(RECOVERED)**. If the availability is currently **ACCESS(SUSPENDED)** this will automatically set it back to **ACCESS(ENABLED)**.

When the owning queue manager is notified that the data set has been recovered, it scans the structure contents to reconstruct the space map, then changes the status to **STATUS(ACTIVE)**. The other queue managers can then start reading the data set again.

Data set recovery with TYPE(NORMAL)

If the contents of a data set have been lost, but the application structure was defined with **RECOVER(YES)** and the appropriate recovery logs are available, the **RECOVER CFSTRUCT** command can be used to recover any persistent messages stored in the structure including persistent message data offloaded to shared message data sets. This command restores the current state using information logged by the **BACKUP CFSTRUCT** command plus all logged changes to persistent messages since the backup time.

The **RECOVER CFSTRUCT** command always recovers all persistent messages in the coupling facility structure together with offloaded message data stored in Db2. For offloaded data stored in shared message data sets, each data set is only selected for recovery processing if it is already marked as **STATUS(FAILED)** or if it is found to be unexpectedly empty or otherwise invalid when opened by recovery processing. Any shared message data set which is marked as active and which passes the validation checks does not need to be recovered, as the existing message data is already correct, but the header is updated to indicate that any saved space map will need to be rebuilt after recovery.

Recovery processing is only possible when the structure has been marked as failed, as the complete contents of the structure need to be reconstructed by recovery processing. However, if at least one shared message data set has been marked as failed the **RECOVER CFSTRUCT** command will automatically mark the structure as failed if necessary to allow recovery processing to proceed.

Recovery may be performed from any queue manager in the queue sharing group, provided that it has been given write access to the relevant data sets.

Only persistent messages are backed up and logged, so normal recovery processing will restore all persistent messages, but will cause any non-persistent messages in the structure to be lost.

When recovery has completed, any data set which was selected for recovery is automatically changed to **STATUS(RECOVERED)**, and if the availability was **ACCESS(SUSPENDED)** it is changed to **ACCESS(ENABLED)**. The queue manager rebuilds the space map for each data set by scanning the messages in the coupling facility, then marks the data set as **STATUS(ACTIVE)** so that it can be used again.

Data set recovery with TYPE(PURGE)

For a recoverable structure, if the data set contents have been lost, but recovery is not possible for some reason, for example because recovery logs are not available or recovery would take too long, the **RECOVER CFSTRUCT** command can be used with **TYPE(PURGE)** to get the structure back to a usable state. This resets the structure to the empty state and marks all of the associated data sets as **STATUS(EMPTY)**.

Deleting the application structure

If a non-recoverable application structure is deleted using the MVS **SETXCF FORCE** command, or as a result of structure failure, then the next time the structure is connected, message CSQE028I is issued to say that the structure has been reset and all existing messages have been discarded, and any

existing data sets are automatically reset to **STATUS(EMPTY)** as well. This action makes a non-recoverable structure usable again after loss of data either in the structure or in any of the associated data sets.

If a recoverable application structure is deleted, it will be treated in the same way as if the structure had failed.

Data set recovery fails

If **RECOVER CFSTRUCT** cannot complete for some reason, for example because a log data set is no longer available, or because the queue manager terminated while recovery was in progress, then any data set for which recovery was at least started will be marked in the header to show that partial recovery has been attempted, and the data set will be left in the **STATUS(FAILED)** state.

In this case, the options are to repeat the original recovery request or to recover with **TYPE(PURGE)** instead, discarding the existing data.

If an attempt is made to mark the data set as **STATUS(RECOVERED)** without actually recovering it, then the next time it is opened the queue manager will see that the header indicates incomplete recovery and mark it as **STATUS(FAILED)** again.

Off site disaster recovery

For off site disaster recovery, persistent shared messages can be re-created using only the logs and the Db2 shared objects containing the CFSTRUCT definitions and associated SMDS status information.

After setting up the Db2 tables containing the definitions, the application structure and the shared message data sets can be set up as empty. When a queue manager connects to them and finds that they are unexpectedly empty, it will mark them as failed, after which a single **RECOVER CFSTRUCT** command can be used to recover all persistent messages for all affected structures.

z/OS *SMDS related commands*

This topic describes and provides access to the commands relating to shared message data sets.

Display and alter the **CFSTRUCT** options relating to large message offload (**OFFLOAD** and offload rules) and shared message data sets (**DSGROUP, DSBLOCK, DSBUFS, DSEXPAND)**:

- DISPLAY CFSTRUCT
- DEFINE CFSTRUCT
- ALTER CFSTRUCT
- DELETE CFSTRUCT

Display **CFSTRUCT** status relating to large message offload **(OFFLDUSE)**:

- DISPLAY CFSTATUS

Display and alter override data set options (**DSEXPAND** and **DSBUFS**) for individual queue managers:

- DISPLAY SMDS
- ALTER SMDS

Display or modify the status and availability of the data sets within the queue sharing group:

- DISPLAY CFSTATUS TYPE(SMDS)
- RESET SMDS

Display SMDS data set space usage and buffer usage information for a queue manager:

- DISPLAY USAGE TYPE(SMDS)

Display or modify the status and availability of the connections (**SMDSCONN**) to the data sets from an individual queue manager:

- DISPLAY SMDSCONN

- START SMDSCONN
- STOP SMDSCONN

Backup and recover shared messages, including large message data in SMDS when necessary:

- BACKUP CFSTRUCT
- RECOVER CFSTRUCT

z/OS *Advantages of using shared queues*

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

The advantages of shared queues

The shared queue architecture, where cloned servers pull work from a single shared queue, has some useful properties:

- It is scalable, by adding new instances of the server application, or even adding a new z/OS image with a queue manager (in the queue sharing group) and a copy of the application.
- It is highly available.
- It naturally performs *pull* workload balancing, based on the available processing capacity of each queue manager in the queue sharing group.

Using shared queues for high availability

The following examples illustrate how you can use a shared queue to increase application availability.

Consider an IBM MQ scenario where client applications running in the network want to make requests of server applications running on z/OS. The client application constructs a request message and places it on a request queue. The client then waits for a reply from the server, sent to the reply-to queue named in the message descriptor of the request message.

IBM MQ manages the transportation of the request message from the client machine to the server's input queue on z/OS and of the response from the server back to the client. By defining the server's input queue as a shared queue, any messages put to the queue can be retrieved on any queue manager in the queue sharing group. This means that you can configure a queue manager on each z/OS image in the sysplex and, by connecting them all to the same queue sharing group, any one of them can access messages on the server's input queue.

Messages on the input queue of the server are still available, even if one of the queue managers terminates abnormally or you have to stop it for administrative reasons. You can take an entire z/OS image offline and the messages will still be available.

To take advantage of this availability of messages on a shared queue, run an instance of the server application on each z/OS image in the sysplex to provide higher server application capacity and availability, as shown in Figure 60 on page 194.

One instance of the server application retrieves a request message from the shared queue and, based on the content, performs its processing, producing a result that is sent back to the client as an IBM MQ message. The response message is destined for the reply-to queue and reply-to queue manager named in the message descriptor of the request message.

There are a number of options that you can use to configure the return path. For more information about these options, see “Distributed queuing and queue sharing groups” on page 213.

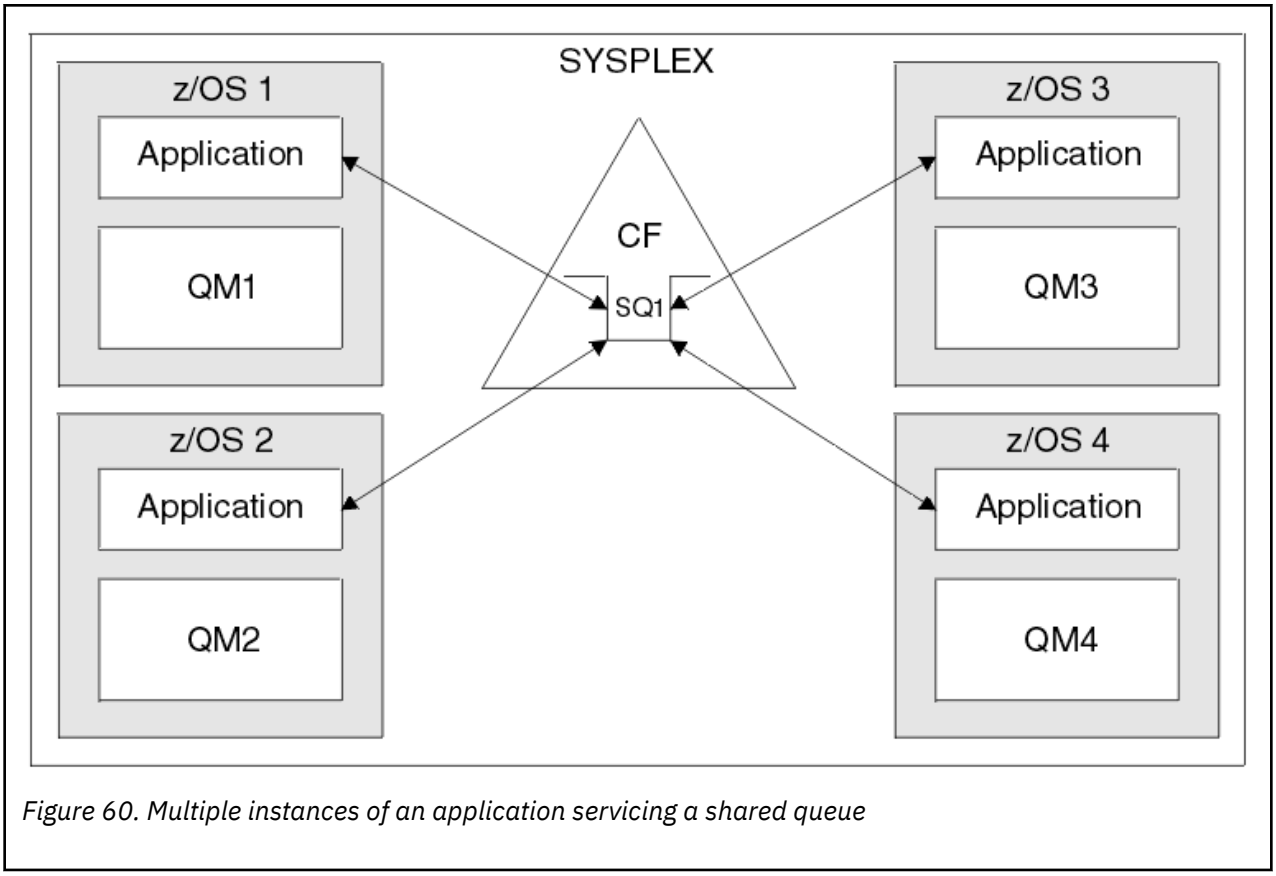


Figure 60. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally and completes units of work for that queue manager that are still pending, where possible. This feature is known as *peer recovery.*

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in sync point, but has not yet put the response message or committed the unit of work. Another queue manager in the queue sharing group detects the failure, and backs out the in-flight units of work being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If IBM MQ cannot resolve a unit of work automatically, you can resolve the shared portion manually to enable another queue manager in the queue sharing group to continue processing that work.

z/OS *Use of storage class memory with shared queues*

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

The z13, zEC12, and zBC12 machines allow the installation of Flash Express cards. These cards contain flash solid-state drives (SSD). After installation, flash storage from the cards can be allocated to one or more LPARs where it is typically known as SCM.

SCM sits between real storage and direct access storage device (DASD) in terms of both I/O latency and cost. Because SCM has no moving parts, it exhibits much lower I/O latencies than DASD.

SCM is also much cheaper than real storage. As a result, a large amount of storage can be installed for a relatively low cost; for example, a pair of Flash Express cards contains 1424 GB of usable storage.

These characteristics mean that SCM is useful when a large amount of data must be taken from real storage in a short period of time, because that data can be written to SCM much quicker than it can be written to DASD. This specific point can be very useful when using coupling facility (CF) list structures containing IBM MQ shared queues.

Why list structures fill up

When a CF structure is defined, it is configured with a SIZE attribute that describes the maximum size of the structure. Because CF structures are always permanently resident in real storage, the sum of the SIZE attributes of the structures that are defined on a CF should be less than the amount of the real storage allocated to the CF.

As a result, there is a constant pressure to keep the SIZE value of any given structure to the minimum value possible, so that more structures can fit into the CF. However, ensuring structures are large enough to achieve their purpose can result in a conflicting pressure, because making a structure too small means that it might fill up, disrupting the applications or subsystems making use of it.

There is a strong need to accurately size a structure based on its expected usage. However, this task is difficult to do because workloads can change over time and accounting for their fluctuations is not easy.

IBM MQ shared queues use CF list structures to store messages. IBM MQ calls CF structures, which contain messages and application structures.

Application structures are referenced using the information stored in IBM MQ CFSTRUCT objects. When a message that is smaller than 63 KB is put on a shared queue, the message is stored entirely in an application structure as a single list entry, and zero or more list elements.

Because IBM MQ shared queues use list structures, the pressures described also affect shared queues. In this case, the maximum number of messages that can be stored on a shared queue is a function of the:

- Size of the messages on the queue
- Maximum size of the structure
- Number of entries and elements available in the structure

Because up to 512 shared queues can use the same structure, and effectively compete for entries and elements, this complicates matters even further

IBM MQ queues are used for the transfer of data between applications, so a common situation is an application putting messages to a queue, when the partner application, which should be getting those messages, is not running.

When this happens the number of messages on the queue increase over time until one or more of the following situations occur:

- The putting application stops putting messages.
- The getting application starts getting messages.
- Existing messages on the queue start expiring, and are removed from the queue.
- The queue reaches its maximum depth in which case an MQRC_Q_FULL reason code is returned to the putting application.
- The structure containing the shared queue reaches its maximum size, or the CF containing the structure runs out of available storage. In either case, an MQRC_STORAGE_MEDIUM_FULL reason code is returned to the putting application.

In the last three cases the queue is full. At this point the putting application has a problem because there is nowhere for its messages to go. The putting application typically solves this problem by using one or more of the following solutions:

- Repeatedly retry putting the message, optionally with a delay between retries.
- Put the messages somewhere else, such as a database or a file. The messages can be accessed later and put to the queue as normal.
- Discard the message if it is nonpersistent.

However, for some classes of applications, for example those with a large volume of incoming messages, or no access to a file system, these solutions are not practical. There is a real need to ensure that queues never, or are extremely unlikely to, fill up in the first place and this is especially pertinent to shared queues.

SMDS and offload rules

The offload rules introduced in IBM WebSphere MQ 7.1 provide a way of reducing the likelihood of an application structure filling up.

Each application structure has three rules associated with it, specified using three pairs of keywords:

- OFFLD1SZ and OFFLD1TH
- OFFLD2SZ and OFFLD2TH
- OFFLD3SZ and OFFLD3TH

Each rule specifies the conditions that must be met for message data to be offloaded to the storage mechanism that is associated with the application structure. Two types of storage mechanisms are currently available:

- Db2
- Group of Virtual Storage Access Method (VSAM) linear data sets, which IBM MQ calls a shared message data set (SMDS).

The following example shows the MQSC command to create an application structure named LIST1, using the DEFINE CFSTRUCT command.

This structure has the default offload rules in place, and uses SMDS as the offload mechanism. This means that when the structure is 70% full (OFFLD1TH), all messages that are 32 KB or larger (OFFLD1SZ) are offloaded to SMDS.

Similarly, when the structure is 80% full (OFFLD2TH) all messages that are 4 KB or larger (OFFLD2SZ) are offloaded. When the structure is 90% full (OFFLD3TH) all messages (OFFLD3SZ) are offloaded.

```
DEFINE CFSTRUCT(LIST1)
CFLEVEL(5)
OFFLOAD(SMDS)
OFFLD1SZ(32K) OFFLD1TH(70)
OFFLD2SZ(4K) OFFLD2TH(80)
OFFLD3SZ(0K) OFFLD3TH(90)
```

An offloaded message is stored in the offload medium, and a pointer to the message is stored in the structure. While the offload rules reduce the chance of the structure filling up, by putting less message data in the structure as it runs out of storage, some data is still written to the structure for each message. That is, the pointer to the offloaded message.

Additionally, the offload rules come with a performance cost. Writing a message to a structure is relatively quick and is largely dominated by the time spent to send the request for the write to the CF. The actual writing to the structure is fast, happening at real storage speeds.

Writing a message to SMDS is much slower because it includes writing to the structure for the message pointer, and writing the message data to SMDS. This second write operation is done at DASD speed and has the potential to add latency. If Db2 is used as the offload mechanism the performance cost is much greater.

z/OS *How storage class memory works with IBM MQ for z/OS*

An overview of the use of storage class memory (SCM) with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

A coupling facility (CF) that is at CFLEVEL 19, or greater, can have SCM allocated to it. The structures defined in that CF can then be configured to make use of SCM to reduce the chances of the structures filling up (known as a structure full condition). When a structure configured to make use of SCM fills up past a system-determined point, the CF starts moving data from the structure into SCM, which frees space in the structure for new data.

Note: Because SCM itself can fill up, allocating SCM to a structure reduces only the likelihood of a structure full condition, but does not entirely remove the chance of one occurring.

A structure is configured to use SCM by specifying both the **SCMALGORITHM** and **SCMMAXSIZE** keywords in the coupling facility resource manager (CFRM) policy, containing the definition of that structure.

Note that after these keywords are specified, and the CFRM policy is applied, the structure must be rebuilt, or deallocated so that they can take effect.

SCMALGORITHM keyword

Because the input/output speed of SCM is slower than that of real storage, the CF uses an algorithm that is tailored to the expected use of the structure in order to reduce the impact of writing to, or reading from, SCM.

The algorithm is configured by the **SCMALGORITHM** keyword in the CFRM policy for the structure, using the *KEYPRIORITY1* value. Note that you should use the *KEYPRIORITY1* value only with list structures used by IBM MQ shared queues.

The *KEYPRIORITY1* algorithm works by assuming that most applications will get messages from a shared queue in priority order; that is, when an application gets a message, it gets the oldest message with the highest priority.

When a structure starts to fill past the system-defined threshold of 90%, the CF starts asynchronously migrating messages that are least likely to be got next. These are messages with lower priorities that were more recently put on the queue.

This asynchronous migration of messages from the structure into SCM is known as "pre-staging".

Pre-staging reduces the performance cost of using SCM because it reduces the chance of an application being blocked during the occurrence of synchronous input/output to SCM.

In addition to pre-staging, the *KEYPRIORITY1* algorithm also asynchronously brings back messages from SCM and into the structure when sufficient free space is available. For the *KEYPRIORITY1* algorithm, this means that when the structure is less than or equal to 70% full.

The act of bringing messages from SCM into the structure is known as "pre-fetching".

Pre-fetching reduces the likelihood of an application trying to get a message that has been pre-staged to SCM and having to wait while the CF synchronously brings back the message into the structure.

SCMMAXSIZE keyword

The **SCMMAXSIZE** keyword defines the maximum amount of SCM that can be used by a structure. Because SCM is allocated to the structure by the CF when it is required, it is possible to specify a **SCMMAXSIZE** that is greater than the total amount of free SCM available. This is known as "over-committing".

Important: Never over-commit SCM. If you do, the applications that are relying on it will not obtain the behavior that they expect. For example, IBM MQ applications using shared queues might get unexpected MQRC_STORAGE_MEDIUM_FULL reason codes.

The CF uses various data structures to track its use of SCM. These data structures reside in the real storage that is allocated to the CF and, as a result, reduce the amount of real storage that can be used by structures. The storage used by these data structures is known as "augmented space".

When a structure is configured with SCM, a small amount of real storage is allocated from the CF to the structure known as fixed augmented space. This is allocated even if the structure never actually uses any SCM. As data from the structure is stored into SCM, extra dynamic augmented space will be allocated from the spare real storage in the CF.

When the data is removed from SCM, the dynamic augmented space is returned to the CF. Augmented space, either fixed or dynamic, is never taken from the real storage that is allocated to a structure.

In addition to augmented storage, when a structure is configured to use SCM, the amount of control storage used by that structure increases. This means that a list structure configured with SCM can contain fewer entries and elements than a structure of the same size without SCM being configured.

To understand the impact of SCM on new or existing structures, use the CFSizer tool.

A final important point to note is that after data is moved from the structure into SCM, and dynamic augmented space has been used, the structure cannot be altered either manually or automatically.

That is, the amount of storage allocated to the structure cannot be increased or decreased, the entry-to-element ratio that is used by the structure cannot be changed, and so on. To make the structure alterable again, the structure must not have any data stored in SCM and must not be making use of dynamic augmented storage.

z/OS *Why use SCM*

Emergency storage and improved performance are two use cases for using SCM with IBM MQ for z/OS.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This section introduces the theory behind the two possible scenarios. For further details on how you set up the scenarios, see:

- “Emergency storage - basic configuration” on page 201
- “Improved performance - basic configuration” on page 207

Important: The use of SCM with CF structures is not dependent on any specific version of IBM MQ. However the emergency storage scenario works only with IBM WebSphere MQ 7.1 and later, because it requires SMDS and the offload rules.

Emergency storage

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

Overview

A single shared queue is configured on an application structure. The putting application puts messages onto the shared queue; the getting application gets messages from the shared queue.

During normal running, the queue depth is expected to be close to zero, but a business requirement indicates that the system must be able to tolerate a two-hour outage of the getting application. This means that the shared queue must be able to contain two hours of messages from the putting application.

Currently, this process is achieved by using the default offload rules, and SMDS, so that the size of the structure is minimized, while reducing the performance cost that is associated with offloading.

The rate of messages being sent to the shared queue is expected to double in the short to medium term. Although the requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure.

Because the CF, which contains the application structure, resides on a zEC12 machine, the possibility exists to associate sufficient SCM with the structure to store enough messages so that a two-hour outage can be tolerated.

Consider what happens over a period of time:

1. Initially, the system is in its steady state. Both the putting and getting application are running normally and the queue depth is close to, or at, zero. The result is that the application structure is largely empty.
2. At a certain time, the getting application suffers an unexpected failure and stops. The putting application continues to put messages to the queue and the application structure starts to fill up.
3. After the structure reaches 70% full, the conditions of the first offload rule are met and all messages with a size greater than or equal to 32 KB are offloaded to SMDS.

   See “SMDS and offload rules” on page 196 for an overview of the offload rules.
4. As messages continue to be put to the shared queue, the structure continues to fill (either because of the message data being stored in the structure, or as a result of the pointers to the offloaded messages being stored in the structure).

   When the structure reaches 80% full, the second offload rule starts to apply and messages that are 4 KB or greater are offloaded to SMDS.
5. When the structure is past 90% full, all messages are offloaded to SMDS and only the message pointers are being placed in the structure.

   About this time, the pre-staging algorithm starts to run, and begins moving data from the structure into SCM. Assuming all messages on the queue are the same priority, the newest messages are pre-staged.

   Because all messages are now being offloaded to SMDS, the data being moved into SCM is not actual message data, but instead the pointers to the messages on SMDS.

   As a result, the number of messages that can be stored on the combination of the structure, and the SCM and SMDS associated with the structure, is very large.

   Performance: During this stage of the outage, the putting application can suffer a degree of performance degradation because of having to write to SMDS. In this case, the use of SCM should not be a limiting factor on the putting application in terms of performance. SCM provides extra space to prevent the structure filling up.
6. Eventually the getting application is available again and the outage is over.

   However SCM is still being used by the structure. The getting application starts reading messages off the queue, getting the oldest, highest priority messages first.

   Because these messages were written before the structure started to fill up, they come out entirely from the real storage portion of the structure.
7. As the structure starts to empty, it goes below the threshold at which pre-staging is active, and so pre-staging stops.
8. The structure usage reduces below the point at which the offload rules take effect, so messages are no longer offloaded to SMDS unless they are more than 63 KB.

   At about this time, the pre-fetch algorithm starts moving data from SCM in to the structure. Because the getting application gets messages from the queue in the order expected by the SCM algorithms, messages are brought in before the getting application needs them.

The result is that the getting application never needs to wait for messages to be brought in synchronously from SCM.

9. As the getting application continues to move down the queue, it starts retrieving messages that were offloaded to SMDS.
10. Finally, the system is in a steady state again. No messages are stored in SCM or SMDS, and the queue depth is close to zero.

Improved performance

This scenario describes using SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

Description

For this scenario, a putting and getting application communicate through a shared queue which is stored in application structure.

The putting application tends to run in bursts, when it puts a large number of messages in a short amount of time. Then, in an extended period of time, it produces no messages at all.

The getting application sequentially processes each message, and performs complex processing on each one. As a result, most of the time the queue depth is zero, except for when the putting application starts to run, where the queue depth starts increasing as messages are being put faster than they are being got.

The queue depth increases until the putting application stops, and the getting application has enough time to process all the messages on the queue.

Notes:

1. In this scenario, the key factor is performance. The messages being sent to the queue are always less than 63 KB and so never need to be offloaded to SMDS.
2. The application structure has been sized so that it is large enough to contain all of the messages that will be placed on it by the putting application in a single "burst."
3. The offload rules must all be disabled so that, even when the structure starts to fill, the messages are not offloaded to SMDS. This is because the performance costs that are associated with writing messages to, and reading messages from, SMDS are deemed unacceptable.

Over time, the number of messages that the putting application send in a burst must increase by several orders of magnitude. Because the getting application must process each message sequentially, the number of messages on the queue increases to the point where the structure fills up.

At this point, the putting application receives a reason code (MQRC_STORAGE_MEDIUM_FULL) when putting a message, and the put operation fails. The putting application can only briefly tolerate periods when it is unable to put messages to the queue. If the period is too long, the application ends.

Assuming that you do not have the time, or skills available, to rewrite either the putting application or getting application, this problem has three possible solutions:

1. Increase the size of the application structure.
2. Add offload rules to the application structure so that messages are offloaded to SMDS as the queue starts to fill up.
3. Associate SCM with the structure.

The first solution is quick to implement, but not enough real storage is available on the CF.

The second solution might also be quick to implement, but the performance impact of offloading to SMDS is considered too significant to use this option.

The third solution, associating SCM with the structure, provides an acceptable balance of cost and performance.

Associating SCM with a structure results in a higher use of real storage in the CF because of the augmented storage that get operations used. However, the actual amount of real storage will be less than the amount used in the first option.

Another consideration is the cost of SCM. However this cost is much cheaper than real storage. These factors combine to make the third option cheaper than the first option.

Although the third option, potentially, might not perform as well as the first option, the pre-fetch and pre-staging algorithms used by the CF can combine to make the differences in performance acceptable, or in some cases negligible.

Certainly the performance can be much better than using SMDS to offload messages.

Consider what happens over a period of time:

1. Initially, the getting application is active and waiting for messages to be delivered to the shared queue. The putting application is not active and the shared queue is empty.
2. At a certain time, the putting application becomes active, and starts putting a large number of messages to the shared queue. The getting application starts getting the messages, but the queue depth rapidly starts to increase because the getting application is slower than the putting application.

   As a result, the application structure starts to fill up.
3. As the time increases, the putting application is still active. The application structure fills up to approximately 90%.

   This is when the SCM pre-staging algorithm starts to move messages from the structure into SCM, freeing space in the structure.

   Because the getting application gets the oldest, highest priority messages from the queue first, it is always getting messages from the structure and does not need to wait for messages to be brought synchronously from SCM in to the structure.
4. The putting application is still active and putting messages to the shared queue. However, the application never receives an MQRC_STORAGE_MEDIUM_FULL reason code, because enough space exists in SCM to store all the messages that do not fit in the structure.
5. Eventually, the putting application stops because it has no more messages to put.

   The pre-staging algorithm stops because the structure falls below 90% in use, and the getting application continues processing the messages in the queue.
6. As the getting application starts to free space in the structure, the pre-fetch algorithm starts to bring messages back from SCM in to the structure.

   Because the getting application processes messages in the order expected by the pre-fetch algorithm, the getting application never becomes blocked waiting for message data to be brought synchronously from SCM in to the structure.
7. Finally, the getting application processes all the messages on the shared queue, and waits until the next message is available. The structure and SCM are empty of messages.

z/OS *Emergency storage - basic configuration*

How you set up a basic scenario for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

For example, your enterprise has an application that puts messages onto the queue and an application that gets messages from the queue. During normal running, you expect the queue depth to be close to zero, but a business requirement indicates that the system be able to tolerate a two-hour outage of the application that gets the messages.

This means that the shared queue being used must be able to contain two hours of messages from the putting application. Currently you achieve this by using the default offload rules, and SMDS.

You expect the rate of messages being sent to the shared queue to double in the short to medium term. Although your requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure. Because the CF containing the application structure resides on a zEC12 machine, you have the capability of associating sufficient SCM with the structure to store enough messages, so that a two-hour outage can be tolerated

This initial scenario uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN1.
- Coupling facility (CF) CF01, in which the SCEN1 application structure is stored as the IBM1SCEN1 structure. This structure has a maximum size of 1 GB.
- Single shared queue, SCEN1.Q that the application structure uses.

This configuration is illustrated in Figure 61 on page 202.

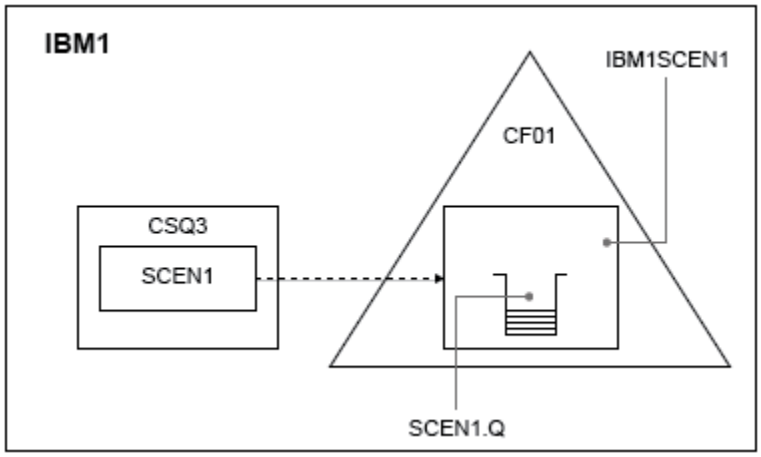


Figure 61. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN1 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Attention: To allow for high availability in your production system, you should include at least two CFs in the PREFLIST for any structures that are used by IBM MQ.

Procedure

1. Refresh the CFRM policy by using the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN1
```

Sample CFRM policy for structure IBM1SCEN1:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
```

```
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

2. Verify that the structure has been created correctly, by using the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

   At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.
3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN1 to create an IBM MQ CFSTRUCT object:

```
DEFINE CFSTRUCT(SCEN1)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 1')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

      .
   b. Validate the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN1.Q shared queue, to use the SCEN1 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN1.Q) QSGDISP(SHARED) CFSTRUCT(SCEN1) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN1.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

   Check in the output from the command, that the STATUS line shows ALLOCATED.

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

Add SMDS and SCM to the initial structure

Related concepts

“Use of storage class memory with shared queues” on page 194

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SMDS and SCM to the initial structure*

How you add SMDS and SCM for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Emergency storage - basic configuration” on page 201. The scenario describes the addition of shared message data sets (SMDS), and then of SCM to the initial structure.

This final configuration is illustrated in Figure 62 on page 204.

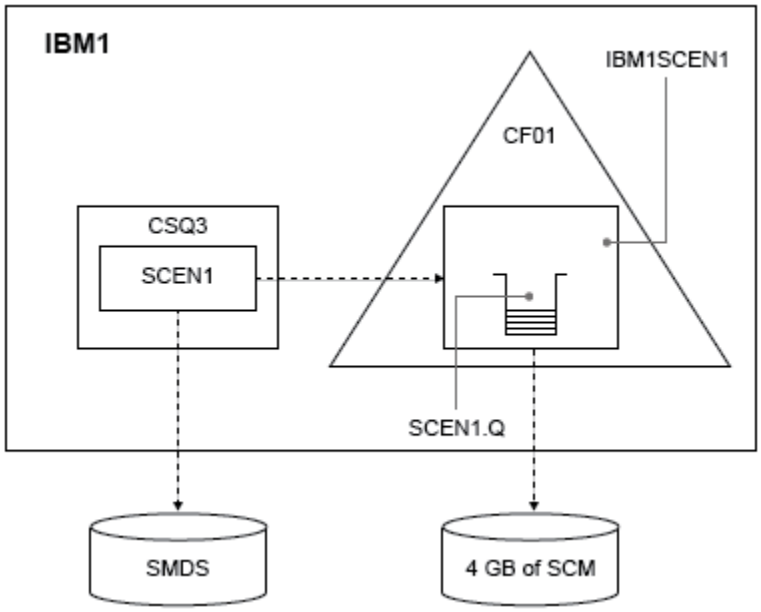


Figure 62. Configuration adding SMDS and SCM for emergency storage

Procedure

1. Create the SMDS data set that the SCEN1 application structure uses, by editing the **CSQ4SMDS** sample JCL, as shown:

```
//CSQ4SMDS JOB NOTIFY=&SYSUID
//*
//* Allocate SMDS
//*
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER           -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS) -
MEGABYTES(5000 3000)     -
LINEAR            -
SHAREOPTIONS(2 3) )      -
DATA                -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS.DATA) )
/*
//*
//* Format the SMDS
//*
//FORM   EXEC PGM=CSQJUFMT,COND=(0,NE),REGION=0M
//STEPLIB DD DSN=MQ800.SCSQANLE,DISP=SHR
//     DD DSN=MQ800.SCSQAUTH,DISP=SHR
//SYSUT1  DD DISP=OLD,DSN=CSQSMDS.SCEN1.CSQ3.SMDS
//SYSPRINT DD SYSOUT=*
```

2. Issue the ALTER CFSTRUCT command to change the SCEN1 application structure, to use SMDS for offloading, implementing the default offload rules:

```
ALTER CFSTRUCT(SCEN1) OFFLOAD(SMDS) OFFLD1SZ(32K) OFFLD2SZ(4K) OFFLD3SZ(0K)
DSGROUP('CSQSMDS.SCEN1.*.SMDS') DSBLOCK(1M)
```

   Note the following:

   - Because SCEN1.Q is the only shared queue on the SCEN1 application structure, the **DSBLOCK** value has been set to 1M, the largest value possible. This should be the most efficient setting for our scenario.
   - Because the messages sent by the putting application are 30 KB, offloading to SMDS does not start until the second offload rule is met, when the structure is 80% full.

3. Run your test application again.

   Note the increased storage of messages on the queue.

4. Add 4 GB of SCM to structure IBM1SCEN1 by carrying out the following procedure:

   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.

   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

5. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=polname
```

6. Rebuild the IBM1SCEN1 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN1
```

Results

You have successfully added SCM to your configuration.

What to do next

Optimize the performance of your system. See “Optimizing storage class memory usage” on page 206 for more information.

z/OS *Optimizing storage class memory usage*

How you improve your use of storage class memory (SCM).

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Run the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

As the structure was already full with message data, because of the previous tests, part of the rebuild involved pre-staging some of the messages from the structure into SCM. This process was initiated by using the previous command.

The output from this command produces, for example:

```
ACTIVE STRUCTURE
----------------
ALLOCATION TIME: 06/17/2014 09:28:50
CFNAME : CF01
COUPLING FACILITY: 002827.IBM.02.00000000B8D7
PARTITION: 3B CPCID: 00
STORAGE CONFIGURATION ALLOCATED MAXIMUM %
ACTUAL SIZE: 1024 M 1024 M 100
AUGMENTED SPACE: 3 M 142 M 2
STORAGE-CLASS MEMORY: 88 M 4096 M 2
ENTRIES: 120120 1089536 11
ELEMENTS: 240240 15664556 1
SPACE USAGE IN-USE TOTAL %
ENTRIES: 84921 219439 38
ELEMENTS: 2707678 3149050 85
EMCS: 2 282044 0
LOCKS: 1024
SCMHIGHTHRESHOLD : 90
SCMLOWTHRESHOLD : 70
ACTUAL SUBNOTIFYDELAY: 5000
PHYSICAL VERSION: CD5186A0 2BD8B85C
LOGICAL VERSION: CD515C50 CE2ED258
SYSTEM-MANAGED PROCESS LEVEL: 9
XCF GRPNAME : IXCLO053
DISPOSITION : KEEP
ACCESS TIME : NOLIMIT
MAX CONNECTIONS: 32
# CONNECTIONS : 1
CONNECTION NAME ID VERSION  SYSNAME JOBNAME  ASID STATE
--------------- -- -------  ------- -------  ---- ----------------
CSQEIBM1CSQ301  01 00010059 SC61    CSQ3MSTR 0091 ACTIVE
```

Note the following from the output of the command:

- That STORAGE_CLASS MEMORY provides confirmation that a **MAXIMUM** of 4096 MB of SCM has been added to the structure.
- The ALLOCATED figure for the amount of STORAGE-CLASS MEMORY used for pre-staging. There is now free space in the structure where there was none before SCM was added.
- The amount of AUGMENTED SPACE used to track SCM usage.
- The point at which the pre-staging algorithm starts to move data from the structure into SCM is when the structure is 90% full. This is indicated by the non-configurable **SCMHIGHTHRESHOLD** property.
- The point below which the prefetching algorithm starts to move data from SCM into the structure is when the structure is 70% full. This is indicated by the non-configurable **SCMLOWTHRESHOLD** property.

You can now test various ways to optimize the use of SCM. Note the following:

- After SCM is used to store messages, you cannot alter the structure until you have removed all the data from SCM.

In this case, that means that the entry-to-element ratio is frozen at the value that was in place when SCM was first used. You must carefully ensure that the structure is in the state you want, before the pre-staging algorithm starts moving data into SCM.

- Is the current structure size correct before using SCM?

  For example, have you increased **INITSIZE** from 512 MB to a SIZE of 1 GB?

  If you do not do this, it is possible that although you enabled your structure for auto-alteration, the pre-staging algorithm will start to move data into SCM before the alteration has a chance to start. As a result, the structure is frozen using 512 MB of real storage.

- Is the entry-to-element ratio correct before using SCM?

  The goal of this scenario is to increase the number of offloaded message pointers that can be stored in the structure and SCM as a whole, as well as keeping as many messages entirely in structure storage as possible. Accessing these messages is faster than accessing messages on SMDS.

  Therefore, you need to have a structure that starts with an entry-to-element ratio that is good for storing messages, and then transitions to a ratio that is good for storing message pointers before the prestage algorithm first starts. This transition can be achieved, in part, by making use of the IBM MQ offload rules.

Change the offload rules by issuing the following command:

```
ALTER CFSTRUCT(SCEN1) OFFLD1SZ(0K)
```

You might have to carry out several runs to optimize the entry-to-element ratios.

The following table shows possible improvements in the number of messages put on the queue during the different phases of the emergency storage scenario.

Table 18. Comparison of results for the emergency storage scenario

Test description	Number of messages	Time to fill queue (seconds)
Basic configuration	27,850	3.2
SMDS with default offload rules	205,000	158
SCM with default offload rules	828,610	469
SCM with adjusted offload rules	1,135,775	679

The last row in the table shows that adjusting the offload rules had the required effect.

You need to examine your system to see if you can improve on these figures in any way. For example, you might run out of available SMDS storage. If you can allocate more SMDS storage you should be able to increase the number of messages on the queue quite significantly.

z/OS *Improved performance - basic configuration*

How you set up a basic scenario for improved performance using shared queues on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This scenario describes the use of SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

This initial scenario is very similar to that used for emergency storage and uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN2.
- Coupling facility (CF) CF01, in which the SCEN2 application structure is stored as the IBM1SCEN2 structure. This structure has a maximum size of 2 GB.
- Single shared queue, SCEN2.Q, which is configured to use the application structure.

This configuration is illustrated in Figure 63 on page 208.

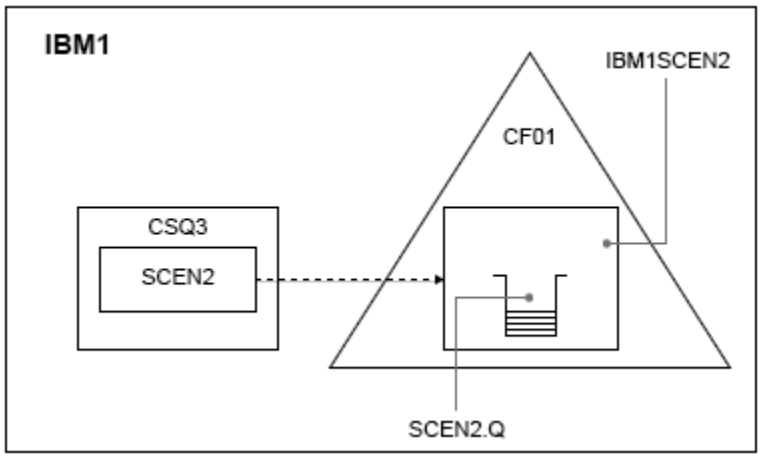


Figure 63. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN2 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Sample CFRM policy for structure IBM1SCEN2:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

Both the **INITSIZE** and **SIZE** keywords have the value 2048M so that the structure cannot resize.

Procedure

1. Refresh the CFRM policy by using the following command:

   ```
   SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
   ```

2. Verify that the structure has been created correctly, by using the following command:

   ```
   D XCF,STR,STRNAME=IBM1SCEN2
   ```

   Issuing the preceding command gives the following output:

   ```
   RESPONSE=SC61
   IXC360I 07.58.51 DISPLAY XCF 581
   STRNAME: IBM1SCEN2
   ```

```
STATUS: NOT ALLOCATED
POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY

EVENT MANAGEMENT: MESSAGE-BASED   MANAGER SYSTEM NAME: SC53
MANAGEMENT LEVEL : 01050107
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN2 to create an IBM MQ CFSTRUCT object.

```
DEFINE CFSTRUCT(SCEN2)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 2')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   b. Check the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN2.Q shared queue, to use the SCEN2 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN2.Q) QSGDISP(SHARED) CFSTRUCT(SCEN2) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN2.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output from the command, a portion of which is shown, and ensure that the STATUS line shows ALLOCATED.

```
RESPONSE=SC61
IXC360I 08.31.27 DISPLAY XCF 703
STRNAME: IBM1SCEN2
STATUS: ALLOCATED
EVENT MANAGEMENT: MESSAGE-BASED
TYPE: SERIALIZED LIST
POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY
```

Additionally, note the values of the fields in the SPACE USAGE section:

- ENTRIES
- ELEMENTS
- EMCS
- LOCKS

An example of the values follows:

```
SPACE USAGE   IN-USE      TOTAL    %
ENTRIES:       344686     345242   99
ELEMENTS:     6548455    6548467   99
EMCS:               2     780318    0
LOCKS:           1024
```

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

You should test the basic scenario. As an example, you can use the following three applications, starting the applications in the order shown and, running them concurrently.

1. Use a PCF application to request the current depth (**CURDEPTH**) value for SCEN2.Q every five seconds. The output can be used to plot the depth of the queue over time.
2. A single threaded getting application repeatedly gets messages from SCEN2.Q, using a get with an infinite wait. To simulate processing of the messages that were removed, the getting application pauses for four milliseconds for every ten messages that it removed.
3. A single threaded putting application puts a total of one million 4 KB non-persistent messages to SCEN2.Q. This application does not pause between putting each message so messages are put on SCEN2.Q faster than the getting application can get them.

   As a result, when the putting application is running, the depth of SCEN2.Q increases.

   When structure IBM1SCEN2 is filled, and the putting application receives a MQRC_STORAGE_MEDIUM_FULL reason code, the putting application sleeps for five seconds before attempting to put the next message to the queue.

You can plot the results of the CURDEPTH application over a period of time. You obtain some form of saw-tooth wave output as the putting application pauses to allow the queue to partially empty.

Go to “Adding SCM to the initial structure” on page 210.

Related concepts

“Use of storage class memory with shared queues” on page 194

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SCM to the initial structure*

How you add SCM for improved performance on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Improved performance - basic configuration” on page 207. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 211.

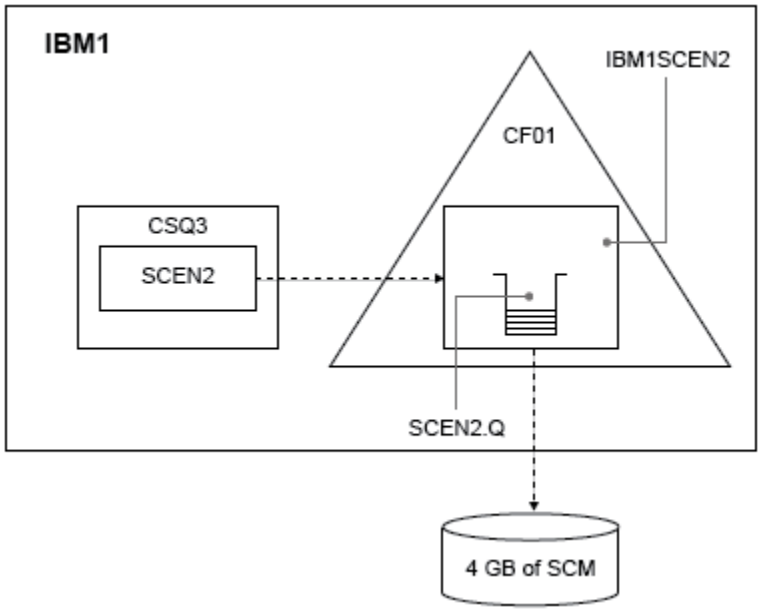


Figure 64. Configuration adding SCM for improved performance

Procedure

1. Add 4 GB of SCM to structure IBM1SCEN2 by carrying out the following procedure:
   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.
   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

2. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
```

3. Rebuild the IBM1SCEN2 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN2
```

4. Issue the following command to confirm the new configuration of the structure:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output of the command, a portion of which follows:

```
SPACE USAGE     IN-USE       TOTAL      %
ENTRIES:          33         342684     0
ELEMENTS:         48        6503697     0
EMCS:              2         575600     0
LOCKS:                         1024
```

Results

Calculate the change in the use of real storage by the increase in control storage required to use SCM.

- Before SCM is added to the structure, the structure has these totals as shown in “Improved performance - basic configuration” on page 207:
  - 345,242 entries
  - 6,548,467 elements
  - 780,318 EMCS
- After SCM is added to the structure, the structure has these totals:
  - 342,684 entries
  - 6,503,697 elements
  - 575,600 EMCS

Using these figures, after the SCM was added, the structure is reduced in size by:

- 2558 entries
- 44,770 elements
- 204,718 EMCS

The amount of structure storage that is used to manage SCM, is as follows for a 2 GB structure with 4 GB of SCM allocated:

```
(2558 + 44,770 + 204,718) * 256 = 61.5 MB
```

Note that adding more SCM is likely to achieve only a marginal reduction of the size of the structure, because the amount of control storage used to track SCM increases, both as the structure size, and the amount of allocated SCM increases.

What to do next

Repeat the tests described in the final section of “Improved performance - basic configuration” on page 207.

You can plot the results of the revised application over a period of time. Comparing the plot to the one obtained previously, you now obtain an output without a saw-tooth wave, as the putting application no longer has to wait for the queue to partially empty.

For more information, refer to MP16: WebSphere MQ for z/OS - Capacity planning & tuning.

z/OS Distributed queuing and queue sharing groups

Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

To complement the high availability of messages on shared queues, the distributed queuing component of IBM MQ has additional functions to provide the following:

- Higher availability to the network.
- Increased capacity for inbound network connections to the queue sharing group.

Figure 65 on page 213 illustrates distributed queuing and queue sharing groups. It shows two queue managers within a sysplex, both of which belong to the same queue sharing group. They can both access shared queue SQ1. Queue managers in the network (on AIX and Windows for example) can put messages onto this queue through the channel initiator of either queue manager. Cloned applications on both queue managers service the queue.

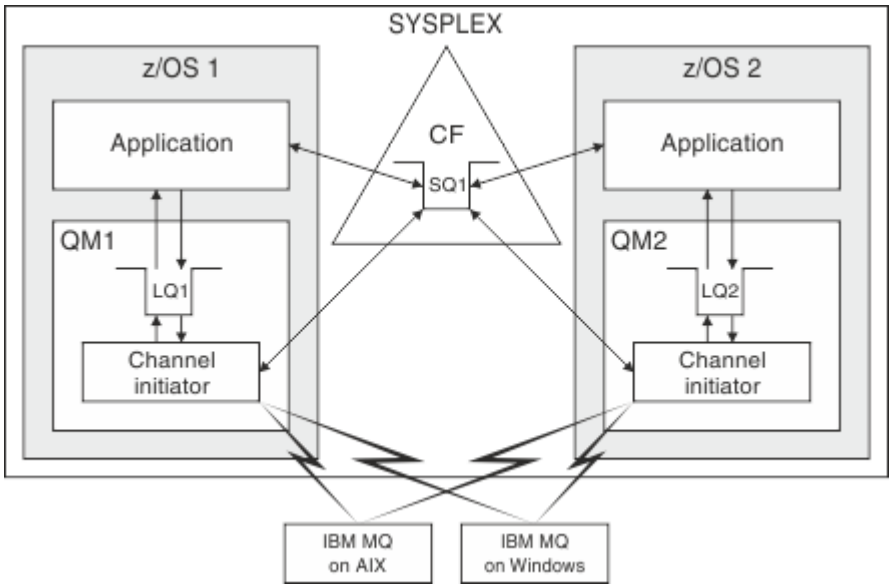


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 213
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 218
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Clusters and queue sharing groups" on page 215
Use this topic to understand how you can use queue sharing groups with clusters.

z/OS *Shared channels*

Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

A number of networking products provide a mechanism to hide server failures from the network, or to balance inbound network requests across a set of eligible servers. The network products make a *generic port* available for inbound network connection requests, and the inbound request can be satisfied by connecting to one of the eligible servers.

These networking products include:

- VTAM generic resources
- SYSPLEX Distributor

The channel initiator takes advantage of these products to use the capabilities of shared queues

There are two types of shared channels, *shared inbound channel*, and the *shared outbound channel*.

- Shared inbound channels
- Shared outbound channels

For further information about channels see

- Shared channel summary
- Shared channel status

Shared inbound channels

Each channel initiator in the queue sharing group starts an additional listener task to listen on a *generic port*. This generic port is made available to the network by one of the supporting technologies (VTAM, TCP/IP). Inbound network attach requests to the generic port are dispatched by the network technology, to any one of the listeners in the queue sharing group (QSG) that are listening on the generic port.

You can start a channel on the channel initiator to which the inbound attach is directed if the channel initiator has access to a channel definition for a channel with that name. You can define a channel definition to be private to a queue manager or stored on the shared repository and so available anywhere (a global definition). This means that you can make a channel definition available on any channel initiator in the queue sharing group by defining it as a global definition.

There is an additional difference when starting a channel through the generic port; channel synchronization is with the queue sharing group and not with an individual queue manager. For example, consider a remote queue manager starting a channel through the generic port. When the channel first starts, it might start on queue manager QM1 and messages flow. If the channel stops and is restarted on queue manager QM2, information about the number of messages that have flowed is still correct because the synchronization is with the queue sharing group.

You can use an inbound channel started through the generic port to put messages to any queue. The remote queue manager does not know whether the target queue is shared or not. If the target queue is a shared queue, the remote queue manager connects through any available channel initiator in a load-balanced fashion and the messages are put to the shared queue.

If the target queue is a private queue, the messages are put to the private queue owned by which ever queue manager the current instance of the channel is connected to. In this environment, known as *replicated local queues*, each queue manager must have the same set of private queues defined.

Configuring SVRCONN channels for a queue sharing group

The optimal configuration for SVRCONN channels in a queue sharing group is to set up private listeners in each CHINIT which use a different port number from the point to point channels. These listener ports are then used as the 'back-end' resources for a new workload distribution mechanism such as Sysplex Distributor using Virtual IP addresses (VIPA). The external VIPA address is then used as the target address for the CLNTCONN definitions in the network. The SVRCONN channel can be defined with QSGDISP(GROUP) so the same definition is available to all queue managers in the QSG. This configuration avoids using a shared listener, and therefore reduces the performance effect of the queue sharing group maintaining shared channel state, which is not needed for client/server channels.

Shared outbound channels

An outbound channel is considered to be a shared channel if it is taking messages from a shared transmission queue. If it is shared, it holds synchronization information at queue sharing group level. This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue sharing group if the communications subsystem, channel initiator, or queue manager fails. Restarting failed channels in this way is a feature of shared channels called *peer channel recovery*.

Workload balancing for shared outbound channels

An outbound shared channel is eligible for starting on any channel initiator within the queue sharing group, if you have not specified that you want it to be started on a particular channel initiator. The channel initiator selected by IBM MQ is determined using the following criteria:

- Is the communications subsystem required currently available to the channel initiator?
- Is a Db2 connection available to the channel initiator?
- Which channel initiator has the lowest current workload? The workload includes channels that are active and retrying.

Shared channel summary

Shared channels differ from private channels in the following ways:

Private channel
Tied to a single channel initiator.

- Outbound channel uses a local transmission queue.
- Inbound channel started through a local port.
- Synchronization information held in SYSTEM.CHANNEL.SYNCQ queue.

Shared Channel
Workload balanced with high availability.

- Outbound channel uses a shared transmission queue.
- Inbound channel started through a generic port.
- Synchronization information held in SYSTEM.QSG.CHANNEL.SYNCQ queue.

You specify whether a channel is private or shared when you start the channel by using CHLDISP options with the START CHANNEL command. A shared channel can be started by triggering in the same way as a private channel. However, when a shared channel is started, IBM MQ performs workload balancing and starts the channel on the most appropriate channel initiator within the queue sharing group. (If required, you can specify that a shared channel is to be started on a particular channel initiator.)

Shared channel status

The channel initiators in a queue sharing group maintain a shared channel-status table in Db2. This records which channels are active on which channel initiators. The shared channel-status table is used if there is a channel initiator or communications system failure. It indicates which channels need to be restarted on a different channel initiator in the queue sharing group.

z/OS *Clusters and queue sharing groups*

Use this topic to understand how you can use queue sharing groups with clusters.

You can make your shared queues available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue sharing group (the shared queue is not advertised as being hosted by the queue sharing group). Clients can start sessions with any members of the queue sharing group to put messages to the same shared queue.

Figure 66 on page 216 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

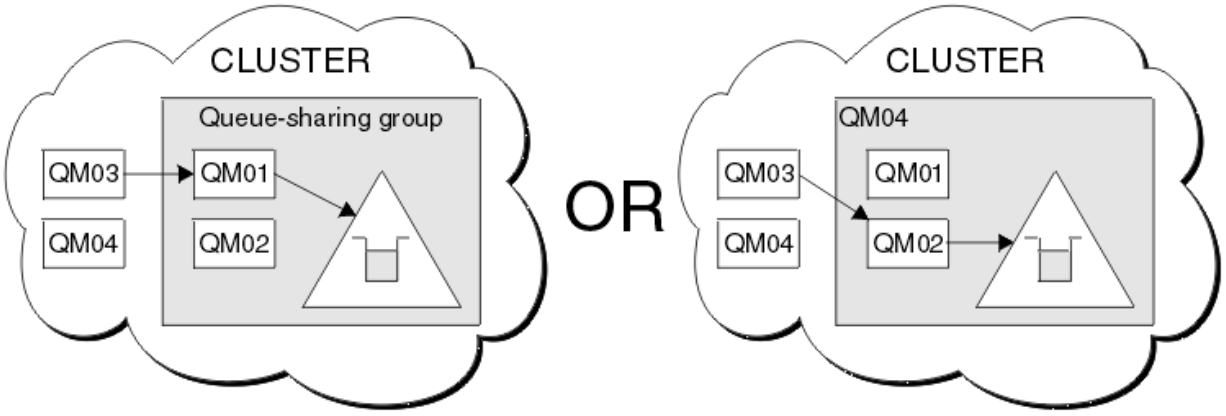


Figure 66. A queue sharing group as part of a cluster

z/OS Influencing workload distribution with shared queues

Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

IBM MQ does not provide workload balancing for shared queues. However, workload distribution in a queue sharing group (QSG) can be influenced in *a pull based fashion*. The choice of which queue manager services a queue (receives a message written to a shared queue) is affected by the available processing capacity of each queue manager in the queue sharing group and the workload management goals defined across the sysplex.

However, it is important to appreciate, the queue manager that performs the MQPUT of a message can also have a large influence in deciding which queue manager gets the message.

A local queue manager is more likely to perform the MQGET

For an application performing an MQPUT, the local queue manager is said to be the queue manager to which the application is connected.

Exactly which queue manager services an MQPUT of a message by performing an MQGET on behalf of a getting application is influenced by the following considerations.

When a message is put to an empty shared queue, the local queue manager is typically posted before any of the other queue manager in the queue sharing group is notified. If the local queue manager is in a position to process the message, it receives a list transition notification from the coupling facility (CF) before any other queue manager in the QSG. (A list transition notification is a notification that the shared queue has changed state from empty to non-empty.)

The possible scenarios, in this case, are as follows:

1. MQPUT of nonpersistent message out of sync point and *fast put to waiting getter*.

   If there is an application with an *MQGET with wait* on the local queue manager for the queue, then the MQPUT of the message is passed directly to the getting application's buffer and not written to the queue. This is true for shared and non-shared queues. This feature is often called *fast put to a waiting getter* mechanism. In the case of shared queues, no other queue manager in the QSG is notified as there is no transition from empty to non-empty of the queue. This means, for example, that provided this queue manager can service all the puts from this application and assuming that no other applications are putting messages to the queue, then no other queue manager in the queue sharing group assists in draining this queue. If however there is no MQGET with wait on the local queue manager, and a message is put to the shared queue then the CF will notify other queue managers in the queue sharing group according to its rules for notifications of list transitions.
2. MQPUT of a persistent or in-syncpoint message.

In this case, if there is an application with an *MQGET with wait* on the local queue manager, then the message is put to the shared queue and the CF notifies other queue managers in the queue sharing group according to its rules for notifications of list transitions. However, the local queue manager does not wait for a transition notification from the CF but honors any local *MQGET with wait* first and usually performs the get of this message on behalf of the application before any other queue manager in the queue sharing group can respond to a CF notification. This is dependent on how busy the local queue manager is. Otherwise, any queue manager notified by the CF due to the arrival of the message on the empty queue will try to service the get first. The first queue manager to respond processes the new message.

3. Finally, if the queue is not drained of messages, where the CF has sent a notification of a state change from empty to non-empty for the queue, all connected queue managers will have an opportunity to assist in the processing of the queue. In this event, the workload is said to be *pull based*.

This design allows for the improved performance over a purely pull based workload distribution. The aim is to take advantage of the high availability offered by queues held in the CF while allowing the queue manager, where possible, to perform the MQGET without needing to reference the CF and so to process the message workload as efficiently as possible.

Alternative approaches can be adopted where emphasis on balance of the workload is more important than the previously described performance enhancements. For example, ensuring that none of the getting applications are connected to the same queue manager that the putting application is connected to. Using this design all messages are put to the queue and all queue managers in the QSG are notified when the queue moves from empty to non-empty, in accordance with the CF algorithm for handling such transitions. In addition, the *fast put to waiting getter* mechanism is not applicable.

z/OS Where to find more information about shared queues and queue sharing groups

Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

Table 19. Where to find more information about shared queues and queue sharing groups

Topic	Where to look
Queue sharing group recovery	“Recovery and restart on z/OS” on page 256
Queue sharing group security	“Security concepts in IBM MQ for z/OS” on page 272
Private and global object definitions Directing commands to different queue managers	Sources from which you can issue commands on z/OS
Planning your coupling facility environment	Defining coupling facility resources
Planning your SMDS environment	Planning your shared message data set (SMDS) environment
Planning your Db2 environment	Planning your Db2 environment
Setting up your shared queues System parameters	“Shared queues and queue sharing groups” on page 174
Utility programs Migrating queues	IBM MQ utilities on z/OS reference

Table 19. Where to find more information about shared queues and queue sharing groups (continued)	
Topic	**Where to look**
Console messages	Messages for IBM MQ for z/OS
MQSC commands	MQSC commands
IBM MQ clusters	Configuring a queue manager cluster
IBM MQ distributed queuing Channel names	Introduction to distributed queue management
Writing applications	Overview of application design
MQCONNX call	MQCONNX

z/OS Intra-group queuing

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

For information about queue sharing groups, see “Shared queues and queue sharing groups” on page 174.

Intra-group queuing concepts

You can perform fast message transfer between queue managers in a queue sharing group without defining channels. This uses a system queue called the SYSTEM.QSG.TRANSMIT.QUEUE, which is a shared transmission queue. Each queue manager in the queue sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue.

Standard name resolution rules are used but, if intra-group queuing (IGQ) is enabled and the target queue manager is within the queue sharing group, the SYSTEM.QSG.TRANSMIT.QUEUE is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

You enable intra-group queuing through a queue manager attribute. Intra-group queuing moves nonpersistent messages outside sync point, and persistent messages within sync point. If it finds a problem delivering messages to the target queue, intra-group queuing tries to put them to the dead-letter queue. If the dead-letter queue is full or undefined, nonpersistent messages are discarded, but persistent messages are backed out and returned to the SYSTEM.QSG.TRANSMIT.QUEUE, and the IGQ agent tries to deliver the messages until it is successful.

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue sharing group can use intra-group queuing to *hop* the message to the correct destination.

There might be times when you want the local queue manager to put a message directly to the target queue if the target queue is a shared queue, rather than the message first being transferred to the target queue manager. You can use the queue manager attribute SQQMNAME to control this. If you set the value of SQQMNAME to USE, the MQOPEN command is performed on the queue manager specified by the **ObjectQMgrName**.

However, if the target queue is a shared queue and you set the value of SQQMNAME to IGNORE, and the **ObjectQMgrName** is that of another queue manager in the queue sharing group, the shared queue is opened on the local queue manager. If the local queue manager cannot open the target queue, or put a

message to the queue, the message is transferred to the specified **ObjectQMgrName** through either IGQ or an IBM MQ channel.

Intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue sharing group. Intra-group queuing also supports large messages, the largest being 100 MB *minus* the length of the transmission queue header.

Note: If you use this feature, users must have the same access to the queues on each queue manager in the queue sharing group.

The following diagram shows a typical example of intra-group queuing.

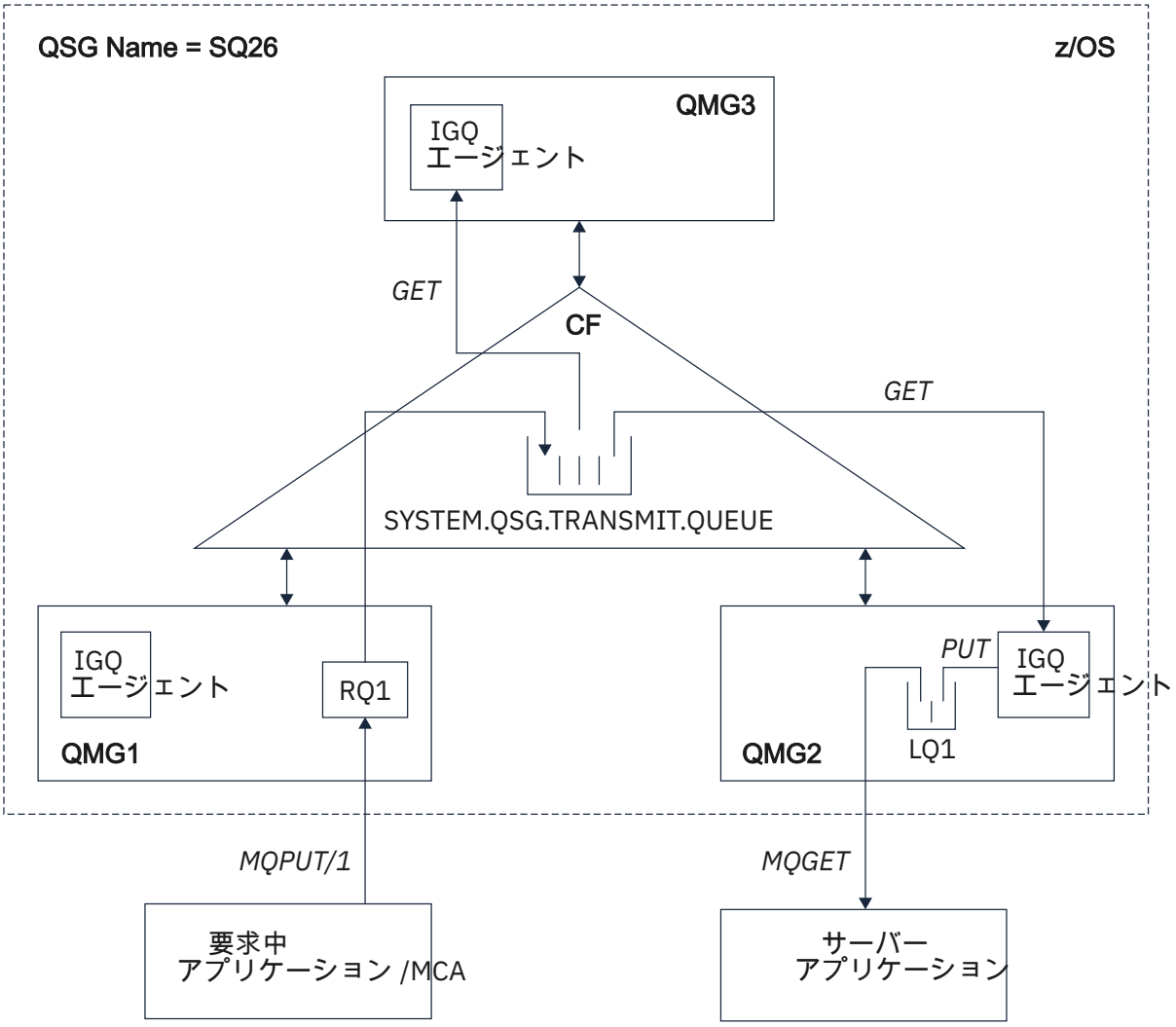


Figure 67. An example of intra-group queuing

The diagram shows:

- IGQ agents running on three queue managers (QMG1, QMG2, and QMG3) that are defined to a queue sharing group called SQ26.
- Shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE that is defined in the coupling facility (CF).
- A remote queue definition that is defined in queue manager QMG1.
- A local queue that is defined in queue manager QMG2.
- A requesting application (this application could be a Message Channel Agent (MCA)) that is connected to queue manager QMG1.

- A server application that is connected to queue manager QMG2.
- A request message being placed on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing and the intra-group queuing agent

An IGQ agent is started during queue manager initialization. When applications open and put messages to remote queues, the local queue manager determines whether intra-group queuing is used for message transfer. If intra-group queuing is to be used, the local queue manager places the message on to the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent on the target remote queue manager retrieves the message and places it on to the destination queue.

Intra-group queuing terminology

Explanations of the terminology: intra-group queuing, shared transmission queue for use by intra-group queuing, and intra-group queuing agent.

Intra-group queuing

Intra-group queuing can effect potentially fast and less expensive message transfer between queue managers in a queue sharing group, without the need to define channels.

Shared transmission queue for use by intra-group queuing

Each queue sharing group has a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE for use by intra-group queuing. If intra-group queuing is enabled, SYSTEM.QSG.TRANSMIT.QUEUE appears in the name resolution path when opening remote queues. When applications (including Message Channel Agents (MCAs)) put messages to a remote queue, the local queue manager determines the eligibility of messages for fast transfer and places them on SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing agent

The IGQ agent is the task, started at queue manager initialization, that waits for suitable messages to arrive on the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent retrieves suitable messages from this queue and delivers them to the destination queues.

The IGQ agent for each queue manager is always started because intra-group queuing is used by the queue manager itself for its own internal processing.

z/OS Benefits of intra-group queuing

The benefits of intra-group queuing are: reduced system definitions, reduced system administration, improved performance, supports migration, and delivery of messages when multi-hopping between queue managers in a queue sharing group.

The benefits of intra-group queuing are:

Reduced system definitions
Intra-group queuing removes the need to define channels between queue managers in a queue sharing group.

Reduced system administration
Because there are no channels defined between queue managers in a queue sharing group, there is no requirement for channel administration.

Improved performance
Because there is only one IGQ agent needed for the delivery of a message to a target queue (instead of two intermediate sender and receiver agents), the delivery of messages using intra-group queuing can be less expensive than the delivery of messages using channels. In intra-group queuing there is only a receiving component, because the need for the sending component has been removed. This saving is because the message is available to the IGQ agent at the destination queue manager for

delivery to the destination queue once the put operation at the local queue manager has completed and, in the case of messages put in sync point scope, committed.

Supports migration

Applications external to a queue sharing group can deliver messages to a queue residing on any queue manager in the queue sharing group, while being connected only to a particular queue manager in the queue sharing group. This is because messages arriving on a receiver channel, destined for a queue on a remote queue manager, can be transparently sent to the destination queue using intra-group queuing. This facility allows applications to be deployed among the queue sharing group without the need to change any systems that are external to the queue sharing group.

A typical configuration is illustrated by the following diagram, in which:

- A requesting application connected to queue manager QMG1 needs to send a message to a local queue on queue manager QMG3.
- Queue manager QMG1 is connected only to queue manager QMG2.
- Queue managers QMG2 and QMG3, which were previously connected using channels, are now members of queue sharing group SQ26.

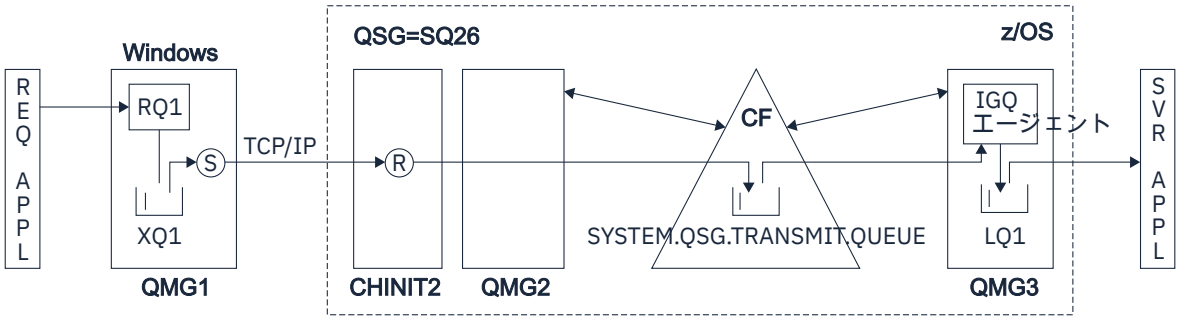


Figure 68. An example of migration support

The flow of operations is as follows:

1. The requesting application puts a message, destined for local queue LQ1 at remote queue manager QMG3, on to remote queue definition RQ1.
2. Queue manager QMG1, running on a Windows NT workstation, places the message on to the transmission queue XQ1.
3. Sender MCA (S) on QM1 transmits the message, using TCP/IP, to the receiver MCA (R) on channel initiator CHINIT2.
4. Receiver MCA (R) on channel initiator CHINIT2 places the message on to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the message from the SYSTEM.QSG.TRANSMIT.QUEUE and places it on to the target local queue LQ1.
6. The server application retrieves the message from the target local queue and processes it.

Delivery of messages when multi-hopping between queue managers in a queue sharing group

The previous diagram in Supports migration also illustrates the delivery of messages when multi-hopping between queue managers in a queue sharing group. Messages arriving on a queue manager within the queue sharing group, but destined for a queue on another queue manager in the queue sharing group, can be easily transmitted to the destination queue on the destination queue manager, using intra-group queuing.

z/OS Limitations of intra-group queuing

The limitations of intra-group queuing are: messages eligible for transfer using intra-group queuing, number of intra-group queuing agents per queue manager, and starting and stopping the intra-group queuing agent.

This topic describes the limitations of intra-group queuing.

Messages eligible for transfer using intra-group queuing
Because intra-group queuing uses a shared transmission queue that is defined in the coupling facility (CF), intra-group queuing is limited to delivering messages of the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Number of intra-group queuing agents per queue manager
Only one IGQ agent is started per queue manager in a queue sharing group.

Starting and stopping the intra-group queuing agent
The IGQ agent is started during queue manager initialization and terminated during queue manager shut-down. It is designed to be a long running, self recovering (in the event of abnormal termination), task. If there is an error with the definition of the SYSTEM.QSG.TRANSMIT.QUEUE (for example, if this queue is Get inhibited) the IGQ agent keeps retrying. If the IGQ agent encounters an error that results in normal termination of the agent while the queue manager is still active, it can be restarted by issuing an ALTER QMGR IGQ(ENABLED) command. This command avoids the need to recycle the queue manager.

Setting the queue manager attribute IGQ to ENABLED or DISABLED
If the queue manager attribute IGQ is set to ENABLED or DISABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See Getting started with intra-group queueing for more information.

z/OS Getting started with intra-group queuing

You can enable, disable, and use intra-group queuing as described in this topic.

Enabling intra-group queuing
To enable intra-group queuing on your queue managers, you need to do the following:

- Define a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE. The definition of this queue can be found in thlqual.SCSQPROC(CSQ4INSS), the CSQINP2 sample for SYSTEM objects for queue sharing groups. This queue must be defined with the correct attributes, as stated in thlqual.SCSQPROC(CSQ4INSS), for intra-group queuing to work properly.
- Because the IGQ agent is always started at queue manager initialization, intra-group queuing is always available for inbound message processing. The IGQ agent processes any messages that are placed on the SYSTEM.QSG.TRANSMIT.QUEUE. However, to enable intra-group queuing for outbound processing, the queue manager attribute IGQ must be set to ENABLED.

  Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 230 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 222 for further information.

Disabling intra-group queuing

To disable intra-group queuing for outbound message transfer, set the queue manager attribute IGQ to DISABLED. If intra-group queuing is disabled for a particular queue manager, the IGQ agent on that queue manager can still process inbound messages that have been placed on the SYSTEM.QSG.TRANSMIT.QUEUE by a queue manager that does have intra-group queuing enabled for outbound transfer.

Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 230 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 222 for further information.

Using intra-group queuing

Once intra-group queuing is enabled, it is available for use and a queue manager uses it whenever possible. That is, when an application puts a message to a remote queue definition, to a fully qualified remote queue, or to a cluster queue, the queue manager determines if the message is eligible to be delivered using intra-group queuing and if it is, places the message on to SYSTEM.QSG.TRANSMIT.QUEUE. There is no need to change user applications, or to application queues, because for eligible messages the queue manager uses the SYSTEM.QSG.TRANSMIT.QUEUE, in preference to any other transmission queue.

z/OS Intra-group queuing configurations

In addition to the typical intra-group queuing configuration, other configurations are possible.

“Intra-group queuing concepts” on page 218 describes the typical configuration.

Related concepts

“Distributed queuing with intra-group queuing (multiple delivery paths)” on page 223
For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

“Clustering with intra-group queuing (multiple delivery paths)” on page 225
It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

“Clustering, intra-group queuing and distributed queuing” on page 227
It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

z/OS *Distributed queuing with intra-group queuing (multiple delivery paths)*

For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

The choice of intra-group queuing over channel communications can be controlled by the CFSTRUCT type level. (3 instead of 4 or 5). The maximum message length as set on the SYSTEM.QSQ.TRANSMIT.QUEUE.

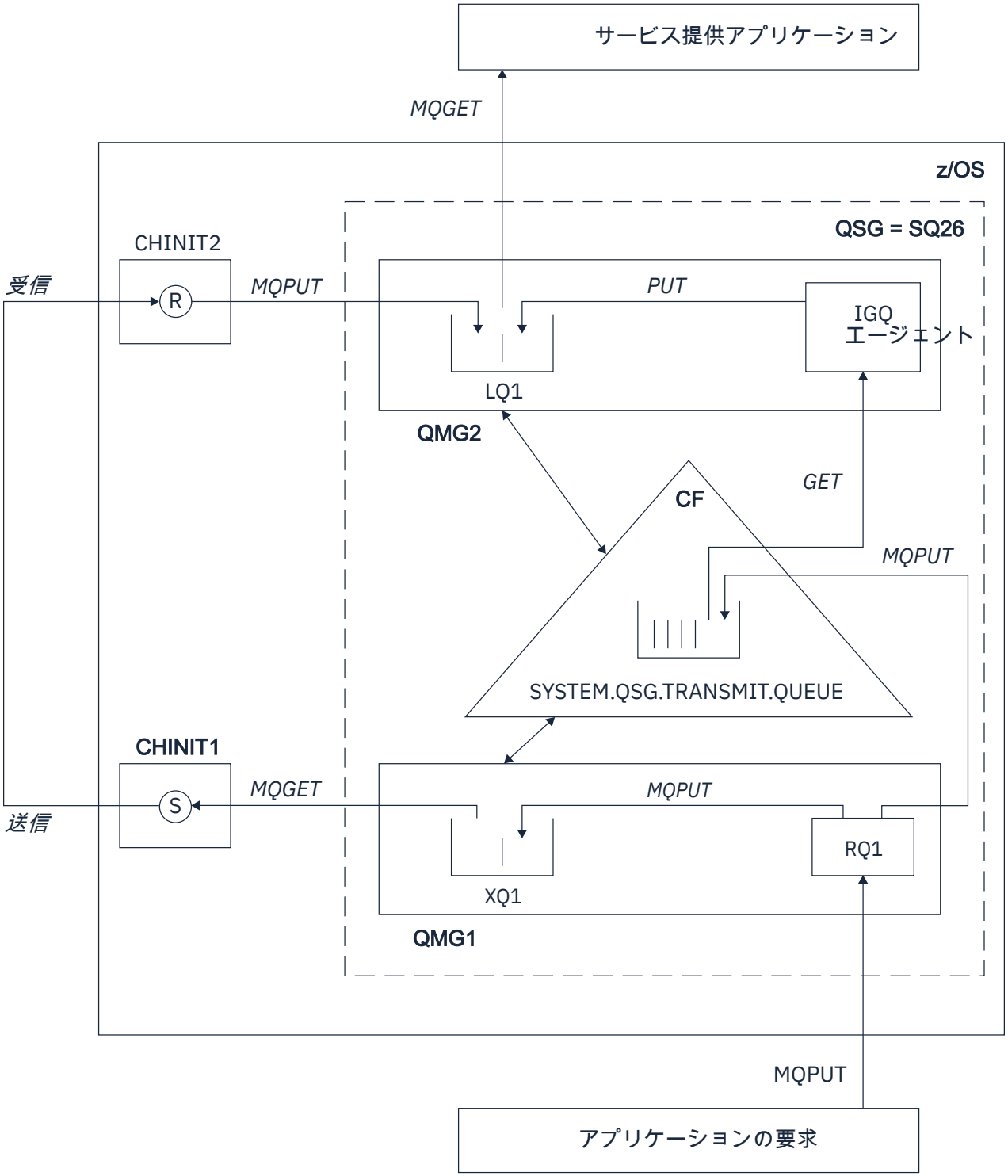


Figure 69. An example configuration

Open/Put processing

1. It is important to note that when the requesting application opens remote queue RQ1, name resolution occurs for both the non-shared transmission queue XQ1 and the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
2. When the requesting application puts a message on to the remote queue, based on whether intra-group queuing is enabled for outbound transfer on the queue manager and on the message characteristics, the message is put to transmission queue XQ1, or to transmission queue

SYSTEM.QSG.TRANSMIT.QUEUE. The queue manager places all large messages on to transmission queue XQ1, and all small messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.

3. If transmission queue XQ1 is full, or is not available, put requests for large messages fail synchronously with a suitable return and reason code. However, put requests for small messages continue to succeed and are placed on transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
4. If transmission queue SYSTEM.QSG.TRANSMIT.QUEUE is full, or cannot be put to, put requests for small messages fail synchronously with a suitable return and reason code. However, put requests for large messages continue to succeed and are placed on transmission queue XQ1. In this case, no attempt is made to put the small messages on to a transmission queue.

Flow for large messages

1. The requesting application puts large messages to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue XQ1.
3. Sender MCA (S) on queue manager QMG1 retrieves the messages from transmission queue XQ1 and sends them to queue manager QMG2.
4. Receiver MCA (R) on queue manager QMG2 receives the messages and places them on to destination queue LQ1.
5. The serving application retrieves and then processes the messages from queue LQ1.

Flow for small messages

1. The requesting application puts small messages on to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. IGQ on queue manager QMG2 retrieves the messages and places them on to the destination queue LQ1.
4. The serving application retrieves the messages from queue LQ1.

Points to note

1. The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
2. A potentially faster message delivery mechanism can be achieved for small messages.
3. Multiple paths are available for message delivery (that is, the normal channel route and the intra-group queuing route).
4. The intra-group queuing route, being potentially faster, is selected in preference to the normal channel route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence (though this delivery is also possible if messages are delivered using only the normal channel route).
5. When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to transmission queue XQ1.

z/OS *Clustering with intra-group queuing (multiple delivery paths)*

It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

When messages are sent to a cluster queue and the local and remote destination queue managers are in the same queue sharing group, intra-group queuing is used for the delivery of small messages (using the SYSTEM.QSG.TRANSMIT.QUEUE), and the delivery of large messages if intra-group queuing supports the size of the message. Also, the SYSTEM.CLUSTER.TRANSMIT.QUEUE is used for the delivery of messages to any queue manager that is in the cluster, but outside the queue sharing group. The following diagram illustrates this configuration (the channel initiators are not shown).

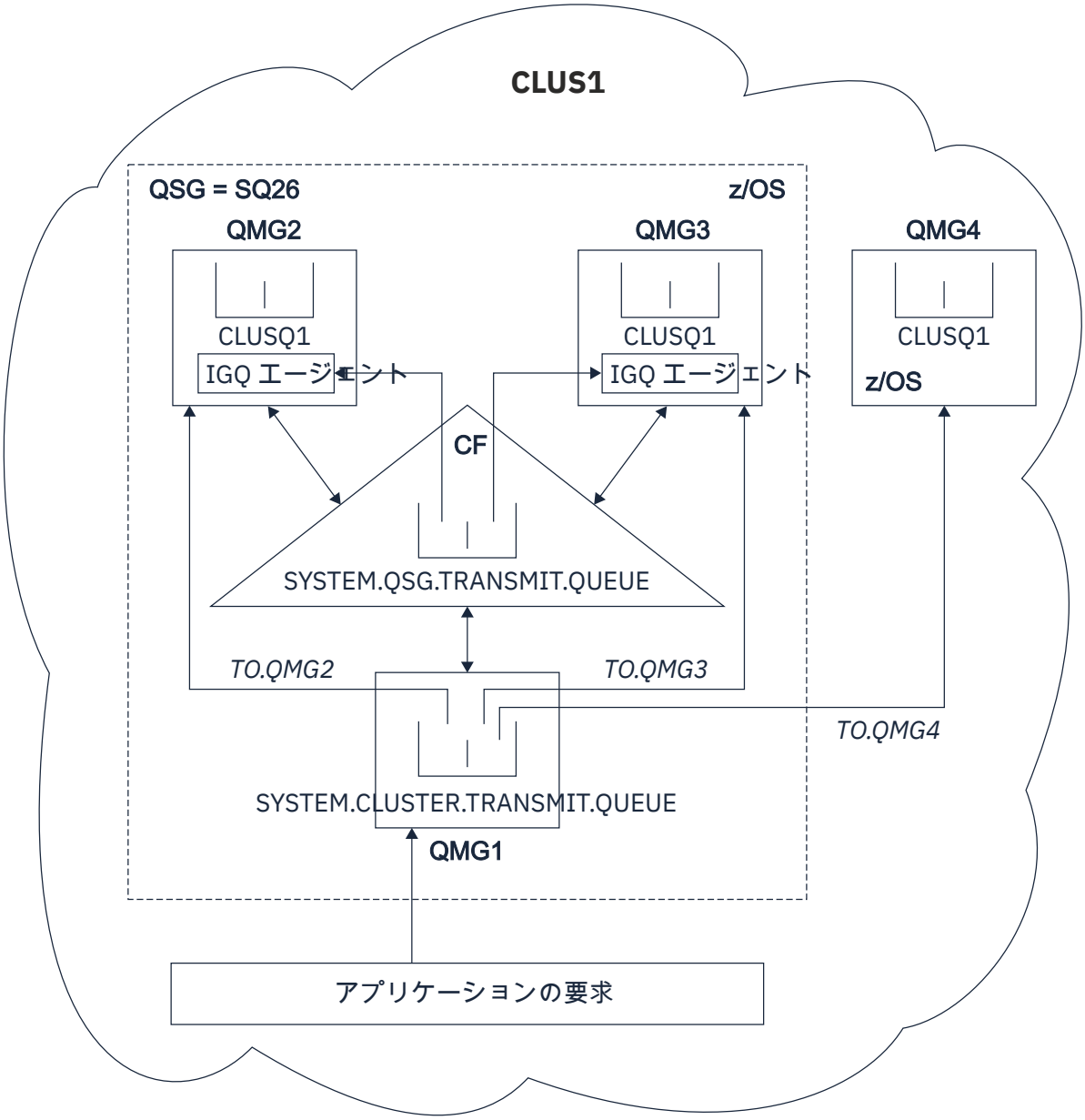


Figure 70. An example of clustering with intra-group queuing

The diagram shows:

- Four z/OS queue managers QMG1, QMG2, QMG3, and QMG4 configured in a cluster CLUS1.
- Queue managers QMG1, QMG2, and QMG3 configured in a queue sharing group SQ26.
- IGQ agents running on queue managers QMG2 and QMG3.
- The local SYSTEM.CLUSTER.TRANSMIT.QUEUE defined in QMG1.

  Note: For clarity, the SYSTEM.CLUSTER.TRANSMIT.QUEUE on the other queue managers not shown.
- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF, which is in an IBM MQ structure configured with the CFLEVEL(3) RECOVER(YES) attribute.
- Cluster channels TO.QMG2 (connecting QMG1 to QMG2), TO.QMG3 (connecting QMG1 to QMG3), and TO.QMG4 (connecting QMG1 to QMG4).
- Cluster queue CLUSQ1 being hosted on queue managers QMG2, QMG3, and QMG4.

Assume that the requesting application opens the cluster queue with the MQOO_BIND_NOT_FIXED option, so that the target queue manager for the cluster queue is selected at put time.

If the selected target queue manager is QMG2:

- All large messages put by the requesting application are:
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1, because SYSTEM.QSG.TRANSMIT.QUEUE is in a CFLEVEL(3) structure; therefore supports messages only up to 63 KB in size.
  - Transferred to cluster queue CLUSQ1 on QMG2 using cluster channel TO.QMG2
- All small messages put by the requesting application are
  - Put to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. This queue is in a structure configured with the RECOVER(YES) attribute, so is used for both persistent, and non-persistent, small messages.
  - Retrieved by the IGQ agent on QMG2
  - Put to the cluster queue CLUSQ1 on QMG2

If the selected target queue manager is QMG4:

- Because QMG4 is not a member of queue sharing group SQ26, all messages put by the requesting application are
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
  - Transferred to cluster queue CLUSQ1 on QMG4 using cluster channel TO.QMG4

Points to note

- The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
- A potentially faster delivery mechanism is achieved for the transfer of small non-persistent messages between queue managers in a queue sharing group (even if the same queue managers are in a cluster).
- Multiple paths are available for message delivery (that is, both the cluster route and the intra-group queuing route).
- The intra-group queuing route, being potentially faster, is selected in preference to the cluster route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence. It is important to note that this delivery is possible without regard to the MQOO_BIND_* option specified by the application. Intra-group queuing distributes messages in the same way as clustering does, depending on whether the MQOO_BIND_NOT_FIXED, MQOO_BIND_ON_OPEN, MQOO_BIND_ON_GROUP, or MQOO_BIND_AS_Q_DEF is specified on open.
- When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

z/OS *Clustering, intra-group queuing and distributed queuing*

It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

This configuration is a combination of distributed queuing with intra-group queuing and clustering with intra-group queuing.

Intra-group queuing is described in “Distributed queuing with intra-group queuing (multiple delivery paths)” on page 223.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 225.

z/OS Intra-group queuing messages

This section describes the messages put to the SYSTEM.QSG.TRANSMIT.QUEUE.

Message structure

Like all other messages that are put to transmission queues, messages that are put to the SYSTEM.QSG.TRANSMIT.QUEUE are prefixed with the transmission queue header (MQXQH).

Message persistence

In IBM WebSphere MQ 5.3 and above, shared queues support both persistent and non-persistent messages.

If the queue manager terminates while the IGQ agent is processing non-persistent messages, or if the IGQ agent terminates abnormally while in the middle of processing messages, non-persistent messages being processed might be lost. Applications must make arrangements for the recovery of non-persistent messages if their recovery is required.

If a put request for a non-persistent message, issued by the IGQ agent, fails unexpectedly, the message being processed is lost.

Delivery of messages

The IGQ agent retrieves and delivers all nonpersistent messages outside of sync point scope, and all persistent messages within sync point scope. In this case, the IGQ agent acts as the sync point coordinator. The IGQ agent therefore processes nonpersistent messages like the way fast, nonpersistent messages are processed on a message channel. See Fast, nonpersistent messages.

Batching of messages

The IGQ agent uses a fixed batch size of 50 messages. Any persistent messages retrieved within a batch are committed at intervals of 50 messages. The agent commits a batch consisting of persistent messages when there are no more messages available for retrieval on the SYSTEM.QSG.TRANSMIT.QUEUE.

Message size

The maximum size of message that can be put to the SYSTEM.QSG.TRANSMIT.QUEUE is the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Default message persistence and default message priority

If the SYSTEM.QSG.TRANSMIT.QUEUE is in the queue name resolution path established at open time, then for messages that are put with default persistence and default priority (or with default persistence or default priority), the normal rules are applied in the selection of the queue that has default priority and persistence values that are used. (See the IBM MQ messages section for more information about the rules of queue selection).

Related concepts

“Undelivered/unprocessed messages” on page 228
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 229
This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

z/OS *Undelivered/unprocessed messages*

This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

If an IGQ agent cannot deliver a message to the destination queue, the IGQ agent:

- Honors the MQRO_DISCARD_MSG report option (if the Report options field of the MQMD for the undelivered message indicates that it must) and discards the undelivered message.

- Attempts to place the undelivered message on to the dead letter queue for the destination queue manager, if the message has not already been discarded. The IGQ agent prefixes the message with a dead letter queue header (MQDLH).

If a dead letter queue is not defined, or if an undelivered message cannot be put to the dead letter queue, and if the undelivered message is:

- persistent, the IGQ agent backs out the current batch of persistent messages that it is processing, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 230.
- non-persistent, the IGQ agent discards the message and continues to process the next message.

If a queue manager in a queue sharing group is terminated before its associated IGQ agent has had time to process all its messages, the unprocessed messages remain on the SYSTEM.QSG.TRANSMIT.QUEUE until the queue manager is next started. The IGQ agent then retrieves and delivers the messages to the destination queues.

If the coupling facility fails before all the messages on the SYSTEM.QSG.TRANSMIT.QUEUE have been processed, any unprocessed non-persistent messages are lost.

IBM recommends that applications do not put messages directly to transmission queues. If an application does put messages directly to the SYSTEM.QSG.TRANSMIT.QUEUE, the IGQ agent might not be able to process these messages and they remain on the SYSTEM.QSG.TRANSMIT.QUEUE. Users then have to use their own methods to deal with these unprocessed messages.

z/OS *Report messages - Intra Group Queuing*

This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

Confirmation of arrival (COA)/confirmation of delivery (COD) report messages
COA and COD messages are generated by the queue manager, when intra-group queuing is used.

Expiry report messages
Expiry report messages are generated by the queue manager.

Exception report messages

Depending on the MQRO_EXCEPTION_* report option specified in the *Report options* field of the message descriptor for the undelivered message, the IGQ agent generates the required exception report and places it on the specified reply-to queue. Intra-group queuing can be used to deliver the exception report to the destination reply-to queue.

The persistence of the report message is the same as the persistence of the undelivered message. If the IGQ agent fails to resolve the name of the destination reply-to queue, or if it fails to put the reply message to a transmission queue (for subsequent transfer to the destination reply-to queue) it attempts to put the exception report to the dead letter queue of the queue manager on which the report message is generated. If it is not possible, then if the undelivered message is:

- persistent, the IGQ agent discards the exception report, backs out the current batch of messages, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 230.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the SYSTEM.QSG.TRANSMIT.QUEUE.

z/OS Security for intra-group queuing

This topic describes the security arrangements for intra-group queuing.

Queue manager attributes IGQAUT (IGQ authority) and IGQUSER (IGQ agent user ID) can be set to control the level of security checking that is performed when the IGQ agent opens destination queues.

Intra-group queuing authority (IGQAUT)

The IGQAUT attribute can be set to indicate the type of security checks to be performed, and hence to determine the userids to be used by the IGQ agent when it establishes the authority to put messages on to the destination queue.

The IGQAUT attribute is analogous to the PUTAUT attribute that is available on channel definitions.

Intra-group queuing user identifier (IGQUSER)

The IGQUSER attribute can be used to nominate a user ID to be used by the IGQ agent when it establishes the authority to put messages on to a destination queue.

The IGQUSER attribute is analogous to the MCAUSER attribute that is available on channel definitions.

z/OS Specific properties of intra-group queuing

This section describes the specific properties of intra-group queuing including Invalidation of object handles, self recovery and retry capability of the intra-group queuing agent, and the intra-group queuing agent and serialization.

Invalidation of object handles (MQRC_OBJECT_CHANGED)

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

Intra-group queuing introduces the following rules for object handle invalidation:

- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was ENABLED at open time, but intra-group queuing is found to be DISABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was not included in the name resolution path during open processing because intra-group queuing was DISABLED at open time, but intra-group queuing is found to be ENABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was enabled at open time, but the SYSTEM.QSG.TRANSMIT.QUEUE definition is found to have changed by put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.

Self recovery of the intra-group queuing agent

If the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent starts again.

Retry capability of the intra-group queuing agent

If the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry.

The IGQ agent observes short and long retry counts and intervals. The values for these counts and intervals, which cannot be changed, are as follows:

Table 20. Short and long retry counts and intervals values

Constant	Value
Short retry count	10
Short retry interval	60 seconds = 1 min
Long retry count	999,999,999

Table 20. Short and long retry counts and intervals values (continued)	
Constant	**Value**
Long retry interval	1200 seconds = 20 min

The intra-group queuing agent and serialization

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail.

If there is a failure of a queue manager in a queue sharing group while the IGQ agent is dealing with uncommitted messages on a shared queue or queues, the IGQ agent ends, and shared queue peer recovery takes place for the failing queue manager. Because shared queue peer recovery is an asynchronous activity, it leaves the possibility for the failing queue manager, and also the IGQ agent for that queue manager, to restart before shared queue peer recovery is complete. Which in turn leaves the possibility for any committed messages to be processed ahead of and out of sequence with the messages still being recovered. To ensure that messages are not processed out of sequence, the IGQ agent serializes access to shared queues by issuing the MQCONNX API call.

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail. An error message is issued and the IGQ agent is put into retry state. When queue manager peer recovery is complete, for example at the time of the next retry, the IGQ agent can start.

z/OS Storage management on z/OS

IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

Related concepts

"Page sets for IBM MQ for z/OS" on page 231
Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

"Storage classes for IBM MQ for z/OS" on page 232
A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

"Buffers and buffer pools for IBM MQ for z/OS" on page 234
IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

Related reference

"Where to find more information about storage management for IBM MQ for z/OS" on page 235
Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

z/OS Page sets for IBM MQ for z/OS

Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

A *page set* is a VSAM linear data set that has been specially formatted to be used by IBM MQ. Page sets are used to store most messages and object definitions.

The exceptions to this are global definitions, which are stored in a shared repository on Db2, and the messages on shared queues. These are not stored on the queue manager page sets. For more information about shared queues, see "Shared queues and queue sharing groups" on page 174, and for more information about global definitions, see Private and global definitions.

IBM MQ page sets can be up to 64 GB in size. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. Each queue manager must have its own page sets.

IBM MQ uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager. For normal operation of IBM MQ it is essential that page set zero does not become full, so do not use it to store messages.

To improve the performance of your system, you should also separate short-lived messages from long-lived messages by placing them on different page sets.

You must format page sets, and IBM MQ provides a FORMAT utility for this; see Formatting page sets (FORMAT). Page sets must also be defined to the IBM MQ subsystem.

IBM MQ for z/OS can be configured to expand a page set dynamically if it becomes full. IBM MQ continues to expand the page set if required until 123 logical extents exist, if there is sufficient disk storage space available. The extents can span volumes if the linear data set is defined in this way, however, IBM MQ cannot expand the page sets beyond 64 GB.

You cannot use page sets from one IBM MQ queue manager on a different IBM MQ queue manager, or change the queue manager name. If you want to transfer the data from one queue manager to another, you must unload all the objects and messages from the first queue manager and reload them onto another.

It is not possible to use page sets greater than 4 GB in a queue manager running a release earlier than V6. During the migration period, when it is likely that you might need to fall back to a previous release of code:

- Do not change page set 0 to be greater than 4 GB.
- Other page sets greater than 4 GB will be left offline when restarting a queue manager with a previous release.

For further information about migrating existing page sets capable of expanding beyond 4 GB, see Defining a page set to be larger than 4 GB.

It is possible for an administrator to dynamically add page sets to a running queue manager, or remove page sets from a running queue manager (except for page set zero). The DEFINE PSID command can run after the queue manager restart has completed, only if the command contains the DSN keyword.

z/OS Storage classes for IBM MQ for z/OS

A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

Introducing storage classes

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored on that page set.

Storage classes allow you to control where non-shared message data is stored for administrative, data set space and load management, or application isolation purposes. You can also use storage classes to define the XCF group and member name of an IMS region if you are using the IMS bridge (described in “IBM MQ and IMS” on page 287).

Shared queues do not use storage classes to obtain a page set mapping because the messages on them are not stored on page sets.

How storage classes work

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you specify the storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

More than one queue can use the same storage class, and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```
DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 71 on page 233, both storage classes ARC1 and MAXI are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

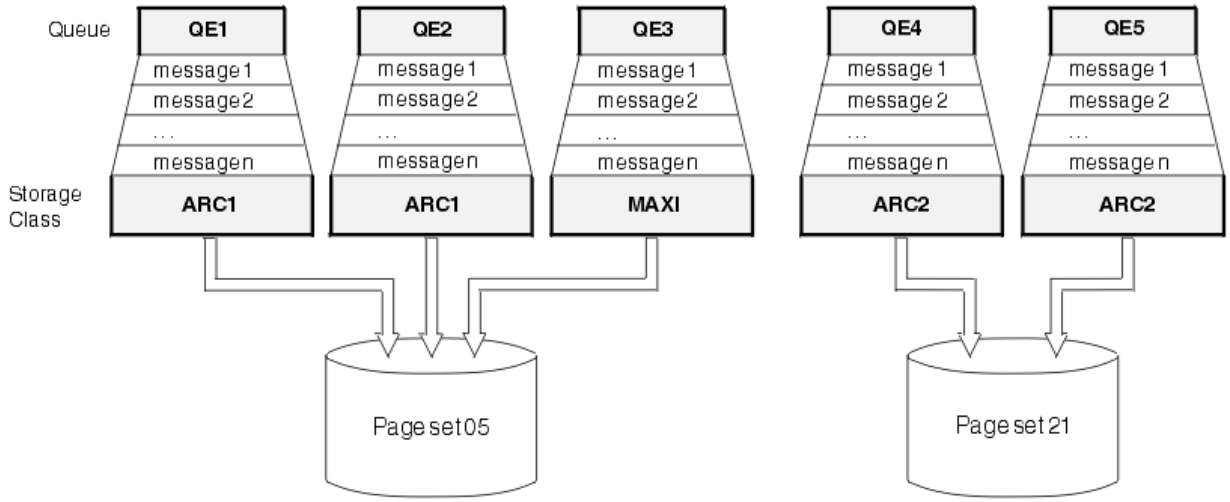


Figure 71. Mapping queues to page sets through storage classes

If you define a queue without specifying a storage class, IBM MQ uses a default storage class.

If a message is put on a queue that names a nonexistent storage class, the application receives an error. You must alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.

You can change a storage class only when:

- All queues that use this storage class are empty, and have no uncommitted activity.
- All queues that use this storage class are closed.

z/OS Buffers and buffer pools for IBM MQ for z/OS

IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

For efficiency, IBM MQ uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. This caching activity is controlled by a buffer manager, which is a component of IBM MQ.

The buffers are organized into *buffer pools*. You can define up to 100 buffer pools (0 through 99) for each queue manager.

You are recommended to use the minimal number of buffer pools consistent with the object and message type separation outlined in Figure 72 on page 234, and any data isolation requirements your application might have. Each buffer is 4 KB long. Buffer pools use 31 bit storage by default, in this mode, the maximum number of buffers is determined by the amount of 31 bit storage available in the queue manager address space; do not use more than about 70% for buffers. Alternatively, buffer pool storage allocation can be made from 64 bit storage (use the LOCATION attribute of the **DEFINE BUFFPOOL** command). Using LOCATION(ABOVE) so that 64 bit storage is used has two benefits. Firstly, there is much more 64 bit storage available so buffer pools can be much bigger, and secondly, 31 bit storage is made available for use by other functions. Typically, the more buffers you have, the more efficient the buffering and the better the performance of IBM MQ.

Figure 72 on page 234 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

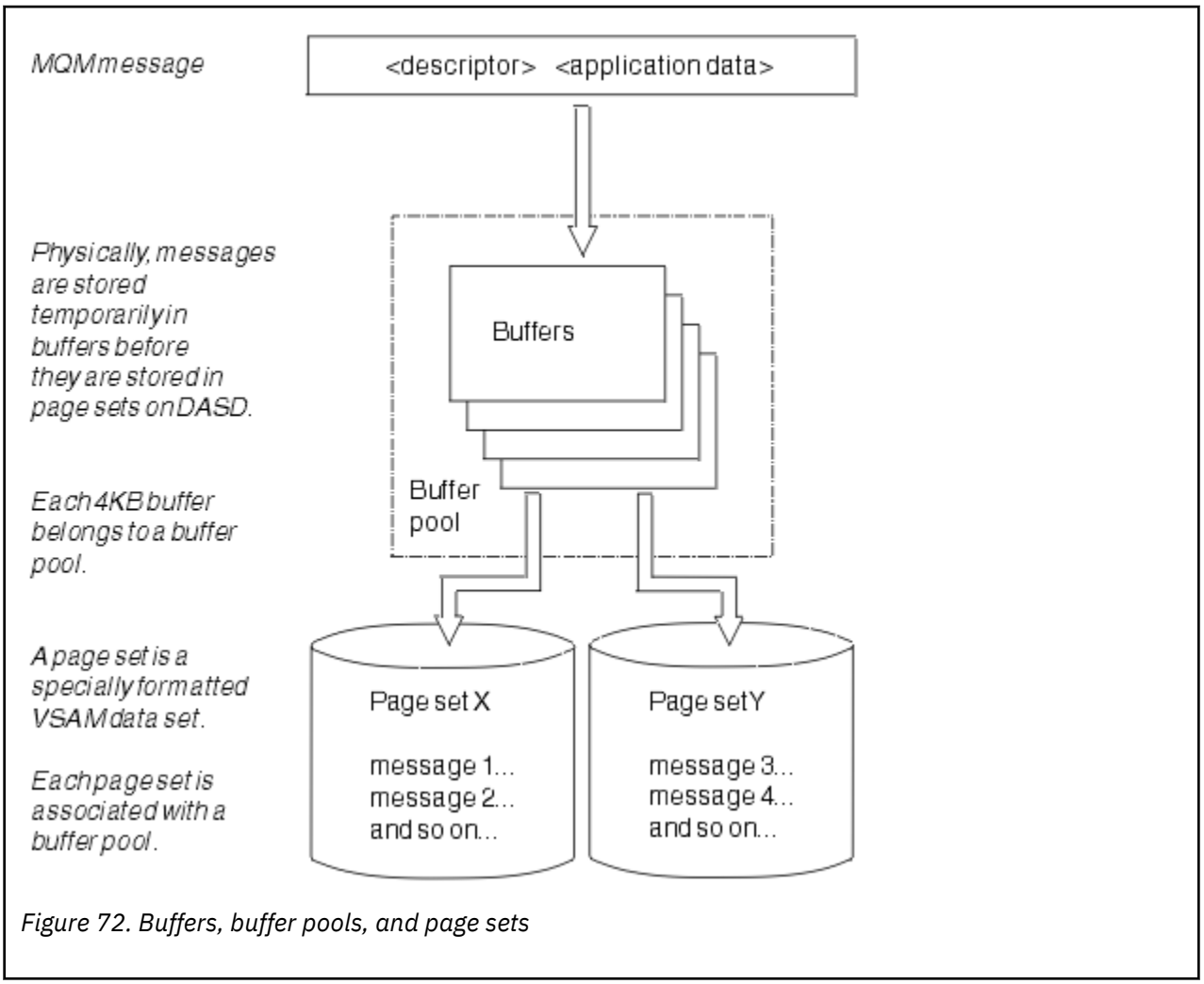


Figure 72. Buffers, buffer pools, and page sets

You can dynamically issue commands to modify buffer pool size, and location, using the **ALTER BUFFPOOL** command. Page sets can be dynamically added by using the **DEFINE PSID** command, or deleted by using the **DELETE PSID** command.

If a buffer pool is too small, IBM MQ issues message CSQP020E. You can then dynamically add more buffers to the affected buffer pool (note that you may have to remove buffers from other buffer pools to do this).

You specify the number of buffers in a pool with the **DEFINE BUFFPOOL** command, and you can dynamically resize buffer pools with the **ALTER BUFFPOOL** command. You determine the current number of buffers in a pool dynamically by displaying a page set that uses the buffer pool, using the **DISPLAY USAGE** command.

For performance reasons, do not put messages and object definitions in the same buffer pool. Use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, keep short-lived messages and long-lived messages in different buffer pools and therefore on different page sets, and in different queues.

The **DEFINE BUFFPOOL** command cannot be used after restart to create a new buffer pool. Instead, if a **DEFINE PSID** command uses the DSN keyword, it can explicitly identify a buffer pool that is not currently defined. That new buffer pool will then be created.

z/OS Where to find more information about storage management for IBM MQ for z/OS

Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

You can find more information about the topics in this section from the following sources:

Table 21. Where to find more information about storage management

Topic	Where to look
How much storage you need	Planning your storage and performance requirements on z/OS
How large to make your page sets and buffer pools	Plan your page sets and buffer pools
Managing page sets	Managing page sets
MQSC commands	The MQSC commands

z/OS Logging in IBM MQ for z/OS

IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

The log does not contain information for statistics, traces, or performance evaluation. For further details about the statistical and monitoring information that IBM MQ collects, see Monitoring and statistics.

For more information about logging, see the following topics:

- “Log files in IBM MQ for z/OS” on page 236
- “How the log is structured” on page 240
- “How the IBM MQ for z/OS logs are written” on page 240
- “Larger log Relative Byte Address” on page 243

- “The bootstrap data set” on page 244

Related tasks

Planning your logging environment

Setting logs using the system parameter module

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS Log files in IBM MQ for z/OS

Log files contain information needed for transaction recovery. Active log files can be archived so that you can keep log data for a long period.

What is a log file

IBM MQ records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- IBM MQ objects, such as queues
- The IBM MQ queue manager

The active log comprises a collection of data sets (up to 310) which are used cyclically.

You can enable log archiving so that when an active log fills a copy is made in an archive data set. Using archiving allows you to keep log data for an extended period. If you do not use archiving, the logs wrap and earlier data is overwritten. To recover a page set, or recover data in a CF structure, you need log data from when the backup of the page set or structure was taken. An archive log can be created on disk or on tape.

Archiving

Because the active log has a fixed size, IBM MQ copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct-access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, IBM MQ uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. You can catalog each data set using the z/OS integrated catalog facility (ICF).

Archiving is an essential component of IBM MQ recovery. If a unit of recovery is a long-running one, log records within that unit of recovery might be found in the archive log. In this case, recovery requires data from the archive log. However, if archiving is disabled, the active log with new log records wraps, overwriting earlier log records. This means that IBM MQ might not be able to back out the unit of recovery and messages might be lost. The queue manager then terminates abnormally.

Therefore, in a production environment, **never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment can you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro, which is described in Using CSQ6LOGP.

To help prevent problems with unplanned long-running units of work, IBM MQ issues a message (CSQJ160I or CSQJ161I) if a long-running unit of work is detected during active log offload processing.

Dual logging

In dual logging, each log record is written to two different active log data sets to minimize the likelihood of data loss problems during restart.

You can configure IBM MQ to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. Each active log data set is a single-extent VSAM linear data set (LDS). With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart.

Log shunting

Log shunting causes the log records for some units of work to be written further down the log. This reduces the amount of log data that must be read at queue manager restart, or backout, for long running or long term in-doubt units of work.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This technique is known as *shunting*. When the whole of the unit of work has been processed, the unit of work is in a *shunted* state. Any backout or restart activity relating to the shunted unit of work can use the shunted log records instead of using the original unit of work log records.

Detecting a long-running unit of work is a function of the checkpoint process. At checkpoint time, each active unit of work is checked to establish whether it needs to be shunted. If the unit of work has been through two prior checkpoints since it was created, or since it was last shunted, the unit of work is suitable to be shunted. This means that a single unit of work might be shunted more than once. This is known as a *multi-shunted* unit of work.

A unit of work is shunted every three checkpoints. However the checkpoint is performed asynchronously to the log-switch (or the writing of the log record which caused LOGLOAD to be exceeded).

There is only a single checkpoint taking place at a time, so there might be multiple log-switches before a checkpoint completes.

This means that if there are not enough active logs, or if they are too small, then shunting of a large unit of work might not complete before all the logs are filled.

Message CSQR027I results if shunting is unable to complete.

If log archiving is turned off, ABEND 5C6 with reason 00D1032A occurs if there is an attempt to back out the unit of work for which shunting failed. To avoid this problem you should use OFFLOAD=YES.

Log shunting is always active, and runs whether log archiving is enabled or not.

Note: Although all log records for a unit of work are shunted, the entire content of each record is not shunted, only the part that is necessary for backout. This means that the amount of log data written is kept to a minimum, and that shunted records cannot be used if a page set failure occurs. A long running unit of work is one that has been running for more than three queue manager checkpoints.

For more information about log shunting, see Managing the logs.

Log compression

You can configure IBM MQ for z/OS to compress and decompress log records as they are written and read from the log data set.

Log compression can be used to reduce the amount of data written to the log for persistent messages on private queues. The amount of compression that is achieved depends on the type of data contained within messages. For example, Run Length Encoding (RLE) works by compacting repeated instances of bytes which can give good results efficiently for structured or record oriented data.

Attention: Persistent messages that are being put to a shared queue are not subject to log compression.

You can use fields within the Log manager section of the System Management Facility 115 (SMF) records to monitor how much data compression is achieved. For more information about SMF, see Using the System Management Facility and Accounting and statistics messages.

Log compression increases the processor utilization of the system. You should only consider using compression if throughput of your queue manager is constrained by the IO bandwidth writing to the log data sets or you are constrained by the disk storage needed to hold log data sets. If you are using shared queues then IO bandwidth constraints can be relieved by adding additional queue managers to the queue sharing group and distributing the workload across more queue managers.

The log compression option can be enabled and disabled as required without the need to stop and restart the queue manager. The queue manager can read any compressed log records regardless of the current log compression setting.

The queue manager supports 3 settings for log compression.

NONE
No log data compression is used. This is the default value.

RLE
Log data compression is performed using run-length encoding (RLE).

ANY
Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. This option results in RLE compression.

You can control the compression of log records using one of the following:

- The SET and DISPLAY LOG commands in MQSC; see SET LOG and DISPLAY LOG
- The Set Log and Inquire Log functions in the PCF interface; see Set log and Inquire log
- The CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

In addition the Log Print utility CSQ1LOGP has support for expanding any compressed log records.

Log data

The log can contain up to 18 million million million ($1.8*10^{19}$) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The RBA is referenced by a 6-byte or 8-byte field giving a total addressable range of 2^{48} bytes, or 2^{64} bytes, depending on whether 6-byte or 8-byte log RBAs are in use.

However, when IBM MQ detects that the used range is beyond F00000000000 (if 6-byte RBAs are in use) or FFFF800000000000 (if 8-byte log RBAs are in use), messages CSQI045, CSQI046, CSQI047, and CSQJ032 are issued, warning you to reset the log RBA.

If the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) the queue manager terminates with reason code 00D10257.

Once the warning messages about the used log range are being issued, you should plan a queue manager outage during which the queue manager can be converted to use 8-byte log RBAs, or the log can be reset. The procedure to reset the log is documented in Resetting the queue manager's log.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the procedure documented in Implementing the larger log Relative Byte Address.

The log consists of *log records*, each of which is a set of log data treated as a single unit. A log record is identified either by the RBA of the first byte of its header, or by its log record sequence number (LRSN). The RBA or LRSN uniquely identifies a record that starts at a particular point in the log.

Whether you use the RBA or LRSN to identify log points depends on whether you are using queue sharing groups. In a queue-sharing environment, you cannot use the relative byte address to uniquely identify a log point, because multiple queue managers can update the same queue at the same time, and each has its own log. To solve this, the log record sequence number is derived from a timestamp value, and does not necessarily represent the physical displacement of the log record within the log.

Each log record has a header that gives its type, the IBM MQ subcomponent that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are four types of log record, described under the following headings:

- Unit of recovery log records
- Checkpoint records
- Page set control records
- CF structure backup records

Unit of recovery log records

Most of the log records describe changes to IBM MQ queues. All such changes are made within units of recovery.

IBM MQ uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that the queue manager has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be reapplied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, IBM MQ takes periodic checkpoints during normal operation. These occur as follows:

- When a predefined number of log records has been written. This number is defined by the checkpoint frequency operand called LOGLOAD of the system parameter macro CSQ6SYSP, described in Using CSQ6SYSP.
- At the end of a successful restart.
- At normal termination.
- Whenever IBM MQ switches to the next active log data set in the cycle.

At the time a checkpoint is taken, IBM MQ issues the DISPLAY CONN command (described in DISPLAY CONN) internally so that a list of connections currently in doubt is written to the z/OS console log.

Page set control records

These records register the page sets and buffer pools known to the IBM MQ queue manager at each checkpoint, and record information about the log ranges required to perform media recovery of the page set at the time of the checkpoint.

Certain dynamic changes to page sets and buffer pools are also written as page set control records, so that the changes can be recovered and automatically reinstated at the next queue manager restart.

CF structure backup records

These records hold data read from a coupling facility list structure in response to a BACKUP CFSTRUCT command. In the unlikely event of a coupling facility structure failure, these records are used, together with unit of recovery records, by the RECOVER CFSTRUCT command to perform media recovery of the coupling facility structure to the point of failure.

Related tasks

Implementing the larger log Relative Byte Address

z/OS How the log is structured

Use this topic to understand the terminology used to describe log records.

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information logged at a particular time forms a *logical* record, with a length that varies independently of the space available in the CI. So one physical record might contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term *log record* refers to the *logical* record, regardless of how many *physical* records are needed to store it.

z/OS How the IBM MQ for z/OS logs are written

Use this topic to understand how IBM MQ processes log file records.

IBM MQ writes each log record to a DASD data set called the *active log*. When the active log is full, IBM MQ copies its contents to a DASD or tape data set called the *archive log*. This process is called *offloading*.

Figure 73 on page 241 illustrates the process of logging. Log records typically go through the following cycle:

1. IBM MQ notes changes to data and significant events in recovery log records.
2. IBM MQ processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM Controls Intervals (CI). Each log record is identified by a relative byte address in the range zero through $2^{64}-1$.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically offloaded to a new archive log data set.

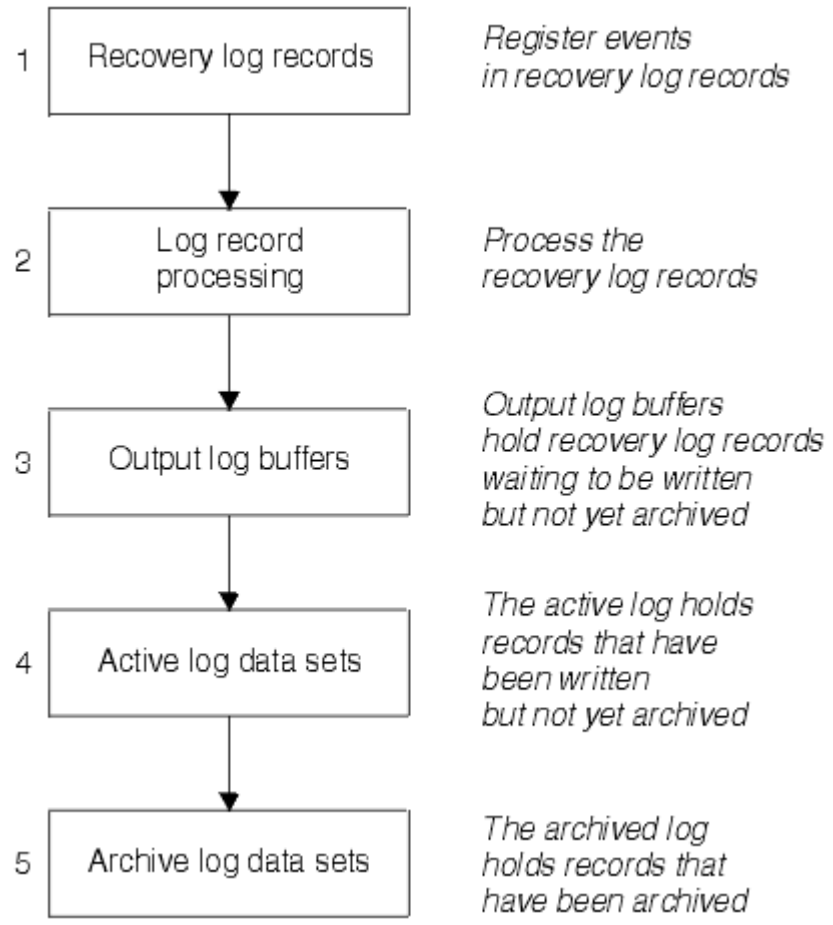


Figure 73. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as a commit point, or when an IBM MQ BACKUP CFSTRUCT command is issued.

When the queue manager is initialized, the active log data sets named in the BSDS are dynamically allocated for exclusive use by the queue manager and remain allocated exclusively to IBM MQ until the queue manager terminates.

Dynamically adding log data sets

It is possible to dynamically define new active log data sets while the queue manager is running. This feature alleviates the problem of a queue manager hang when archiving is not able to offload active logs due to a transient problem. See the DEFINE LOG command for more information.

Note: To redefine or remove active logs you must terminate and restart the queue manager.

IBM MQ and Storage Management Subsystem

IBM MQ parameters enable you to specify Storage Management Subsystem (MVS™/DFP SMS) storage classes when allocating IBM MQ archive log data sets dynamically. IBM MQ initiates the archiving of log data sets, but you can use SMS to perform allocation of the archive data set.

Related reference

"When the IBM MQ for z/OS archive log is written" on page 242
Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

z/OS When the IBM MQ for z/OS archive log is written

Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 74 on page 242.

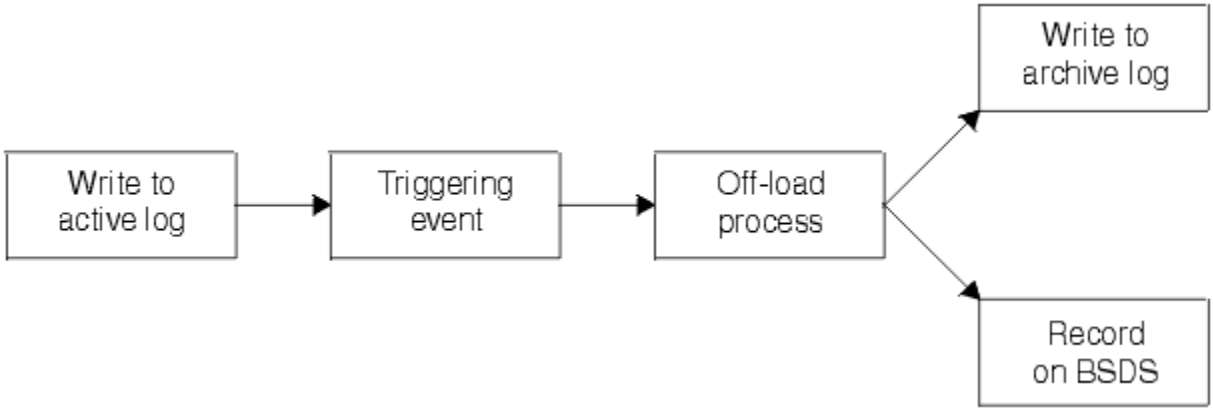


Figure 74. The offloading process

Triggering the offloading process

The offload process of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.
- Using the MQSC ARCHIVE LOG command.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that was not written becomes the first record of the new data set. Offloading is triggered for the truncated data set as for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

Message CSQJ110E is issued when the last available active log is 5% full and at 5% increments thereafter, stating the percentage of the log's capacity that is in use. If all the active logs become full, IBM MQ stops processing, until offloading occurs, and issues this message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The offload process

When all the active logs become full, IBM MQ runs the offloading process and halts processing until the offloading process has been completed. If the offload processing fails when the active logs are full, IBM MQ abends.

When an active log is ready to be offloaded, a request is sent to the z/OS console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR logging option (for further information, see Using CSQ6ARVP) determines whether the request is received. If you are using tape for offloading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the offload process. It does not affect IBM MQ performance unless the operator delays the response for so long that IBM MQ runs out of active logs.

The operator can respond by canceling the offload process. In this case, if the allocation is for the first copy of dual archive data sets, the offload process is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Interruptions and errors while offloading

A request to stop the queue manager does not take effect until offload processing has finished. If IBM MQ fails while offloading is in progress, offloading begins again when the queue manager is restarted.

Messages during offload processing

Offloaded messages are sent to the z/OS console by IBM MQ and the offloading process. You can use these messages to find the RBA ranges in the various log data sets.

z/OS Larger log Relative Byte Address

This function improves the availability of the queue manager by increasing the period of time before you have to reset the log.

Recovery data is written to the log so that persistent messages are available when the queue manager is restarted. The term log Relative Byte Address (log RBA) is used to refer to the location of data as an offset from the beginning of the log.

Before IBM MQ 8.0, the 6 byte log RBA could address up to 256 terabytes of data. Before this quantity of log records has been written, you have to reset the queue manager's log by following the procedure documented in Resetting the queue manager's log.

Resetting the logs of queue managers is not a quick process, and can require an extended outage, due to the need to reset the page sets as part of the process. For a high use queue manager this operation might typically be done once a year.

From IBM MQ 8.0, the log RBA can be 8 bytes long and the queue manager can now address over 64,000 times as much data (16 exabytes) before the log RBA needs to be reset. The impact of using the larger log RBA is that the size of the log data written increases by a few bytes.

When is this function enabled?

Queue managers created at IBM MQ 9.3.0 or later already have this function enabled.

If the current log RBA is approaching the end of the log RBA range, consider converting the queue manager to use an 8 byte log RBA rather than resetting the queue manager's log. Converting a queue manager to use 8 byte log RBAs requires a shorter outage than resetting the log, and significantly increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6 byte or 8 byte log RBAs are in use.

How is this function enabled?

8 byte log RBA is enabled by starting the queue manager with a version 2 format BSDS. In summary, this is achieved by:

1. Ensuring that all queue managers in the queue sharing group meet the requirements for enabling 8 byte log RBA
2. Shutting down the queue manager cleanly
3. Running the BSDS conversion utility to create a copy of the BSDS in version 2 format.
4. Restarting the queue manager with the converted BSDS.

Once a queue manager has been converted to use 8 byte log RBAs, it cannot go back to using 6 byte log RBA.

See Implementing the larger log Relative Byte Address for the detailed procedure on how to enable 8 byte log RBAs.

Related tasks

Planning to increase the maximum addressable log range

Related reference

The BSDS conversion utility (CSQJUCNV)

z/OS The bootstrap data set

The bootstrap data set is required by IBM MQ as a mechanism to reference log data sets, and log records. This information is required during normal processing, and restart recovery.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by IBM MQ. It contains the following:

- An inventory of all active and archived log data sets known to IBM MQ. IBM MQ uses this inventory to:
  - Track the active and archived log data sets
  - Locate log records so that it can satisfy log read requests during normal processing
  - Locate log records so that it can handle restart processing

  IBM MQ stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each portion of the log held in that data set.
- A *wrap-around* inventory of all recent IBM MQ activity. This is needed if you have to restart the queue manager.

The BSDS is required if the queue manager has an error and you have to restart it. IBM MQ **must** have a BSDS. To minimize the likelihood of problems during a restart, you can configure IBM MQ with dual BSDSs, each recording the same information. Using dual BSDSs is known as running in *dual mode*. If possible, place the copies on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. Use dual BSDSs rather than dual write to DASD.

The BSDS is set up when IBM MQ is customized and you can manage the inventory using the change log inventory utility (CSQJU003). For more information about this utility, see IBM MQ for z/OS の管理. It is referenced by a DD statement in the queue manager startup procedure.

Normally, IBM MQ keeps duplicate copies of the BSDS. If an I/O error occurs, it deallocates the failing copy and continues with a single BSDS. You can restore dual-mode operation, this is described in the IBM MQ for z/OS の管理.

The active logs are first registered in the BSDS when IBM MQ is installed. You cannot replace the active logs without terminating and restarting the queue manager.

Archive log data sets are allocated dynamically. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets (IBM MQ does not have an automated method). Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiry date.

You can use the following MQSC command to determine the extent of the log, and the name of the active or archive log data set holding the earliest log RBA, required for various types of media or queue manager recovery:

```
DISPLAY USAGE TYPE(DATASET)
```

If the system parameter module specifies that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

The BSDS version

The format of the BSDS varies according to its version. Increasing the version of the BSDS allows new features to be used. The following BSDS versions are supported by IBM MQ:

Version 1
Supported by all releases of IBM MQ. A version 1 BSDS supports 6-byte log RBA values.

Version 2
Supported by IBM MQ 8.0 and later. A version 2 BSDS enables 8-byte log RBA values, and up to 310 data sets in each active log copy.

Enabled by default for queue managers created at IBM MQ 9.3.0 or later.

Version 3
Supported by IBM MQ 8.0 and later. The BSDS is automatically converted to version 3, from version 2, when more than 31 data sets are added to either active log copy.

You can determine the version of a BSDS by running the print log map utility (CSQJU004). To convert a BSDS from Version 1 to Version 2, run the BSDS conversion utility (CSQJUCNV).

See “Larger log Relative Byte Address” on page 243 for more information on 6-byte and 8-byte log RBAs.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name
CSQ.ARCHLOG1.E00186.T2336229. *A* 0000001

BSDS copy name
CSQ.ARCHLOG1.E00186.T2336229. *B* 0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the offloading to the new archive log data set continues without the BSDS copy.

z/OS System definition on z/OS

IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

Setting system parameters

In IBM MQ for z/OS, a system parameter module controls the logging, archiving, tracing, and connection environments that IBM MQ uses in its operation. The system parameters are specified by three assembler macros, as follows:

CSQ6SYSP
System parameters, including setting the connection and tracing environment.

CSQ6LOGP
Logging parameters.

CSQ6ARVP
Log archive parameters.

Default parameter modules are supplied with IBM MQ for z/OS. If these do not contain the values that you want to use, you can create your own parameter modules using the sample supplied with IBM MQ. The sample is `thlqual.SCSQPROC(CSQ4ZPRM)`.

You can alter some system parameters while a queue manager is running. See the SET SYSTEM, SET LOG, and SET ARCHIVE commands in The MQSC commands.

For more information about defining , see the following topics:

- “Defining system objects for IBM MQ for z/OS” on page 246
- “Tuning your queue manager on IBM MQ for z/OS” on page 251
- “Sample definitions supplied with IBM MQ for z/OS” on page 252

Related concepts
Customize the sample initialization input data sets
Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS
Related tasks
Administering z/OS
Configuring clusters
Monitoring IBM MQ

z/OS Defining system objects for IBM MQ for z/OS

IBM MQ for z/OS requires additional predefined objects for publish/subscribe applications, cluster, and channel control and other system administration functions.

The system objects required by IBM MQ for z/OS can be divided into the following categories:

- Publish/subscribe objects
- System default objects
- System command objects
- System administration objects
- Channels queues
- Cluster queues

- Queue sharing group queues
- Storage classes
- Defining the system object dead-letter queue
- Default transmission queue
- Internal queues
- “Channel authentication queue” on page 250

Publish/subscribe objects

There are several system objects that you need to define before you can use publish/subscribe applications with IBM MQ for z/OS. Sample definitions are supplied with IBM MQ to help you define these objects. These samples are described in CSQ4INSG.

To use publish/subscribe you need to define the following objects:

- A local queue called SYSTEM.RETAINED.PUB.QUEUE, which is used to hold a copy of each retained publication in the queue manager. Each full topic name could have up to one retained publication stored on this queue. If your applications will make use of retained publications on many different topics, or if your retained publication messages are large messages, the requirements for storage for this queue should be carefully planned, including assigning it to its own page set if the storage requirements for it are large. To improve performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.SUBSCRIBER.QUEUE, which is used to hold a persistent copy of the durable subscriptions in the queue manager. To improve performance, you should define this queue with an index type of CORRELID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.MODEL.QUEUE, which is used as a model for managed durable subscriptions.
- A local queue called SYSTEM.NDURABLE.MODEL.QUEUE, which is used as a model for managed non-durable subscriptions.
- A namelist called SYSTEM.QPUBSUB.QUEUE.NAMELIST, which contains a list of queue names monitored by the queued publish/subscribe interface.
- A namelist called SYSTEM.QPUBSUB.SUBPOINT.NAMELIST, which contains a list of topic objects used by the queued publish/subscribe interface to match topic objects to subscription points.
- A topic called SYSTEM.BASE.TOPIC, which is used as a base topic for resolving attributes.
- A topic called SYSTEM.BROKER.DEFAULT.STREAM, which is the default stream used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.DEFAULT.SUBPOINT, which is the default RFH2 subscription point used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.ADMIN.STREAM, which is the admin stream used by the queued publish/subscribe interface.
- A subscription called SYSTEM.DEFAULT.SUB, which is a default subscription object used to provide default values on DEFINE SUB commands.

System default objects

System default objects are used to provide default attributes when you define an object and do not specify the name of another object to base the definition on.

The names of the default system object definitions begin with the characters "SYSTEM.DEFAULT" or "SYSTEM.DEF." For example, the system default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE.

These objects define the system defaults for the attributes of these IBM MQ objects:

- Local queues

- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels
- Storage classes
- Authentication information

Shared queues are a special type of local queue, so when you define a shared queue, the definition is based on the SYSTEM.DEFAULT.LOCAL.QUEUE. You need to remember to supply a value for the Coupling Facility structure name because one is not specified in the default definition. Alternatively, you could define your own default shared queue definition to use as a basis for shared queues so that they all inherit the required attributes. Remember that you need to define a shared queue on one queue manager in the queue sharing group only.

System command objects

The names of the system command objects begin with the characters SYSTEM.COMMAND. You must define these objects before you can use the IBM MQ operations and control panels to issue commands to an IBM MQ subsystem.

There are two system command objects:

1. The system-command input queue is a local queue on which commands are put before they are processed by the IBM MQ command processor. It must be called SYSTEM.COMMAND.INPUT. An alias named SYSTEM.ADMIN.COMMAND.QUEUE should also be defined, for compatibility with IBM MQ for Multiplatforms, and for use by the IBM MQ Console and administrative REST API.
2. SYSTEM.COMMAND.REPLY.MODEL is a model queue that defines the system-command reply-to queue.

There are two extra objects for use by the IBM MQ Explorer:

- SYSTEM.MQEXPLORER.REPLY.MODEL queue
- SYSTEM.ADMIN.SVRCONN channel

SYSTEM.REST.REPLY.QUEUE is the reply queue used by the IBM MQ administrative REST API.

Commands are normally sent using nonpersistent messages so both the system command objects should have the DEFPSIST(NO) attribute so that applications using them (including the supplied applications like the utility program and the operations and control panels) get nonpersistent messages by default. If you have an application that uses persistent messages for commands, set the DEFTYPE(PERMDYN) attribute for the reply-to queue, because the reply messages to such commands are persistent.

System administration objects

The names of the system administration objects begin with the characters SYSTEM.ADMIN.

There are seven system administration objects:

- The SYSTEM.ADMIN.CHANNEL.EVENT queue
- The SYSTEM.ADMIN.COMMAND.EVENT queue
- The SYSTEM.ADMIN.CONFIG.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.QMGR.EVENT queue

- The SYSTEM.ADMIN.TRACE.ROUTE.QUEUE queue
- The SYSTEM.ADMIN.ACTIVITY.QUEUE queue

Channels queues

To use distributed queuing, you need to define the following objects:

- A local queue with the name SYSTEM.CHANNEL.SYNCQ, which is used to maintain sequence numbers and logical units of work identifiers (LUWID) of channels. To improve channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue with the name SYSTEM.CHANNEL.INITQ, which is used for channel commands.

You cannot define these queues as shared queues.

Cluster queues

To use IBM MQ clusters, you need to define the following objects:

- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons, you should define this queue with an index type of CORRELID (as shown in the sample queue definition).

These queues typically contain large numbers of messages.

You cannot define these queues as shared queues.

Queue sharing group queues

To use shared channels and intra-group queuing, you need to define the following objects:

- A shared queue with the name SYSTEM.QSG.CHANNEL.SYNCQ, which is used to hold synchronization information for shared channels.
- A shared queue with the name SYSTEM.QSG.TRANSMIT.QUEUE, which is used as the transmission queue for intra-group queuing. If you are running in a queue sharing group, you must define this queue, even if you are not using intra-group queuing.

Storage classes

You are recommended to define the following six storage classes. You must define four of them because they are required by IBM MQ. The other storage class definitions are recommended because they are used in the sample queue definitions.

DEFAULT (required)
This storage class is used for all message queues that are not performance critical and that don't fit in to any of the other storage classes. It is also the supplied default storage class if you do not specify one when defining a queue.

NODEFINE (required)
This storage class is used if the storage class specified when you define a queue is not defined.

REMOTE (required)
This storage class is used primarily for transmission queues, that is, system related queues with short-lived performance-critical messages.

SYSLNGLV
This storage class is used for long-lived, performance-critical messages.

SYSTEM (required)
This storage class is used for performance critical, system related message queues, for example the SYSTEM.CHANNEL.SYNQ and the SYSTEM.CLUSTER.* queues.

SYSVOLAT
This storage class is used for short-lived, performance-critical messages.

You can modify their attributes and add other storage class definitions as required.

Defining the system object dead-letter queue

The dead-letter queue is used if the message destination is not valid. IBM MQ puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, you should regard it as essential, especially if you are using distributed queuing or one of the IBM MQ bridges.

Do **not** define the dead-letter queue as a shared queue. A put to a local queue on one queue manager might get put to the dead letter queue. If the dead letter queue was a shared queue, a dead letter queue handler on a different system could process the message and put it on a queue with the same name, but because this is on a different queue manager, it would be the wrong queue, or have a different security profile. If the queue did not exist, it would fail to reprocess it.

If you decide to define a dead-letter queue, you must also tell the queue manager its name. To do this use the ALTER QMGR DEADQ(*queue-name*) command. For more information see Displaying and altering queue manager attributes.

Default transmission queue

The default transmission queue is used when no other suitable transmission queue is available for sending messages to another queue manager. If you define a default transmission queue, you must also define a channel to serve the queue. If you do not do this, messages that are put on to the default transmission queue are not transmitted to the remote queue manager and remain on the queue.

If you decide to define a default transmission queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command.

Internal queues

- **Pending data queue**
  - A queue defined for internal use, SYSTEM.PENDING.DATA.QUEUE, supports the use of durable subscriptions in a JMS publish/subscribe environment.
- **JMS 2.0 delivery delay staging queue**
  - If the delivery delay functionality provided by JMS 2.0 is used then an internal staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, must be defined. This queue is used by the queue manager to temporarily store messages sent with a non-zero delivery delay until the delivery delay is completed, and the message is put to its target destination. A sample definition for this queue is provided, commented out, in CSQ4INSG.
  - When you define the SYSTEM.DDELAY.LOCAL.QUEUE queue, you must set the STGCLASS, MAXMSGL and MAXDEPTH attributes for the anticipated number of messages that will be sent with a delivery delay. Additionally when defining the SYSTEM.DDELAY.LOCAL.QUEUE queue ensure that only the queue manager can put messages to this queue. Care should be taken to ensure that no user identifier has the authority to put messages to this queue.

Channel authentication queue

For internal use of channel authentication the SYSTEM.CHLAUTH.DATA.QUEUE queue is required. Sample definitions are supplied with IBM MQ to help you define these objects. This sample is described in CSQ4INSA, which also defines some default rules.

z/OS Tuning your queue manager on IBM MQ for z/OS

There are few simple steps that you can take to ensure that your queue manager is tuned to avoid basic performance problems.

There are a number of ways in which you can improve the performance of your queue manager, which are controlled by queue manager attributes set by the ALTER QMGR command. This section contains information about how you can do this by setting the maximum number of messages allowed on the queue manager, or by performing 'housekeeping' on the queue manager. IBM MQ SupportPac MP16 - IBM MQ for z/OS キャパシティー・プランニング & のチューニング gives more information on performance and tuning.

Syncpoints

One of the roles of the queue manager is syncpoint control within an application. An application constructs a unit of work containing any number of MQPUT or MQGET calls terminated with an MQCMIT call.

As the number of MQPUT or MQGET calls within the scope of one MQCMIT increases, the performance cost of the commit increases significantly. Applications, in general, should be designed to not MQPUT/MQGET a large number of messages in a single synchpoint.

You can administratively limit the number of messages within any single syncpoint by using the MAXUMSGS queue manager attribute. If an application exceeds this limit it receives MQRC_SYNCPOINT_LIMIT_REACHED on the MQPUT,MQPUT1, or MQGET call which exceeds the limit. The application should then issue MQCMIT or MQBACK as appropriate.

The default value of MAXUMSGS is 10000. This value can be lowered if you want to enforce a lower limit, which can also help protect against looping applications. Before reducing MAXUMSGS make sure you understand your existing applications to ensure they do not exceed the limit, or can tolerate the MQRC_SYNCPOINT_LIMIT_REACHED return code

Expired messages

Messages that have expired are discarded by the next appropriate MQGET call. However, if no such call occurs, the expired messages are not discarded, and, for some queues, particularly those where message retrieval is done by MessageId, CorrelId, or GroupId and the queue is indexed for performance, many expired messages can accumulate. The queue manager can periodically scan any queue for expired messages, which are then deleted. You can choose how often this scanning takes place, if at all. There are two ways of doing this:

Explicit request
: You can control which queues are scanned and when. Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want to be scanned.

Periodic scan
: You can specify an expiry interval in the queue manager object by using the EXPRYINT attribute. The queue manager maintains information about the expired messages on each queue, and knows at what time a scan for expired messages is worthwhile. Each time that the EXPRYINT interval is reached, the queue manager looks for candidate queues that are worth scanning for expired messages, and scans only those queues that it deems to be worthwhile. It does not scan all queues. This avoids any processor time being wasted on unnecessary scans.

Shared queues are only scanned by one queue manager in the queue sharing group. Generally, the first queue manager to restart or the first to have the EXPRYINT set performs the scan.

Note: You must set the same EXPRYINT value for all queue managers within a queue sharing group.

z/OS Sample definitions supplied with IBM MQ for z/OS

Use this topic as a reference for the sample JCL, and code supplied with IBM MQ for z/OS.

The following sample definitions are supplied with IBM MQ in the thlqual.SCSQPROC library. You can use them to define the system objects and to customize your own objects. You can include some of them in the initialization input data sets (described in Initialization commands).

Table 22. IBM MQ sample definitions for system objects

Initialization input data set	Sample name
CSQINP1	CSQ4INP1 CSQ4INPR
CSQINP2	CSQ4INSA CSQ4INYS [1] CSQ4INSX CSQ4INSG CSQ4INSR CSQ4INSS CSQ4INSJ CSQ4INSM CSQ4INYG CSQ4INYR CSQ4INYC CSQ4INYD CSQ4INSC
CSQINPT	CSQ4INST CSQ4INYT
Other	CSQ4DISP CSQ4INPX CSQ4IVPQ CSQ4IVPG CSQ4MSTR CSQ4MSRR CSQ4QMIN

Note:

1. The order of these sample definitions is important: an error occurs if INYS, INSX, and INSG are ordered incorrectly.

CSQINP1 samples

Use the sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) when you are using one page set for each class of message, or thlqual.SCSQPROC(CSQ4INPR) when using multiple page sets for the major classes of message. It contains definitions of buffer pools, page set to buffer pool associations, and an

ALTER SECURITY command. Include the sample in the CSQINP1 concatenation of your queue manager started task procedure.

CSQINP2 samples

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for the following system objects for general use:

- System default objects
- System command objects
- System administration objects
- Other objects for system use

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across queue manager shutdown and restart. You must not change the object names, but you can change their attributes if required.

When the following conditions are met, one message is put to the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue (even if publish subscribe is not active):

- The QMGR attribute PSMODE is set to DISABLED
- The sample object CSQ4INST statement `DEFINE SUB('SYSTEM.DEFAULT.SUB')` is present.

To avoid this, delete or comment out the `DEFINE SUB('SYSTEM.DEFAULT.SUB')` statement.

The JMS 2.0 delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE only need be defined if JMS 2.0 delivery delay is used. By default, the queue definition is commented out, which you can uncomment if required.

CSQ4INSA system object and authentication sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSA) contains the channel authentication system queue definition. This queue holds the channel authentication records. It also contains the default channel authentication rules.

You must define the objects in this sample if CHLAUTH is ENABLED on the queue manager and you want to run channels, or you want to SET or DISPLAY CHLAUTH record. You only need to define them once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across a queue manager shutdown and restart, you must not change the queue name.

CSQ4INSS system object sample

You can define additional system objects if you are using queue sharing groups.

Sample data set thlqual.SCSQPROC(CSQ4INSS) contains sample commands for use with CF structures and a set of definitions for the system objects required for shared channels and intra-group queuing.

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required commands.

When you are defining group or shared objects, you need to include them in the CSQINP2 DD concatenation for only one queue manager in the queue sharing group.

CSQ4INSX system object sample

You must define additional system objects if you are using distributed queuing and clustering.

Sample data set thlqual.SCSQPROC(CSQ4INSX) contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:

- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

CSQ4INSJ system JMS object sample

Defines queues used in the JMS publish/subscribe domain.

CSQ4INSM system object sample

If you are using advanced message security you must define additional system objects. Sample data set thlqual.SCSQPROC(CSQ4INSM) contains the queue definitions required.

CSQ4INSR object sample

Defines queues used by WebSphere Application Server and brokers.

CSQ4INYD object sample

If you are using distributed queuing and you need to set up your own queues, processes, and channels.

Sample data set thlqual.SCSQPROC(CSQ4INYD) contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:

- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don't have to redefine these objects each time you restart the queue manager).

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed - a cluster-receiver channel for the cluster and a cluster-sender definition to at least one cluster repository queue manager.

The sample data set: thlqual.SCSQPROC(CSQ4INYC) contains the following sample definitions that you can use for customizing your clustering objects:

- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

CSQ4INYG object sample

The sample data set: thlqual.SCSQPROC(CSQ4INYG) contains the following sample definitions that you can use for customizing your own objects for general use:

- Dead-letter queue
- Default transmission queue
- CICS adapter objects

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you can make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Default transmission queue
Read the Default transmission queue description before you decide whether you want to define a default transmission queue.

- If you decide that you do want to define a default transmission queue, remember that you must also define a channel to serve it.
- If you decide that you do not want to define one, remember to remove the DEFXMITQ statement from the ALTER QMGR command in the sample.

CICS adapter objects
The sample defines an initiation queue named CICS01.INITQ. This queue is used by the IBM MQ -supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement.

CSQ4INYS/CSQ4INYR object samples

Storage class definitions for using:

- one page set for each class of message
- multiple page sets for major classes of message

For example, SYSTEM.COMMAND.INPUT uses STGCLASS('SYSVOLAT'), and SYSTEM.CLUSTER.TRANSMIT.QUEUE uses STGCLASS('REMOTE'). In CSQ4INYS, both of those storage classes use the same page set. In CSQ4INYR, those storage classes use different page sets in order to lessen the impact of the transmission queue filling.

CSQINPT samples

CSQ4INST

The sample data set: thlqual.SCSQPROC(CSQ4INST) contains the definition for the system default subscription.

You must define the object in this sample, but you need to do it only once when the publish/subscribe engine is first started. Including the definition in the CSQINPT data set is the best way to do this. It is maintained across queue manager shutdown and restart. You must not change the object name, but you can change their attributes if required.

CSQ4INYT
The sample data set: thlqual.SCSQPROC(CSQ4INYT) contains a set of commands that you might want to run when the publish/subscribe engine is started. This sample displays Topic and Subscription information.

Other

CSQ4DISP display sample

The sample data set: thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your queue manager. This includes the definitions for all IBM MQ objects and definitions such as storage classes and trace. These commands can generate a large amount of output. You can use this sample in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4INPX sample

The sample data set: thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

CSQ4IVPQ and CSQ4IVPG samples

The sample data sets: thlqual.SCSQPROC(CSQ4IVPQ) and thlqual.SCSQPROC(CSQ4IVPG) contain sets of DEFINE commands that are required to run the installation verification programs (IVPs).

You can include these samples in the CSQINP2 data set. When you have run the IVPs successfully, you do not need to run them again each time the queue manager is restarted. Therefore, you do not need to keep these samples in the CSQINP2 concatenation permanently.

CSQ4MSTR and CSQ4MSRR samples
These are sample started task procedures for the queue manager: thlqual.SCSQPROC(CSQ4MSTR) and thlqual.SCSQPROC(CSQ4MSRR).

CSQ4MSRR uses CSQ4INYR in the CSQINP2 concatenation so that important queues are spread across different page sets.

You can remove the comments, so that you can use the CSQMINI card for newly created queue managers if required.

CSQ4QMIN sample

A sample QMINI data set, thlqual.SCSQPROC(CSQ4QMIN).

See QMINI data set for details of the QMINI data set and the **TransportSecurity** stanza.

z/OS Recovery and restart on z/OS

Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

IBM MQ for z/OS has robust features for restart and recovery. For information about how a queue manager recovers after it has stopped, and what happens when it is restarted, see the following subtopics:

- “How changes are made to data in IBM MQ for z/OS” on page 257
- “How consistency is maintained in IBM MQ for z/OS” on page 258
- “What happens during termination in IBM MQ for z/OS” on page 260
- “What happens during restart and recovery in IBM MQ for z/OS” on page 261
- “How in-doubt units of recovery are resolved” on page 263
- “Shared queue recovery” on page 266

Related concepts

z/OS IBM MQ for z/OS recovery actions

Related tasks

Planning for backup and recovery

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS How changes are made to data in IBM MQ for z/OS

IBM MQ for z/OS must interact with other subsystems to keep all the data consistent. This topic contains information about *units of recovery*, what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single queue manager for an application program, that changes IBM MQ data from one point of consistency to another. A *point of consistency* - also called a *syncpoint* or *commit point* - is a point in time when all the recoverable data that an application program accesses is consistent.

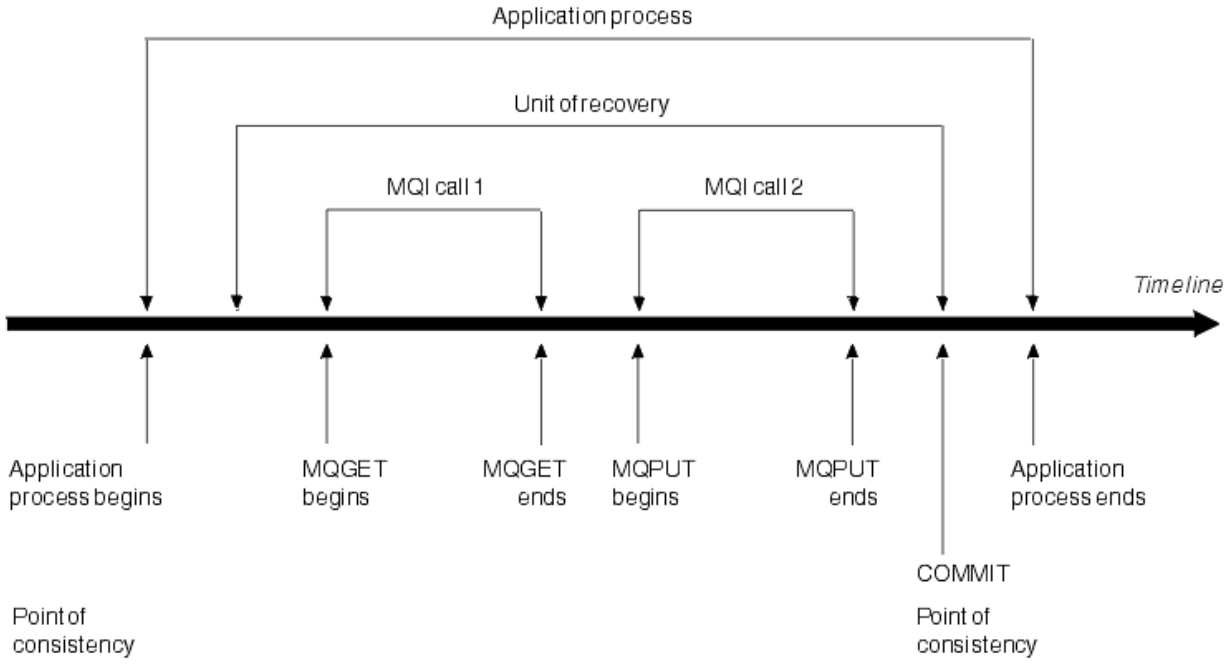


Figure 75. A unit of recovery within an application program

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 75 on page 257 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and IBM MQ cannot commit. They become consistent when the amount is added to account B. When both steps are complete, the program

can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, IBM MQ removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, IBM MQ backs out the work. The events are shown in Figure 76 on page 258.

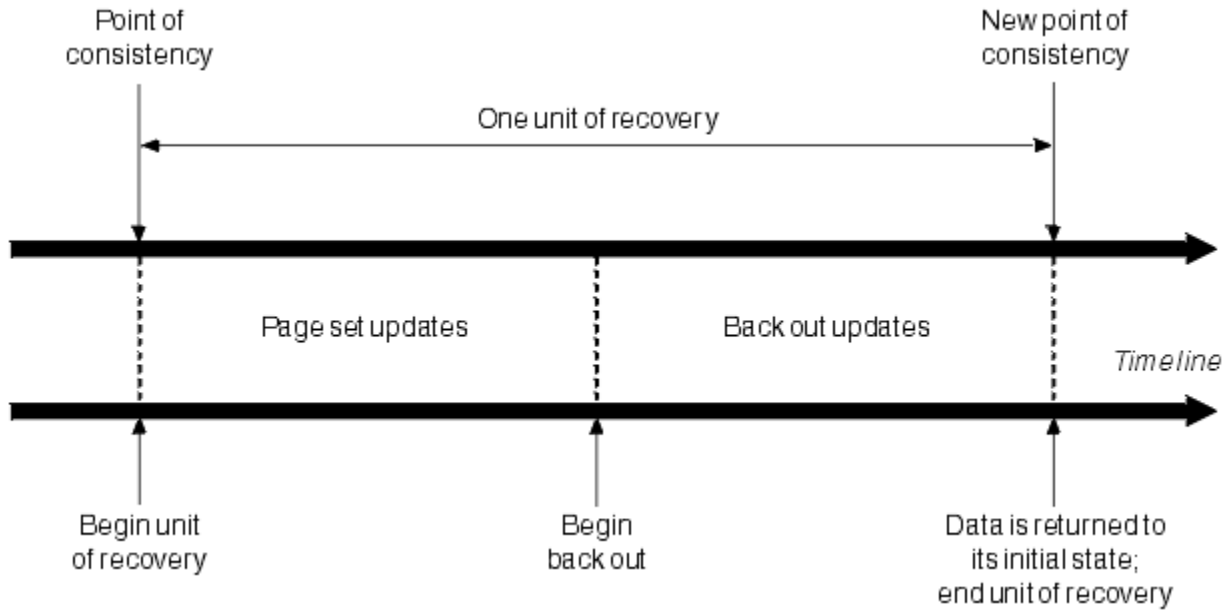


Figure 76. A unit of recovery showing back out

z/OS How consistency is maintained in IBM MQ for z/OS

Data in IBM MQ for z/OS must be consistent with batch, CICS, IMS, or TSO. Any data changed in one must be matched by a change in the other.

Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*; the other is the *participant*. CICS or IMS is always the coordinator in interactions with IBM MQ, and IBM MQ is always the participant. In the batch or TSO environment, IBM MQ can participate in two-phase commit protocols coordinated by z/OS RRS.

During a *single-phase commit* (for example under TSO or batch), IBM MQ is always the coordinator in the interactions and completely controls the commit process.

In a WebSphere Application Server environment, the semantics of the JMS session object determine whether single-phase or two-phase commit coordination is used.

Consistency with CICS or IMS

The connection between IBM MQ and CICS or IMS supports the following syncpoint protocols:

- Two-phase commit - for transactions that update resources owned by more than one resource manager.

  This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.
- Single-phase commit - for transactions that update resources owned by a single resource manager (IBM MQ).

  This protocol is optimized for logging and message flows.
- Bypass of syncpoint - for transactions that involve IBM MQ but which do nothing in the queue manager that requires a syncpoint (for example, browsing a queue).

In each case, CICS or IMS acts as the syncpoint manager.

The stages of the two-phase commit that IBM MQ uses to communicate with CICS or IMS are as follows:

1. In phase 1, each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.

   At the end of the phase, the systems communicate. If they agree, each begins the next phase.
2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 77 on page 259 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in IBM MQ on the lower line.

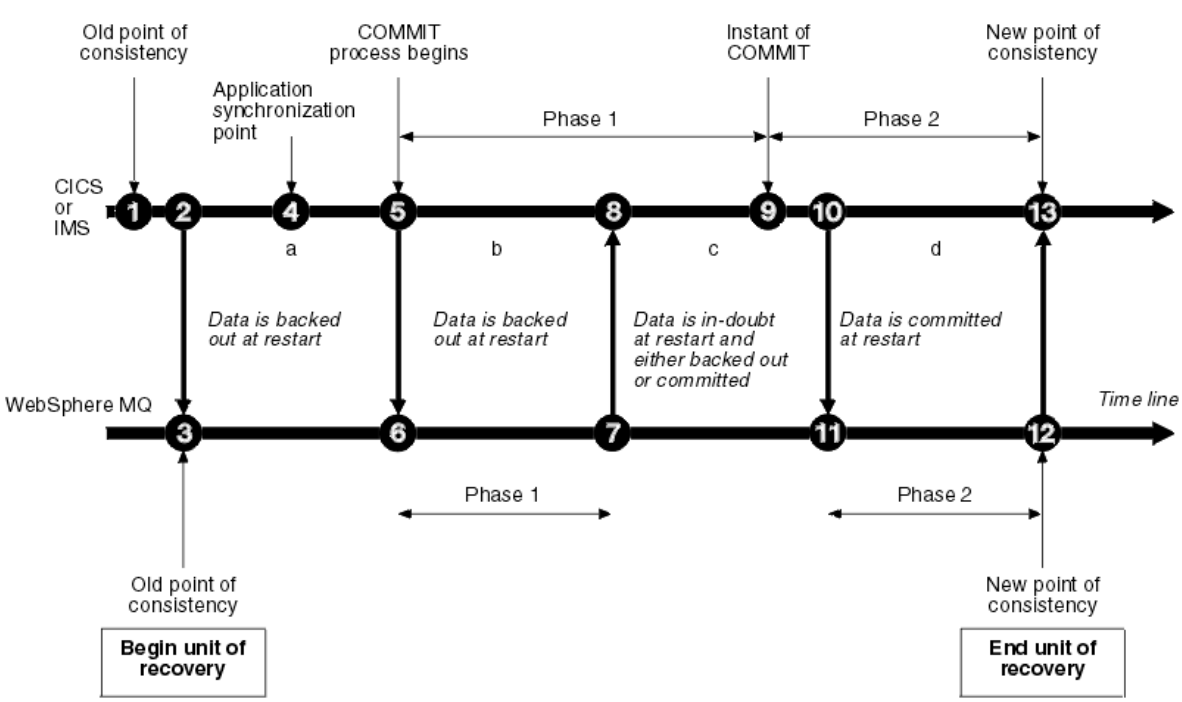


Figure 77. The two-phase commit process

The numbers in the following section are linked to those shown in the figure.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls IBM MQ to update a queue by adding a message.
3. This starts a unit of recovery in IBM MQ.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using a

CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.

6. As the coordinator begins phase 1 processing, so does IBM MQ.
7. IBM MQ successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit - the irrevocable decision of the two subsystems to make the changes.

   The coordinator now begins phase 2 of the processing - the actual commitment.
10. The coordinator notifies IBM MQ to begin its phase 2.
11. IBM MQ logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for IBM MQ. IBM MQ then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

How consistency is maintained after an abnormal termination

When a queue manager is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For some units of recovery, IBM MQ has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 77 on page 259 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be one of the following:

In flight
The queue manager ended before finishing phase 1 (period a or b); during restart, IBM MQ backs out the updates.

In doubt
The queue manager ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, IBM MQ must back out its changes; if it happened after, IBM MQ must make its changes and commit them. At restart, IBM MQ waits for information from the coordinator before processing this unit of recovery.

In commit
The queue manager ended after it began its own phase 2 processing (period d); it makes committed changes.

In backout
The queue manager ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure) during restart, IBM MQ continues to back out the changes.

z/OS What happens during termination in IBM MQ for z/OS

A queue manager terminates normally in response to the STOP QMGR command. If a queue manager stops for any other reason, the termination is abnormal.

Note, that during queue manager termination, IBM MQ internally issues the command

```
DISPLAY CONN(*) TYPE(CONN) ALL WHERE (APPLTYPE NE SYSTEMAL)
```

so that you are aware of what threads might prevent the queue manager from completing shutdown.

SYSTEMAL matches APPLTYPES of either SYSTEM or CHINIT, so the DISPLAY CONN command filtering application types not matching SYSTEMAL, returns to the joblog information about threads that could be preventing normal shutdown.

Normal termination

In a normal termination, IBM MQ stops all activity in an orderly way. You can stop IBM MQ using either quiesce, force, or restart mode. The effects are given in Table 23 on page 261.

Table 23. Termination using QUIESCE, FORCE, and RESTART

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, stopping a queue manager in quiesce mode stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

If you stop a queue manager in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when IBM MQ is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop a queue manager, in any mode, the steps are:

1. Connections are ended.
2. IBM MQ ceases to accept commands.
3. IBM MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by IBM MQ so that the restart RBA is recorded on the z/OS console log.
5. The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify quiesce mode do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

IBM MQ resolves any data inconsistencies arising from abnormal termination during restart and recovery.

z/OS What happens during restart and recovery in IBM MQ for z/OS

IBM MQ uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent IBM MQ checkpoint in the log.

Introduction to restart and recovery

After IBM MQ has been initialized, the queue manager restart process takes place as follows:

- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:

- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until IBM MQ recognizes and acts on the in-doubt decision.
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (described in “Rebuilding queue indexes” on page 263).

If dual BSDSs are in use, IBM MQ checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, IBM MQ tests whether the two time stamps are equal. If they are not, IBM MQ issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while the queue manager was stopped. IBM MQ detects the situation at restart.
- If one copy of the BSDS was de-allocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, IBM MQ might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Batch applications are not notified when restart occurs *after* the application has requested a connection.

Understanding the log range required for recovery

During restart, the range of log data which must be read is dependent on many factors:

- At the time of an abnormal termination, there are typically many incomplete units of work in the system. As described earlier, restart processing will bring the system to a state of consistency, which may involve backing out inflight units of work, or recovering locks on indoubt units of work. Unit of work recovery requires that all unit of work log records for inflight, in-backout, and in-doubt units of work are available. IBM MQ will 'shunt' old units of work, so that unit of work recovery can be performed using a much smaller range of log data.
- At the time of an abnormal termination, there are typically many persistent updates which are only held in the buffer pool cache. They have not yet been written to disk. These changes must be read from the log, and reapplied to the data held in page sets. Page set recovery RBAs in the checkpoint describe the lowest log RBA required for updating the page sets to a consistent state.
- If old page sets have been introduced into the system, for example, a page set backup has been introduced to recover from a media failure, all the changes must be read from the log from the time the backup was taken. These changes are reapplied to the data held in the page set being recovered. Page set recovery RBAs held in page 0 of the page set describe the lowest log RBA required for media recovery of a page set.
- If using persistent messages on shared queues, a range of log data is required to recover CFSTRUCTs which are holding persistent messages. The earliest log data that would be required to perform a CFSTRUCT recovery, is from around the time of the old CFSTRUCT BACKUP.

During normal running, the DISPLAY USAGE TYPE(DATASET) command can be used to view the recovery log range associated with these factors (it is unable to provide information due to reintroducing old page sets, of course). To avoid any issues that might prolong a queue manager restart in the event of an abnormal termination, regularly monitor the values output from DISPLAY USAGE TYPE(DATASET).

In addition, the queue manager issues informational messages relating to these factors:

- CSQJ160I and CSQJ161I warn of long running units of work.
- CSQR026I and CSQR027I provide information about whether these long running units of work have been successfully shunted.
- CSQE040I and CSQE041E warn that structure backups are getting old, and consequently a RECOVER CFSTRUCT operation would take a long time.

Determining which application has a long running unit of work

It is possible to determine the application with the long-running unit of work. To do this, you use the DISPLAY CONN command.

The DISPLAY CONN command returns connection information for all the applications connected to the queue manager, together with additional information that helps you determine which application(s) currently have a long-running unit of work. The information returned by the DISPLAY CONN command is similar to the information returned by the DISPLAY QSTATUS command, but the main difference is that DISPLAY CONN displays information about objects, and transactional information for a particular connection, rather than details of which connections are associated with a particular object.

For each connected application, the DISPLAY CONN command returns the following information:

- Basic information including the Connection Id and PID.
- Transactional information for that connection, including the time and date when the transaction was created (that is, when the first MQGET/PUT was made under syncpoint), and when the transaction first wrote to the log.
- Log time information indicating which application still has a long running unit of work.
- A list of all objects that the connection currently has open. Details for each object are returned as a separate message, with the Connection Id used as a key. Because there are different types of object such as queues and queue managers, the information displayed with the object is specific to its object type.

Rebuilding queue indexes

To increase the speed of MQGET operations on a queue where messages are not retrieved sequentially, you can specify that you want IBM MQ to maintain an index of the message or correlation identifiers or groupid for all the messages on that queue.

When a queue manager is restarted, these indexes are rebuilt for each queue. This applies only to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this increases the time taken to restart the queue manager.

You can choose to have indexes rebuilt asynchronously to queue manager startup by using the QINDXBLD parameter of the CSQ6SYSP macro. If you set QINDXBLD=NOWAIT, IBM MQ restarts without waiting for indexes to rebuild.

z/OS How in-doubt units of recovery are resolved

If IBM MQ loses its connection to another resource manager, it typically attempts to recover all inconsistent objects at restart.

If IBM MQ loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information required to resolve in-doubt units of recovery must come from the coordinating system. The next sections describe the process of resolution for different environments.

- How in-doubt units of recovery are resolved from CICS
- How in-doubt units of recovery are resolved from IMS
- How in-doubt units of recovery are resolved from RRS
- How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

How in-doubt units of recovery are resolved from CICS

Under some circumstances, CICS cannot run the IBM MQ process to resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the IBM MQ for z/OS のメッセージ、完了コード、および理由コード manual.

The resolution of in-doubt units does not effect CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was a log record marking the beginning of the commit. The existence of in-doubt objects does not lock CICS resources while IBM MQ is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and IBM MQ. If a queue manager abends while connected to CICS, it is possible for CICS to commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*.

IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to CICS is restarted or reconnected.

A process to resolve in-doubt units of recovery is initiated during startup of the CICS adapter. The process starts when the adapter requests a list of in-doubt units of recovery. Then:

- The adapter receives a list of in-doubt units of recovery for this connection ID from IBM MQ, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own log. CICS determines from its own list what action it took for each in-doubt unit of recovery.

For all resolved units, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units can remain after restart. Resolve them by the methods described in the IBM MQ for z/OS の管理.

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMSdoes not effect DL/I resources. IMS is in control of recovery coordination and, when it restarts, automatically commits or backs out incomplete DL/I work. The decision to commit or back out for online regions (non-fast-path) is on the presence or absence of IMS log record types X'3730' and X'3801'. The existence of in-doubt units of recovery does not imply that DL/I records are locked until IBM MQ connects.

During queue manager restart, IBM MQ makes a list of in-doubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

During reconnection of an IMS region to IBM MQ, IMS indicates to IBM MQ whether to commit or back out units of work marked in IBM MQ as in doubt.

When in-doubt units are resolved:

1. If IBM MQ recognizes that it has marked an entry for commit and IMS has marked it to be backed out, IBM MQ issues message CSQQ010E. IBM MQ issues this message for all inconsistencies of this type between IBM MQ and IMS.
2. If IBM MQ has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, IBM MQ updates queues as necessary and releases the corresponding locks.

IBM MQ maintains locks on in-doubt work that was not resolved. This can cause a backlog in the system if important locks are being held. The connection remains active so you can resolve the IMS RREs. Recover the in-doubt threads by the methods described in the IBM MQ for z/OS の管理.

All in-doubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. In-doubt resolution by the IMS control region takes place in two circumstances:

1. At the start of the connection to IBM MQ, during which resolution is done synchronously.
2. When a program abends, during which the resolution is done asynchronously.

How in-doubt units of recovery are resolved from RRS

One of the functions of the RRS adapter is to keep data synchronized between IBM MQ and other RRS-participating resource managers. If a failure occurs when IBM MQ has completed phase one of the commit and is waiting for a decision from RRS (the commit coordinator), the unit of recovery enters the in-doubt state.

When communication is reestablished between RRS and IBM MQ, RRS automatically commits or backs out each unit of recovery, depending on whether there was a log record marking the beginning of the commit. IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages to the z/OS console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

For details of what these messages mean, see the IBM MQ for z/OS のメッセージ、完了コード、および理由コード manual.

For all resolved units of recovery, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units of recovery can remain after restart. Resolve them by the method described in the IBM MQ for z/OS の管理.

How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

In-doubt transactions that have a GROUP unit of recovery disposition can be resolved by the transaction coordinator by any queue manager in the queue sharing group (QSG) where the GROUPUR queue manager attribute is enabled. Whenever a transaction coordinator reconnects it typically requests a list of any outstanding in-doubt transactions and then resolves them using information from its logs.

When a transaction coordinator, that has connected with a GROUP unit of recovery disposition, requests the list of in-doubt transactions, the list returned comprises all in-doubt transactions with a GROUP unit of recovery disposition that exist throughout the queue sharing group. This list is not dependent on which queue manager those in-doubt transactions were started on. A queue manager processing such a request compiles the list by communicating with all other active queue managers in the queue sharing group using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The queue manager then reads the logs of any inactive queue

managers, from their last checkpoint, to identify any additional in-doubt transactions that they would have reported had they been active.

When a transaction coordinator requests the resolution of an in-doubt transaction, the queue manager to which it is connected identifies whether the transaction was originated on itself and if so resolves it in the same way as transactions with a QMGR unit of recovery disposition. If the transaction was originated on another active queue manager in the QSG, a request to complete the resolution is routed to that queue manager using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. In the case where the transaction was originated on an inactive queue manager in the QSG, any shared-queue work is resolved immediately, and a request to resolve any remaining private queue work is placed on the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The inactive queue manager processes this request upon start-up before accepting new work. In this scenario, the original queue manager's logs still reflect that the unit of recovery is in doubt until it has restarted and processed the request.

z/OS Shared queue recovery

Use this topic to understand IBM MQ recovery and resilience of various components in the queue sharing group environment.

Transactional recovery

When an application issues a MQBACK call or terminates abnormally (for example, because of an EXEC CICS ROLLBACK or an IMS abend) thread-level information stored in the queue manager ensures that the in-flight unit of work is rolled back. MQPUT and MQGET operations within syncpoint on shared queues are rolled back in the same way as updates to non-shared queues.

Peer recovery

If a queue manager fails, it disconnects abnormally from the coupling facility structures that it is currently connected to. If the connection between the z/OS instance and the coupling facility fails (for example, physical link failure or power-off of a coupling facility or partition) this is also detected as an abnormal termination of the connection between the queue manager and the coupling facility structures involved. Other queue managers in the same queue sharing group that remain connected to that structure detect the abnormal disconnection and all attempt to initiate *peer recovery* for the failed queue manager on that structure. Only one of these queue managers initiates peer recovery successfully, but all the other queue managers cooperate in the recovery of units of work that were owned by the queue manager that failed.

If a queue manager fails when there are no peers connected to a structure, recovery is performed when another queue manager connects to that structure, or when the queue manager that failed restarts.

Peer recovery, often referred to as Peer Level Recovery (PLR), is performed on a structure by structure basis and it is possible for a single queue manager to participate in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that IBM MQ resets any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared Db2 repository used by the queue sharing group. Ensure that adequate procedures are in place for the backup and recovery of the Db2 tables used to hold IBM MQ objects. You can also use the IBM MQ CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine IBM MQ objects, including shared queue and group definitions stored in Db2.

Logging

Queue sharing groups can support persistent messages, because the messages on shared queues can be logged in the queue manager logs.

Coupling facility and structure failures

There are two types of failure that can be reported for a coupling facility (CF) structure: structure failure and loss of connectivity. Sysplex services for data sharing (XES) inform IBM MQ of a CF structure failure or a CF failure with a structure failure event. If XES creates a loss of connectivity event this does not necessarily indicate that there is a problem with the structure, it might be that there is no connection available to communicate with the structure. It is possible that not all of the queue managers receive a loss of connectivity event for the structure; it depends on the configuration of connections to the CF. A loss of connectivity event can also be received because of operator commands, for example VARY PATH OFFLINE or CONFIG CHP OFFLINE.

The CF structures that are used by IBM MQ can be configured to use system-managed duplexing. This means that if there is a single failure, system-managed failover processing hides the failure of a structure or the loss of connectivity, and the queue manager is not informed of the failure. If there is a failure of both instances of a duplexed structure or connection, the queue manager receives the appropriate event and handles it in the same way as a failure event for a simplex structure. Details of how the queue manager handles the events are described in Scenarios.

In the unlikely event of a CF or structure failure, any nonpersistent messages stored in the affected application structures are lost. You can recover persistent messages using the RECOVER CFSTRUCT command. If a recoverable application structure has failed, any further application activity to this structure is prevented until the structure has been recovered.

To ensure that you can recover a CF structure in a reasonable period of time, take frequent backups, using the BACKUP CFSTRUCT command. You can choose to perform the backups on any queue managers in the queue sharing group or dedicate one queue manager to perform all the backups. Automate the process of taking backups to ensure that they are taken on a regular basis.

Each backup is written to the active log data set of the queue manager taking the backup. The shared queue Db2 repository records the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager's log, and the backup time.

The administration structure contains information about incomplete units of work on shared queues at the time of any application structure failure so the administration structure must be available during RECOVER CFSTRUCT processing. If the administration structure has failed, all the queue managers in the queue sharing group must have rebuilt their administration structure entries before you can issue the RECOVER CFSTRUCT command.

Queue managers rebuild their administration structure entries automatically and without terminating. If a queue manager is not running at the time of the failure, its administration structure entries can be rebuilt by another queue manager in the queue sharing group that is running at the same or higher level.

To recover an application structure, issue a RECOVER CFSTRUCT command to the queue manager that you want to perform the recovery. You can recover a single CF structure or you can recover several CF structures simultaneously. You can recover using any queue manager in the queue sharing group, it does not have to be the one that performed the backup, or one that has been connected previously to the failed structure.

The RECOVER CFSTRUCT command uses the backup, located through the Db2 repository information (Db2 must therefore be available on the queue manager where recovery is being carried out), and recovers this to the point of failure.

The RECOVER CFSTRUCT command does this by applying log records from every queue manager in the queue sharing group that has performed an MQPUT or MQGET between the start of the backup and the time of failure, to any shared queue that maps to the CF structure. The resulting merging of the logs might require reading a considerable amount of log data because all the log data written by participating queue managers since the backup is read. You are strongly recommended to make frequent (for example, hourly) backups, especially if there are large messages within the backup.

Structure failure scenarios

Scenarios

If a failure is reported for a CF structure, the action taken by connected queue managers depends on the following:

- The type of failure reported by the XES component of z/OS to IBM MQ.
- The structure type (application or administration)
- The queue manager level
- The CFLEVEL of the IBM MQ CFSTRUCT object (2, 3, 4 or 5. This is not the CFLEVEL of the CFCC microcode)
- The RECAUTO attribute of an IBM MQ CFSTRUCT object at CFLEVEL(5)

The following scenarios describe what happens when a failure is reported for the administration structure:

- If a structure failure event is received for the administration structure, the structure is reallocated and rebuilt automatically without the queue manager terminating. A new instance of the structure is allocated by XES when a queue manager attempts to connect to it.

  When the queue manager has connected to the new instance of the structure, the queue manager writes the entries for itself into the structure. This processing is carried out by the queue manager and is not part of XES rebuild processing.

  If a queue manager was not running at the time of the failure, or terminates before recovery of its part of the administration structure has been completed, its administration structure entries can be rebuilt by another queue manager in the queue sharing group.

  Administration structure entries of a queue manager can only be rebuilt by another queue manager running at the same level or higher. If administration structure entries of a queue manager cannot be rebuilt by another queue manager in the queue sharing group, restart the queue manager so that it can complete the rebuild of its part of the structure.

Certain actions are suspended until the administration structure entries for all queue managers have been rebuilt. The suspended actions include the following:

- Opening and closing of shared queues.
- Committing or backing out units of recovery.
- Serialized applications connecting to or disconnecting from the queue manager.
- Backing up or recovering an application structure.

Any serialized applications that have already connected to the queue manager can continue processing. Any serialized application attempting to connect with the MQCNO_SERIALIZE_CONN_TAG_QSG or MQCNO_RESTRICT_CONN_TAG_QSG parameters receive the MQRC_CONN_TAG_NOT_USABLE return code.

When the administration structure entries for the queue manager have been rebuilt, the suspended actions are resumed.

The following scenarios describe what happens when a failure is reported for an application structure:

- If a structure failure event is received for an application structure, and the CFLEVEL is 1 or 2, the queue manager terminates. Restart the queue manager. The first queue manager to attempt to connect to the structure again causes XES to allocate a new instance of the structure.
- If a structure failure event is received for an application structure, and the CFLEVEL is 3, 4, or 5 the queue managers connected to the structure continue to run. Applications that do not use the queues in the failed structure can continue normal processing.

  However, applications that attempt operations on queues in the failed structure receive an MQRC_CF_STRUC_FAILED error until the structure has been successfully rebuilt, at which point the application can open the queues again.

  Structure rebuild is started automatically for CFLEVEL(5) application structures defined with RECAUTO(YES). Otherwise, the structure will be rebuilt when the RECOVER CFSTRUCT command is issued.

Resilience to coupling facility connectivity failures

What is resilience to coupling facility connectivity failures?

Resilience to coupling facility connectivity failures refers to the ability of queue managers in a queue sharing group to tolerate loss of connectivity to a coupling facility structure without terminating. This function also attempts to rebuild the structure in another coupling facility with better connectivity in order to regain access to shared queues as soon as possible.

What is partial loss of connectivity?

IBM MQ defines partial loss of connectivity as a situation where one or more systems in the sysplex lose connectivity to the coupling facility where the structure being accessed by the system is allocated, but at least one system in the sysplex maintains connectivity to the same coupling facility.

What is total loss of connectivity?

IBM MQ defines a total loss of connectivity as a situation where no systems in the sysplex have connectivity to the coupling facility and the structure allocated within it.

Why would you enable this function?

Resilience to coupling facility connectivity failures improves the availability of IBM MQ, allowing non-shared queues to remain available after a queue manager has lost connectivity to one or more coupling facility structures. Additionally, queue managers that lose connectivity to a coupling facility structure automatically attempt to rebuild the structure in another available coupling facility, improving the availability of the shared queues within the queue sharing group.

Considerations when enabling this function

A queue manager that tolerates loss of connectivity to coupling facility structures without terminating may not be able to reconnect to a coupling facility structure for some time if there is no alternative coupling facility available. Shared queues defined on a structure that has suffered loss of connectivity remain unavailable until connectivity to the structure is restored. In this situation, applications that connect into members of the queue sharing group in order to perform shared queue work may find that the shared queues they need to access are not available. To avoid this situation it is recommended that queue managers should be configured to terminate when connectivity to a coupling facility structure is lost. This termination forces applications to connect to another member of the queue sharing group that has connectivity to the coupling facility structures where the shared queues the application requires are defined.

Managing Resilience to coupling facility connectivity failures

How do I enable this functionality?

The following steps must be performed in order to enable resilience to coupling facility connectivity

1. Ensure that the CFRM couple data set has been formatted to support system-managed rebuild. This allows queue managers to initiate a system-managed rebuild to re-create a structure into an available coupling facility. Use the **DISPLAY XCF,COUPLE,TYPE=CFRM** command to determine the format of the CFRM couple data set. To support system-managed rebuild, the CFRM couple data set should be formatted by specifying:

   ```
   "ITEM NAME(SMREBLD) NUMBER(1)"
   ```

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information on formatting a CFRM couple data set.
2. Ensure that an alternative coupling facility is available and is in the CFRM preference list for all IBM MQ coupling facility structures. This enables the queue managers to attempt to rebuild structures into an alternative available coupling facility to restore access to the structures as soon as possible.

   IBM MQ structures must be defined with ENFORCEORDER(NO) in CFRM policy, so that XCF is able to choose the optimum CF in the configuration if IBM MQ needs to reallocate the structure.

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information about structure preference lists.
3. Alter all application coupling facility structures that need to tolerate loss of connectivity to CFLEVEL(5). This is the minimum level that can tolerate a loss of connectivity.
4. Determine the values required for the **QMGR CFCONLOS** and the **CFSTRUCT CFCONLOS** attributes and alter these accordingly. The **QMGR CFCONLOS** attribute controls whether loss of connectivity to the administration structure is tolerated, and the **CFSTRUCT CFCONLOS** attribute controls whether loss of connectivity is tolerated by each application coupling facility structure. If the default values for these attributes are retained, the queue manager terminates following loss of connectivity to any coupling facility structure.
5. Determine the values required for the **CFSTRUCT RECAUTO** attribute for each application coupling facility structure, and alter these accordingly. This attribute controls whether coupling facility structures should be automatically recovered using logged data following total loss of connectivity. If the default value for this attribute is retained, no automatic recovery is performed for application structures following total loss of connectivity.

Scenario 1 - Loss of connectivity to the administration structure

Queue managers can tolerate loss of connectivity to the administration structure without terminating.

When connectivity to the administration structure is lost by any queue manager that has been configured to tolerate loss of connectivity to the administration structure, all members of the queue sharing group disconnect from the administration structure. All active queue managers in the queue

sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in the coupling facility with the best connectivity to all systems in the sysplex, and rebuild the administration structure data.

Note: This may not necessarily be the coupling facility which has the best connectivity to all systems that have active queue managers.

If a queue manager cannot reconnect to the administration structure, for example because none of the coupling facilities in the CFRM preference list for the administration structure are available, some shared queue operations remain unavailable until the queue manager can successfully reconnect to the administration structure and rebuild its administration structure data. Reconnection occurs automatically when a suitable coupling facility becomes available on the system.

Failure to connect to the administration structure during queue manager startup as a result of a lack of connectivity to the coupling facility, or no suitable coupling facility available to allocate the structure, is not tolerated. All active queue managers in the queue sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in another coupling facility if one is available, and rebuild the administration structure data.

Scenario 2- Loss of connectivity to the application structure

Loss of connectivity to application structures at **CFLEVEL(5)** or higher can be tolerated without the queue manager terminating. Queue managers connected to application structures at **CFLEVEL(4)** or lower, or structures at **CFLEVEL(5)** that have not been configured to tolerate loss of connectivity, abend with reason code 00C510AB when connectivity to the structure is lost.

When connectivity is lost to an application structure that has been configured to tolerate loss of connectivity, all queue managers that lost connectivity to the structure disconnect. The subsequent behavior of the queue manager depends on whether the loss of connectivity is partial or total.

Partial loss of connectivity to an application structure

If the loss of connectivity is determined to be partial, queue managers that have lost connectivity to the structure attempt to initiate a system-managed rebuild in order to move the structure to another coupling facility with improved connectivity. If this rebuild is successful, both persistent and non-persistent messages in the structure are copied to the other coupling facility, and access to queues on the structure is restored. Queue managers that did not lose connectivity remain connected to the structure, however, operations that access the structure are delayed during the system-managed rebuild process.

If an application structure cannot be rebuilt to another coupling facility with improved connectivity, or some queue managers still do not have connectivity to the structure after it has been rebuilt in another coupling facility, queues defined on the structure remain unavailable on the queue managers that do not have connectivity to the structure until connectivity is restored to the coupling facility. Queue managers automatically reconnect to the structure when it becomes available and access to the shared queues defined on the structure are restored.

Total loss of connectivity to an application structure

If all MVS systems in the sysplex have lost connectivity to the coupling facility that the application structure is allocated in, z/OS deallocates the structure from the coupling facility whenever an attempt is made to reconnect to the structure. It is possible for the queue manager to attempt to reconnect to the structure for several reasons, such as an attempt by an application to open a shared queue, or a notification from the system that new coupling facility resources may have become available. It is therefore likely that all non-persistent messages in the affected structure are lost following total loss of connectivity to an application structure.

Recoverable application structures are automatically recovered following total loss of connectivity, if they have been defined with **RECAUTO(YES)**. The recovery starts almost immediately if an alternative coupling facility is available to allocate the structure in, or whenever such a coupling facility becomes available. If a structure has not been defined with **RECAUTO(YES)**, recovery can be started by issuing the **RECOVER CFSTRUCT** command. This recovers all persistent messages in the structure, but all non-persistent messages are lost. As this process involves reading the queue manager log it can take

some time to complete, therefore it is recommended that structure backups be taken regularly to reduce the time until access to the shared queues on the structure is restored.

Queue managers attempt to reconnect to non-recoverable application structures as soon as an application attempts to open a shared queue that is defined on the structure or a notification is received from the system that new coupling facility resources have become available. If a suitable coupling facility is available to allocate the structure in, a new structure is allocated and access to the shared queues defined on the structure is restored. As persistent messages cannot be put to queues defined in non-recoverable structures, all messages on the shared queues are lost.

Operational behavior

If an IBM WebSphere MQ 7.1, or later, queue manager, configured to tolerate loss of connectivity to a particular coupling facility structure loses connectivity, the members of the queue sharing group attempt to automatically recover from the failure and reconnect to the structure. This activity may involve reallocating the structure in another coupling facility with better connectivity if one is available. However, operator intervention may still be required to recover from the loss of connectivity.

Typically the required operator action is to:

1. Resolve the cause of the failure that resulting in the loss of connectivity.
2. Ensure that a coupling facility where the IBM MQ structures can be allocated is available on all systems in the sysplex

Any structures that have been automatically reallocated in another coupling facility after the loss of connectivity event, can be moved to the coupling facility with the optimal connectivity to all queue managers in the queue sharing group. If required, this can be done by initiating the system-managed rebuild command **SETXCF START,REBUILD** as documented in z/OS 「MVS」 システム・コマンド解説書.

In the case of a partial loss of connectivity to an application structure, the queue managers that lost connectivity to the structure attempt to initiate a system-managed rebuild. This process only allocates the structure in another coupling facility if that coupling facility has connectivity to all active queue managers currently connected to the structure. Therefore, it is possible that where the majority of queue managers in a queue sharing group have lost connectivity to an application structure, they are unable to rebuild the structure into another coupling facility due to the queue managers that are still connected to the original structure. In this situation the queue managers that are still connected to the original structure can be shut down to allow the structure to be rebuilt, or the **RESET CFSTRUCT ACTION(FAIL)** command can be issued to fail the structure. Recovery can be initiated on applicable structures by issuing the **RECOVER CFSTRUCT** command.

Note: When failing and recovering the structure, all non-persistent messages on the structure are lost.

z/OS Security concepts in IBM MQ for z/OS

Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

Why you must protect IBM MQ resources

IBM MQ handles the transfer of information that is potentially valuable. Applying security ensures that the resources IBM MQ owns and manages are protected from unauthorized access. Such access might lead to the loss or disclosure of the information.

You should ensure that none of the following resources are accessed or changed by any unauthorized user or process:

- Connections to IBM MQ
- IBM MQ objects such as queues, processes, and namelists
- IBM MQ transmission links, that is, IBM MQ channels
- IBM MQ system control commands

- IBM MQ messages
- Context information associated with messages

To provide the necessary security, IBM MQ uses the z/OS system authorization facility (SAF) to route authorization requests to an External Security Manager (ESM), for example Security Server (previously known as RACF). IBM MQ does no security verification of its own. Where distributed queuing or clients are being used, you might require additional security measures, for which IBM MQ provides channel authentication records, channel exits, the MCAUSER channel attribute, and TLS.

The decision to allow access to an object is made by the ESM and IBM MQ follows that decision. If the ESM cannot make a decision, IBM MQ prevents access to the object.

What happens if you do not protect IBM MQ resources

If you do nothing about security, the most likely effect is that *all* users can access and change *every* resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for z/OS.

To enable security checking you must do the following:

- Install and activate an ESM (for example, Security Server).
- Define the MQADMIN class if you are using an ESM other than Security Server.
- Activate the MQADMIN class.

You must consider whether using mixed-case resource names would be beneficial to your enterprise. If you do use mixed-case resource names in your ESM profiles you must define and activate the MXADMIN class.

z/OS Data Set Encryption

Data Set Encryption (DSE) provides the capability to encrypt z/OS data sets, so that the data they contain can only be viewed or modified by user IDs granted the specific permission. This provides encryption of data at rest in the file system, and prevents inadvertent disclosure of sensitive information to users who have a legitimate business need and permissions to manage the data sets themselves.

Prior to IBM MQ for z/OS 9.1.4, IBM MQ for z/OS does not support use of DSE with the active logs, page sets, and shared message data sets (SMDS) that provide the primary persistence mechanisms for IBM MQ messages.

Instead, Advanced Message Security provides an end-to-end encryption solution for IBM MQ messaging, which encompasses the entire IBM MQ network, encryption of data in flight, at rest, and even inside the runtime IBM MQ processes.

Other VSAM and sequential data sets used in an IBM MQ subsystem can be encrypted using DSE. For example:

- Bootstrap data set (BSDS)
- Sequential files holding system configuration (MQSC) commands read at startup using CSQINPx DDNAMEs
- IBM MQ archive logs, often used for long term archival of IBM MQ log data for audit purposes.

You can encrypt using DSE by allocating a dataclass that is defined with a data set key label. For more information, see Planning your log archive storage.

From IBM MQ for z/OS 9.1.4, IBM MQ for z/OS supports use of DSE with the active logs and page sets in addition to the support provided in earlier releases.

IBM MQ for z/OS does not support use of DSE for shared message data sets (SMDS).

See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

Related concepts
Security concepts
Channel authentication records
Authority to work with IBM MQ objects on z/OS
Cryptographic security protocols: TLS
Related tasks
Setting up security on z/OS
Comparing link level security and application level security
Related reference
Messages for IBM MQ for z/OS

z/OS Security controls and options in IBM MQ for z/OS

You can specify whether security is turned on for the whole IBM MQ subsystem, and whether you want to perform security checks at queue manager or queue sharing group level. You can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done for the whole queue manager. If you do not require security checking (for example, on a test system), or if you are satisfied with the level of security on all the resources that can connect to IBM MQ (including clients and channels), you can turn security checking off for the queue manager or queue sharing group so that no further security checking takes place.

This is the only check that can turn security off completely and determine whether any other security checks are performed or not. That is, if you turn off checking for the queue manager or queue sharing group, no other IBM MQ checking is done; if you leave it turned on, IBM MQ checks your security requirements for other IBM MQ resources.

You can also turn security on or off for particular sets of resources, such as commands.

Queue manager or queue sharing group level checking

You can implement security at queue manager level or at queue sharing group level. If you implement security at queue sharing group level, all the queue managers in the group share the same profiles. This means that there are fewer profiles to define and maintain, making security management easier. It also makes it easy to add a new queue manager to the queue sharing group because it inherits the existing security profiles.

It is also possible to implement a combination of both if your installation requires it, for example, during migration or if you have one queue manager in the queue sharing group that requires different levels of security to the other queue managers in the group.

Queue sharing group level security

Queue sharing group level security checking is performed for the entire queue sharing group. It enables you to simplify security administration because it requires you to define fewer security profiles. The authorization of a user ID to use a particular resource is handled at the queue sharing group level, and is independent of which queue manager that user ID is using to access the resource.

For example, say a server application runs under user ID SERVER and wants access to a queue called SERVER.REQUEST, and you want to run an instance of SERVER on each z/OS image in the sysplex. Rather than permitting SERVER to open SERVER.REQUEST on each queue manager individually (queue manager level security), you can permit access only at the queue sharing group level.

You can use queue sharing group level security profiles to protect all types of resource, whether local or shared.

Queue manager level security
You can use queue manager level security profiles to protect all types of resource, whether local or shared.

Combination of both levels
You can use a combination of both queue manager and queue sharing group level security.

You can override queue sharing group level security settings for a particular queue manager that is a member of that group. This means that you can perform a different level of security checks on an individual queue manager to those performed on the other queue managers in the group.

For more information, see Profiles to control queue sharing group or queue manager level security.

Controlling the number of user IDs checked

RESLEVEL is a Security Server profile that controls the number of user IDs checked for IBM MQ resource security. Normally, when a user attempts to access an IBM MQ resource, Security Server checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

These controls are done on a connection by connection basis, and last for the life of the connection.

There is only one RESLEVEL profile for each queue manager. Control is implemented by the access that a user ID has to this profile.

Mixed case or uppercase IBM MQ RACF classes

You can now use mixed case RACF profile support, which allows you to use mixed case resource names and define IBM MQ RACF profiles to protect them.

You can choose to either:

- Continue using uppercase only IBM MQ RACF Classes as in previous releases, or
- Use the new mixed case IBM MQ RACF classes.

Without the use of mixed case RACF profiles, you can still use mixed case resource names in IBM MQ for z/OS ; however, these resource names can only be protected by generic RACF profiles in the uppercase IBM MQ classes. When using mixed case IBM MQ RACF profile support you can provide a more granular level of protection by defining IBM MQ RACF profiles in the mixed case IBM MQ classes.

z/OS Resources you can protect in IBM MQ for z/OS

When a queue manager starts, or when instructed by an operator command, IBM MQ for z/OS determines which resources you want to protect.

You can control which security checks are performed for each individual queue manager. For example, you can implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager. It is done by issuing an MQCONN or MQCONNX request, or when the channel initiator, or CICS or IMS adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager. However, if you do this any user can connect to that queue manager.

For the CICS adapter, only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to IBM MQ, the IMS address space user ID is checked. For the channel initiator, the user ID used by the channel initiator address space is checked.

You can turn connection security checking on or off at either queue manager or queue sharing group level.

Command security

Command security checking is carried out when a user issues an MQSC command from any of the sources described in Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS. You can make a separate check on the resource specified by the command as described in “Command resource security” on page 276.

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSC commands are entered from a console, the console must have the z/OS SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You must control who is allowed to update these data sets through normal data set protection.

You can turn command security checking on or off at either queue manager or queue sharing group level.

Command resource security

Some MQSC commands, for example defining a local queue, involve the manipulation of IBM MQ resources. When command resource security is active, each time a command involving a resource is issued, IBM MQ checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning "PAYROLL". If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

You can turn command resource security checking on or off at either queue manager or queue sharing group level.

Channel security considerations

Channel security

When you are using channels, the security features available depend on which communications protocol you are going to use. If you use TCP, there are no security features provided with the communications protocol, although you can use TLS. If you are using APPC, you can flow user ID information from the sending MCA through the network to the destination MCA for verification.

For both protocols, you can specify which user IDs you want to check for security purposes, and how many. Again, the choices available to you depend on which protocol you are using, what you specify when you define the channel, and the RESLEVEL settings for the channel initiator.

For more information about the types of channel security available see Channel authentication records and Security exit overview

Related reference

"API-resource security in IBM MQ for z/OS" on page 277

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

z/OS *API-resource security in IBM MQ for z/OS*

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into the following checks:

- Queue
- Process
- Namelist
- Alternate user
- Context

No security checks are performed when opening the queue manager object or when accessing storage class objects.

Queue

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (using the MQOO_BROWSE option), but not to remove messages from the queue (using one of the MQOO_INPUT_* options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an MQOPEN or MQPUT1 call).

You can turn queue security checking on or off at either queue manager or queue sharing group level.

Process

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

You can turn process security checking on or off at either queue manager or queue sharing group level.

Namelist

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

You can turn namelist security checking on or off at either queue manager or queue sharing group level.

Alternate user

Alternate user security controls whether one user ID can use the authority of another user ID to open an IBM MQ object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.

- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternative user ID when opening the reply-to queue.

The alternative user ID is specified in the *AlternateUserId* field of the object descriptor (MQOD).

You can use alternative user IDs on any IBM MQ object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for z/OS data set security.

If alternate user security is not active, any user can use any other user ID as an alternative user ID.

You can turn alternate user security checking on or off at either queue manager or queue sharing group level.

Context

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section
The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section
The application that put the message on the queue where it is currently stored. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQPUT or an MQPUT1 call is made. The application might generate the data, the data might be passed on from another message, or the queue manager might generate the data by default. For example, server programs can use context data to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternative user.

You use context security to control whether the user can specify any of the context options on any MQOPEN or MQPUT call. For information about the context options, see the MQOPEN options relating to message context. For descriptions of the message descriptor fields relating to context, see MQMD - Message descriptorMQMD - Message descriptor.

If you turn context security checking off, any user can use any of the context options that the queue security allows.

You can turn context security checking on or off at either queue, queue manager or queue sharing group level.

z/OS Availability on z/OS

IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

Several features of IBM MQ can increase system availability if the queue manager or channel initiator fails. For more information about these features, see the following sections:

- Sysplex considerations
- Shared queues
- Shared channels
- IBM MQ network availability
- Using the z/OS Automatic Restart Manager (ARM)
- Using the z/OS Extended Recovery Facility (XRF)
- Using the z/OS GROUPUR attribute for recovery in a queue sharing group
- Where to find more information about availability

Sysplex considerations

In a *sysplex*, a number of z/OS operating system images collaborate in a single system image and communicate using a coupling facility. IBM MQ can use the facilities of the sysplex environment for enhanced availability.

Removing the affinities between a queue manager and a particular z/OS image allows a queue manager to be restarted on a different z/OS image in the event of an image failure. The restart mechanism can be manual, use ARM, or use system automation, if you ensure the following:

- All page sets, logs, bootstrap data sets, code libraries, and queue manager configuration data sets are defined on shared volumes.
- The subsystem definition has sysplex scope and a unique name within the sysplex.
- The level of *early code* installed on every z/OS image at IPL time is at the same level.
- TCP virtual IP addresses (VIPA) is available on each TCP stack in the sysplex, and you have configured IBM MQ TCP listeners and inbound connections to use VIPAs rather than default host names.

For more information about using TCP in a sysplex, see *TCP/IP in a sysplex*, SG24-5235, an IBM Redbooks® publication.

You can additionally configure multiple queue managers running on different operating system images in a sysplex to operate as a queue sharing group, which can take advantage of shared queues and shared channels for higher availability and workload balancing.

Shared queues

In the queue sharing group environment, an application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access the same set of shared queues, the application does not depend on the availability of a particular queue manager; any queue manager in the queue sharing group can service any queue. This gives greater availability if a queue manager stops because all the other queue managers in the queue sharing group can continue processing the queue. For information about high availability of shared queues, see “Advantages of using shared queues” on page 193.

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally, and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery* and is described in “Peer recovery” on page 266.

Peer recovery cannot recover units of work that were in doubt at the time of the failure. You can use the Automatic Restart Manager (ARM) to restart all the systems involved in the failure (CICS, Db2, and IBM MQ for example), and to ensure that they are all restarted on the same new processor. This means that they can resynchronize, and gives rapid recovery of in-doubt units of work. This is described in “Using the z/OS Automatic Restart Manager (ARM)” on page 280.

Shared channels

In the queue sharing group environment, IBM MQ provides functions that give high availability to the network. The channel initiator enables you to use networking products that balance network requests across a set of eligible servers and hide server failures from the network (for example, VTAM generic resources). IBM MQ uses a generic port for inbound requests so that attach requests can be routed to any available channel initiator in the queue sharing group. This is described in “Shared channels” on page 213.

Shared outbound channels take the messages they send from a shared transmission queue. Information about the status of a shared channel is held in one place for the whole queue sharing group level. This means that a channel can be restarted automatically on a different channel initiator in the queue sharing group if the channel initiator, queue manager, or communications subsystem fails. This is called *peer channel recovery* and is described in Shared outbound channels.

IBM MQ network availability

IBM MQ messages are carried from queue manager to queue manager in an IBM MQ network using channels. You can change the configuration at a number of levels to improve the network availability of a queue manager, and the ability of an IBM MQ channel to detect a network problem and to reconnect.

TCP *Keepalive* is available for TCP/IP channels. It causes TCP to send packets periodically between sessions to detect network failures. The KAINT channel attribute determines the frequency of these packets for a channel.

AdoptMCA allows a channel, blocked in receive processing as a result of a network outage, to be terminated and replaced by a new connection request. You control AdoptMCA using the ADOPTMCA queue manager property with the MQSC utility or the AdoptNewMCAType property with the Programmable Command Formats interface.

ReceiveTimeout prevents a channel from being permanently blocked in a network receive call. The RCVTIME and RCVTMIN channel initiator parameters, determine the receive timeout characteristics for channels, as a function of their heartbeat interval. See Queue manager parameter for more details.

Using the z/OS Automatic Restart Manager (ARM)

You can use IBM MQ for z/OS in conjunction with the z/OS automatic restart manager (ARM). If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS fails, a whole group of related subsystems and applications also fail. ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a cross-system restart.

ARM enables rapid recovery of in-doubt transactions in the shared queue environment. It also gives higher availability if you are not using queue sharing groups.

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure.

To enable automatic restart, you must do the following:

1. Set up an ARM coupling data set.
2. Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.
3. Start the ARM policy.

If you want to restart queue managers in different z/OS images automatically, every queue manager in each z/OS image on which that queue manager might be restarted must be defined with a sysplex-wide unique 4-character subsystem name.

Using ARM with IBM MQ is described in Using ARM in an IBM MQ network.

Using the z/OS Extended Recovery Facility (XRF)

You can use IBM MQ in an extended recovery facility (XRF) environment. All IBM MQ-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternative XRF processors.

If you use XRF for recovery, you must stop the queue manager on the active processor and start it on the alternative processor. For CICS, you can do this using the command list table (CLT) provided by CICS, or the system operator can do it manually. For IMS, this is a manual operation and you must do it after the coordinating IMS system has completed the processor switch.

IBM MQ utilities must be completed or terminated before the queue manager can be switched to the alternative processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent the queue manager starting on the alternative processor before the queue manager on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of IBM MQ on the two systems. You must include the BSDS as a protected resource, and you must include the active and alternative XRF processors in the GRS ring.

Using the z/OS GROUPUR attribute for recovery in a queue sharing group

Queue sharing groups (QSG) allow additional transactional facilities which are described in this topic. The GROUPUR attribute allows XA client applications to have any in-doubt transaction recovery that may be required, performed on any member of the QSG.

If an XA client application connects to a queue sharing group (QSG) through a Sysplex it cannot guarantee which specific queue manager it connects to. Use of the GROUPUR attribute by queue managers within the queue sharing group can enable any in-doubt transaction recovery that may be necessary to occur on any member of the QSG. Even if the queue manager to which the application was initially connected is not available, transaction recovery can take place.

This feature frees the XA client application from any dependency on specific members of the QSG and thus extends the availability of the queue manager. The queue sharing group appears to the transactional application as a single entity providing all the IBM MQ features and without a single queue manager point of failure.

This functionality is not apparent to the transactional application.

Where to find more information about availability

You can find more information about these topics from the following sources:

Table 24. Where to find more information about availability

Topic	Where to look
Queue sharing groups	“Shared queues and queue sharing groups” on page 174
System parameters	Configuring system parameters
Using the Automatic Restart Manager Utility programs	Using ARM in an IBM MQ network
MQSC commands	MQSC commands

z/OS Monitoring and statistics on IBM MQ for z/OS

IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

IBM MQ supplies facilities for monitoring the system and collecting statistics. For further information about these facilities, see the following sections:

- "Online monitoring" on page 282
- "IBM MQ trace" on page 282
- "Events" on page 282

Online monitoring

IBM MQ includes the following commands for monitoring the status of IBM MQ objects:

- DISPLAY CHSTATUS displays the status of a specified channel.
- DISPLAY QSTATUS displays the status of a specified queue.
- DISPLAY CONN displays the status of a specified connection.

For more information about these commands, see The MQSC commands.

IBM MQ trace

IBM MQ supplies a trace facility that you can use to gather the following information while the queue manager is running:

Performance statistics

The statistics trace gathers the following information to help you monitor performance and tune your system:

- Counts of different MQI requests (message manager statistics)
- Counts of different object requests (data manager statistics)
- Information about Db2 usage (Db2 manager statistics)
- Information about Coupling Facility usage (Coupling Facility manager statistics)
- Information about SMDS usage (shared message data set statistics)
- Information about buffer pool usage (buffer manager statistics)
- Information about logging (log manager statistics)
- Information about storage usage (storage manager statistics)
- Information about lock requests (lock manager statistics)

Accounting data

- The accounting trace gathers information about the processor time spent processing MQI calls and about the number of MQPUT and MQGET requests made by a particular user.
- IBM MQ can also gather information about each task using IBM MQ. This data is gathered as a thread-level accounting record. For each thread, IBM MQ also gathers information about each queue used by that thread.

The data generated by the trace is sent to the System Management Facility (SMF) or the generalized trace facility (GTF).

Events

IBM MQ events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple IBM MQ applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView®, to monitor reports and create the appropriate alerts.

Related tasks

Using IBM MQ trace

Using IBM MQ events

z/OS Unit of recovery disposition on z/OS

Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Transactions started by applications that have connected using the queue sharing group name also have a GROUP unit of recovery disposition.

When a transactional application connects with a GROUP unit of recovery disposition it is logically connected to the queue sharing group and does not have an affinity to any specific queue manager. Any 2-phase commit transactions that it has started that have completed phase-1 of the commit process, that is, they are in doubt, can be inquired and resolved, when connected to any queue manager within the QSG. In a recovery scenario this means that the transaction coordinator does not have to reconnect to the same queue manager, which may be unavailable at that time.

Applications that connect with a QMGR unit of recovery disposition have a direct affinity to the queue manager to which they are connected. In a recovery scenario the transaction coordinator must reconnect to the same queue manager to resolve any in-doubt transactions, irrespective of whether the queue manager belongs to a queue sharing group.

When applications specify a queue sharing group name, and thus connect to a queue manager in a QSG with a GROUP unit of recovery disposition, the queue sharing group is logically a separate resource manager. This means that in-doubt transactions are only visible to an application if it reconnects with the same unit of recovery disposition. In-doubt transactions with a QMGR unit of recovery disposition are not visible to applications that have connected with a GROUP unit of recovery disposition and vice versa.

Related concepts

“Enabling GROUP units of recovery” on page 283
A queue sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 284
Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

z/OS Enabling GROUP units of recovery

A queue sharing group can configure and enable support for GROUP units of recovery.

To use GROUP units of recovery on a queue manager within a QSG, enable the GROUPUR queue manager attribute. For more information about this concept, see “Unit of recovery disposition on z/OS” on page 283 before reading the rest of this topic.

When the GROUPUR queue manager attribute is enabled, the queue manager accepts new connections with a GROUP unit of recovery disposition. If you disable this attribute new connections with this disposition are not accepted, although applications already connected is unaffected until they disconnect.

When an application connects with a GROUP unit of recovery disposition and either inquires what transactions are in doubt or attempts to resolve a transaction that was started elsewhere in the queue

sharing group (QSG), the queue manager to which it is now connected must be able to communicate with the other members of the queue sharing group so that it can process the request. To do this it uses a shared queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE. This queue must be on a recoverable application structure called CSQSYSAPPL. The structure must be recoverable because persistent messages are stored on this queue when processing resolution requests.

Before you can enable GROUP units of recovery, you must ensure that the coupling facility structure and the shared queue are defined. You can use the definitions in the CSQ4INSS sample. When the queue is defined, or detected during startup, each queue manager in the queue sharing group opens the queue so that it can receive incoming requests. If you want to delete or move the queue because it has been defined incorrectly you can request that the queue managers close their open handles on it by updating the queue object to inhibit MQGET requests. When you have made the necessary corrections, permitting applications to get messages from the queue once more directs each queue manager to reopen it. Use the DISPLAY QSTATUS command to identify what handles are open on a queue.

When you have completed this setup you can then enable GROUP units of recovery on each queue manager that you want transactional applications to be able to connect to with a GROUP unit of recovery disposition. This need not be all of the queue managers within the queue sharing group but if you choose to only enable this functionality on a subset of the queue sharing group you must ensure that your applications only attempt to connect to queue managers on which you have enabled it. For more information, see “Application support” on page 284.

When you attempt to enable the GROUPUR queue manager attribute, a number of configuration checks are performed. The queue manager checks that:

- It belongs to a queue sharing group.
- The shared-queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE has been defined, according to the the definition in CSQ4INSS.
- The SYSTEM.QSG.UR.RESOLUTION.QUEUE is on a recoverable CF structure called CSQSYSAPPL.

If any of the above checks fail, the GROUPUR attribute remains disabled and a message code is returned.

These configuration checks are also performed at queue manager startup if the queue manager attribute is enabled. If any of the checks fail during startup GROUP units of recovery is disabled and the queue manager issues a message identifying which check failed. When you have performed the necessary corrective action you must then re-enable the queue manager attribute.

z/OS Application support

Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

Support for the GROUP unit of recovery disposition is limited to certain types of transactional applications for which IBM MQ for z/OS is a resource manager but not the transaction coordinator. Currently supported transactional applications are:

- IBM MQ extended transactional client applications
- IBM MQ classes for JMS applications running in an application server, such as WebSphere Application Server.
- CICS applications running in CICS Transaction Server 4.2 or later, when the CICS MQCONN resource definition is configured with RESYNCMEMBER(GROUPRESYNC).

Related concepts

“ IBM MQ extended transactional client applications” on page 285
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 285
Use this page to determine how CICS can use the GROUP unit of recovery disposition.

z/OS *IBM MQ extended transactional client applications*

Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

An example of an IBM MQ extended transactional client application is one that uses JMS and runs in WebSphere Application Server, connecting to IBM MQ over TCP/IP, rather than local bindings. These client applications connect to IBM MQ for z/OS over network connections, such as via TCP/IP. For these applications, it is the value specified for the QMNAME parameter of the xa_info string passed in the xa_open call that specifies whether a QMGR or GROUP unit of recovery disposition is used. For more information about xa_open, see The format of an xa_open string and Additional error processing for xa_open. For JMS applications this is done by specifying the name of the queue sharing group (QSG) in the ConnectionFactory instead of the name of a specific queue manager.

For XA client applications to take advantage of using the GROUP unit of recovery disposition you must configure your TCP/IP setup to allow your client applications to be routed to the queue managers in the queue sharing group that have the GROUPUR attribute enabled, rather than a specific queue manager. One of the dynamic virtual IP address technologies that you can use to do this is the z/OS SysPlex Distributor. See z/OS Communications Server and z/OS 基本スキル: 動的仮想アドレッシング for more details. If you want to enable GROUP units of recovery on a subset of the queue managers in your queue sharing group, ensure that your client applications cannot be routed to those on which it is not enabled.

Your client applications do not have to connect to the queue sharing group using shared channels.

z/OS *CICS applications*

Use this page to determine how CICS can use the GROUP unit of recovery disposition.

CICS 4.2 and later provides the group resynchronization option, RESYNCMEMBER(GROUPRESYNC) in an MQCONN resource definition. A CICS configured with this option can connect to any suitable queue manager in a queue sharing group which is running on the same LPAR as that CICS region. To support the CICS GROUPRESYNC option, a queue manager must be running at MQ V7.1 or later, and be enabled for GROUPUR support.

Transactions running within a CICS region connected to MQ using GROUPRESYNC create units of work with GROUP unit of recovery disposition.

You can use RESYNCMEMBER(GROUPRESYNC) to enable faster recovery after a queue manager failure as it enables the CICS region to immediately connect to an alternative eligible queue manager running on the same LPAR, resolving any indoubt transactions as necessary, without waiting for queue manager restart.

RESYNCMEMBER(GROUPRESYNC) also enables more flexible restart options for CICS. A CICS region with its MQ connection configured to use GROUPRESYNC and MQ shared queues can be restarted on any LPAR where there is a queue manager running as a member of the same queue sharing group.

z/OS IBM MQ and other z/OS products

Use this topic to understand how IBM MQ can work with other z/OS products.

Related concepts

"IBM MQ and CICS" on page 286
All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

"IBM MQ for z/OS and WebSphere Application Server" on page 292
Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Related reference

"IBM MQ and IMS" on page 287
Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

"IBM MQ and the z/OS Batch, TSO, and RRS adapters" on page 290
Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

z/OS IBM MQ and CICS

All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

For more information about configuring the IBM MQ CICS adapter, and the IBM MQ CICS bridge components, see the Configuring connections to IBM MQ section of the CICS documentation.

Related tasks

Using IBM MQ with CICS

z/OS CICS group attach

CICS group attach provides the ability for a CICS region to connect to any active member of an IBM MQ queue sharing group on the same LPAR rather than specifying an individual queue manager. CICS still connects to a single queue manager at a time.

You require at least two queue managers on the LPAR to support CICS group attach. Using group attach provides higher availability as you do not need a particular queue manager to be active. CICS connects to any queue manager in the queue sharing group on the LPAR.

For more information, see the CICS documentation on the MQCONN resource.

CICS attempts to connect to MQNAME passed as if it were a queue manager:

- If the queue manager exists and is active, the connection will work.
- If the connection fails, CICS queries the status of queue managers in the group to ascertain which are active on same LPAR.
- If multiple queue managers are active, CICS checks for RESYNCMEMBER(YES) and the UOW status to determine whether CICS needs to connect, or should connect, to a particular member, or wait if not active.
- If there is no need to connect to a particular member, CICS selects a queue manager (using a randomizing algorithm).
- CICS attempts to connect to chosen queue manager.
- If the attempt fails then, depending upon the return code, CICS chooses the next member, then goes through the selection loop again.
- If no queue managers are active, CICS issues multiple connections to the list of queue managers and waits on ECBLIST until the first queue manager becomes available.

Related concepts

“Group units of recovery (GROUPUR) for CICS” on page 286
The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

Related information

Support for IBM MQ queue sharing groups

z/OS Group units of recovery (GROUPUR) for CICS

The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

If a CICS region is working with a queue manager, and the queue manager ends abnormally, then any indoubt transactions are recovered. This eliminates the need for the CICS region to wait for the queue

manager that it was working with to restart, and then resolve any in doubt units of work. This means that you need at least two queue managers on the LPAR, so that CICS can connect to another queue manager in the event of an abnormal termination of the first queue manager.

The new RESYNCMEMBER(GROUPRESYNC) setting on the CICS MQCONN definition:

- Uses the IBM MQ group attach function and peer recovery.
- Requires a queue manager with the GROUPUR attribute enabled.
- Still supports the existing CICS MQCONN RESYNCMEMBER settings (YES and NO):
  - Uses the existing CICS group attach function and no peer recovery.
  - Changing RESYNCMEMBER settings takes effect next time CICS connects to IBM MQ.

Related concepts

"Enabling GROUP units of recovery" on page 283

A queue sharing group can configure and enable support for GROUP units of recovery.

z/OS IBM MQ and IMS

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

The optional additional IBM MQ - IMS bridge enables applications to run an IMS application that does not use the MQI. This means that you can use your legacy applications with IBM MQ, without the need to rewrite them.

For more information about these components, see the following subtopics:

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Setting up the IMS adapter

Setting up the IMS bridge

Operating the IMS adapter

Related reference

MQIIH - IMS information header

z/OS The IMS adapter

The IMS adapter is an interface between IMS application programs and an IBM MQ subsystem.

The IBM MQ adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an IBM MQ subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to IBM MQ using the External Subsystem Attach Facility (ESAF) provided by IMS. Usually, IMS connects to IBM MQ automatically without operator intervention.

The IMS adapter provides access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ.

The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ with IMS acting as the syncpoint coordinator. Conversations where IMS is not the syncpoint coordinator, for example APPC- protected (SYNCLVL=SYNCPT) conversations, are not supported by the IMS adapter.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in "The IMS trigger monitor" on page 288.

You can use IBM MQ with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error.

Note: As of IMS 15.2 Extended Recovery Facility (XRF) is no longer supported. See the IMS documentation for more information.

Using the adapter

The application programs and the IMS adapter run in the same address space. The queue manager is separate, in its own address space.

You must link-edit each program that issues one or more MQI calls to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the IBM MQ-supplied API stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to IBM MQ. However, the IMS terminal operator has no control over the IBM MQ address space. For example, the operator cannot shut down IBM MQ from an IMS address space.

Restrictions

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

The IMS trigger monitor

The IMS trigger monitor (**CSQQTRMN**) is an IBM MQ-supplied IMS application that starts an IMS transaction when an IBM MQ event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user-defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, you can start an instance of CSQQTRMN to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an INSERT (ISRT) to the IMS message queue. The started IMS application reads the message from the application message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. When it has started, the trigger monitor runs until IBM MQ or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions that are started by the trigger monitor contain the following:

- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is protected by Security Server (previously known as RACF), you might need to define CSQQTRMN as a user ID to Security Server.

z/OS The IBM MQ - IMS bridge

The IBM MQ - IMS bridge is the component of IBM MQ for z/OS that allows direct access from IBM MQ applications to applications on your IMS system.

The IBM MQ - IMS bridge enables *implicit MQI support*. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by IBM MQ messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access* (OTMA) client.

In bridge applications there are no IBM MQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. IBM MQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals. If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one IBM MQ message.

The IMS bridge is illustrated in Figure 78 on page 289.

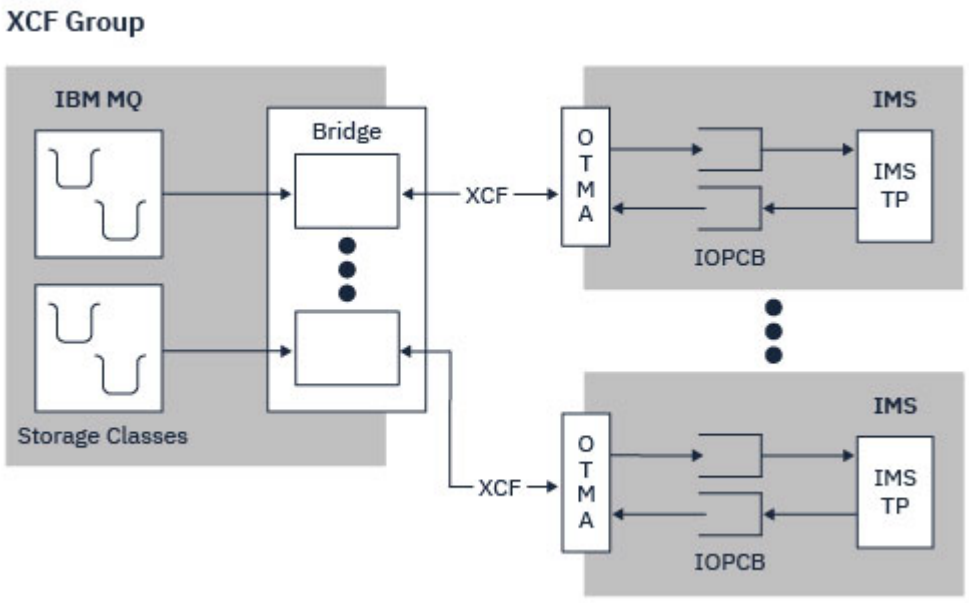


Figure 78. The IBM MQ - IMS bridge

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

See Setting up the IMS bridge for information on setting up an IMS bridge and adding an additional IMS connection to the same queue manager.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS. It functions as an interface for host-based communications servers accessing IMS TM applications through the z/OS Cross Systems coupling facility (XCF).

OTMA enables clients to connect to IMS to provide high performance for interactions between clients and IMS for a large network or large number of sessions. OTMA is implemented in a z/OS sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA Resource Monitoring

Support for the x'3C' OTMA protocol messages, available in IMS v10 or higher, is included in the IBM MQ - IMS bridge in IBM MQ for z/OS. These messages are sent to OTMA clients by IMS to report its health status.

If an IMS partner is unable to process the volume of transaction requests being sent then it will notify IBM MQ that a flood warning has occurred. In response IBM MQ will slow down the rate at which requests are sent over the bridge.

If IMS is still unable to process the transaction requests and a full flood condition occurs all TPIPEs to the IMS partner are suspended. Upon notification from the IMS partner that the flood or flood-warning condition has been relieved IBM MQ will resume all suspended TPIPEs, if appropriate, and gradually increase the rate at which transaction requests are sent until the maximum rate is achieved. Console messages are issued by IBM MQ in response to a change in the status of IMS partners.

If IMS v10 partners are being used you should ensure that PTF UK45082 has been applied.

Submitting IMS transactions from IBM MQ

To submit an IMS transaction that uses the bridge, applications put messages on an IBM MQ queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the IBM MQ - IMS bridge to make assumptions about the data in the message.

IBM MQ then puts the message to an IMS queue (it is queued in IBM MQ first to enable the use of syncpoints to assure data integrity). The storage class of the IBM MQ queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the IBM MQ - IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on IBM MQ for z/OS.

Data returned from the IMS system is written directly to the IBM MQ reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the **ReplyToQMgr** field of the MQMD.)

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Customizing the IMS bridge

Related reference

“IBM MQ and IMS” on page 287

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

z/OS IBM MQ and the z/OS Batch, TSO, and RRS adapters

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

Introduction to the Batch adapters

The Batch/TSO adapters are the interface between IBM MQ and z/OS application programs running under JES, TSO, or z/OS UNIX System Services. These adaptors enable z/OS application programs to use the MQI.

The adapters provide access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

Connections between application programs and IBM MQ are at the task level. The adapters provide a connection thread from an application task control block (TCB) to IBM MQ.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ. It does not support multi-phase commit protocols. The RRS adapter enables IBM MQ applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by z/OS Resource Recovery Services (RRS).

The adapters use the z/OS STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the IBM MQ termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in IBM MQ (for example, a signal or a wait).

The Batch/TSO adapter

The IBM MQ Batch/TSO adapter provides IBM MQ support for z/OS Batch and TSO applications. All application programs that run under z/OS Batch or TSO must have the API stub program CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

The RRS adapter

Resource Recovery Services (RRS) is a subcomponent of z/OS that provides a system-wide service for coordinating two-phase commit across z/OS products. The IBM MQ Batch/TSO RRS adapter (the RRS adapter) provides IBM MQ support for z/OS Batch and TSO applications that want to use these services. The RRS adapter enables IBM MQ to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, Db2).

The RRS adapter provides two stubs; you must link-edit application programs that want to use RRS with one of these stubs.

CSQBRSTB
: This stub allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**.

  You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code MQRC_ENVIRONMENT_ERROR.

CSQBRRSI
: This stub allows you to use MQI calls **MQCMIT** and **MQBACK** ; IBM MQ actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see The RRS batch adapter.

Where to find more information about the z/OS Batch, TSO, and RRS adapters

You can find more information about the topics in this section in the following sources:

Table 25. Where to find more information about using z/OS Batch with IBM MQ	
Topic	**Where to look**
Setting up the Batch adapters	Task 19: Set up Batch, TSO, and RRS adapters
RRS callable resource recovery services	*MVS Programming: Callable Services for High Level Languages*

z/OS IBM MQ for z/OS and WebSphere Application Server

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Applications written in Java that are running under WebSphere Application Server can use the Java Message Service (JMS) specification to perform messaging. Point-to-point messaging in this environment can be provided by an IBM MQ for z/OS queue manager.

A benefit of using an IBM MQ for z/OS queue manager to provide the messaging is that connecting JMS applications can participate fully in the functionality of an IBM MQ network. For example, they can use the IMS bridge, or exchange messages with queue managers running on other platforms.

Connection between WebSphere Application Server and a queue manager

See Using IBM MQ and WebSphere Application Server together for more information.

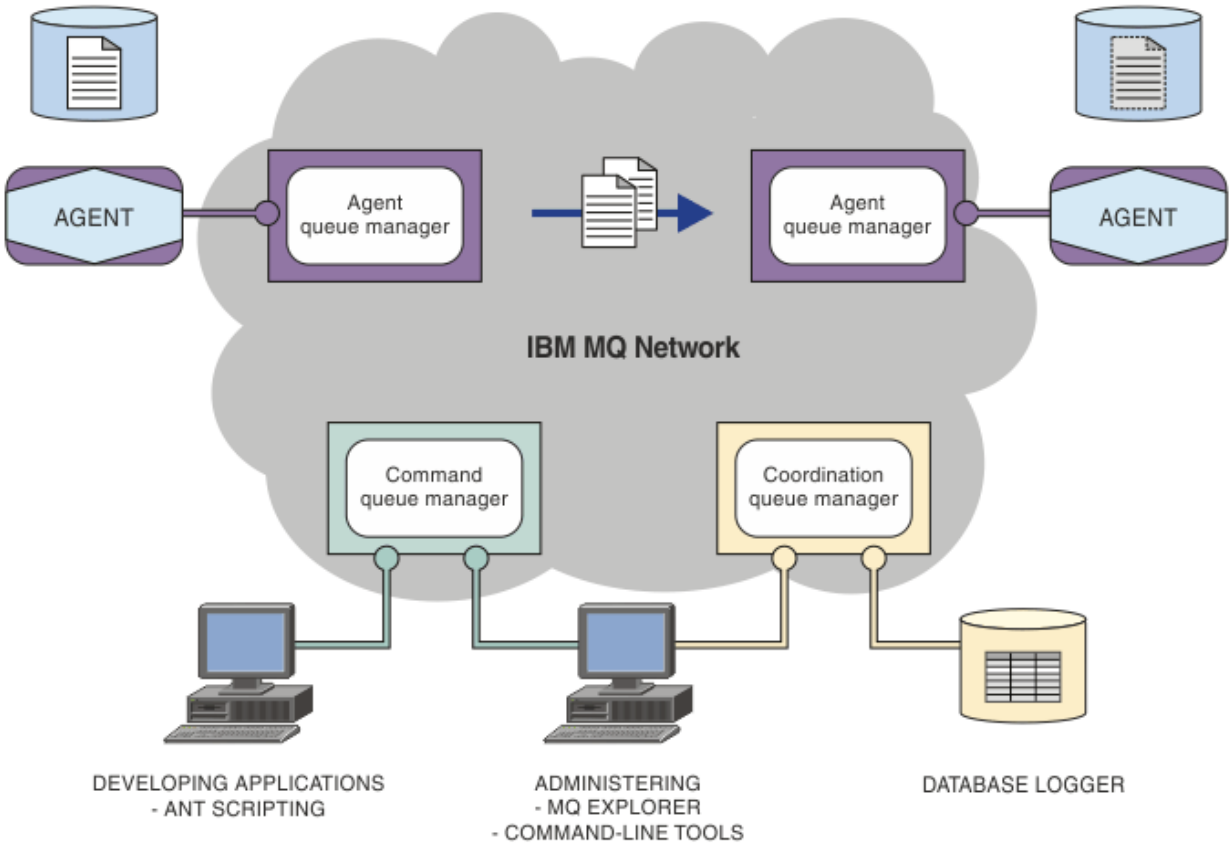
Using IBM MQ functions from JMS applications

By default, JMS messages held on IBM MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy IBM MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for IBM MQ Workflow applications. For more details about these special considerations, see Mapping JMS messages onto IBM MQ messages.

Managed File Transfer

Managed File Transfer は、ファイルのサイズや使用するオペレーティング・システムにかかわりなく、システム間のファイル転送を管理下に置いて実行できます。監査も可能です。

Managed File Transfer を使用すれば、ファイル転送を管理し、確認し、保護するために、カスタマイズしたスケーラブルな自動化ソリューションを構築できます。 Managed File Transfer によって、コストのかかる冗長性を除去し、保守コストを削減し、既存の IT 投資を最大限に活用することが可能になります。



この図は、単純な Managed File Transfer トポロジーを示しています。 2 つのエージェントがあり、それぞれが IBM MQ ネットワーク内の独自のエージェント・キュー・マネージャーに接続しています。 図の一方にあるエージェントから IBM MQ ネットワークを経由して、図のもう一方にあるエージェントにファイルを転送します。 さらに、IBM MQ ネットワークには、調整キュー・マネージャーとコマンド・キュー・マネージャーもあります。 アプリケーションおよびツールは、これらのキュー・マネージャーに接続して、Managed File Transfer ネットワーク内の IBM MQ アクティビティーを構成、管理、操作、およびログに記録します。

Managed File Transfer には、オペレーティング・システムと全体的なセットアップに応じて、4 種類のインストール・オプションがあります。 これらのオプションは、Managed File Transfer Agent、Managed File Transfer Logger、Managed File Transfer Service、または Managed File Transfer Tools です。 詳しくは、Managed File Transfer 製品のオプションを参照してください。

Managed File Transfer を使用して、次のタスクを実行できます。

- 管理対象ファイル転送を作成します。
  - Windows Linux Linux または Windows プラットフォーム上の IBM MQ Explorer から新規ファイル転送を作成します。
  - サポートされているすべてのオペレーティング・システムで、コマンド・ラインから新しいファイル転送を作成できます。
  - ファイル転送機能を Apache Ant ツールに組み込みます。
  - エージェント・コマンド・キューにメッセージを PUT することによって、Managed File Transfer を制御するアプリケーションを作成します。
  - ファイル転送は、後の時点で実行されるようにスケジュールに入れます。 また、スケジュール済みファイル転送を、一定の範囲のファイル・システム・イベント (例えば、新規ファイルの作成など) に基づいてトリガーすることもできます。

  - 例えばディレクトリーなどのリソースを継続的にモニターして、そのリソースの内容が事前定義の条件に一致した場合にタスクを開始します。 このタスクは、ファイル転送、Ant スクリプト、または JCL ジョブにすることができます。
  - IBM MQ キューとの間のファイル転送が可能です。
  - FTP サーバー、FTPS サーバー、または SFTP サーバーとの間でファイル転送が可能です。
  - Connect:Direct® ノードとの間のファイル転送が可能です。
  - テキスト・ファイルとバイナリー・ファイルの両方の転送が可能です。 テキスト・ファイルの場合、ソース・システムと宛先システムの間でコード・ページと行の終わり規則が自動的に変換されます。
  - 転送は、SSL (Secure Socket Layer) ベース接続の業界規格を使用して保護できます。
- 転送の進行状況を表示することや、ネットワーク内のすべての転送に関する情報をログに記録することが可能です。
  - Windows Linux Linux または Windows プラットフォーム上の IBM MQ Explorer から、進行中の転送の状況を表示します。
  - Windows Linux Linux または Windows プラットフォームで IBM MQ Explorer を使用して、完了した転送の状況を確認します。
  - Managed File Transfer のデータベース・ロガー機能を使用して、Db2 または Oracle データベースにログ・メッセージを保存します。

Managed File Transfer は、IBM MQ の基盤の上に構築されている製品であり、アプリケーション間の 1 回限りのメッセージ配信を確実に実行できるようになっています。 IBM MQ のさまざまなフィーチャーを活用することができます。 例えば、チャネル圧縮を使用して、IBM MQ チャネルを介してエージェント間で送信するデータを圧縮し、SSL チャネルを使用して、エージェント間で送信するデータを保護することができます。 ファイルは安全に転送され、ファイル転送を行う媒体となるインフラストラクチャーで発生した障害に対処する機能があります。 ネットワーク障害が発生した場合、接続が復元されたときに、ファイル転送は中止された位置から再開します。

ファイル転送を既存の IBM MQ ネットワークと統合することにより、2 つの別個のインフラストラクチャーを保守して必要なリソースを浪費するということを避けられます。 まだ IBM MQ のお客様でない場合は、Managed File Transfer をサポートする IBM MQ ネットワークを作成することにより、将来の SOA 実装のためのバックボーンを構築します。 既にお客様が IBM MQ 顧客であれば、Managed File Transfer は IBM MQ や IBM MQ Internet Pass-Thru などの既存の IBM Integration Bus インフラストラクチャーを利用できます。

IBM MQ 高可用性ソリューションを利用して、 Managed File Transfer 構成の回復力を向上させることができます。 エージェントが複製データ・キュー・マネージャー (RDQM) を使用する場合は、浮動 IP アドレス機能を使用するように構成する必要があります。 これは、エージェントが同じ IP アドレスを使用して、現在実行中の 3 つの RDQM インスタンスのいずれかと通信し、フェイルオーバー時に自動的に再接続することを意味します (RDQM 高可用性 および 浮動 IP アドレスの作成と削除を参照してください)。 複数インスタンス・キュー・マネージャー・ソリューションを使用する場合、アプリケーションは、各インスタンスとの通信に異なる IP アドレスを使用します。これは、フェイルオーバー時にクライアント再接続によって処理されます (複数インスタンス・キュー・マネージャー および チャネルとクライアントの再接続を参照)。

Managed File Transfer は、以下のように、他の多くの IBM 製品との統合が可能です。

IBM Integration Bus

Managed File Transfer によって転送されたファイルを IBM Integration Bus フローの一部として処理できます。 詳しくは、Working with MFT from IBM Integration Bus を参照してください。

IBM Sterling Connect:Direct

Managed File Transfer Connect:Direct ブリッジを使用して、既存の Connect:Direct ネットワークとの間でファイルを転送します。 詳しくは、Connect:Direct ブリッジを参照してください。

IBM Tivoli® Composite Application Manager

IBM Tivoli Composite Application Manager には、調整キュー・マネージャーにパブリッシュされた情報をモニターするために使用できるエージェントが用意されています。

関連概念

Managed File Transfer 製品のオプション

295 ページの『MFT トポロジーの概要』
Managed File Transfer エージェントが IBM MQ ネットワーク内の調整キュー・マネージャーとどのように接続されるかについての概要。

295 ページの『MFT と IBM MQ の連動について』
Managed File Transfer は、さまざまな方法で IBM MQ と対話します。

MFT と IBM MQ の連動について

Managed File Transfer は、さまざまな方法で IBM MQ と対話します。

- Managed File Transfer は、各ファイルを 1 つ以上のメッセージに分割し、それらメッセージを IBM MQ ネットワークを介して送信することにより、エージェント・プロセス間でファイルを転送します。
- エージェント・プロセスは、IBM MQ ログに対する影響を最小化するために、非永続メッセージを使用してファイル・データを移動します。 エージェント・プロセスは、相互にやり取りすることにより、ファイル・データが含まれるメッセージのフローを調整します。 このようにして、ファイル・データが含まれているメッセージが IBM MQ 伝送キューに蓄積される状況が回避され、いずれかの非永続メッセージが送信されなかった場合にファイル・データが確実に再送信されるようになります。
- Managed File Transfer エージェントは、いくつかの IBM MQ キューを使用します。 詳しくは、MFT システム・キューおよびシステム・トピックを参照してください。
- これらのキューの一部は内部使用に限られていますが、エージェントは、読み取り先の特定のキューに送信される特殊形式のコマンド・メッセージの形で要求を受け入れることができます。 コマンド行コマンドおよび IBM MQ Explorer・プラグインの両方は、IBM MQ メッセージをエージェントに送信し、対象となるアクションを実行するようにエージェントに指示します。 このような方法でエージェントと対話する IBM MQ アプリケーションを作成できます。 詳しくは、エージェント・コマンド・キューへのメッセージの書き込みによる MFT の制御を参照してください。
- Managed File Transfer エージェントは、その状態と、転送の進行状況と結果に関する情報を、調整キュー・マネージャー として指定されている MQ キュー・マネージャーに送信します。 この情報は、調整キュー・マネージャーによりパブリッシュされ、転送の進行状況のモニターまたは発生した転送の記録を行うアプリケーションによってサブスクライブできます。 コマンド行コマンドおよび IBM MQ Explorer・プラグインの両方で、パブリッシュされた情報を利用できます。 この情報を使用する IBM MQ アプリケーションを作成できます。 情報がパブリッシュされるトピックについて詳しくは、 SYSTEM.FTE トピック。
- Managed File Transfer のキー・コンポーネントは、IBM MQ キュー・マネージャーの機能を利用してメッセージのストア・アンド・フォワード処理を行います。 これは、故障が発生した場合、インフラストラクチャーの中で影響を受けていない部分はファイルの転送を続行できることを意味します。 このことは調整キュー・マネージャーにも当てはまります。ストア・アンド・フォワードと永続サブスクリプションの組み合わせにより、調整キュー・マネージャーは、使用不可状態になっても行われたファイル転送に関する主要な情報を失うことなく対処できます。

MFT トポロジーの概要

Managed File Transfer エージェントが IBM MQ ネットワーク内の調整キュー・マネージャーとどのように接続されるかについての概要。

Managed File Transfer エージェントは、転送されるファイルを送受信します。 エージェントはそれぞれ、関連付けられているキュー・マネージャーに対して独自の一連のキューを持ちます。 エージェントはバインディング・モードまたはクライアント・モードのいずれかでキュー・マネージャーに接続されます。 また、エージェントはそのキュー・マネージャーとして調整キュー・マネージャーを使用することもできます。

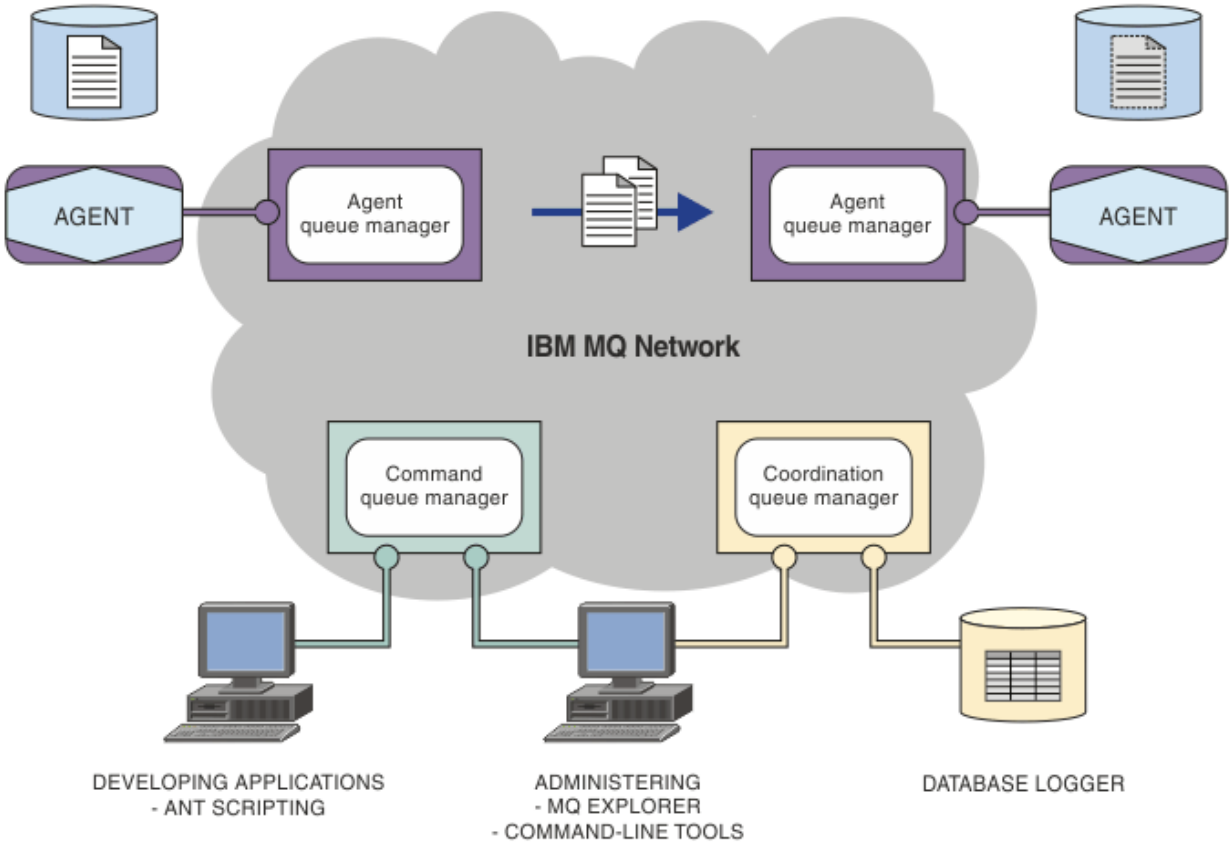
調整キュー・マネージャーは、監査およびファイル転送情報をブロードキャストします。 調整キュー・マネージャーは、エージェント、転送状況、および転送監査の情報を収集するためのシングル・ポイントの役割を果たします。 調整キュー・マネージャーが使用可能な状態になっていることは、転送を実行するための必要条件ではありません。 調整キュー・マネージャーが一時的に使用できなくなった場合でも、転送処理は、通常どおり続行されます。 監査メッセージと状況メッセージは、調整キュー・マネージャーが使

用できるようになって、通常の処理が可能になるまで、エージェント・キュー・マネージャーに格納されます。

エージェントは調整キュー・マネージャーに登録され、詳細をそのキュー・マネージャーにパブリッシュします。 このエージェント情報は、IBM MQ Explorer からの転送を開始できるようにするために Managed File Transfer プラグインによって使用されます。 各種のコマンドも、エージェントの情報と状況を表示するために、調整キュー・マネージャーで収集されるエージェント情報を使用します。

転送状況と転送監査の情報は、調整キュー・マネージャーでパブリッシュされます。 Managed File Transfer プラグインはその転送状況と転送監査の情報を使用して、IBM MQ Explorer から転送の進行状況をモニターします。 監査能力を確保するために、調整キュー・マネージャーに格納される転送監査の情報を保存することもできます。

IBM MQ ネットワークに接続する場合にはこのコマンド・キュー・マネージャーが使用され、このコマンド・キュー・マネージャーが Managed File Transfer のコマンドを発行するときに接続されるキュー・マネージャーとなります。



関連概念

292 ページの『Managed File Transfer』
Managed File Transfer は、ファイルのサイズや使用するオペレーティング・システムにかかわりなく、システム間のファイル転送を管理下に置いて実行できます。監査も可能です。

295 ページの『MFT と IBM MQ の連動について』
Managed File Transfer は、さまざまな方法で IBM MQ と対話します。

Managed File Transfer シナリオ

MFT REST API の概要

REST API は、転送のリスト表示やファイル転送エージェントの詳細など、特定の Managed File Transfer コマンドをサポートします。

REST API には、現在のすべての Managed File Transfer 転送をリストするためのオプションと、Managed File Transfer エージェントの状況を照会するためのオプションが含まれています。 詳しくは、Getting started with the REST API MFT を参照してください。

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru (MQIPT) は IBM MQ のオプション・コンポーネントで、インターネットを介してリモート・サイト間のメッセージング・ソリューションを実装するために使用できます。

IBM MQ 9.4.x 用の MQIPT インストール・ファイルを入手するには、IBM Fix Central for IBM MQ にアクセスしてください。

MQIPT を使用して、サポートされている任意のバージョンの IBM MQ を接続できます。 MQIPT と同じバージョンの他の IBM MQ コンポーネントをインストールする必要はありません。

IBM MQ のライセンスを購入した場合は、必要な数の MQIPT のコピーをインストールできます。 MQIPT インストール済み環境は、購入した IBM MQ ライセンスを消費するものとしてカウントされません。 IBM MQ ライセンスについて詳しくは、 IBM MQ ライセンス情報を参照してください。

注 : この資料は、 IBM MQ 9.4 の MQIPT に関連しています。 IBM Documentation の MQIPT サポート・パック (バージョン 2.1) の資料については、 IBM MQ 9.0 資料の MQIPT (SupportPac MS81) を参照してください。

注 : MQIPT 2.1 以前を使用している場合は、MQIPT サポート・パックのサポート終了日が 2020 年 9 月 30 日 30th ため、 MQIPT for IBM MQ 9.4 にアップグレードすることをお勧めします。

IBM MQ Internet Pass-Thru は、2 つの IBM MQ キュー・マネージャー間、または IBM MQ クライアントと IBM MQ キュー・マネージャー間の IBM MQ メッセージ・フローを受け取って転送することができる、スタンドアロンのサービスとして実行されます。

クライアントとサーバーが同じ物理ネットワーク上にない場合、MQIPT によってこのような接続が使用可能になります。

MQIPT の 1 つ以上のインスタンスを、2 つの IBM MQ キュー・マネージャー間の通信パス、または IBM MQ クライアントと IBM MQ キュー・マネージャーの間の通信パスに配置することができます。 MQIPT のインスタンスによって、2 つの IBM MQ システムが、2 つのシステム間に直接 TCP/IP 接続がなくてもメッセージを交換することができます。 このアーキテクチャーは、ファイアウォール構成で 2 つのシステム間の直接 TCP/IP 接続が禁止されている場合に役立ちます。

MQIPT は、1 つ以上の TCP/IP ポートで着信接続を listen します。 これらの接続は、通常の IBM MQ メッセージ、HTTP 内部でトンネリングされた IBM MQ メッセージ、または Transport Layer Security (TLS) または Secure Sockets Layer (SSL) を使用して暗号化されたメッセージのいずれかを伝送することができます。 MQIPT は複数の同時接続を処理することができます。

最初に TCP/IP 接続要求を行う IBM MQ チャネルは呼び出し側 と呼ばれ、接続先のチャネルは応答側 と呼ばれます。また、最終的な接続先となるキュー・マネージャーは宛先キュー・マネージャー と呼ばれます。

MQIPT は、ソースから宛先にデータを転送する際、そのデータをメモリー内に保持します。 データはディスクに保存されません (ただし、オペレーティング・システムがディスクにページングするメモリーを除きます)。 MQIPT がディスクに明示的にアクセスするのは、その構成ファイルを読み取り、接続ログおよびトレース・レコードを書き込む場合のみです。

IBM MQ チャネル・タイプの全範囲は、MQIPT の 1 つ以上のインスタンスを介して接続できます。 通信パスに MQIPT が存在しても、接続されている IBM MQ コンポーネントの機能特性には影響しません。 ただし、メッセージ転送のパフォーマンスが影響を受ける可能性があります。

MQIPT は、 301 ページの『MQIPT の考えられる構成』 で説明されているように、 IBM MQ で使用できます。

MQIPT をインストールするには、MQIPT のインストールを参照してください。

関連タスク

IBM MQ Internet Pass-Thru の構成

IBM MQ Internet Pass-Thru の管理および構成

関連資料
IBM MQ Internet Pass-Thru 構成リファレンス

MQIPT の用途

IBM MQ Internet Pass-Thru (MQIPT) は以下のようなさまざまな用途に使用することができます。

MQIPT をチャネル・コンセントレーターとして使用する

この用途で MQIPT を使用することによって、ファイアウォールで、複数の個別のホストとの間のチャネルをすべて MQIPT ホストとの間のチャネルであるかのように認識できます。 これにより、ファイアウォールのフィルタリング・ルールの定義と管理が容易になります。

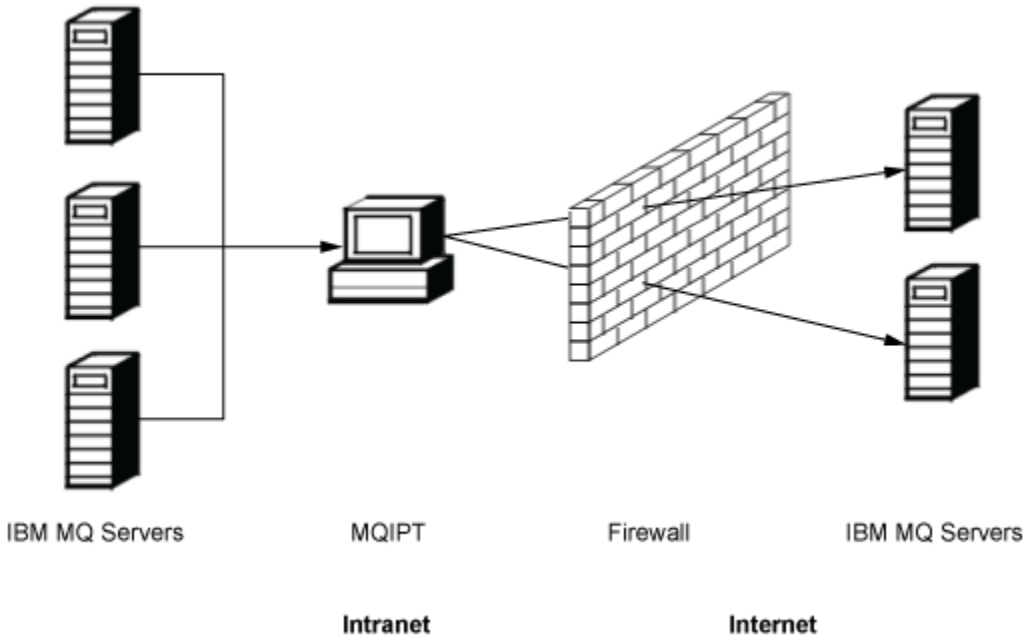


図 *79.* チャネル・コンセントレーターとしての *MQIPT* の例

MQIPT を DMZ に配置して単一アクセス・ポイントを提供する

信頼された既知のインターネット・プロトコル (IP) アドレスを持つコンピューター上で、DMZ ファイアウォール内に MQIPT が配置されている場合 (ローカル・エリア・ネットワークを保護するためのファイアウォール構成)、MQIPT を使用して、着信 IBM MQ チャネル接続を listen し、それを信頼されたイントラネットに転送することができます。内側のファイアウォールでは、この信頼されたコンピューターがインバウンド接続を確立できるよう許可する必要があります。 この構成では、MQIPT により、外部のアクセス要求が、信頼されたイントラネット内のコンピューターの本当の IP アドレスを受信することができません。 このようにして、MQIPT は、単一アクセス・ポイントを提供します。 必要に応じて、TLS 接続を受け入れ、別個の TLS 接続を使用してデータを宛先に転送するように MQIPT を構成できます。これにより、DMZ 内の TLS セッションが終了します。

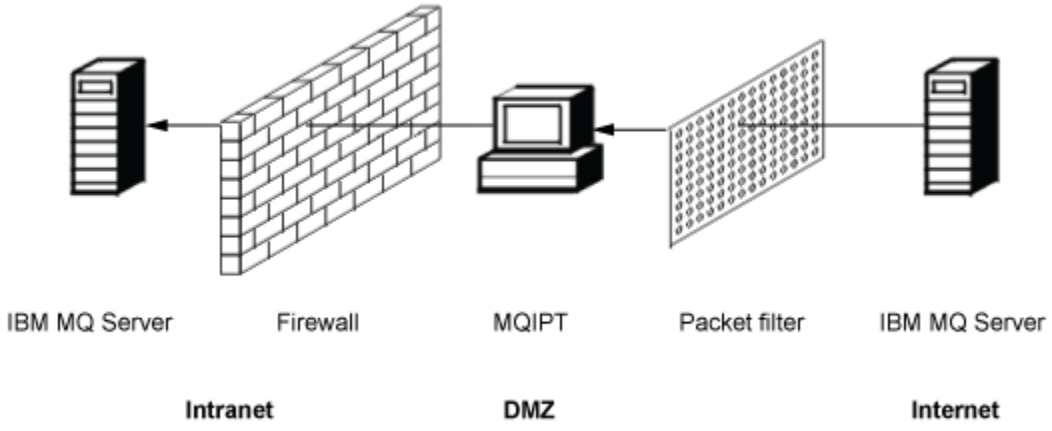


図 *80. DMZ* ファイアウォール内の *MQIPT* の例

HTTP トンネリングによって通信できる MQIPT

直列でデプロイされている 2 つの MQIPT インスタンスは、HTTP を使用して通信することができます。 HTTP トンネリング機能によって、既存の HTTP プロキシーを使用することによりファイアウォールを介して要求を伝送することが可能です。 最初の MQIPT は HTTP に IBM MQ プロトコルを挿入し、2 番目のものはその HTTP ラッパーから IBM MQ プロトコル を抽出して、それを宛先キュー・マネージャーに送ります。

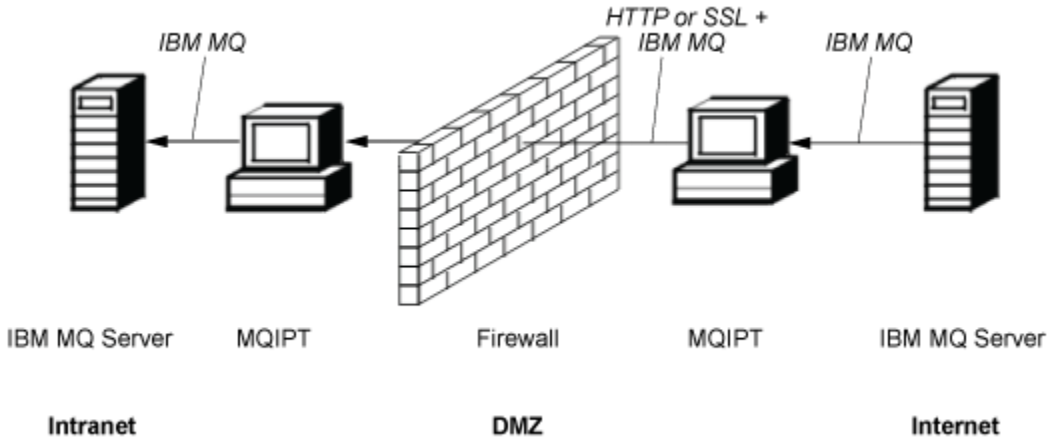


図 *81. MQIPT* および *HTTP* トンネリングの例

MQIPT でメッセージを暗号化する

前の例のように MQIPT が構成されている場合、要求は、暗号化した後にファイアウォールを介して送信することができます。 1 つ目の MQIPT がデータを暗号化し、2 つ目が SSL/TLS を使用して復号した後、宛先キュー・マネージャーに送信します。

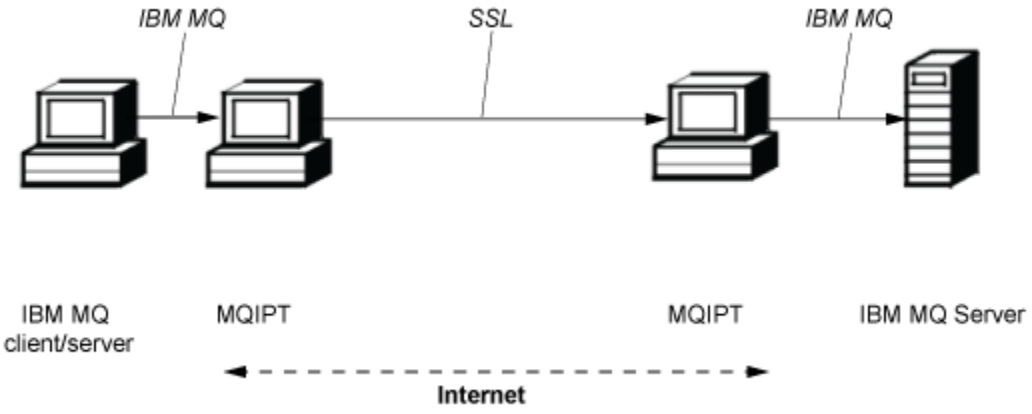


図 *82. MQIPT* および *SSL/TLS* の例

MQIPT の動作

最も単純な構成では、MQIPT は IBM MQ の プロトコル・フォワーダーとして機能します。 TCP/IP ポートを listen し、IBM MQ チャネルからの 接続要求を受け入れます。

正しい形式の要求を受信すると、MQIPT は、 自身と宛先 IBM MQ キュー・マネージャーとの間で TCP/IP 接続を 確立します。 次に、着信接続から受信したすべてのプロトコル・パケットを 宛先キュー・マネージャーに渡してから、宛先キュー・マネージャーからのプロトコル・パケットを元の着信接続に戻します。

両側とも中継の存在について直接認識していないため、IBM MQ プロトコル (クライアント/サーバーまたはキュー・マネージャー対キュー・マネージャー) に変更を 加える必要はありません。 IBM MQ クライアントまたはサーバーのコードの新規バージョンは 必要ありません。

MQIPT を使用するには、宛先キュー・マネージャーの ホスト名とポートではなく、MQIPT のホスト名とポートを使用するように、呼び出し元チャネルを構成する必要があります。 これは、 IBM MQ チャネルの **CONNAME** プロパティーで定義されます。 MQIPT は 着信データを読み取り、単純にそれを宛先キュー・マネージャーに渡します。 クライアント/サーバー・チャネル内の ユーザー ID やパスワードなどの他の構成フィールドも、同様に宛先キュー・マネージャーに渡されます。

複数のキュー・マネージャー

MQIPT を使用して、複数の宛先キュー・マネージャーへのアクセスを許可できます。 このためには、どのキュー・マネージャーに接続するかを MQIPT に通知するためのメカニズムが 必要です。MQIPT は、着信 TCP/IP ポート番号を使用して、 接続するキュー・マネージャーを判別します。

したがって、複数の TCP/IP ポートを listen するように MQIPT を構成できます。 各リスニング・ポートは、MQIPT の経路 を介して 宛先キュー・マネージャーにマップされます。 リスニング TCP/IP ポートを宛先キュー・マネージャーのホスト名とポートに関連付ける経路を、最大 100 まで定義できます。 これは、宛先キュー・マネージャーのホスト名 (IP アドレス) は、発生元のチャネルからは見えないことを意味します。各経路は、リスニング・ポートと宛先間の複数の接続を処理できます。各接続は独立して機能します。

MQIPT 構成ファイル

MQIPT は `mqipt.conf` という構成ファイルを使用します。 このファイルには、すべての経路およびその関連プロパティーの定義が 含まれています。 `mqipt.conf` について詳しくは、「IBM MQ Internet Pass-Thru の管理および構成」を参照してください。

MQIPT を起動すると、構成ファイルにリストされている各経路が 開始されます。 各経路の状況を示すメッセージがシステム・コンソールに 書き込まれます。 経路についてメッセージ MQCPI078 が表示される場合、その経路で 接続要求を受け入れる準備ができたことを示しています。

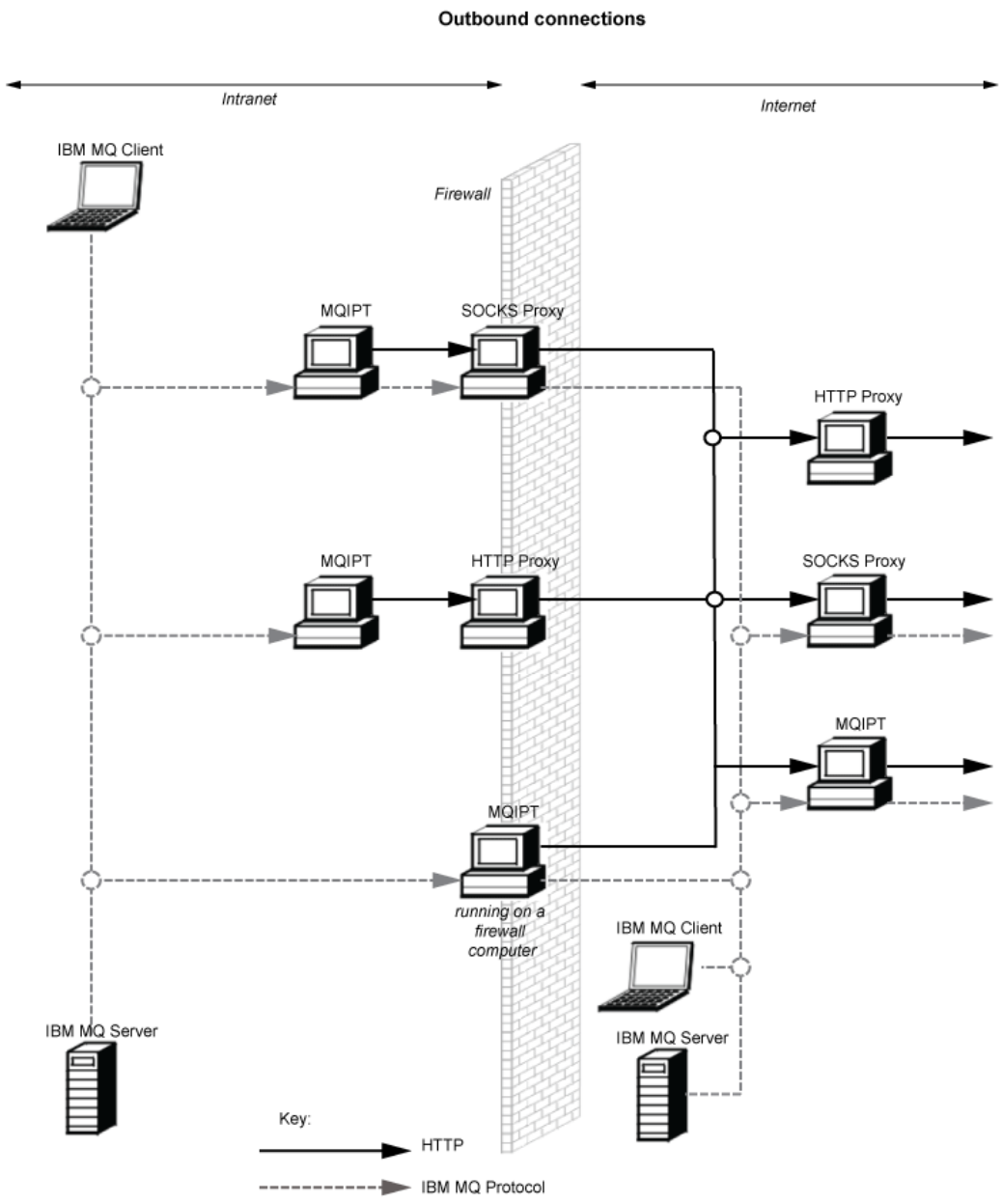
MQIPT の考えられる構成

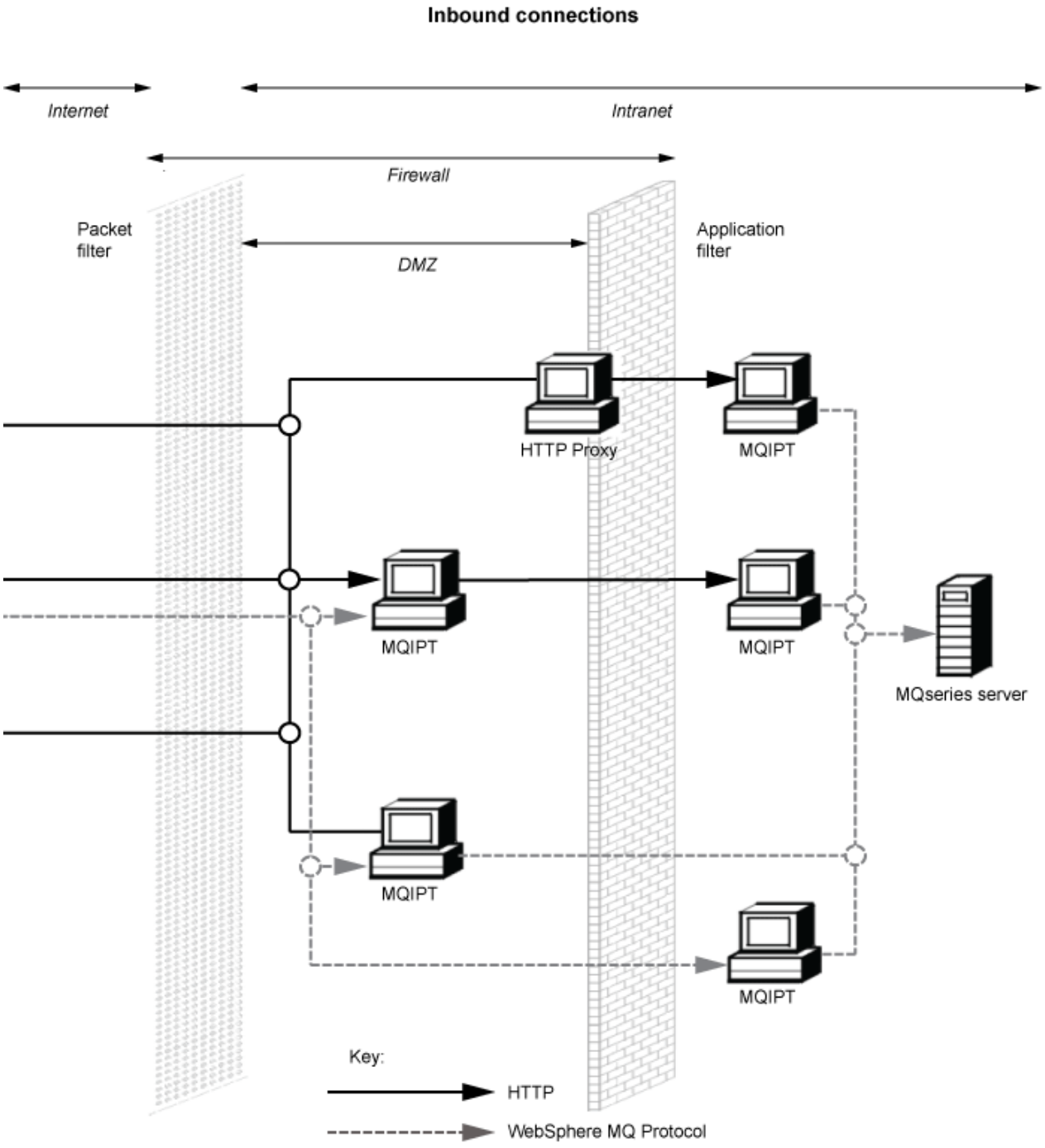
MQIPT は IBM MQ および IBM Integration Bus と組み合わせて使用できます。

以下のマルチパートの図は、IBM MQ トポロジー内の MQIPT で可能な構成の多くを示しています。 ここには、MQIPT がメッセージを送信できるさまざまな方法が示されています。 イントラネット、ファイアウォールの内側、およびファイアウォールの外側のインターネット上にあるクライアントとサーバーが、メッセージを MQIPT、HTTP プロキシー、または SOCKS プロキシーに渡して転送する様子が示されています。

インバウンド・ファイアウォールを介してサーバーにメッセージが渡される前に、DMZ の MQIPT プロキシーまたは HTTP プロキシーがメッセージを受信します。

ファイアウォールのイントラネット側の HTTP プロキシー、SOCKS プロキシー、および MQIPT コンピューターは、インターネット上で複数のコンピューターが相互にチェーニングされる可能性があることを示していることに注意してください。 例えば、MQIPT コンピューターは、ターゲットに達する前に、1 つ以上の SOCKS または HTTP プロキシー・コンピューター、さらに MQIPT コンピューターを介して通信する可能性があります。


Outbound connections
Intranet
Internet
IBM MQ Client
Firewall
MQIPT
SOCKS Proxy
HTTP Proxy
MQIPT
HTTP Proxy
SOCKS Proxy
MQIPT
MQIPT
running on a firewall computer
IBM MQ Client
IBM MQ Server
IBM MQ Server
Key:
HTTP
IBM MQ Protocol



互換性のある構成

IBM MQ クライアントまたはキュー・マネージャーが MQIPT と通信する、互換性のある接続シナリオ。 同一または 2 つ目の MQIPT 経路が、宛先キュー・マネージャーとの通信に使用されます。

単一の MQIPT 経路を使用した互換性のある構成

単一の MQIPT 経路を使用して、IBM MQ と通信することができます。

304 ページの表 26 の列には、以下の情報が含まれます。

1. IBM MQ と MQIPT の経路の間で使用されるプロトコル。 接続は IBM MQ クライアントまたはキュー・マネージャーのいずれかから作成でき、IBM MQ Formats and Protocols (FAP) または SSL/TLS プロトコルのいずれかを使用できます。
2. MQIPT 経路が作動するモード。 MQIPT と IBM MQ との間のインターネット経由での通信形式は、MQIPT 経路の構成によって決まります。 表で SSL が言及されている場合は TLS も使用することができます。
3. MQIPT 経路と宛先キュー・マネージャーの間で使用されるプロトコル。

表 *26. MQIPT* の単一インスタンスでの有効な構成

1 秒. IBM MQ ソース・プロトコル	**2. MQIPT 経路のモード**	**3. IBM MQ 宛先プロトコル**
FAP	FAP プロキシー (デフォルト)	FAP
	FAP サーバーおよび SSL クライアント	SSL/TLS
SSL/TLS	SSL プロキシー	SSL/TLS
	SSL サーバーおよび FAP クライアント	FAP
	SSL サーバーおよび SSL クライアント	SSL/TLS

複数の MQIPT 経路を使用した互換性のある構成

1 つ以上の MQIPT インスタンスで複数の経路を使用して、IBM MQ と通信するよう選択することができます。

305 ページの表 27 の列には、以下の情報が含まれます。

1. IBM MQ と 1 つ目の MQIPT 経路の間で使用されるプロトコル。 接続は IBM MQ クライアントまたはキュー・マネージャーのいずれかから作成でき、IBM MQ Formats and Protocols (FAP) または SSL/TLS プロトコルのいずれかを使用できます。
2. 1 つ目の MQIPT 経路が作動するモード。 MQIPT と IBM MQ との間のインターネット経由での通信形式は、MQIPT 経路の構成によって決まります。 表で SSL が言及されている場合は TLS も使用することができます。
3. 2 つ目の MQIPT 経路が作動するモード。
4. 2 つ目の MQIPT 経路と宛先キュー・マネージャーの間で使用されるプロトコル。

表 27. *MQIPT* の複数インスタンスでの有効な構成

1 秒. IBM MQ ソース・プロトコル	2. 1 つ目の MQIPT 経路のモード	3. 2 つ目の MQIPT 経路のモード	4. IBM MQ 宛先プロトコル
FAP (デフォルト)	FAP プロキシー (デフォルト)	FAP プロキシー (デフォルト)	FAP
	FAP サーバーおよび SSL クライアント	SSL プロキシー	SSL/TLS
		SSL サーバーおよび FAP クライアント	FAP
		SSL サーバーおよび SSL クライアント	SSL/TLS
	HTTP クライアント	HTTP サーバーおよび SSL クライアント	SSL/TLS
	HTTPS クライアント	HTTPS サーバーおよび SSL クライアント	SSL/TLS
	HTTP クライアント	HTTP サーバー	FAP
	HTTPS クライアント	HTTPS サーバー	FAP
SSL/TLS	SSL プロキシー	SSL プロキシー	SSL/TLS
		SSL サーバーおよび FAP クライアント	FAP
		SSL サーバーおよび SSL クライアント	SSL/TLS
	HTTP クライアント	HTTP サーバー	FAP
	HTTPS クライアント	HTTPS サーバー	SSL/TLS
	HTTP クライアント	HTTP サーバーおよび SSL クライアント	FAP
	HTTPS クライアント	HTTPS サーバーおよび SSL クライアント	SSL/TLS

サポートされるチャネルの構成

IBM MQ のすべてのチャネル・タイプがサポートされますが、 構成は TCP/IP 接続に制限されます。 IBM MQ クライアントや キュー・マネージャーには、MQIPT は宛先キュー・マネージャーのように 見えます。 チャネル構成で宛先ホストとポート番号が必要な場合は、MQIPT の ホスト名とリスナー・ポート番号が指定されます。

クライアント/サーバー・チャネル
MQIPT は着信クライアント接続要求を listen し、HTTP トンネリング か SSL/TLS を使用するか、または標準の IBM MQ プロトコル・パケット として、その要求を転送します。 MQIPT が HTTP トンネリング または SSL/TLS を使用している場合、接続上で要求を 2 番目 の MQIPT に転送します。 HTTP トンネリングを使用していない場合は、 接続上で宛先キュー・マネージャーのように見える宛先に要求を転送します (これはさらに 先の MQIPT である場合があります)。 宛先キュー・マネージャーが クライアント接続を受け入れると、クライアントとサーバー間でパケットがリレーされます。

クラスター送信側/受信側チャネル
クラスター送信側チャネルから着信要求を受信すると、MQIPT は キュー・マネージャーが SOCKS 対応であると見なし、真の宛先アドレスが SOCKS ハンドシェーク・プロセス中に 取得されます。 要求は、クライアント接続チャネルの場合と同様に、次の MQIPT または 宛先キュー・マネージャーに転送されます。 これには、自動定義のクラスター送信側チャネルも含まれます。

送信側/受信側
送信側チャネルから着信要求を 受信すると、MQIPT は、クライアント接続チャネルの場合と同様に、要求を次の MQIPT または宛先キュー・マネージャーに転送します。 宛先キュー・マネージャーは 着信要求を検証し、適切な場合は受信側チャネルを開始します。 送信側と受信側チャネル間の すべての通信 (セキュリティー・フローなど) がリレーされます。

要求側/サーバー
この組み合わせは、前の構成と同様の方法で 処理されます。 接続要求の検証は、宛先キュー・マネージャーのサーバー・チャネルで実行されます。

要求側/送信側
2 つのキュー・マネージャーで相互に直接接続を 確立することが許可されていないが、両方とも MQIPT への接続は許可されており、 そこからの接続を受け入れることができる場合は、「コールバック」構成が有用です。

サーバー/要求側およびサーバー/受信側
これらは、 Sender/Receiver 構成を処理する場合と同じ方法で MQIPT によって処理されます。

チャネルの終了と障害の状態

MQIPT は IBM MQ チャネルのクローズ (正常または異常) を検出すると、そのチャネル・クローズを伝搬します。 MQIPT を使用して経路をクローズすると、その経路を通過するすべてのチャネルがクローズされます

MQIPT は、オプションのアイドル・タイムアウト機能を備えています。 MQIPT は、チャネルがタイムアウトを超過して一定期間アイドル状態になっていることを検出すると、該当する 2 つの接続の即時シャットダウンを実行します。

チャネルの両端の IBM MQ システムは、これらの異常シャットダウン状態をネットワーク障害、またはパートナーによるチャネルの終了と見なします。 その後、MQIPT を使用していない場合と同じように、チャネルを再始動してリカバリーできます (障害がプロトコル未確定期間中に発生した場合)。

メッセージの安全性

IBM MQ 分散キュー管理によって、 メッセージが確実に適切な方法で送達されます。 これは、チャネルの両側に MQIPT が存在する場合にも当てはまります。 MQIPT は、メッセージ・データを保管することや、適切なメッセージの送達を確実にする同期点プロシージャーに関与することはしません。

高速の非持続 IBM MQ メッセージを使用している場合、IBM MQ メッセージの転送中に MQIPT 経路に障害が発生したり、経路が再始動したりすると、メッセージが失われる可能性があります。 経路を再始動する前に、 MQIPT 経路を使用するすべての IBM MQ チャネルが非アクティブであることを確認してください。

IBM MQ でのメッセージの安全性について詳しくは、メッセージの安全性を参照してください。

複数インスタンス・キュー・マネージャーおよび高可用性

MQIPT は、高可用性環境で複数インスタンス・キュー・マネージャーとともに使用することができます。

MQIPT には持続状態がないため、MQIPT を別のシステムにフェイルオーバーすることには利点がありません。 代わりに、さまざまなシステム上で実行される同一の MQIPT 構成ファイルを持つ複数インスタンスの mqipt.conf を使用します。 MQIPT の各インスタンスのアベイラビリティーをモニターし、必要な場合は再始動します (同じシステム上で)。 これにより、接続の経路指定に使用することができる同一の MQIPT インスタンスのセットが提供されます。 IBM MQ が接続を MQIPT に経路指定できるようにし、MQIPT がこれらの接続を宛先キュー・マネージャーに転送できるようにする必要があります。

アウトバウンド IBM MQ チャネルは、以下の例に示すさまざまな方法で、使用可能な MQIPT インスタンスにダイレクトすることができます。

- WebSphere Edge Components 製品の IBM Network Dispatcher などのロード・バランサーまたは高可用性ルーターを使用します。
- コンマ区切りリストを使用して、IBM MQ チャネル定義で複数の接続名を指定します。 これにより IBM MQ は、使用可能な MQIPT インスタンスが見つかるまで、各 MQIPT アドレスへの接続を順番に試行します。

また、MQIPT からの接続を宛先キュー・マネージャーにダイレクトする必要があります。 高可用性構成で IP アドレスが宛先キュー・マネージャーを使用してフェイルオーバーするようになっている場合は、特別な MQIPT 構成は必要ありません。**Destination** 経路プロパティーに宛先 IP アドレスを指定し、フェイルオーバー操作でキュー・マネージャーを使用して IP アドレスを移動できるようにします。

ただし、フェイルオーバー後にキュー・マネージャーの IP アドレスが変更された場合は、MQIPT が接続を正しい宛先に転送できるように調整する必要があります。 これは、以下のいずれかの方法で行うことができます。

- どの IP アドレスとポート番号がアクセス可能かをチェックするルーティング出口を作成し、各接続の経路の宛先を指定変更します。 MQIPT には、いくつかのサンプルのルーティング出口が用意されています。これらをこの目的に合うように調整することができます。
- 高可用性ロード・バランサーを使用して、接続をリダイレクトします。
- キュー・マネージャーが実行されている可能性がある各 IP アドレスとポートに 1 つずつ、複数の MQIPT 経路を定義します。 次に、例えば、アウトバウンド・チャネルの接続名のコンマ区切りリストに経路 IP アドレスとポート番号をすべてリストすることによって、IBM MQ 接続をさまざまな MQIPT 経路にダイレクトします。

また、ネットワーク・パス上にあるすべてのエンドツーエンド・コンポーネントを調整することも重要です。

1. 使用不可のシステムへの接続の試行は、再接続が試行が使用可能な最初の宛先に進めるように即座に失敗する必要があります。

   MQIPT SSL 経路については、使用不可の宛先の場合に接続が即座に失敗するように **SSLClientConnectTimeout** 経路プロパティーを調整します。 IBM MQ チューニング・パラメーターの詳細については、IBM MQ の資料を参照してください。 また、オペレーティング・システム向けの TCP/IP の調整の詳細については、ご使用のオペレーティング・システムの資料を参照してください。 すべての場合において、失敗した接続の試行は、迅速にネットワーク障害 (TCP リセット・パケットなど) を戻すか、または過度の遅延なしにタイムアウトになる必要があります。
2. 障害が発生したシステムへのアクティブな接続は、新規接続を確立できるように即座に切断される必要があります。

   また、接続でアクティブに MQIPT が使用されているときにフェイルオーバーすることの影響を考慮する必要があります。 フェイルオーバー中はネットワーク接続が切断される可能性があります。 クライアント・アプリケーションの場合は、IBM MQ 自動クライアント再接続機能を使用して、切断された接続を再確立することができます。 メッセージ・チャネルの場合は、チャネルが即座に再接続するように、短期再試行間隔を指定することができます。 自動クライアント再接続およびメッセージ・チャネルの再試行構成の詳細については、IBM MQ の資料を参照してください。

IBM MQ Console および REST API

IBM MQ Console および REST API を使用して、 IBM MQ を管理し、HTTP を使用してメッセージング操作を実行することができます。

- IBM MQ Console を使用して、Web ブラウザーから基本的な管理タスクを実行できます。 詳しくは、IBM MQ Console による管理を参照してください。
- administrative REST API を使用して、キュー・マネージャーやキューなどの IBM MQ オブジェクト、Managed File Transfer エージェントおよび転送を管理できます。 詳しくは、REST API による管理を参照してください。
- messaging REST API を使用して、単純な Point-to-Point メッセージングおよびパブリッシュ・メッセージングを実行できます。 詳しくは、 REST API を使用したメッセージングを参照してください。

インストールのオプション

IBM MQ Console および REST API は、mqweb という WebSphere Liberty サーバーで実行されます。 IBM MQ 9.3.5 以降、mqweb サーバーは、 IBM MQ インストール済み環境のオプション・コンポーネントとしてインストールすることも、スタンドアロンの IBM MQ Web Server インストール済み環境としてインストールすることもできます。

Linux V 9.4.0 スタンドアロン IBM MQ Web Server インストール済み環境

IBM MQ 9.4.0 以降、mqweb サーバーは IBM MQ Web Server のスタンドアロン・インストール済み環境で実行できます。 スタンドアロンの IBM MQ Web Server インストール済み環境では、IBM MQ インストール済み環境とは別のシステムに mqweb サーバーをインストールして実行することができます。 スタンドアロン IBM MQ Web Server をインストールすると、mqweb サーバーを実行するために選択するシステムとシステムの数に関して柔軟性が向上します。 必要に応じて、mqweb サーバーの複数のインスタンスを異なるマシン上で実行して、必要なスケーラビリティーと可用性を提供することができます。

IBM MQ ライセンスを購入した場合は、スタンドアロン IBM MQ Web Server に必要な数だけコピーをインストールできます。 IBM MQ Web Server インストール済み環境は、購入した IBM MQ ライセンスを消費するものとしてカウントされません。 IBM MQ のライセンス交付について詳しくは、IBM MQ のライセンス情報を参照してください。

スタンドアロン IBM MQ Web Server インストール済み環境では、以下の制約事項が適用されます。

- IBM MQ Console は、リモート・キュー・マネージャーの管理にのみ使用できます。
- messaging REST API は、リモート・キュー・マネージャーでのみ使用できます。
- administrative REST API は使用できません。

スタンドアロン IBM MQ Web Server は、 Linux プラットフォームでのみサポートされます。

スタンドアロン IBM MQ Web Server のインストールについて詳しくは、 スタンドアロン IBM MQ Web Server のインストールを参照してください。

IBM MQ インストール済み環境のオプション・コンポーネント

IBM MQ Console および REST API コンポーネントを IBM MQ インストールの一部としてインストールすることを選択できます。

IBM MQ Console および REST API のすべての機能は、mqweb サーバーが IBM MQ インストール済み環境で実行されている場合に使用可能です。

- IBM MQ Console を使用して、ローカルおよびリモートのキュー・マネージャーを管理できます。
- messaging REST API は、ローカルおよびリモートのキュー・マネージャーで使用できます。
- administrative REST API を使用して、ローカルおよびリモートのキュー・マネージャーを管理できます。

IBM MQ Console および REST API コンポーネントを使用するには、 IBM MQ インストールの一部として以下のコンポーネントをインストールします。

- AIX で、`mqm.web.rte` ファイルセットをインストールします。
- IBM i の場合は、WEB コンポーネントをインストールします。
- Linux で、`MQSeriesWeb` コンポーネントをインストールします。
- Windows で、`Web Administration` フィールドをインストールします。
- z/OS で、`IBM MQ for z/OS UNIX System Services Web Components` フィールドをインストールします。

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。

IBM 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
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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が、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証できません。 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にお願いします。

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、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。これらの名称はすべて架空のものであり、 名称や住所が類似する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 あることをほのめかしたり、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 MQ のサービスを使用できるように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 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 氏の米国およびその他の国における登録商標です。

この製品には、 Eclipse Project (https://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: