9.4

IBM MQ -Technische Übersicht

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen” auf Seite 317 gelesen werden.

Diese Ausgabe bezieht sich auf Version 9 Release 4 von IBM® MQ und alle nachfolgenden Releases und Modifikationen, bis dieser Hinweis in einer Neuausgabe geändert wird.

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

Inhaltsverzeichnis

IBM MQ - Technische Übersicht

Mit IBM MQ können Sie Verbindungen für Ihre Anwendungen herstellen und die Verteilung von Informationen in Ihrem Unternehmen verwalten.

IBM MQ ermöglicht Programmen, über ein Netz unähnlicher Komponenten (Prozessoren, Betriebssysteme, Subsysteme und Übertragungsprotokolle) mithilfe einer konsistenten Anwendungsprogrammierschnittstelle miteinander zu kommunizieren. Anwendungen, die mit dieser Schnittstelle entworfen und geschrieben wurden, werden als Message-Queuing-Anwendungen bezeichnet (Anwendungen zur Steuerung von Nachrichtenwarteschlangen).

In den folgenden Abschnitten erhalten Sie Informationen zur Steuerung von Nachrichtenwarteschlangen und anderen Features von IBM MQ.

Zugehörige Konzepte

Einführung in IBM MQ

Informationen zu Produktanforderungen und zum Support

Zugehörige Tasks

IBM MQ-Architektur planen

Zugehörige Verweise

„Hauptmerkmale und Vorteile des Message-Queuing” auf Seite 7

In diesem Abschnitt werden einige Merkmale und Vorteile des Message-Queuing hervorgehoben. Sie finden hier beispielsweise eine Beschreibung der Message-Queuing-Merkmale in Bezug auf Sicherheit und Datenintegrität.

Einführung in die Nachrichtenwarteschlangensteuerung

Mit den IBM MQ-Produkten können Programme über ein Netz ungleicher Komponenten (Prozessoren, Betriebssysteme, Subsysteme und Kommunikationsprotokolle) unter Verwendung einer konsistenten Anwendungsprogrammierschnittstelle miteinander kommunizieren.

Anwendungen, die über diese Schnittstelle entwickelt und geschrieben wurden, werden als *Message-Queuing*-Anwendungen bezeichnet, da sie *Messaging* und *Queuing* (Warteschlangensteuerung) verwenden:

- Messaging bedeutet, dass Programme miteinander kommunizieren, indem sie sich gegenseitig Daten in Nachrichten zusenden, anstatt sich direkt anzurufen.
- Eine Warteschlangensteuerung bedeutet, dass Nachrichten in Warteschlangen gespeichert werden. Programme können somit unabhängig voneinander mit unterschiedlicher Verarbeitungsgeschwindigkeit und zu unterschiedlichen Zeiten an verschiedenen Standorten ausgeführt werden, ohne dass eine logische Verbindung zwischen ihnen bestehen muss.

Message-Queuing ist in der Datenverarbeitung seit vielen Jahren im Einsatz. Heutzutage wird es im Allgemeinen in E-Mails verwendet. Ohne Queuing (Warteschlangensteuerung) muss beim Senden einer elektronischen Nachricht über weite Entfernungen jeder einzelne Knoten auf der Route zur Weiterleitung von Nachrichten verfügbar sein. Außerdem müssen die Empfänger angemeldet sein und wissen, dass Sie ihnen eine Nachricht senden möchten. In einem Warteschlangensystem werden die Nachrichten an Zwischenknoten gespeichert, bis das System zu ihrer Weiterleitung bereit ist. Am Zielort werden sie in einer elektronischen Mailbox gespeichert, bis der Empfänger bereit ist, sie zu lesen.

Trotz alledem werden viele komplexe Geschäftstransaktionen heutzutage ohne Queuing verarbeitet. In einem großen Netz kann das System möglicherweise viele Tausende Verbindungen sofort einsatzfähig halten. Tritt in einem Teil des Systems ein Problem auf, können viele Systemteile nicht mehr eingesetzt werden.

Sie können sich Message-Queuing als E-Mail für Programme vorstellen. In einer Message-Queuing-Umgebung führt jedes Programm einer Anwendungssuite eine klar strukturierte eigenständige Funktion als Antwort auf eine bestimmte Anforderung aus. Für die Kommunikation mit einem anderen Programm

muss ein Programm eine Nachricht in eine vordefinierte Warteschlange einreihen. Diese Nachricht wird von dem anderen Programm aus der Warteschlange abgerufen und die Anforderungen und Angaben aus der Nachricht werden verarbeitet. Message-Queuing stellt somit eine Art der Kommunikation zwischen Programmen dar.

Queuing bezeichnet den Mechanismus, mit dem Nachrichten zurückbehalten werden, bis sie von einer Anwendung verarbeitet werden können. Queuing bietet folgende Möglichkeiten:

- Kommunikation zwischen Programmen (die sogar in verschiedenen Umgebungen ausgeführt werden können), ohne dass Kommunikationscode geschrieben werden muss.
- Auswahl der Reihenfolge, in welcher ein Programm Nachrichten verarbeitet.
- Ausgleich der Arbeitslast in einem System durch Festlegung mehrerer Programme, die für eine Warteschlange zuständig sind, wenn die Anzahl der Nachrichten einen Grenzwert überschreitet.
- Erhöhte Verfügbarkeit der Anwendungen, durch Festlegung eines alternativ für die Warteschlangen zuständigen Systems, wenn das primäre System nicht verfügbar ist.

Was ist unter einer Nachrichtenwarteschlange zu verstehen?

Bei einer Nachrichtenwarteschlange, die auch einfach als Warteschlange bezeichnet wird, handelt es sich um ein benanntes Ziel, an das Nachrichten gesendet werden können. Nachrichten sammeln sich in Warteschlangen an, bis sie von Programmen abgerufen werden, die für diese Warteschlangen zuständig sind.

Warteschlangen befinden sich auf einem Warteschlangenmanager, von dem sie auch verwaltet werden (siehe Abschnitt „Terminologie zum Message-Queuing" auf Seite 9). Die physische Natur einer Warteschlange hängt von dem Betriebssystem ab, unter dem der Warteschlangenmanager ausgeführt wird. Warteschlangen können flüchtige Pufferbereiche im Hauptspeicher eines Computers oder aber Datenmengen in einer permanenten Speichereinheit (wie z. B. einer Festplatte) sein. Die physische Verwaltung der Warteschlangen ist Aufgabe der Warteschlangenmanager, für die beteiligten Anwendungsprogramme ist sie nicht offensichtlich.

Programme greifen auf Warteschlangen lediglich über die externen Services des Warteschlangenmanagers zu. Sie können eine Warteschlange öffnen, Nachrichten darin einreihen und daraus abrufen und die Warteschlange schließen. Außerdem können sie die Warteschlangenattribute festlegen und abfragen.

Verschiedene Arten des Message-Queuing

Punkt-zu-Punkt

Eine Nachricht wird in die Warteschlange eingereiht und dann von einer Anwendung empfangen.

Beim Punkt-zu-Punkt-Messaging muss eine sendende Anwendung Informationen zu einer anderen empfangenden Anwendung erhalten, damit sie eine Nachricht an diese Anwendung senden kann. Die sendende Anwendung muss beispielsweise den Namen der Warteschlange kennen, an die die Informationen gesendet werden sollen, und möglicherweise muss auch der Name eines Warteschlangenmanagers angegeben werden.

Publish/Subscribe

Jeder interessierten Anwendung wird eine Kopie jeder Nachricht bereitgestellt, die von einer publizierenden Anwendung veröffentlicht wird. Dabei ist die Anzahl der interessierten Anwendungen - keine, eine oder mehrere - unbedeutend. Beim Publish/Subscribe wird eine interessierte Anwendung als Subskribent bezeichnet, und die Nachrichten werden in eine durch eine Subskription festgelegte Warteschlange eingereiht.

Mithilfe des Publish/Subscribe-Messaging kann der Informationsanbieter von den Nutzern der Informationen abgekoppelt werden. Die Sender- und Empfängeranwendungen müssen zur Übertragung der Informationen nichts übereinander wissen. Weitere Informationen finden Sie unter „Publish/Subscribe-Messaging" auf Seite 67.

Vorteile des Message-Queuing für Anwendungsentwickler

In IBM MQ können Anwendungsprogramme *Message-Queuing* einsetzen und somit an einer nachrichtengesteuerten Verarbeitung teilnehmen. Unter Verwendung der entsprechenden Message-Queuing-Softwareprodukte können Anwendungsprogramme über verschiedene Plattformen hinweg miteinander kommunizieren. Beispielsweise können z/OS-Anwendungen über IBM MQ for z/OS kommunizieren. Die Anwendungen sind von den Mechanismen der zugrunde liegenden Kommunikation abgeschirmt. Einige weitere Vorteile des Message-Queuing:

- Es ist möglich, für die Anwendungsentwicklung kleine Programme einzusetzen, die von vielen Anwendungen gemeinsam genutzt werden können.
- Durch die Wiederverwendung dieser logischen Bausteine können schnell neue Anwendungen erstellt werden.
- Anwendungen, die für die Verwendung von Message-Queuing-Verfahren geschrieben wurden, sind von Änderungen in der Funktionsweise der Warteschlangenmanager nicht betroffen.
- Es werden keine Kommunikationsprotokolle benötigt. Alle Aspekte der Kommunikation werden vom Warteschlangenmanager übernommen.
- Programme, die Nachrichten empfangen, müssen nicht aktiv sein, wenn ihnen Nachrichten zugesendet werden. Die Nachrichten verbleiben in Warteschlangen.

Geringere Kosten für die Anwendungen, da eine schnellere Entwicklung mit weniger Entwicklern möglich ist und weniger Programmierkenntnisse erforderlich sind als für Anwendungen, die Message-Queuing nicht einsetzen.

Überall, wo die Anwendungen ausgeführt werden, implementiert IBM MQ eine einheitliche Anwendungsprogrammierschnittstelle, das sogenannte *Message Queue Interface* (MQI). Auf diese Weise können Anwendungsprogramme leichter auf andere Plattformen portiert werden.

Details zum Message Queue Interface finden Sie im Abschnitt Übersicht über das Message Queue Interface.

Hauptmerkmale und Vorteile des Message-Queuing

In diesem Abschnitt werden einige Merkmale und Vorteile des Message-Queuing hervorgehoben. Sie finden hier beispielsweise eine Beschreibung der Message-Queuing-Merkmale in Bezug auf Sicherheit und Datenintegrität.

Anwendungen, die Message-Queuing-Verfahren einsetzen, weisen folgende Hauptmerkmale auf:

- „Keine Direktverbindungen zwischen Programmen” auf Seite 7
- „Zeitunabhängige Kommunikation” auf Seite 8
- „Kleine Programme” auf Seite 8
- „Nachrichtengesteuerte Verarbeitung” auf Seite 9
- „Ereignisgesteuerte Verarbeitung” auf Seite 9
- „Nachrichtenpriorität” auf Seite 9
- „Sicherheit” auf Seite 9
- „Datenintegrität” auf Seite 9
- „Unterstützung für die Wiederherstellung” auf Seite 9

Anmerkung: Für IBM MQ-Clients und -Server gilt Folgendes: Es ist keine Änderung einer Serveranwendung erforderlich, damit weitere IBM MQ MQI clients auf neuen Plattformen unterstützt werden können. Entsprechend kann der IBM MQ MQI client ohne Änderung mit weiteren Servertypen eingesetzt werden.

Keine Direktverbindungen zwischen Programmen

Message-Queuing stellt ein Verfahren für die indirekte Kommunikation zwischen Programmen dar. Dieses Verfahren kann in jeder Anwendung eingesetzt werden, in der Programme miteinander kommunizieren. Die Kommunikation erfolgt, indem ein Programm Nachrichten in eine (einem Warteschlangenmanager

zugehörige) Warteschlange einreiht und ein anderes Programm die Nachrichten aus der Warteschlange abruft.

Programme können Nachrichten abrufen, die von anderen Programmen in eine Warteschlange eingereiht wurden. Die anderen Programme können mit demselben Warteschlangenmanager wie das empfangende Programm oder aber mit einem anderen Warteschlangenmanager verbunden sein. Dieser andere Warteschlangenmanager kann sich auf einem anderen System, in einem anderen Computersystem oder sogar in einem anderen Unternehmen befinden.

Zwischen Programmen, die Nachrichtenwarteschlangen für die Kommunikation miteinander einsetzen, bestehen keine physischen Verbindungen. Ein Programm sendet Nachrichten an eine Warteschlange eines Warteschlangenmanagers und ein anderes Programm ruft Nachrichten aus der Warteschlange ab (siehe Abschnitt Abbildung 1 auf Seite 8).

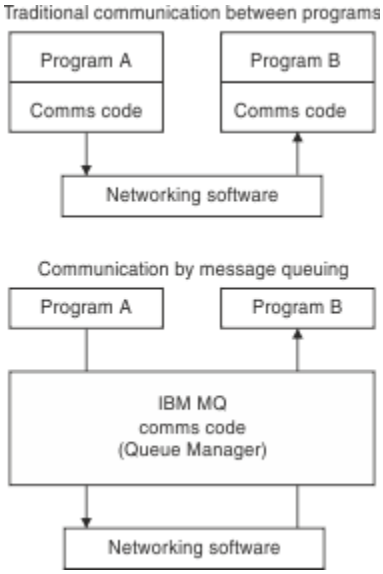


Abbildung 1. Message-Queuing im Vergleich zur konventionellen Kommunikation

Wie bei E-Mail durchqueren die einzelnen zu einer Transaktion gehörigen Nachrichten ein Netz auf Basis des Store-and-forward-Verfahrens. Schlägt eine Verbindung zwischen Knoten fehl, bleibt die Nachricht erhalten, bis die Verbindung wiederhergestellt ist oder der Bediener bzw. das Programm die Nachricht umleitet.

Der Mechanismus, mit dem Nachrichten von einer Warteschlange zur nächsten gelangen, bleibt vor den Programmen verborgen. Dies vereinfacht die Programme.

Zeitunabhängige Kommunikation

Programme, die andere Programme zur Ausführung von Arbeiten auffordern, müssen nicht auf die Antwort zu der Anforderung warten. Sie können weitere Arbeiten erledigen und die Antwort bei Erhalt oder zu einem späteren Zeitpunkt verarbeiten. Beim Schreiben einer Messaging-Anwendung ist es irrelevant, wann ein Programm eine Nachricht sendet bzw. wann das Ziel die Nachricht empfangen kann. Die Nachricht geht nicht verloren, sie wird vom Warteschlangenmanager beibehalten, bis sie vom Ziel verarbeitet werden kann. Die Nachricht bleibt in der Warteschlange, bis sie von einem Programm daraus entfernt wird. Die sendende und die empfangende Anwendung sind also voneinander abgekoppelt. Der Sender kann seine Verarbeitung fortsetzen und muss nicht warten, bis der Empfänger den Erhalt der Nachricht bestätigt. Die Zielanwendung muss nicht einmal aktiv sein, wenn die Nachricht gesendet wird. Sie kann die Nachricht nach dem Start abrufen.

Kleine Programme

Bei Einsatz des Message-Queuing können Sie die Vorteile kleiner unabhängiger Programme nutzen. Anstelle eines einzelnen umfangreichen Programms, das nacheinander alle Teile eines Jobs ausführt,

kann der Job auf mehrere kleinere unabhängige Programme verteilt werden. Das aufrufende Programm sendet Nachrichten an jedes einzelne dieser separaten Programme und fordert diese zur Ausführung ihrer jeweiligen Funktion auf. Sobald die einzelnen Programme mit der Verarbeitung fertig sind, werden die Ergebnisse in Form einer oder mehrerer Nachrichten zurückgesendet.

Nachrichtengesteuerte Verarbeitung

Wenn Nachrichten in einer Warteschlange ankommen, können sie mithilfe der *Auslösefunktion* automatisch eine Anwendung starten. Gegebenenfalls können die Anwendungen nach erfolgter Verarbeitung der Nachricht(en) gestoppt werden.

Ereignisgesteuerte Verarbeitung

Programme können entsprechend dem Warteschlangenstatus gesteuert werden. So können Sie beispielsweise festlegen, dass bei Eintreffen einer Nachricht in einer Warteschlange ein Programm gestartet werden soll, oder Sie können angeben, dass das Programm erst gestartet werden soll, wenn sich in der Warteschlange beispielsweise 10 Nachrichten über einer gewissen Priorität oder aber mit einer beliebigen Priorität befinden.

Nachrichtenpriorität

Wenn ein Programm eine Nachricht in eine Warteschlange einreiht, kann es der Nachricht eine Priorität zuordnen. Diese legt fest, an welcher Stelle in der Warteschlange die neue Nachricht hinzugefügt wird.

Programme können entweder die Nachrichten der Reihenfolge nach aus einer Warteschlange abrufen oder aber eine bestimmte Nachricht abrufen. (So könnte es z. B. sein, dass ein Programm eine bestimmte Nachricht abrufen möchte, wenn es auf die Antwort auf eine zuvor gesendete Anforderung wartet.)

Sicherheit

Für bestimmte Vorgänge stehen bestimmte Sicherheitsfunktionen zur Verfügung. Zum Beispiel die Authentifizierung einer Anwendung beim Zugriff auf einen Warteschlangenmanager, Berechtigungsprüfungen, wenn eine Anwendung Ressourcen wie eine Warteschlange auf einem Warteschlangenmanager verwendet, oder die Verschlüsselung der Nachrichtendaten während der Übertragung im Netz und der Aufbewahrung in Warteschlangen. Weitere Informationen zur Sicherheit finden Sie im Abschnitt Übersicht über die Sicherheit.

Datenintegrität

Datenintegrität wird über Arbeitseinheiten bereitgestellt. Die Synchronisation des Anfangs und Endes von Arbeitseinheiten wird bei jedem Aufruf MQGET oder MQPUT als Option vollständig unterstützt, sodass die Ergebnisse der Arbeitseinheit festgeschrieben bzw. rückgängig gemacht werden können. Die Synchronisationspunktunterstützung funktioniert entweder IBM MQ-intern oder -extern, je nachdem, welche Art der Synchronisationspunktkoordination für die Anwendung ausgewählt wurde.

Unterstützung für die Wiederherstellung

Um eine Wiederherstellung zu ermöglichen, werden alle permanenten IBM MQ-Updates protokolliert. Sollte eine Wiederherstellung erforderlich sein, werden alle persistenten Nachrichten zurückgeschrieben, für alle unvollständigen Transaktionen erfolgt ein Rollback und alle Synchronisationspunktfestschreibungen und -Backouts werden nach dem für den zuständigen Synchronisationspunktmanager normalen Verfahren bearbeitet. Weitere Informationen zu persistenten Nachrichten finden Sie im Abschnitt Nachrichtenpersistenz.

Terminologie zum Message-Queuing

In diesem Abschnitt werden einige Begriffe im Zusammenhang mit Message-Queuing erläutert.

Dazu gehören:

- Kanäle
- Cluster
- IBM MQ MQI client
- z/OS Gruppeninterne Steuerung von Warteschlangen
- Nachricht
- Nachrichtenkanalagent
- Nachrichtendeskriptor
- Punkt-zu-Punkt
- Publish/Subscribe
- Queue
- Queue Manager
- z/OS Gruppe mit gemeinsamer Warteschlange
- z/OS Gemeinsam genutzte Warteschlange
- Subskription
- Thema

Kanäle

Kanäle werden für die Übertragung von Nachrichten zwischen Warteschlangenmanagern verwendet und schirmen Anwendungen von den verwendeten Kommunikationsprotokollen ab. Die Warteschlangenmanager können sich dabei auf demselben oder einem anderen System auf derselben Plattform oder auf verschiedenen Plattformen befinden. Die gesendeten Nachrichten können aus vielen Quellen stammen:

- Vom Benutzer geschriebene Anwendungsprogramme, die Daten von einem Knoten an einen anderen übertragen
- Vom Benutzer geschriebene Verwaltungsanwendungen, die PCF-Befehle oder die MQAI verwenden
- Die IBM MQ Explorer.
- Warteschlangenmanager, die Instrumentierungsereignisnachrichten an einen anderen Warteschlangenmanager senden
- Warteschlangenmanager, die Fernverwaltungsbefehle an einen anderen Warteschlangenmanager senden (z. B. mit MQSC-Befehlen oder über die administrative REST API)

Weitere Informationen zu Kanälen finden Sie unter „Kanal- definitionen” auf Seite 34.

Cluster

Ein *Cluster* ist ein Netz von Warteschlangenmanagern, die logisch in gewisser Weise zusammengehören.

In einem IBM MQ-Netz mit verteilter Steuerung von Warteschlangen ohne Clustering ist jeder Warteschlangenmanager unabhängig. Wenn ein Warteschlangenmanager Nachrichten an einen anderen Warteschlangenmanager senden muss, müssen eine Übertragungswarteschlange sowie ein Kanal zu dem fernen Warteschlangenmanager definiert sein.

Für die Verwendung von Clustern gibt es zwei Gründe: geringerer Systemverwaltungsaufwand sowie verbesserte Verfügbarkeit und besserer Lastausgleich.

Schon bei Einrichtung eines sehr kleinen Clusters profitieren Sie von einer vereinfachten Systemverwaltung. Zu einem Cluster gehörige Warteschlangenmanager benötigen weniger Definitionen, wodurch sich das Risiko von Fehlern bei den Definitionen verringert.

Weitere Informationen zu Clustering finden Sie im Abschnitt Cluster.

IBM MQ MQI client

IBM MQ MQI-*Clients* sind unabhängig voneinander installierbare Komponenten von IBM MQ. Auf einem MQI-Client können IBM MQ-Anwendungen über ein Kommunikationsprotokoll ausgeführt werden, es ist eine Interaktion mit einem oder mehreren MQI-Servern (MQI = Message Queue Interface) auf anderen Plattformen möglich und es kann eine Verbindung zu den zugehörigen Warteschlangenmanagern hergestellt werden.

Ausführliche Informationen zur Installation und Verwendung von IBM MQ MQI client-Komponenten finden Sie in den folgenden Abschnitten:

- 

IBM MQ-Client unter AIX installieren
- 

IBM MQ-Client unter Linux® installieren
- 

IBM MQ-Client unter Windows installieren
- 

IBM MQ-Client unter IBM i installieren

und Verbindungen zwischen Server und Client konfigurieren.

Einreihung in Warteschlange innerhalb von Gruppen

Warteschlangenmanager in einer Gruppe mit gemeinsamer Warteschlange können über normale Kanäle miteinander kommunizieren, es kann aber auch das Verfahren der *gruppeninternen Warteschlangensteuerung* mit schneller Nachrichtenübertragung ohne Definition von Kanälen eingesetzt werden. Dies gilt nur für IBM MQ for z/OS.

Weitere Informationen zur gruppeninternen Warteschlangensteuerung finden Sie unter „Intra-group queuing” auf Seite 224.

Nachricht

Beim Message-Queuing bezeichnet der Begriff 'Nachricht' eine Datensammlung, die von einem Programm gesendet wurde und an ein anderes Programm gerichtet ist. Siehe IBM MQ-Nachrichten.

Informationen zu Nachrichtentypen finden Sie im Abschnitt Nachrichtentypen.

Nachrichtenkanalagent

An jedem Ende eines Kanals befindet sich ein Nachrichtenkanalagent. Ein Kanal besteht somit aus einem Paar aus Nachrichtenkanalagenten, einem sendenden und einem empfangenden, über die Nachrichten von einem zum nächsten Warteschlangenmanager übertragen werden.

Informationen zur Verwendung von Nachrichtenkanalagenten finden Sie unter Einführung in das verteilte Warteschlangenmanagement.

Nachrichtendeskriptor

Eine IBM MQ-Nachricht besteht aus Steuerinformationen und Anwendungsdaten.

Die Steuerinformationen sind in einer Nachrichtendeskriptorstruktur (MQMD) definiert und umfassen u. a. folgende Angaben:

- Nachrichtentyp
- ID für die Nachricht
- Zustellungspriorität der Nachricht

Struktur und Inhalt der Anwendungsdaten werden durch die beteiligten Programme und nicht durch IBM MQ bestimmt.

Weitere Informationen finden Sie unter MQMD.

Punkt-zu-Punkt-Messaging

Beim Punkt-zu-Punkt-Messaging wird jede Nachricht von einer generierenden Anwendung an eine konsumierende Anwendung übertragen. Zur Übertragung stellt die generierende Anwendung die Nachricht in eine Warteschlange, aus der die konsumierende Anwendung die Nachricht abruft.

Publish/Subscribe-Messaging

Beim Publish/Subscribe-Messaging wird jeder interessierten Anwendung eine Kopie jeder Nachricht bereitgestellt, die von einer publizierenden Anwendung veröffentlicht wird. Dabei ist die Anzahl der interessierten Anwendungen - keine, eine oder mehrere - unbedeutend. Beim Publish/Subscribe wird eine interessierte Anwendung als Subskribent bezeichnet, und die Nachrichten werden in eine durch eine Subskription festgelegte Warteschlange eingereiht.

Weitere Informationen finden Sie unter „Publish/Subscribe-Messaging” auf Seite 67.

Warteschlange

Ein benanntes Ziel, an das Nachrichten gesendet werden können. Nachrichten sammeln sich in Warteschlangen an, bis sie von Programmen abgerufen werden, die für diese Warteschlangen zuständig sind.

Weitere Informationen finden Sie unter „Warteschlangen” auf Seite 20.

Warteschlangenmanager

Ein *Warteschlangenmanager* ist ein Systemprogramm, welches Warteschlangensteuerungsservices für Anwendungen bereitstellt.

Der Warteschlangenmanager stellt eine Anwendungsprogrammierschnittstelle zur Verfügung, über welche Programme Nachrichten in Warteschlangen einreihen und daraus abrufen können. Darüber hinaus bietet der Warteschlangenmanager weitere Funktionen, mit denen Administratoren neue Warteschlangen erstellen, die Eigenschaften bestehender Warteschlangen ändern und den Betrieb des Warteschlangenmanagers steuern können.

Die Message-Queueing-Services von IBM MQ sind auf einem System nur verfügbar, wenn ein Warteschlangenmanager aktiv ist. Auf einem einzelnen System können mehrere Warteschlangenmanager aktiv sein (beispielsweise um ein Testsystem von einem *Live*system zu trennen). Jeder Warteschlangenmanager wird gegenüber einer Anwendung über eine *Verbindungskennung* (*Hconn*) identifiziert.

Die Services des Warteschlangenmanagers können von vielen verschiedenen Anwendungen, die komplett unabhängig voneinander sein können, gleichzeitig verwendet werden. Damit ein Programm die Services eines Warteschlangenmanagers verwenden kann, muss es eine Verbindung zu dem betreffenden Warteschlangenmanager herstellen.

Anwendungen können Nachrichten nur dann an Anwendungen senden, die mit anderen Warteschlangenmanagern verbunden sind, wenn die betreffenden Warteschlangenmanager in der Lage sind, miteinander zu kommunizieren. Für die sichere Nachrichtenübermittlung zwischen solchen Anwendungen implementiert IBM MQ ein *Store-and-forward*-Protokoll.

Weitere Informationen finden Sie unter „Warteschlangenmanager” auf Seite 30.

Gruppe mit gemeinsamer Warteschlange

Die Warteschlangenmanager, die auf dieselbe Gruppe gemeinsam genutzter Warteschlangen zugreifen können, bilden eine sogenannte *Gruppe mit gemeinsamer Warteschlange*. Sie kommunizieren über eine Coupling-Facility (CF) miteinander, auf der die gemeinsam genutzten Warteschlangen gespeichert werden. Dies gilt nur für IBM MQ for z/OS.

Weitere Informationen finden Sie unter „Shared queues and queue sharing groups” auf Seite 180.

Gemeinsam genutzte Warteschlange

z/OS

Eine *gemeinsam genutzte Warteschlange* ist eine bestimmte Art von lokaler Warteschlange, auf deren Nachrichten ein oder mehrere Warteschlangenmanager in einem Sysplex zugreifen können. Dies ist nicht zu verwechseln mit einer Warteschlange, die von mehreren Anwendungen gemeinsam genutzt wird, die denselben Warteschlangenmanager verwenden. Dies gilt nur für IBM MQ for z/OS.

Abonnement

Eine Publish/Subscribe-Anwendung kann ihr Interesse an Nachrichten zu bestimmten Themen registrieren lassen. Mit der Registrierung wird die Anwendung zu einem Subskribenten. Der Begriff 'Subskription' definiert dabei, auf welche Weise passende Nachrichten zur Verarbeitung in Warteschlangen eingereiht werden.

Eine Subskription enthält Informationen zur Identität des Subskribenten und der Zielwarteschlange, auf der Veröffentlichungen eingereiht werden sollen. Sie enthält außerdem Informationen darüber, wie eine Veröffentlichung in der Zielwarteschlange eingereiht wird.

Weitere Informationen finden Sie unter „Subskribenten und Subskriptionen” auf Seite 71.

Thema

Ein Thema ist eine Zeichenfolge, die den Gegenstand der Informationen beschreibt, die in einer Publish/Subscribe-Nachricht veröffentlicht werden.

Themen sind wichtig für die erfolgreiche Nachrichtenübermittlung in einem Publish/Subscribe-System. Anstatt eine bestimmte Zieladresse in die einzelnen Nachrichten einzufügen, ordnet ein Publisher jeder Nachricht ein Thema zu. Der Warteschlangenmanager gleicht das Thema mit einer Liste von Abonnenten ab, die das Thema abonniert haben, und stellt jedem dieser Abonnenten die Nachricht zu.

Weitere Informationen finden Sie unter „Themen” auf Seite 74.

Nachrichten und Warteschlangen

Nachrichten und Warteschlangen sind die Basiskomponenten eines Message-Queuing-Systems.

Was ist eine Nachricht?

Eine *Nachricht* ist eine Bytefolge, die Informationen für die Anwendungen bereitstellt, die diese Nachricht verwendet. Mithilfe von Nachrichten werden Informationen von einem Anwendungsprogramm an ein anderes oder zwischen den verschiedenen Teilen ein und derselben Anwendung übertragen. Die Anwendungen können dabei auf derselben oder auf verschiedenen Plattformen aktiv sein.

Eine IBM MQ-Nachricht setzt sich aus folgenden Komponenten zusammen:

- *Den Anwendungsdaten.*Inhalt und Struktur der Anwendungsdaten werden vom Anwendungsprogramm vorgegeben, das diese Daten nutzt.
- *Einem Nachrichtendeskriptor.*Der Nachrichtendeskriptor dient zur Kennzeichnung der Nachricht und enthält zusätzliche Steuerinformationen, z. B. den Nachrichtentyp und die Priorität, die der Nachricht von der sendenden Anwendung zugewiesen wurde.

  Das Format des Nachrichtendeskriptors wird von IBM MQ definiert. Eine umfassende Beschreibung des Nachrichtendeskriptors finden Sie im Abschnitt MQMD – Nachrichtendeskriptor.
- *Nachrichteneigenschaften.* Die Metadaten zur Nachricht. Der Inhalt der Nachrichteneigenschaften wird durch die Anwendungsprogramme vorgegeben, die diese Eigenschaften nutzen. Weitere Informationen finden Sie im Abschnitt Nachrichteneigenschaften.

Nachrichtenlänge

Die maximale Nachrichtenlänge ist standardmäßig 4 MB; diesen Wert können Sie auf maximal 100 MB (1 MB = 1,048,576 Byte) erhöhen. Tatsächlich kann die Nachrichtenlänge durch folgende Faktoren begrenzt werden:

- Die für die Empfangswarteschlange definierte maximale Nachrichtenlänge
- Die für den Warteschlangenmanager definierte maximale Nachrichtenlänge
- Die von der Warteschlange definierte maximale Nachrichtenlänge
- Die von der sendenden oder empfangenden Anwendung definierte maximale Nachrichtenlänge
- Dem für die Nachricht verfügbaren Speicherplatz

In manchen Fällen sind mehrere Nachrichten erforderlich, um alle notwendigen Informationen an eine Anwendung zu übertragen.

Wie werden Nachrichten von Anwendungen gesendet und empfangen?

Anwendungsprogramme senden und empfangen Nachrichten über **MQI-Aufrufe**.

Um beispielsweise eine Nachricht in eine Warteschlange einzureihen, sind folgende Schritte erforderlich:

1. Die Anwendung öffnet die betreffende Warteschlange, indem sie den MQ-Aufruf `MQOPEN` ausgibt.
2. Anschließend gibt die Anwendung den MQI-Aufruf `MQPUT` aus, um die Nachricht in die Warteschlange einzureihen.

Diese Nachricht kann von einer anderen Anwendung mit dem MQI-Aufruf `MQGET` aus der Warteschlange abgerufen werden.

Weitere Informationen zu MQI-Aufrufen finden Sie im Abschnitt MQI-Aufrufe.

Was ist eine Warteschlange?

Eine *Warteschlange* ist eine Datenstruktur, in der Nachrichten gespeichert werden.

Jede Warteschlange ist einem *Warteschlangenmanager* zugeordnet, der damit Eigner der Warteschlange ist. Der Warteschlangenmanager ist zuständig für die Verwaltung aller ihm zugeordneten Warteschlangen; er muss außerdem alle Nachrichten, die er empfängt, in den entsprechenden Warteschlangen speichern. Die Nachrichten können entweder von den Anwendungsprogrammen oder vom Warteschlangenmanager (als Teil des normalen Warteschlangenmanagerbetriebs) in die Warteschlangen eingereiht werden.

Vordefinierte und dynamische Warteschlangen

Warteschlangen werden anhand der Art und Weise ihrer Erstellung unterschieden:

- **Vordefinierte Warteschlangen** werden vom Administrator mit den entsprechenden MQSC- oder PCF-Befehlen erstellt.Vordefinierte Warteschlangen sind permanent; sie sind unabhängig von den Anwendungen vorhanden, von denen sie verwendet werden, und sind auch nach einem Neustart von IBM MQ noch vorhanden.
- **Dynamische Warteschlangen** werden erstellt, wenn eine Anwendung eine `MQOPEN`-Anforderung unter Angabe des Namens einer *Modellwarteschlange* ausgibt. Die Warteschlange wird auf Basis einer *Definition für eine Warteschlangenschablone*, einer sogenannten Modellwarteschlange, erstellt.Modellwarteschlangen können mit dem WebSphere MQ-Scriptbefehl DEFINE QMODEL erstellt werden. Die Attribute einer Modellwarteschlange (beispielsweise die Anzahl der Nachrichten, die maximal in der Warteschlange gespeichert werden können) werden von allen dynamischen Warteschlangen übernommen, die auf Basis dieser Modellwarteschlange erstellt werden.

  Modellwarteschlangen haben Attribute, die angeben, ob die dynamische Warteschlange permanent oder temporär sein soll. Permanente Warteschlangen sind auch nach dem Neustart einer Anwendung oder des Warteschlangenmanagers noch vorhanden, temporäre Warteschlangen hingegen nicht.

Nachrichten aus Warteschlangen abrufen

Anwendungen mit der entsprechenden Berechtigung können Nachrichten anhand der folgenden Abrufalgorithmen aus einer Warteschlange abrufen:

- FIFI (First In/First Out).
- Nachrichtenpriorität (ist im Nachrichtendeskriptor festgelegt). Nachrichten mit derselben Priorität werden nach dem FIFI-Prinzip (First In/First Out) abgerufen.
- Programmanforderung für eine bestimmte Nachricht.

Der Algorithmus ist in der MQGET-Anforderung der Anwendung festgelegt.

IBM MQ-Objekte

Die Eigenschaften von IBM MQ-Objekten werden durch Warteschlangenmanager festgelegt. Die Werte dieser Eigenschaften bestimmen, wie diese Objekte von IBM MQ verarbeitet werden. Objekte werden mit den von IBM MQ bereitgestellten Befehlen und Schnittstellen erstellt und verwaltet. In den Anwendungen werden Objekte über die MQI (Message Queue Interface) gesteuert. Programme verweisen auf Objekte anhand eines IBM MQ *Objektdeskriptors* (MQOD).

Objektverwaltung

Die Objektverwaltung beinhaltet folgende Tasks:

- Warteschlangenmanager starten und stoppen
- Objekte, insbesondere Warteschlangen, für Anwendungen erstellen
- Objektattribute anzeigen und ändern
- Objekte löschen
- Die Arbeit mit Kanälen, um Kommunikationspfade zu Warteschlangenmanagern auf anderen (fernen) Systemen zu erstellen.
- *Warteschlangenmanager-Cluster* erstellen, um die Verwaltung zu vereinfachen und die Arbeitslast gleichmäßig zu verteilen.

Mit Ausnahme dynamischer Warteschlangen müssen Objekte im Warteschlangenmanager definiert sein, damit sie verwendet werden können.

Wenn Sie eine Objektverwaltungsoperation mit einem IBM MQ-Befehl ausführen, prüft der Warteschlangenmanager, ob Sie über die entsprechende Berechtigung zur Ausführung dieses Vorgangs verfügen. Ebenso überprüft der Warteschlangenmanager auch bei Anwendungen, die versuchen, ein Objekt mit einem MQOPEN-Aufruf zu öffnen, ob diese über die entsprechenden Berechtigungen verfügen, bevor der Zugriff auf das Objekt gestattet wird. Dabei werden die Prüfungen für den Namen des Objekts vorgenommen, das geöffnet werden soll.

Objekte können wie folgt definiert und verwaltet werden:

- Mit den in den Abschnitten Referenz zu Programmable Command Formats und Verwaltungstasks automatisieren beschriebenen PCF-Befehlen.
- Die im Abschnitt MQSC-Befehle beschriebenen MQSC-Befehle
- 

Die Operationen und Steuerkonsolen von IBM MQ for z/OS , beschrieben in IBM MQ for z/OS verwalten
- 

Über IBM MQ Explorer (nur Windows- und Linux for Intel-Systeme). Weitere Informationen finden Sie im Abschnitt Einführung in MQ Explorer.

Weitere Möglichkeiten für die Objektverwaltung sind:

- Steuerbefehle, die über die Tastatur eingeben werden. Siehe IBM MQ for Multiplatforms mit Steuerbefehlen verwalten.

- MQAI-Aufrufe (IBM MQ Administration Interface) in einem Programm. Siehe IBM MQ Administration Interface (MQAI).

ALW Für IBM MQ-Befehlsfolgen unter AIX, Linux, and Windows können Sie die MQSC-Funktion verwenden, um eine Reihe von Befehlen in einer Datei auszuführen. Weitere Informationen finden Sie unter IBM MQ mit MQSC-Befehlen verwalten.

IBM i Für häufig verwendete IBM MQ for IBM i-Befehlsfolgen können Sie CL-Programme schreiben. Weitere Informationen finden Sie im Abschnitt IBM MQ for IBM i mit CL-Befehlen verwalten.

z/OS Für häufig verwendete IBM MQ for z/OS-Befehlsfolgen können Sie auch Verwaltungsprogramme schreiben, die Nachrichten mit diesen Befehlen erstellen und in die Eingabewarteschlange für Systembefehle einreihen. Der Warteschlangenmanager verarbeitet die Nachrichten in dieser Warteschlange wie Befehle, die über die Befehlszeile oder die Betriebs- und Steueranzeigen eingegeben werden. Dieses Verfahren wird unter Programme für die Verwaltung von IBM MQ schreiben erläutert und in der mit IBM MQ for z/OS gelieferten Beispielanwendung 'Mail Manager' veranschaulicht. Eine Beschreibung dieses Beispiels finden Sie unter Beispielprogramme für IBM MQ for z/OS .

Objektattribute

Die Eigenschaften eines Objektes werden über die Objektattribute definiert. Einige dieser Attribute können Sie angeben, andere hingegen können nur angezeigt werden.

Das Attribut **MaxMsgLength** einer Warteschlange gibt beispielsweise die maximal mögliche Länge für Nachrichten an, die in dieser Warteschlange eingereiht werden können. Das Attribut **DefinitionType** gibt an, wie die Warteschlange erstellt wurde; dieses Attribut kann nur angezeigt werden.

In IBM MQ gibt es für den Verweis auf Attribute zwei Möglichkeiten:

- Über den PCF-Namen des Attributs (beispielsweise **MaxMsgLength**).
- Über den WebSphere MQ-Scriptbefehlsnamen des Attributs (beispielsweise MAXMSGL).

Gruppen mit gemeinsamer Warteschlange

z/OS

Warteschlangenmanager, die auf dieselbe Gruppe gemeinsam genutzter Warteschlangen zugreifen können, bilden eine sogenannte *Gruppe mit gemeinsamer Warteschlange*. Sie kommunizieren über eine Coupling-Facility miteinander, auf der die gemeinsam genutzten Warteschlangen gespeichert werden. Beachten Sie, dass eine Gruppe mit gemeinsamer Warteschlange kein striktes Objekt ist.

Eine gemeinsam genutzte Warteschlange ist eine bestimmte Art von lokaler Warteschlange, auf deren Nachrichten ein oder mehrere Warteschlangenmanager aus einer Gruppe mit gemeinsamer Warteschlange zugreifen können. Dies ist nicht zu verwechseln mit einer Warteschlange, die von mehreren Anwendungen gemeinsam genutzt wird, die denselben Warteschlangenmanager verwenden.

Die Namen von Gruppen mit gemeinsamer Warteschlange können bis zu vier Zeichen lang sein. Der Name muss in Ihrem Netz eindeutig sein und muss sich von allen WS-Managernamen unterscheiden.

Wichtig: Gemeinsam genutzte Warteschlangen und Gruppen mit gemeinsamer Warteschlange werden nur in IBM MQ for z/OS unterstützt.

Weitere Informationen finden Sie unter „Shared queues and queue sharing groups” auf Seite 180.

Systemstandardwertobjekte

Systemstandardobjekte sind Objektdefinitionen, die bei der Erstellung eines Warteschlangenmanagers automatisch erstellt werden. Sie können diese Objektdefinitionen für die Verwendung in den Anwendungen Ihrer Installation kopieren und entsprechend ändern.

Standardobjektnamen beginnen mit SYSTEM; die lokale Standardschlange beispielsweise hat die Bezeichnung SYSTEM.DEFAULT.LOCAL.QUEUE, der Standardempfängerkanal die Bezeichnung SYS-

TEM.DEF.RECEIVER. Die Standardobjekte mit diesen Namen sind erforderlich; sie können nicht umbenannt werden.

Alle Attribute, die bei der Definition eines Objekts nicht angegeben werden, werden explizit aus dem entsprechenden Standardobjekt kopiert. Wenn Sie beispielsweise eine lokale Warteschlange definieren, werden die von Ihnen nicht angegebenen Attribute aus der Standardwarteschlange SYSTEM.DEFAULT.LOCAL.QUEUE übernommen.

Weitere Informationen finden Sie unter System-und Standardobjekte .

Objekttypen

Bei vielen Verwaltungstasks werden verschiedene Arten von IBM MQ-*Objekten* bearbeitet.

Informationen zur Benennung von IBM MQ-Objekten finden Sie im Abschnitt „IBM MQ-Objekte benennen” auf Seite 39.

Informationen zu den auf einem Warteschlangenmanager erstellten Standardobjekten finden Sie im Abschnitt „Systemstandardwertobjekte” auf Seite 16.

Informationen zu den verschiedenen Typen von IBM MQ -Objekten finden Sie in den folgenden Abschnitten:

Authentifizierungsdatenobjekte

Ein Authentifizierungsdatenobjekt stellt die Definitionen bereit, die für die Überprüfung auf widerrufene Zertifikate erforderlich sind.

Das Authentifizierungsdatenobjekt eines Warteschlangenmanagers ist Teil der TLS-Unterstützung (Transport Layer Security) von IBM MQ. Es stellt die Definitionen bereit, die für eine Überprüfung auf widerrufene Zertifikate erforderlich sind. Zertifikate, die nicht mehr als vertrauenswürdig eingestuft werden, werden von Zertifizierungsstellen widerrufen.

Sie können mit dem MQSC-Befehl **DEFINE AUTHINFO** ein Authentifizierungsdatenobjekt definieren. Weitere Informationen zu den Attributen von Authentifizierungsinformationsobjekten finden Sie unter **DEFINE AUTHINFO**.

Sie können folgende IBM MQ-Steuerbefehle für ein Authentifizierungsdatenobjekt verwenden:

- **setmqaut** (Berechtigung erteilen oder entziehen)
- **dspmqaut** (Vergabe der Objektberechtigung anzeigen)
- **dmpmqaut** (Speicherauszugsberechtigungen)
- **rcrmqobj** (Objekt erneut erstellen)
- **rcdmqimg** (Medienimage aufzeichnen)
- **dspmqfls** (Dateinamen anzeigen)

Eine Übersicht über TLS und die Verwendung der Authentifizierungsinformationsobjekte finden Sie unter TLS-Konzepte (Transport Layer Security) und TLS-Sicherheitsprotokolle in IBM MQ.

Kanäle

Kanäle sind Objekte, die einen Kommunikationspfad von einem Warteschlangenmanager zu einem anderen bereitstellen.

Weitere Informationen finden Sie unter „Kanäle” auf Seite 32.

Kommunikationsinformationsobjekte

IBM MQ Multicast bietet eine geringe Latenzzeit, eine hohe Ausgabefächerung und eine zuverlässige Multicasting-Nachrichtenübertragung. Für die Multicasting-Übertragung ist ein Kommunikationsinformationsobjekt (COMMINFO-Objekt) erforderlich.

Weitere Informationen finden Sie unter „IBM MQ Multicasting” auf Seite 115.

Bei COMMINFO-Objekten handelt es sich um IBM MQ-Objekte, die Attribute in Zusammenhang mit Multicast-Übertragungen enthalten. Weitere Informationen zu diesen Attributen finden Sie im Abschnitt DEFINE COMMINFO. Weitere Informationen zur Erstellung von COMMINFO-Objekten finden Sie im Abschnitt Erste Schritte mit Multicasting.

Empfangsprogramme

Empfangsprogramme sind Prozesse, die Netzanforderungen von anderen Warteschlangenmanagern oder Clientanwendungen annehmen und die zugeordneten Kanäle starten.

Listenerprozesse können mit dem Steuerbefehl **runmqlsr** gestartet werden.

Empfangsprogrammobjekte sind IBM MQ-Objekte, mit denen Empfangsprogrammprozesse auf Warteschlangenmanagerebene gestartet und gestoppt werden können. Über die Definition der Attribute eines Empfangsprogramms können Sie:

- den Empfangsprogrammprozess definieren;
- angeben, ob der Empfangsprogrammprozess beim Starten bzw. Stoppen des Warteschlangenmanagers automatisch gestartet bzw. gestoppt werden soll.

Wichtig: z/OS Empfangsprogrammobjekte werden in IBM MQ for z/OS nicht unterstützt. Weitere Informationen zur Implementierung der Empfangsbereitschaft in IBM MQ for z/OS mithilfe des Kanalinitiators finden Sie im Abschnitt „The channel initiator on z/OS” auf Seite 176.

Namenslisten

Eine *Namensliste* ist ein IBM MQ-Objekt, das eine Liste mit Clusternamen, Warteschlangennamen oder Namen von Authentifizierungsdatenobjekten enthält. In einem Cluster kann dieses Objekt eine Liste der Cluster enthalten, für die der Warteschlangenmanager die Repositorys enthält.

Eine Namensliste ist ein IBM MQ-Objekt, das eine Liste anderer IBM MQ-Objekte enthält. Namenslisten werden in der Regel von Anwendungen wie beispielsweise Auslösemonitoren verwendet (in diesem Fall ist in diesem Objekt eine Gruppe von Warteschlangen angegeben). Namenslisten haben den Vorteil, dass sie unabhängig von Anwendungen verwaltet werden können; sie können aktualisiert werden, ohne dass eine der Anwendungen gestoppt werden muss, die dieses Objekt verwenden. Der Ausfall einer Anwendung hat keinerlei Auswirkungen auf die Namensliste - sie kann von den anderen Anwendungen weiterhin verwendet werden.

Namenslisten werden auch mit Warteschlangenmanagerclustern verwendet, um eine Liste von Clustern zu verwalten, auf die von mehreren IBM MQ -Objekten verwiesen wird.

Sie können Namenslisten mit den MQSC-Befehlen DEFINE NAMELIST und ALTER NAMELIST definieren und ändern.

Anmerkung: z/OS Alternativ können Sie unter z/OSdie Operationen und Steuerkonsolen von IBM MQ for z/OS verwenden.

Programme können über die MQI feststellen, welche Warteschlangen in diesen Namenslisten aufgeführt sind. Die Verwaltung der Namenslisten ist Aufgabe des Anwendungsentwicklers und des Systemadministrators.

Eine Liste der verfügbaren Namenslistenattribute finden Sie unter Attribute für Namenslisten.

Prozessdefinitionen

Über Prozessdefinitionsobjekte können Anwendungen ohne Bedienereingriff gestartet werden; dazu werden die Attribute der Anwendung für den Warteschlangenmanager definiert.

Prozessdefinitionsobjekte definieren eine Anwendung, die auf ein Auslöserereignis auf einem IBM MQ-Warteschlangenmanager hin gestartet wird. Zu den Prozessdefinitionsattributen gehören die Anwen-

dungs-ID, der Anwendungstyp sowie die anwendungsspezifischen Daten. Weitere Informationen finden Sie unter *Initialisierungswarteschlangen* im Abschnitt „Für bestimmte Zwecke von IBM MQ verwendete Warteschlangen” auf Seite 29.

Damit eine Anwendung ohne Bedienereingriff gestartet werden kann (siehe Abschnitt IBM MQ-Anwendungen mithilfe von Auslösten starten), müssen die Attribute der Anwendung dem Warteschlangenmanager bekannt sein. Diese Attribute werden in einem *Prozessdefinitionsobjekt* definiert.

Das Attribut **ProcessName** wird bei der Erstellung des Objekts festgelegt. Andere Attribute können Sie jedoch mithilfe von IBM MQ-Befehlen ändern.

Anmerkung: z/OS Alternativ können Sie unter z/OSdie Operationen und Steuerkonsolen von IBM MQ for z/OS verwenden.

Informationen zu den Werten aller Attribute finden Sie im Abschnitt MQINQ – Objektattribute abfragen.

Eine Liste der verfügbaren Prozessdefinitionsattribute finden Sie unter Attribute für Prozessdefinitionen.

Warteschlangen

Eine IBM MQ-*Warteschlange* ist ein benanntes Objekt, in das Anwendungen Nachrichten einreihen und aus dem Anwendungen Nachrichten abrufen können.

Weitere Informationen finden Sie unter „Warteschlangen” auf Seite 20.

Warteschlangenmanager

IBM MQ Warteschlangenmanager stellen Services zur Warteschlangensteuerung für Anwendungen bereit und verwalten die Warteschlangen, die zu ihnen gehören.

Weitere Informationen finden Sie unter „Warteschlangenmanager” auf Seite 30.

Services

Über *Serviceobjekte* können die Programme definiert werden, die beim Starten oder Stoppen eines Warteschlangenmanagers ausgeführt werden sollen.

Es kann sich um folgende Programmtypen handeln:

Server
Für Serviceobjekte des Typs *Server* ist der Parameter SERVTYPE auf SERVER gesetzt. Bei einem Serviceobjekt des Typs SERVER handelt es sich um die Definition eines Programms, das beim Start des angegebenen Warteschlangenmanagers ausgeführt wird. Es kann immer nur jeweils eine Instanz eines Serverprozesses ausgeführt werden. Der Status eines Serverprozesses kann während der Ausführung mit dem WebSphere MQ-Scriptbefehl DISPLAY SVSTATUS überwacht werden. Serverserviceobjekte sind in der Regel Definitionen von Programmen wie beispielsweise Steuerroutinen für nicht zustellbare Nachrichten oder Auslösemonitore; die Programme, die ausgeführt werden können, sind allerdings nicht auf die von IBM MQ bereitgestellten Programme beschränkt. Darüber hinaus kann ein Serverserviceobjekt auch einen Befehl enthalten, mit dem das Programm nach der Beendigung des angegebenen Warteschlangenmanagers beendet wird.

Befehle
Ein *Befehl* ist ein Serviceobjekt, für das der Parameter SERVTYPE auf COMMAND gesetzt ist. Ein Befehlsserviceobjekt ist die Definition eines Programms, das beim Starten oder Stoppen des angegebenen Warteschlangenmanagers ausgeführt wird. Mehrere Instanzen eines Befehlsprozesses können gleichzeitig ausgeführt werden. Befehlsserviceobjekte unterscheiden sich von Serverserviceobjekten darin, dass das Programm bei der Ausführung nicht vom Warteschlangenmanager überwacht wird. Befehlsserviceobjekte sind Definitionen von Programmen mit einer kurzen Lebensdauer, mit denen eine bestimmte Aufgabe wie beispielsweise das Starten einer oder mehrerer Tasks ausgeführt wird.

Wichtig: z/OS Serviceobjekte werden unter IBM MQ for z/OS nicht unterstützt.

Weitere Informationen hierzu finden Sie unter Mit Services arbeiten.

Speicherklassen

z/OS

Eine Speicherklasse ordnet einer Seitengruppe eine oder mehrere Warteschlangen zu.

Folglich werden Nachrichten für die Warteschlange in der entsprechenden Seitengruppe gespeichert (gepuffert).

Speicherklassen werden nur in IBM MQ for z/OS unterstützt.

Weitere Informationen zu Speicherklassen finden Sie unter IBM MQ -Umgebung unter z/OSplanen.

Topic-Objekte

Ein *Themenobjekt* ist ein IBM MQ-Objekt, mit dem Themen bestimmte, nicht standardmäßige Attribute zugewiesen werden können.

Ein *Thema* wird von einer Anwendung definiert, die für eine bestimmte *Themenzeichenfolge* eine Veröffentlichung oder Subskription durchführt. Eine Themenzeichenfolge kann eine Hierarchie von Themen angeben, indem die Themen durch einen Schrägstrich (/) voneinander getrennt werden. Dies kann durch eine *Themenstruktur* dargestellt werden. Wenn eine Anwendung beispielsweise Veröffentlichungen für die Themenzeichenfolgen `/Sport/American Football` und `/Sport/Soccer` durchführt, wird eine Themenstruktur mit dem übergeordneten Knoten `Sport` und den zwei untergeordneten Knoten `American Football` und `Soccer` erstellt.

Themen übernehmen ihre Attribute vom ersten übergeordneten Verwaltungsknoten, der in ihrer Themenstruktur gefunden wird. Wenn eine bestimmte Themenstruktur keine administrativen Themenknoten enthält, übernehmen alle Themen ihre Attribute vom Basisthemenobjekt SYSTEM.BASE.TOPIC.

Sie können jedem Knoten einer Themenstruktur ein Themenobjekt hinzufügen, indem Sie die Themenzeichenfolge des betreffenden Knotens im Attribut TOPICSTR des Themenobjekts angeben. Sie können auch weitere Attribute für den administrativen Themenknoten definieren. Weitere Informationen zu diesen Attributen finden Sie in den Abschnitten MQSC-Befehle bzw. Verwaltung mithilfe von PCF-Befehlen automatisieren. Jedes Themenobjekt übernimmt standardmäßig seine Attribute von dem übergeordneten administrativen Themenknoten, der ihm am nächsten liegt.

Themenobjekte können auch dazu verwendet werden, die vollständige Themenstruktur vor Anwendungsentwicklern zu verbergen. Wenn ein Themenobjekt mit dem Namen `FOOTBALL.US` für das Thema `/Sport/American Football`erstellt wird, kann eine Anwendung das Objekt mit dem Namen `FOOTBALL.US` anstelle der Zeichenfolge `/Sport/American Football` mit demselben Ergebnis veröffentlichen oder subskribieren.

Wenn Sie in einer Themenzeichenfolge für ein Themenobjekt das Zeichen #, +, / oder * eingeben, wird es wie ein normales Zeichen innerhalb der Zeichenfolge behandelt, d. h., es wird als Teil der Themenzeichenfolge betrachtet, die einem Themenobjekt zugeordnet ist.

Weitere Informationen zu Themenobjekten finden Sie im Abschnitt „Publish/Subscribe-Messaging” auf Seite 67.

Zugehörige Konzepte

„Einführung in die Nachrichtenwarteschlangensteuerung” auf Seite 5
Mit den IBM MQ-Produkten können Programme über ein Netz ungleicher Komponenten (Prozessoren, Betriebssysteme, Subsysteme und Kommunikationsprotokolle) unter Verwendung einer konsistenten Anwendungsprogrammierschnittstelle miteinander kommunizieren.

Zugehörige Verweise

Die MQSC-Befehle

Warteschlangen

Einführung in IBM MQ-Warteschlangen und Warteschlangenattribute

Nachrichten werden in einer Warteschlange gespeichert. Wenn also die Anwendung, welche die Nachricht einreiht, eine Antwort auf ihre Nachricht erwartet, kann sie währenddessen andere Arbeiten erledigen. Anwendungen greifen über die im Abschnitt Übersicht über das Message Queue Interface beschriebene Message Queue Interface auf eine Warteschlange zu.

Bevor Nachrichten in eine Warteschlange eingereiht werden können, muss diese bereits erstellt sein. Eine Warteschlange gehört zu einem Warteschlangenmanager, der wiederum Eigner vieler Warteschlangen sein kann. Die Namen der einzelnen Warteschlangen müssen allerdings innerhalb des betreffenden Warteschlangenmanagers eindeutig sein.

Eine Warteschlange wird von einem Warteschlangenmanager verwaltet. In den meisten Fällen werden die einzelnen Warteschlangen von dem jeweiligen Warteschlangenmanager physisch verwaltet, ohne dass dies den Anwendungsprogrammen offensichtlich ist. Gemeinsam genutzte Warteschlangen in IBM MQ for z/OS können von jedem Warteschlangenmanager in der Gruppe mit gemeinsamer Warteschlange verwaltet werden.

Warteschlangen können mithilfe von IBM MQ-Befehlen (MQSC), PCF-Befehlen oder plattformspezifischen Schnittstellen erstellt werden. Die Betriebs- und Steueranzeigen von IBM MQ for z/OS sind beispielsweise plattformspezifisch.

Lokale Warteschlangen für temporäre Jobs können *dynamisch* von der Anwendung aus erstellt werden. So können beispielsweise *Warteschlangen für zu beantwortende Nachrichten* erstellt werden (die nach Beendigung der Anwendung nicht mehr benötigt werden). Weitere Informationen finden Sie in „Dynamische und Modellwarteschlangen” auf Seite 26.

Bevor Sie eine Warteschlange verwenden können, müssen Sie sie zunächst öffnen und dabei angeben, wofür die Warteschlange vorgesehen ist. So können Warteschlangen beispielsweise für folgende Zwecke geöffnet werden:

- Ausschließlich zum Durchsuchen (nicht zum Abrufen) von Nachrichten
- Zum Abrufen von Nachrichten (mit gemeinsamem Zugriff mit anderen Programmen oder mit exklusivem Zugriff)
- Zum Einreihen von Nachrichten in die Warteschlange
- Zur Abfrage der Warteschlangenattribute
- Zur Definition der Warteschlangenattribute

Eine vollständige Liste der Optionen, die beim Öffnen einer Warteschlange angegeben werden können, finden Sie im Abschnitt MQOPEN - Objekt öffnen.

Warteschlangenattribute

Einige Warteschlangenattribute werden bei der Definition der Warteschlange angegeben und können später nicht mehr geändert werden (z. B. der Warteschlangentyp). Andere Warteschlangenattribute können danach eingeteilt werden, von wem sie geändert werden können:

- Attribute, die vom Warteschlangenmanager während der Warteschlangenverarbeitung geändert werden können (z. B. die aktuelle Warteschlangenlänge)
- Attribute, die nur über Befehle geändert werden können (z. B. die Textbeschreibung der Warteschlange)
- Attribute, die von Anwendungen unter Verwendung des Aufrufs MQSET geändert werden können (z. B. ob Einreihvorgänge in die Warteschlange erlaubt sein sollen).

Die Werte aller Attribute können mit dem Aufruf MQINQ abgerufen werden.

Folgende Attribute werden für mehrere Warteschlangentypen verwendet:

QName
Name der Warteschlange

QType
Typ der Warteschlange

QDesc
Textbeschreibung der Warteschlange

InhibitGet
Gibt an, ob Programme Nachrichten aus der Warteschlange abrufen dürfen. Es können jedoch niemals Nachrichten aus fernen Warteschlangen abgerufen werden.

InhibitPut
Gibt an, ob Programme Nachrichten in die Warteschlange einreihen dürfen.

DefPriority
Gibt die Standardpriorität für in die Warteschlange eingereihte Nachrichten an.

DefPersistence
Die Standardpersistenz für Nachrichten, die in die Warteschlange eingereiht werden.

Bereich
Steuert, ob auch in einem Namensservice ein Eintrag für diese Warteschlange vorhanden ist.

z/OS Das Attribut **Scope** wird unter z/OS nicht unterstützt.

Eine umfassende Beschreibung dieser Attribute finden Sie im Abschnitt Attribute für Warteschlangen.

Methoden zur Definition von Warteschlangen

Sie können Warteschlangen für IBM MQ mit dem MQSC-Befehl DEFINE oder dem PCF-Befehl Create Queue definieren. Mit den Befehlen werden Typ und Attribute der Warteschlange angegeben. Über die Attribute einer lokalen Warteschlange wird beispielsweise festgelegt, was geschieht, wenn Anwendungen in MQI-Aufrufen auf diese Warteschlange verweisen. Hier einige Beispiele für Attribute:

- Ob Anwendungen Nachrichten aus der Warteschlange abrufen können (GET-aktiviert).
- Ob Anwendungen Nachrichten in die Warteschlange einreihen können (PUT-aktiviert).
- Ob nur eine Anwendung oder aber mehrere Anwendungen auf die Warteschlange zugreifen können.
- Die maximale Anzahl an Nachrichten, die gleichzeitig in der Warteschlange gespeichert werden können (max. Warteschlangenlänge).
- Die Länge, die Nachrichten maximal haben dürfen, damit sie in die Warteschlange eingereiht werden können.

Es gibt auch verschiedene plattformspezifische Schnittstellen, die Sie verwenden können, um Warteschlangen zu definieren.

Zugehörige Konzepte

„Clusterwarteschlangen” auf Seite 63
Eine Clusterwarteschlange wird von einem Clusterwarteschlangenmanager anderen Warteschlangenmanagern im Cluster zur Verfügung gestellt.

„Warteschlangen für nicht zustellbare Nachrichten” auf Seite 53
Die Warteschlange für nicht zustellbare Nachrichten (oder Warteschlange für nicht zugestellte Nachrichten) ist die Warteschlange, an die Nachrichten gesendet werden, wenn sie nicht an ihr eigentliches Ziel weitergeleitet werden können. In der Regel hat jeder Warteschlangenmanager eine solche Warteschlange.

Verwaltung mithilfe von PCF-Befehlen automatisieren

IBM MQ Console: Mit Warteschlangen arbeiten

Zugehörige Tasks

IBM MQ mit MQSC-Befehlen verwalten

Warteschlangenmanager und Objekte mit MQ Explorer erstellen und konfigurieren

IBM i IBM MQ for IBM i mithilfe von CL-Befehlen verwalten

z/OS Quellen, aus denen Sie MQSC-und PCF-Befehle unter IBM MQ for z/OS ausgeben können

Zugehörige Verweise

„Comparison between shared queues and cluster queues” auf Seite 63

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

Zugehörige Informationen

„What is a shared queue?" auf Seite 180

Lokale Warteschlangen

Übertragungs-, Initialisierungs-, Befehls-, Standard-, Kanal- und Ereigniswarteschlangen sowie Warteschlangen für nicht zustellbare Nachrichten sind verschiedene Typen lokaler Warteschlangen.

Eine Warteschlange gilt für ein Programm als *lokal*, wenn sie dem Warteschlangenmanager zugeordnet ist, mit dem das Programm verbunden ist. Nachrichten können aus lokalen Warteschlangen abgerufen und darin eingereiht werden.

Das Warteschlangendefinitionsobjekt enthält die Definitionsinformationen der Warteschlange sowie die in die Warteschlange eingereihten physischen Nachrichten.

Jeder Warteschlangenmanager kann über einige lokale Warteschlangen verfügen, die für spezielle Zwecke eingesetzt werden:

Übertragungswarteschlangen

Wenn eine Anwendung eine Nachricht an eine ferne Warteschlange sendet, speichert der lokale Warteschlangenmanager die Nachricht in einer speziellen lokalen Warteschlange, der sogenannten *Übertragungswarteschlange*. Anwendungen können Nachrichten direkt in eine Übertragungswarteschlange einreihen oder indirekt mithilfe der Definition einer fernen Warteschlange.

Wenn ein Warteschlangenmanager Nachrichten an einen fernen Warteschlangenmanager sendet, ermittelt er die Übertragungswarteschlange in der folgenden Reihenfolge:

1. Die Übertragungswarteschlange, die im Attribut XMITQ der lokalen Definition einer fernen Warteschlange angegeben ist.
2. Eine Übertragungswarteschlange mit demselben Namen wie dem des fernen Warteschlangenmanagers. Dies ist der Standardwert des Attributs XMITQ der lokalen Definition einer fernen Warteschlange.
3. Die Übertragungswarteschlange, die im Attribut DEFXMITQ des lokalen Warteschlangenmanagers angegeben ist.

Ein *Nachrichtenkanalagent* ist ein der Übertragungswarteschlange zugeordnetes Kanalprogramm, das die Nachricht an das nächste Ziel übermittelt. Bei dem nächsten Ziel handelt es sich um den Warteschlangenmanager, mit dem der Nachrichtenkanal verbunden ist. Dieser Warteschlangenmanager muss nicht zwangsläufig das endgültige Ziel der Nachricht darstellen. Nach ihrer Übermittlung an das nächste Ziel wird die Nachricht aus der Übertragungswarteschlange gelöscht. Möglicherweise muss die Nachricht auf ihrem Weg zum Zielort viele Warteschlangenmanager durchlaufen. Auf jedem Warteschlangenmanager entlang der Route muss eine Übertragungswarteschlange definiert werden. Diese enthalten jeweils Nachrichten, die auf die Übertragung zum nächsten Ziel warten. Eine normale Übertragungswarteschlange enthält Nachrichten für das nächste Ziel, die jedoch nicht alle denselben endgültigen Zielort haben müssen. In einer Clusterübertragungswarteschlange sind Nachrichten für mehrere Ziele enthalten. Die Korrelations-ID (`correlID`) der einzelnen Nachrichten gibt den Kanal an, in den die Nachricht zur Übertragung an das nächste Ziel gestellt wird.

An einem Warteschlangenmanager können mehrere Übertragungswarteschlangen definiert werden. Sie können auch mehrere Übertragungswarteschlangen für dasselbe Ziel definieren, die jeweils für eine andere Berechtigungsklasse verwendet werden. So ist es unter Umständen sinnvoll, für kurze und lange Nachrichten an dasselbe Ziel unterschiedliche Übertragungswarteschlangen zu erstellen. Die Nachrichten können dann über verschiedene Nachrichtenkanäle übertragen werden, sodass die kürzeren Nachrichten nicht von den längeren aufgehalten werden. Alle Nachrichten an Clusterwarteschlangen oder Cluster-Topics werden standardmäßig in eine einzige Clusterübertragungswarteschlange namens `SYSTEM.CLUSTER.TRANSMIT.QUEUE` gestellt. Sie haben die Möglichkeit, die Standardeinstellung zu ändern und die Nachrichtenübertragung an verschiedene Cluster-Warteschlangenmanager auf mehrere Clusterübertragungswarteschlangen aufzuteilen. Wenn Sie das Warteschlangenmana-

ger-Attribut DEFCLXQ auf CHANNEL setzen, erstellt jeder Clustersenderkanal eine separate Clusterübertragungswarteschlange. Eine weitere Möglichkeit besteht darin, manuell Clusterübertragungswarteschlangen zu definieren, die von den Clustersenderkanälen verwendet werden sollen.

Mithilfe von Übertragungswarteschlangen kann ein Nachrichtenkanalagent veranlasst werden, Nachrichten weiterzuleiten. Informationen hierzu finden Sie im Abschnitt IBM MQ-Anwendungen mithilfe von Auslösern starten.

z/OS Wenn Sie unter IBM MQ for z/OS eine gruppeninterne Warteschlangensteuerung verwenden, ist für die Übertragungswarteschlange ein *Agent für die gruppeninterne Warteschlangensteuerung* zuständig. Bei Einsatz der gruppeninternen Warteschlangensteuerung in IBM MQ for z/OS wird eine gemeinsam genutzte Übertragungswarteschlange verwendet.

Initialisierungswarteschlangen

Eine *Initialisierungswarteschlange* ist eine lokale Warteschlange, in die der Warteschlangenmanager eine Auslösenachricht einreiht, wenn in einer Anwendungswarteschlange ein Auslöserereignis eintritt.

Ein Auslöserereignis ist ein Ereignis, das ein Programm dazu veranlassen soll, die Verarbeitung einer Warteschlange zu starten. Bei einem solchen Ereignis könnte es sich beispielsweise um den Eingang von mehr als 10 Nachrichten handeln. Weitere Informationen zur Funktionsweise von Auslösern finden Sie im Abschnitt IBM MQ-Anwendungen mithilfe von Auslösern starten.

Warteschlange für nicht zustellbare Nachrichten

Eine *Warteschlange für nicht zustellbare Nachrichten* ist eine lokale Warteschlange, in die der Warteschlangenmanager Nachrichten einreiht, die nicht zugestellt werden können.

Wenn der Warteschlangenmanager eine Nachricht in die Warteschlange für nicht zustellbare Nachrichten stellt, fügt er ihr einen Header hinzu. In den Headerinformationen ist unter anderem auch angegeben, warum die Nachricht vom Warteschlangenmanager in die Warteschlange für nicht zustellbare Nachrichten gestellt wurde. Darüber hinaus sind der Zielort der ursprünglichen Nachricht sowie der Zeitpunkt (Datum und Uhrzeit) aufgeführt, an dem der Warteschlangenmanager die Nachricht in die Warteschlange für nicht zustellbare Nachrichten eingereiht hat.

Auch Anwendungen können die Warteschlange für Nachrichten verwenden, die sie nicht zustellen können. Weitere Informationen finden Sie im Abschnitt Verwendung der Warteschlange für nicht zustellbare Nachrichten.

Systembefehlswarteschlange

Eine *Systembefehlswarteschlange* ist eine Warteschlange, an die entsprechend berechtigte Anwendungen IBM MQ-Befehle senden können. Je nachdem, welche Befehle auf Ihrer Plattform unterstützt werden, empfangen diese Warteschlangen PCF-, MQSC- und CL-Befehle, die dann vom Warteschlangenmanager bearbeitet werden können.

z/OS Unter IBM MQ for z/OS heißt die Warteschlange SYSTEM.COMMAND.INPUT ; Auf anderen Plattformen heißt sie SYSTEM.ADMIN.COMMAND.QUEUE. Welche Befehle akzeptiert werden, hängt von der Plattform ab. Details hierzu finden Sie im Abschnitt Programmierbare Befehlsformate-Referenz.

Systemstandardwarteschlangen

In den *Systemstandardwarteschlangen* sind die ursprünglichen Definitionen der Warteschlangen für Ihr System enthalten. Wenn Sie eine Warteschlangendefinition erstellen, wird die Definition vom Warteschlangenmanager aus der entsprechenden Systemstandardwarteschlange kopiert. Die Vorgehensweise beim Erstellen einer Warteschlangendefinition ist anders als beim Erstellen einer dynamischen Warteschlange. Die Definition der dynamischen Warteschlange basiert auf der als Schablone für die dynamische Warteschlange ausgewählten Modellwarteschlange.

Ereigniswarteschlangen
In *Ereigniswarteschlangen* sind Ereignisnachrichten enthalten. Diese Nachrichten werden vom Warteschlangenmanager oder einem Kanal gemeldet.

Ferne Warteschlangen

Eine Warteschlange ist für ein Programm *fern*, wenn sie zu einem Warteschlangenmanager gehörig ist, mit dem das Programm nicht verbunden ist.

Wenn eine Kommunikationsverbindung eingerichtet ist, kann ein Programm eine Nachricht an eine ferne Warteschlange senden. Nicht möglich ist es dagegen, dass ein Programm eine Nachricht aus einer fernen Warteschlange abruft.

In dem bei der Definition einer fernen Warteschlange erstellten Warteschlangendefinitionsobjekt sind nur die Informationen enthalten, die erforderlich sind, damit der lokale Warteschlangenmanager die Warteschlange, an welche die Nachricht gehen soll, ausfindig machen kann. Dieses Objekt wird auch als *lokale Definition einer fernen Warteschlange* bezeichnet. Alle Attribute der fernen Warteschlange sind auf dem Warteschlangenmanager gespeichert, der Eigner der Warteschlange ist, da es sich für diesen Warteschlangenmanager um eine lokale Warteschlange handelt.

Beim Öffnen einer fernen Warteschlange ist zur Identifikation der Warteschlange eine der folgenden Angaben erforderlich:

- Der Name der lokalen Definition der fernen Warteschlange. Vom Blickpunkt einer Anwendung aus besteht kein Unterschied zum Öffnen einer lokalen Warteschlange. Für die Anwendung ist es nicht relevant, ob es sich um eine lokale oder eine ferne Warteschlange handelt.

  Verwenden Sie den Befehl DEFINE QREMOTE , um auf allen Plattformen außer IBM ieine lokale Definition einer fernen Warteschlange zu erstellen.

  IBM i Verwenden Sie unter IBM i den Befehl CRTMQMQ.
- Der Name des fernen Warteschlangenmanagers und der Name, unter dem dieser ferne Warteschlangenmanager die Warteschlange kennt.

Lokale Definitionen ferner Warteschlangen umfassen zusätzlich zu den im Abschnitt „Warteschlangenattribute” auf Seite 21 beschriebenen allgemeinen Attributen drei weitere Attribute. Bei diesen drei Attributen handelt es sich um die folgenden:

RemoteQName
Der Name, unter dem der Warteschlangenmanager, zu dem die Warteschlange gehört, sie kennt.

RemoteQMgrName
Der Name des Warteschlangenmanagers, zu dem die Warteschlange gehört.

XmitQName
Der Name der lokalen Übertragungswarteschlange, die zur Weiterleitung von Nachrichten an andere Warteschlangenmanager verwendet wird.

Weitere Informationen zu diesen Attributen finden Sie im Abschnitt Attribute für Warteschlangen.

Wenn Sie den Aufruf MQINQ für die lokale Definition einer fernen Warteschlange verwenden, gibt der Warteschlangenmanager nur die Attribute der lokalen Definition zurück, also die Namen der fernen Warteschlange, des fernen Warteschlangenmanagers und der Übertragungswarteschlange. Die Attribute der entsprechenden lokalen Warteschlange im fernen System werden nicht zurückgegeben.

Weitere Informationen finden Sie auch im Abschnitt Übertragungswarteschlangen.

Aliaswarteschlangen

Eine *Aliaswarteschlange* ist ein IBM MQ-Objekt, mit dessen Hilfe der Zugriff auf eine andere Warteschlange oder ein Thema möglich ist. Somit können mehrere Programme mit derselben Warteschlange arbeiten, indem sie für den Zugriff jeweils einen anderen Namen verwenden.

Die aus der Auflösung eines Aliasnamens resultierende Warteschlange, die sogenannte Basiswarteschlange, kann einer der folgenden Warteschlangentypen sein, wie von der Plattform unterstützt:

- Eine lokale Warteschlange

- die lokale Definition einer fernen Warteschlange
- z/OS Eine gemeinsam genutzte Warteschlange, ein lokaler Warteschlangentyp, der nur in IBM MQ for z/OS verfügbar ist.
- eine vordefinierte Warteschlange
- eine dynamische Warteschlange

Ein Aliasname kann auch in ein Thema aufgelöst werden. Wenn eine Anwendung gerade Nachrichten in eine Warteschlange einreiht, kann der Warteschlangenname zu einem Alias für ein Thema gemacht und die Anwendung auf diese Weise zu Veröffentlichungen zu dem Thema gebracht werden. Hierfür muss der Anwendungscode nicht geändert werden.

Anmerkung: Ein Alias kann nicht direkt in ein anderes Alias auf demselben Warteschlangenmanager aufgelöst werden.

Ein Beispiel für die Verwendung von Aliaswarteschlangen: Ein Systemadministrator erteilt für den Basiswarteschlangennamen (also die Warteschlange, in die der Aliasname aufgelöst wird) und den Aliaswarteschlangennamen verschiedene Zugriffsberechtigungen. Es kann dann passieren, dass Programme oder Benutzer berechtigt sind, die Aliaswarteschlange zu verwenden, nicht jedoch die Basiswarteschlange.

Alternativ dazu kann die Berechtigung so definiert werden, dass Einreihoperationen für die Basiswarteschlange erlaubt, für den Aliasnamen dagegen nicht zulässig sind.

Bei manchen Anwendungen bietet sich Systemadministratoren durch die Verwendung von Aliaswarteschlangen die Möglichkeit, ganz einfach die Definition eines Aliaswarteschlangenobjekts zu ändern, ohne dass eine Änderung der Anwendung erforderlich ist.

Wenn Programme versuchen, den Aliasnamen zu verwenden, nimmt IBM MQ Berechtigungsprüfungen für diesen Namen vor. Dabei wird nicht geprüft, ob das Programm auch zum Zugriff auf den Namen berechtigt ist, in den der Aliasname aufgelöst wird. Es kann also sein, dass ein Programm zum Zugriff auf einen Aliaswarteschlangennamen berechtigt ist, jedoch über keine Berechtigung für den aufgelösten Warteschlangennamen verfügt.

Zusätzlich zu den im Abschnitt „Warteschlangen” auf Seite 20 beschriebenen allgemeinen Warteschlangenattributen gibt es zu Aliaswarteschlangen noch das Attribut **BaseQName**. Hierbei handelt es sich um den Basiswarteschlangennamen, in den der Aliasname aufgelöst wird. Eine detailliertere Beschreibung dieser Attribute finden Sie im Abschnitt BaseQName (MQCHAR48).

Die Aliaswarteschlangenattribute *InhibitGet* und **InhibitPut** (siehe Abschnitt „Warteschlangen” auf Seite 20) gehören zum Aliasnamen. Wird beispielsweise der Aliaswarteschlangenname ALIAS1 in den Basiswarteschlangennamen BASE aufgelöst, gelten Beschränkungen für ALIAS1 nur für ALIAS1 und nicht für BASE. Beschränkungen für BASE dagegen gelten auch für ALIAS1.

Auch die Attribute *DefPriority* und **DefPersistence** gehören zum Aliasnamen. Somit können Sie beispielsweise verschiedenen Aliassen derselben Basiswarteschlange unterschiedliche Standardprioritäten zuordnen. Diese Prioritäten können außerdem geändert werden, ohne dass eine Änderung der Anwendungen, welche die Aliasse verwenden, erforderlich ist.

Dynamische und Modellwarteschlangen

In diesem Abschnitt erhalten Sie einen Einblick in dynamische Warteschlangen, Sie finden hier Erläuterungen zu den Eigenschaften temporärer und permanenter dynamischer Warteschlangen, zur Verwendung dynamischer Warteschlangen und zu dabei besonders zu berücksichtigenden Aspekten sowie zu Modellwarteschlangen.

Wenn ein Anwendungsprogramm einen Aufruf MQOPEN absetzt, um eine Modellwarteschlange zu öffnen, erstellt der Warteschlangenmanager mit den Attributen der Modellwarteschlange dynamisch eine Instanz einer lokalen Warteschlange. Je nachdem, welcher Wert im Feld *DefinitionType* der Modellwarteschlange angegeben ist, erstellt der Warteschlangenmanager eine temporäre oder eine permanente dynamische Warteschlange (siehe Dynamische Warteschlangen erstellen).

Eigenschaften temporärer dynamischer Warteschlangen

Temporäre dynamische Warteschlangen weisen folgende Eigenschaften auf:

- z/OS Die Warteschlangen können nicht gemeinsam genutzt werden, es ist also kein Zugriff von Warteschlangenmanagern aus einer Gruppe mit gemeinsamer Warteschlange möglich.

  Beachten Sie, dass Gruppen mit gemeinsamer Warteschlange nur in IBM MQ for z/OS verfügbar sind.
- Sie enthalten nur nicht persistente Nachrichten.
- Sie sind nicht wiederherstellbar.
- Beim Start des Warteschlangenmanagers werden sie gelöscht.
- Sie werden auch gelöscht, wenn die Anwendung, die den Aufruf MQOPEN zum Erstellen der Warteschlange ausgegeben hat, die Warteschlange schließt oder selbst beendet wird.
  - Eventuell vorhandene festgeschriebene Nachrichten in der Warteschlange werden gelöscht.
  - Stehen zu diesem Zeitpunkt nicht festgeschriebene Aufrufe MQGET, MQPUT oder MQPUT1 für die Warteschlange aus, wird die Warteschlange als logisch gelöscht markiert und wird erst physisch gelöscht (nach Festschreibung dieser Aufrufe), wenn der Vorgang zum Schließen der Warteschlange verarbeitet oder die Anwendung beendet wird.
  - Wird die Warteschlange zu diesem Zeitpunkt gerade verwendet (von der erstellenden oder einer anderen Anwendung), wird sie als logisch gelöscht markiert und erst physisch gelöscht, wenn sie von der letzten Anwendung, welche die Warteschlange verwendet, geschlossen wird.
  - Zugriffsversuche auf eine logisch gelöschte Warteschlange schlagen mit dem Ursachencode MQRC_Q_DELETED fehl (es sei denn, der Zugriff erfolgt zum Schließen der Warteschlange).
  - MQCO_NONE, MQCO_DELETE und MQCO_DELETE_PURGE werden alle als MQCO_NONE behandelt, wenn sie in einem Aufruf MQCLOSE zu dem entsprechenden Aufruf MQOPEN angegeben sind, mit dem die Warteschlange erstellt wurde.

Eigenschaften permanenter dynamischer Warteschlangen

Permanente dynamische Warteschlangen weisen folgende Eigenschaften auf:

- Sie enthalten persistente und nicht persistente Nachrichten.
- Im Falle eines Systemfehlers sind sie wiederherstellbar.
- Sie werden gelöscht, wenn sie von einer Anwendung mit der Option MQCO_DELETE oder MQCO_DELETE_PURGE erfolgreich geschlossen werden. (Bei dieser Anwendung muss es sich nicht zwangsläufig um die Anwendung handeln, die den Aufruf MQOPEN zum Erstellen der Warteschlange ausgegeben hat.)
  - Falls sich noch (festgeschriebene oder nicht festgeschriebene) Nachrichten in der Warteschlange befinden, schlägt eine Schließanforderung mit der Option MQCO_DELETE fehl. Eine Schließanforderung mit der Option MQCO_DELETE_PURGE kann selbst dann erfolgreich ausgeführt werden, wenn sich festgeschriebene Nachrichten in der Warteschlange befinden (wobei die Nachrichten im Rahmen des Schließvorgangs gelöscht werden), sie schlägt jedoch fehl, wenn für die Warteschlange nicht festgeschriebene Aufrufe MQGET, MQPUT oder MQPUT1 ausstehen.
  - Wenn die Löschanforderung erfolgreich ist, die Warteschlange jedoch gerade (von der erstellenden oder einer anderen Anwendung) verwendet wird, wird sie als logisch gelöscht markiert und erst physisch gelöscht, wenn sie von der letzten Anwendung, die die Warteschlange verwendet, geschlossen wird.
- Die Warteschlangen werden nicht gelöscht, wenn sie von einer Anwendung geschlossen werden, die nicht zum Löschen der Warteschlange berechtigt ist, es sei denn, diese Anwendung hat den Aufruf MQOPEN zum Erstellen der Warteschlange ausgegeben. Für die Benutzer-ID (bzw. bei Angabe von MQOO_ALTERNATE_USER_AUTHORITY für die alternative Benutzer-ID), mit welcher der entsprechende Aufruf MQOPEN validiert wurde, werden Berechtigungsprüfungen vorgenommen.
- Sie können wie normale Warteschlangen gelöscht werden.

Verwendung dynamischer Warteschlangen

Dynamische Warteschlangen können in folgenden Fällen eingesetzt werden:

- Für Anwendungen, nach deren Beendigung keine Warteschlangen beibehalten werden müssen.
- Für Anwendungen, bei denen Antworten auf Nachrichten von einer anderen Anwendung verarbeitet werden müssen. Solche Anwendungen können durch Öffnen einer Modellwarteschlange dynamisch eine Empfangswarteschlange für Antworten erstellen. Eine Clientanwendung kann beispielsweise folgende Aktionen ausführen:

  1. Eine dynamische Warteschlange erstellen.
  2. Im Feld **ReplyToQ** der Nachrichtendeskriptorstruktur der Anforderungsnachricht deren Namen angeben.
  3. Die Anforderung in eine von einem Server verarbeitete Warteschlange stellen.

  Der Server kann die Antwortnachricht dann in die Empfangswarteschlange für Antworten einreihen. Der Client wiederum kann nun die Antwort verarbeiten und die Empfangswarteschlange für Antworten mit der Löschoption schließen.

Bei der Verwendung dynamischer Warteschlangen zu berücksichtigende Aspekte

Bei der Verwendung dynamischer Warteschlangen sind folgende Punkte zu beachten:

- In einem Client-Server-Modell müssen die einzelnen Clients jeweils eine eigene dynamische Empfangswarteschlange für Antworten erstellen und verwenden. Falls eine dynamische Empfangswarteschlange für Antworten von mehreren Clients gemeinsam genutzt wird, kann es sein, dass die Empfangswarteschlange für Antworten erst mit Verzögerung gelöscht wird, da für die Warteschlange nicht festgeschriebene Aktivitäten ausstehen oder die Warteschlange gerade von einem anderen Client verwendet wird. Außerdem wird die Warteschlange möglicherweise als logisch gelöscht markiert. Nachfolgende API-Anforderungen (außer MQCLOSE) können dann nicht mehr auf die Warteschlange zugreifen.
- Müssen in Ihrer Anwendungsumgebung dynamische Warteschlangen von mehreren Anwendungen gemeinsam genutzt werden, ist darauf zu achten, dass die Warteschlange erst geschlossen wird (mit der Löschoption), wenn die gesamte Aktivität für die Warteschlange festgeschrieben wurde. Diese Aktion sollte der letzte Benutzer ausführen. Auf diese Weise wird gewährleistet, dass die Warteschlange ohne Verzögerungen gelöscht wird. Außerdem wird der Zeitraum, in dem die Warteschlange nicht zugänglich ist, da sie als logisch gelöscht markiert ist, auf ein Minimum reduziert.

Modellwarteschlangen

Eine *Modellwarteschlange* ist eine Schablone einer Warteschlangendefinition, die beim Erstellen einer dynamischen Warteschlange verwendet wird.

Von einem IBM MQ-Programm aus kann dynamisch eine lokale Warteschlange erstellt werden, wobei die als Schablone für die Warteschlangenattribute zu verwendende Modellwarteschlange angegeben wird. Zu diesem Zeitpunkt können einige Attribute der neuen Warteschlange geändert werden. Eine Änderung des Attributs **DefinitionType** ist jedoch nicht möglich. Falls Sie also beispielsweise eine permanente Warteschlange benötigen, wählen Sie eine Modellwarteschlange aus, für die als Definitionstyp 'permanent' angegeben ist. Manche Dialoganwendungen können für Antworten auf ihre Anfragen dynamische Warteschlangen verwenden, da diese Warteschlangen nach Verarbeitung der Antworten vermutlich nicht beibehalten werden müssen.

Den Namen einer Modellwarteschlange geben Sie im *Objektdeskriptor* (MQOD) des Aufrufs MQOPEN an. Unter Verwendung der Attribute der Modellwarteschlange wird vom Warteschlangenmanager dynamisch eine lokale Warteschlange erstellt.

Sie können einen (vollständigen) Namen für die dynamische Warteschlange angeben oder aber nur den Namensstamm (z. B. ABC), dem dann vom Warteschlangenmanager ein eindeutiger Teil hinzugefügt wird, Sie können aber auch die Zuordnung eines eindeutigen Namens komplett dem Warteschlangenmanager überlassen. Falls der Warteschlangenmanager den Namen zuordnet, stellt er ihn in die MQOD-Struktur.

Es ist nicht möglich, direkt für eine Modellwarteschlange einen Aufruf MQPUT1 auszugeben, für die dynamische Warteschlange, die durch Öffnen einer Modellwarteschlange erstellt wurde, kann der Aufruf MQPUT1 dagegen ausgegeben werden.

MQSET und MQINQ können nicht an einer Modellwarteschlange ausgeführt werden. Wenn Sie eine Modellwarteschlange mit MQOO_INQUIRE oder MQOO_SET öffnen, werden an der dynamisch erstellten Warteschlange weitere MQINQ- und MQSET-Aufrufe ausgeführt.

Bei den Attributen einer Modellwarteschlange handelt es sich um eine Untergruppe der Attribute einer lokalen Warteschlange. Eine detailliertere Beschreibung finden Sie im Abschnitt Attribute für Warteschlangen.

Für bestimmte Zwecke von IBM MQ verwendete Warteschlangen

In IBM MQ werden einige lokale Warteschlangen für bestimmte Aspekte des Betriebsablaufs verwendet.

Diese Warteschlangen müssen erst definiert werden, damit sie von IBM MQ verwendet werden können.

Initialisierungswarteschlangen

Initialisierungswarteschlangen werden für die Auslösefunktion (Triggering) eingesetzt. Wenn ein Auslöserereignis eintritt, reiht der Warteschlangenmanager eine Auslösenachricht in eine Initialisierungswarteschlange ein. Ein Auslöserereignis ist eine logische Kombination an Bedingungen, die von einem Warteschlangenmanager erkannt wird. So kann beispielsweise ein Auslöserereignis generiert werden, wenn die Anzahl der Nachrichten in einer Warteschlange die vorgegebene Warteschlangengröße erreicht hat. Bei Eintreten dieses Ereignisses reiht der Warteschlangenmanager eine Auslösenachricht in die angegebene Initialisierungswarteschlange ein. Diese Auslösenachricht wird von einem *Auslösemonitor* abgerufen, bei dem es sich um eine besondere Anwendung für die Überwachung einer Initialisierungswarteschlange handelt. Der Auslösemonitor startet daraufhin das in der Auslösenachricht angegebene Anwendungsprogramm.

Soll die Auslösefunktion in einem Warteschlangenmanager verwendet werden, muss mindestens eine Initialisierungswarteschlange für ihn definiert sein. Weitere Informationen finden Sie in den Abschnitten Objekte für Auslösefunktion verwalten, runmqtrm und IBM MQ-Anwendungen mithilfe von Auslösern starten

Übertragungswarteschlangen

In Übertragungswarteschlangen werden kurzfristig Nachrichten gespeichert, die an einen fernen Warteschlangenmanager gesendet werden sollen. Sie müssen zumindest eine Übertragungswarteschlange für jeden fernen Warteschlangenmanager definieren, an den der lokale Warteschlangenmanager direkt Nachrichten senden soll. Diese Warteschlangen werden auch bei der Fernverwaltung verwendet (siehe Fernverwaltung von einem lokalen Warteschlangenmanager aus). Informationen zur Verwendung von Übertragungswarteschlangen in der verteilten Steuerung von Warteschlangen finden Sie im Abschnitt IBM MQ-Verfahren für verteilte Steuerung von Warteschlangen.

Für jeden Warteschlangenmanager kann eine Standard-Übertragungs-WS definiert werden. Wenn ein Warteschlangenmanager, der zu keinem Cluster gehört, eine Nachricht in eine ferne Warteschlange einreiht, wird standardmäßig die Standard-Übertragungs-WS verwendet. Wenn eine Übertragungswarteschlange mit demselben Namen wie dem des Ziel-Warteschlangenmanagers vorhanden ist, wird die Nachricht in diese Übertragungswarteschlange gestellt. Ist eine Definition eines Warteschlangenmanager-Aliasnamens vorhanden, deren Parameter **RQMNAME** mit dem Zielwarteschlangenmanager übereinstimmt, und ist der Parameter **XMITQ** angegeben, wird die Nachricht in die in **XMITQ** benannte Übertragungswarteschlange eingereiht. Wenn kein **XMITQ**-Parameter vorhanden ist, wird die Nachricht in die in der Nachricht angegebene lokale Warteschlange gestellt.

Clusterübertragungswarteschlangen

Zu jedem Warteschlangenmanager in einem Cluster gibt es eine Clusterübertragungswarteschlange `SYSTEM.CLUSTER.TRANSMIT.QUEUE` und eine Modell-Clusterübertragungswarteschlange `SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE`. Definitionen dieser Warteschlangen werden standardmäßig bei der Definition eines Warteschlangenmanagers erstellt. Wenn das Warteschlangenmanager-Attribut **DEFCLXQ** auf `CHANNEL` gesetzt ist, wird automatisch eine permanente dynamische Cluster-Übertragungs-WS für jeden erstellten Clustersenderkanal erstellt. Die Warteschlangen heißen `SYSTEM.CLUSTER.TRANSMIT.` *`ChannelName`*. Cluster-Übertragungs-WS können auch manuell definiert werden.

Ein Warteschlangenmanager, der zu einem Cluster gehört, sendet Nachrichten in einer dieser Warteschlangen an andere Warteschlangenmanager im selben Cluster.

Bei der Namensauflösung hat eine Cluster-Übertragungs-WS Vorrang vor der standardmäßigen Übertragungswarteschlange und eine spezifische Cluster-Übertragungs-WS hat Vorrang vor SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Warteschlangen für nicht zustellbare Nachrichten
In Warteschlangen für nicht zustellbare Nachrichten werden Nachrichten gespeichert, die nicht an ihr vorgegebenes Ziel gesendet werden können. Eine Nachricht kann nicht weitergeleitet werden, wenn beispielsweise die Zielwarteschlange voll ist. Die bereitgestellte Warteschlange für nicht zustellbare Nachrichten hat den Namen SYSTEM.DEAD.LETTER.QUEUE.

Bei einer verteilten Steuerung von Warteschlangen muss für jeden beteiligten Warteschlangenmanager eine Warteschlange für nicht zustellbare Nachrichten definiert werden.

Befehlswarteschlangen
Die Befehlswarteschlange SYSTEM.ADMIN.COMMAND.QUEUE ist eine lokale Warteschlange, an die entsprechend berechtigte Anwendungen WebSphere MQ-Scriptbefehle zur Verarbeitung senden können. Diese Befehle werden anschließend von einer IBM MQ-Komponente, dem sogenannten Befehlsserver, abgerufen. Der Befehlsserver überprüft die Befehle, sendet die gültigen Befehle weiter zur Verarbeitung durch den Warteschlangenmanager und gibt Antworten an die entsprechende Empfangswarteschlange für Antworten zurück.

Die Befehlswarteschlange wird automatisch bei der Erstellung eines Warteschlangenmanagers erstellt.

Empfangswarteschlangen für Antworten
Wenn eine Anwendung eine Anforderungsnachricht sendet, kann die Anwendung, die diese Nachricht enthält, eine Antwortnachricht an die Anwendung zurückgeben, von der die Nachricht stammt. Diese Nachricht wird in eine sogenannte Empfangswarteschlange für Antworten eingereiht, bei der es sich in der Regel um eine lokale Warteschlange der Anwendung handelt, die die Nachricht sendet. Der Name der Empfangswarteschlange wird von der sendenden Anwendung als Teil des Nachrichtendeskriptors angegeben.

Ereigniswarteschlangen
Mithilfe von Instrumentierungsereignissen können Warteschlangenmanager unabhängig von MQI-Anwendungen überwacht werden.

Bei Eintreten eines Instrumentierungsereignisses reiht der Warteschlangenmanager eine Ereignisnachricht in eine Ereigniswarteschlange ein. Diese Nachricht kann von einer Überwachungsanwendung gelesen werden, die entweder den Administrator informieren oder aber eine Korrekturmaßnahme einleiten kann, wenn das Ereignis auf ein Problem hindeutet.

Anmerkung: Auslöserereignisse und Instrumentierungsereignisse sollten nicht verwechselt werden. Auslöserereignisse treten unter anderen Bedingungen auf und generieren auch keine Ereignisnachrichten.

Weitere Informationen zu Instrumentierungsereignissen finden Sie im Abschnitt Instrumentierungsereignisse.

Warteschlangenmanager

Dieser Abschnitt enthält eine Einführung in *Warteschlangenmanager* und die Services für die Warteschlangensteuerung, die sie für die Anwendungen bereitstellen.

Damit ein Programm die Services eines Warteschlangenmanagers nutzen kann, muss es eine Verbindung zu diesem Warteschlangenmanager herstellen. Diese Verbindung kann vom Programm direkt (über den MQCONN- oder MQCONNX-Aufruf) oder indirekt (abhängig von der Plattform und der Umgebung, auf der bzw. in der das Programm aktiv ist) hergestellt werden.

Ein IBM MQ -Warteschlangenmanager stellt die folgenden Aktionen sicher:

- Objektattribute entsprechend den empfangenen Befehlen geändert werden;
- besondere Ereignisse wie Auslöser- oder Instrumentierungsereignisse generiert werden, wenn die entsprechenden Bedingungen erfüllt sind;

- Nachrichten entsprechend dem MQPUT-Aufruf einer Anwendung in die richtige Warteschlange eingereiht werden. Ist dies nicht möglich, wird die Anwendung unter Angabe des entsprechenden Ursachencodes darüber informiert.

Jede Warteschlange gehört nur zu einem Warteschlangenmanager und wird als *lokale Warteschlange* dieses Warteschlangenmanagers bezeichnet. Der Warteschlangenmanager, mit dem die Anwendung verbunden ist, wird als *lokaler Warteschlangenmanager* dieser Anwendung bezeichnet. Für die Anwendung sind die Warteschlangen ihres lokalen Warteschlangenmanagers lokale Warteschlangen.

Eine *ferne Warteschlange* ist eine Warteschlange, die zu einem anderen Warteschlangenmanager gehört. Als *fern* werden alle Warteschlangenmanager bezeichnet, bei denen es sich nicht um den lokalen Warteschlangenmanager handelt. Ein ferner Warteschlangenmanager kann sich auf einer fernen Maschine im Netz oder auf derselben Maschine wie der lokale Warteschlangenmanager befinden. IBM MQ unterstützt das Vorhandensein mehrerer Warteschlangenmanager auf ein und derselben Maschine.

In einigen MQI-Aufrufen können Warteschlangenmanagerobjekte verwendet werden. So können Sie beispielsweise mit dem MQI-Aufruf MQINQ die Attribute des Warteschlangenmanager-Objekts abfragen.

Attribute von Warteschlangenmanagern

Jedem Warteschlangenmanager sind eine Reihe von Attributen (oder Eigenschaften) zugeordnet, die diesen Warteschlangenmanager definieren.Einige dieser Attribute werden bei der Erstellung eines Warteschlangenmanagers festgelegt; andere Attribute hingegen können über IBM MQ-Befehle geändert werden. Mit Ausnahme der für die TLS-Verschlüsselung (Transport Layer Security) verwendeten Attribute können Sie die Werte aller Attribute mit dem MQINQ-Aufruf abfragen.

Zu den festgelegten Attributen gehören:

- Der Name des Warteschlangenmanagers.
- Die Plattform, auf der der Warteschlangenmanager aktiv ist (z. B. Windows).
- Die Systemsteuerbefehle, die der Warteschlangenmanager unterstützt.
- Die Priorität, die den vom Warteschlangenmanager verarbeiteten Nachrichten maximal zugeordnet werden kann.
- Der Name der Warteschlange, an die Programme IBM MQ-Befehle senden können.
- Die maximale Länge der Nachrichten, die der Warteschlangenmanager verarbeiten kann z/OS (nur in IBM MQ for z/OS festgelegt)
- Ob der Warteschlangenmanager beim Einreihen und Abrufen von Nachrichten Synchronisationspunkte unterstützt.

Zu den Attribute, die *geändert* werden können, gehören die folgenden:

- Eine Beschreibung des Warteschlangenmanagers im Textformat.
- Die ID des Zeichensatzes, den der Warteschlangenmanager bei der Verarbeitung von MQI-Aufrufen für Zeichenfolgen verwendet.
- Das Zeitintervall, mit dem der Warteschlangenmanager die Anzahl von Auslösenachrichten einschränkt.
- z/OS Das Zeitintervall, mit dem der Warteschlangenmanager vorgibt, wie oft Warteschlangen auf abgelaufene Nachrichten durchsucht werden sollen (nur in IBM MQ for z/OS).
- Der Name der Warteschlange für nicht zustellbare Nachrichten, die dem Warteschlangenmanager zugeordnet ist.
- Der Name der Übertragungswarteschlange des Warteschlangenmanagers.
- Die maximale Anzahl geöffneter Kennungen für jede Verbindung.
- Die Aktivierung und Inaktivierung verschiedener Kategorien von Ereignisberichten.
- Die maximale Anzahl nicht festgeschriebener Nachrichten innerhalb einer Arbeitseinheit.

Warteschlangenmanager und Workload-Management

Sie können einen Warteschlangenmanager-Cluster einrichten, für den mehrere Definitionen ein und derselben Warteschlange vorhanden sind (bei den Warteschlangenmanagern im Cluster kann es sich beispielsweise um Klone handeln). Nachrichten für eine bestimmte Warteschlange können von jedem Warteschlangenmanager verarbeitet werden, der über eine Instanz dieser Warteschlange verfügt. Mithilfe eines Workload-Management-Algorithmus wird der Warteschlangenmanager ermittelt, der die Nachricht verarbeiten soll. Auf diese Weise wird die Arbeitslast auf die einzelnen Warteschlangenmanager verteilt. Weitere Informationen hierzu finden Sie im Abschnitt Der Cluster-Workload-Management-Algorithmus.

Kanäle

Ein *Kanal* ist eine logische Kommunikationsverbindung, die von verteilten Warteschlangenmanagern verwendet wird, zwischen einem IBM MQ MQI client und einem IBM MQ-Server oder zwischen zwei IBM MQ-Servern.

Kanäle werden für die Übertragung von Nachrichten zwischen Warteschlangenmanagern verwendet und schirmen Anwendungen von den verwendeten Kommunikationsprotokollen ab. Die Warteschlangenmanager können sich dabei auf demselben oder einem anderen System auf derselben Plattform oder auf verschiedenen Plattformen befinden. Die gesendeten Nachrichten können aus vielen Quellen stammen:

- Vom Benutzer geschriebene Anwendungsprogramme, die Daten von einem Knoten an einen anderen übertragen
- Vom Benutzer geschriebene Verwaltungsanwendungen, die PCF-Befehle oder die MQAI verwenden
- Die IBM MQ Explorer.
- Warteschlangenmanager, die Instrumentierungsereignisnachrichten an einen anderen Warteschlangenmanager senden
- Warteschlangenmanager, die Fernverwaltungsbefehle an einen anderen Warteschlangenmanager senden (z. B. mit MQSC-Befehlen oder über die administrative REST API)

Ein Kanal verfügt über zwei Definitionen, nämlich je eine an jedem Ende der Verbindung. Damit Warteschlangenmanager miteinander kommunizieren können, müssen Sie einen Kanal auf der Seite des Warteschlangenmanagers definieren, der Nachrichten senden soll, sowie einen Kanal auf der Seite des Warteschlangenmanagers, der diese Nachrichten empfangen soll. An beiden Enden muss derselbe *Kanalname* verwendet werden und die *Kanaltypen* müssen kompatibel sein.

In IBM MQ gibt es drei Kanalkategorien mit jeweils verschiedenen Kanaltypen:

- Unidirektionale Nachrichtenkanäle, über die Nachrichten von einem Warteschlangenmanager an einen anderen gesendet werden.
- Bidirektionale MQI-Kanäle, über die MQI-Aufrufe von einem IBM MQ MQI client an einen Warteschlangenmanager und Antworten von einem Warteschlangenmanager an einen IBM MQ-Client übertragen werden.
- Bidirektionale AMQP-Kanäle, die einen AMQP-Client mit einem Warteschlangenmanager auf einer Servermaschine verbinden. IBM MQ verwendet AMQP-Kanäle, um AMQP-Aufrufe und -Antworten zwischen AMQP-Anwendungen und Warteschlangenmanagern zu übertragen.

Nachrichtenkanäle

Über Nachrichtenkanäle werden Nachrichten zwischen den Warteschlangenmanagern übertragen. In der Client/Server-Umgebung sind keine Nachrichtenkanäle erforderlich.

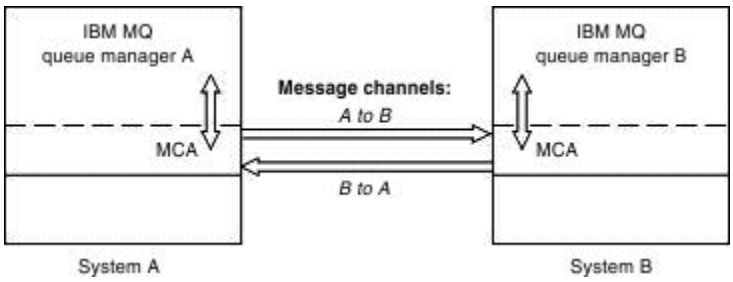


Abbildung 2. Nachrichtenkanäle zwischen zwei Warteschlangenmanagern

Ein Nachrichtenkanal ist eine unidirektionale Verbindung, Wenn ein ferner Warteschlangenmanager auf Nachrichten antworten soll, die von einem lokalen Warteschlangenmanager gesendet werden, müssen Sie einen zweiten Kanal einrichten, um Antworten zurück an den lokalen Warteschlangenmanager zu senden.

Ein Nachrichtenkanal verbindet zwei Warteschlangenmanager über *Nachrichtenkanalagenten*. An jedem Ende eines Kanals gibt es einen Nachrichtenkanalagenten. Sie können einem Nachrichtenkanalagenten ermöglichen, Nachrichten unter Verwendung mehrerer Threads zu übertragen. Dieser Prozess ist als *Pipelining* bekannt. Mithilfe von Pipelining kann der Nachrichtenkanalagent Nachrichten effizienter übertragen, was die Kanalleistung verbessert. Weitere Informationen zum Pipelining finden Sie im Abschnitt Attribute von Kanälen.

Weitere Informationen zu Kanälen finden Sie in den Abschnitten Kanalexitaufrufe und Datenstrukturen und „Komponenten der verteilten Steuerung von Warteschlangen” auf Seite 50.

MQI-Kanäle

Ein MQI-Kanal (Message Queue Interface) verbindet einen IBM MQ MQI client mit einem Warteschlangenmanager auf einer Servermaschine und wird eingerichtet, wenn Sie einen `MQCONN` -oder `MQCONNX` -Aufruf von einer IBM MQ MQI client -Anwendung absetzen.

MQI-Kanäle sind bidirektionale Verbindungen, die nur für die Übertragung von MQI-Aufrufen und -Antworten verwendet werden (z. B. `MQPUT`-Aufrufe, die Nachrichtendaten enthalten, und `MQGET`-Aufrufe, mit denen Nachrichtendaten zurückgegeben werden). Es gibt verschiedene Möglichkeiten, Kanaldefinitionen zu erstellen und zu verwenden (siehe MQI-Kanäle definieren).

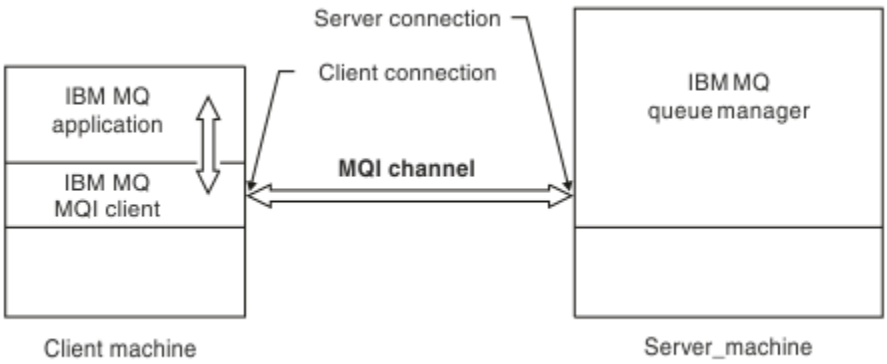


Abbildung 3. Clientverbindung und Serververbindung über einen MQI-Kanal

z/OS Mit einem MQI-Kanal kann ein Client mit einem einzelnen Warteschlangenmanager oder mit einem Warteschlangenmanager, der zu einer Gruppe mit gemeinsamer Warteschlange gehört, verbunden werden (siehe Client mit einer Gruppe mit gemeinsamer Warteschlange verbinden).

Es gibt zwei Kanaltypen für MQI-Kanaldefinitionen. Sie beschreiben den bidirektionalen MQI-Kanal.

Clientverbindungskanal
Dieser Typ ist für den IBM MQ MQI client.

Serververbindungskanal
Dieser Typ gilt für den Server, auf dem der Warteschlangenmanager ausgeführt wird, mit dem die IBM MQ -Anwendung in einer IBM MQ MQI client -Umgebung kommunizieren soll.

AMQP-Kanäle

Es gibt nur einen AMQP-Kanaltyp.

Über den Kanal wird eine AMQP-Messaging-Anwendung mit einem Warteschlangenmanager verbunden, damit die Anwendung Nachrichten mit IBM MQ-Anwendungen austauschen kann. Ein AMQP-Kanal ermöglicht es Ihnen, mithilfe von MQ Light eine Anwendung zu entwickeln, die dann als Unternehmensanwendung eingesetzt werden kann, um die auf Unternehmen abgestimmten Funktionen, die von IBM MQ bereitgestellt werden, zu nutzen.

Clientverbindungskanäle

Clientverbindungskanäle sind Objekte, die einen Kommunikationspfad von einem IBM MQ MQI client zu einem Warteschlangenmanager bereitstellen.

Clientkommunikationskanäle werden bei der verteilten Steuerung von Warteschlangen für die Übertragung von Nachrichten zwischen einem Warteschlangenmanager und einem Client verwendet. Sie schirmen Anwendungen von den verwendeten Kommunikationsprotokollen ab. Der Client kann sich auf derselben Plattform wie der Warteschlangenmanager oder auf einer anderen Plattform befinden.

Kanal- definitionen

Beschreibungen der einzelnen Kanaltypen finden Sie im Abschnitt „Kanal- definitionen” auf Seite 34.

Zugehörige Konzepte

„Verteilte Warteschlangen und Cluster” auf Seite 46
Durch die verteilte Steuerung von Warteschlangen können Nachrichten von einem Warteschlangenmanager an einen anderen gesendet werden. Der empfangende Warteschlangenmanager kann sich auf demselben System oder auf einem anderen System ganz in der Nähe oder am anderen Ende der Welt befinden. Er kann auf derselben Plattform wie der lokale Warteschlangenmanager oder jeder anderen von IBM MQ unterstützten Plattform aktiv sein. In einer Umgebung mit verteilter Steuerung von Warteschlangen können Sie alle Verbindungen manuell definieren oder Sie können einen Cluster erstellen und IBM MQ den größten Teil der Verbindungskonfiguration überlassen.

Übersicht über Message Queue Interface

Zugehörige Tasks

Ferne IBM MQ-Objekte verwalten

MQI-Kanäle stoppen

Verbindungen zwischen dem Server und dem Client konfigurieren

Zugehörige Verweise

Kanalexitaufrufe und Datenstrukturen

„Kommunikation” auf Seite 39
IBM MQ MQI clients kommunizieren über MQI-Kanäle mit dem Server.

Kanal- definitionen

Tabellen zur Beschreibung der verschiedenen Arten von Nachrichtenkanälen und MQI-Kanälen, die von IBM MQ verwendet werden.

Bei den Erläuterungen zu Nachrichtenkanälen wird das Wort Kanal gleichbedeutend mit dem Begriff Kanaldefinition verwendet. Aus dem Kontext geht im Allgemeinen hervor, ob von einem vollständigen Kanal (mit zwei Enden) oder von einer Kanaldefinition (die sich nur auf ein Kanalende bezieht) die Rede ist.

Nachrichtenkanäle

Nachrichtenkanaldefinitionen können zu einer der folgenden Arten gehören:

Art der Nachrichtenkanaldefinition	Beschreibung
Sender	Ein Senderkanal ist ein Nachrichtenkanal, über den ein Warteschlangenmanager Nachrichten an andere Warteschlangenmanager sendet. Damit Sie Nachrichten über einen Senderkanal senden können, müssen Sie auf dem anderen Warteschlangenmanager auch einen Empfangskanal mit demselben Namen wie der Senderkanal erstellen. Bei Verwendung eines Callback-Mechanismus kann der Senderkanal auch mit Requester-Kanälen verwendet werden.
Server	Ein Serverkanal ist ein Nachrichtenkanal, über den ein Warteschlangenmanager Nachrichten an andere Warteschlangenmanager sendet. Damit Sie Nachrichten über einen Serverkanal senden können, müssen Sie auf dem anderen Warteschlangenmanager auch einen Empfangskanal mit demselben Namen wie der Serverkanal erstellen. Sie können auch Serverkanäle in Verbindung mit Requester-Kanälen verwenden. In diesem Fall fordert die Requesterkanaldefinition am anderen Ende des Kanals die Serverkanaldefinition zum Start auf. Der Server sendet Nachrichten an den Requester. Auch der Server kann die Datenübertragung initiieren, sofern ihm der Verbindungsname des Partnerkanals bekannt ist.
Empfänger	Ein Empfängerkanal ist ein Nachrichtenkanal, über den ein Warteschlangenmanager Nachrichten von anderen Warteschlangenmanagern empfängt. Damit Sie Nachrichten über einen Empfangskanal empfangen können, müssen Sie auf dem anderen Warteschlangenmanager auch einen Senderkanal oder Serverkanal mit demselben Namen wie der Empfangskanal erstellen.

Art der Nachrichtenkanaldefinition	Beschreibung
Requester	Ein Requesterkanal ist ein Nachrichtenkanal, über den ein Warteschlangenmanager Nachrichten von anderen Warteschlangenmanagern empfängt. Ein Requesterkanal kann eine Startanforderung an den am fernen Ende definierten Partnerkanal senden. Wenn der Partnerkanal ein Serverkanal ist, akzeptiert der Serverkanal die Startanforderung und beginnt damit, Nachrichten aus der Übertragungswarteschlange, die in der Kanaldefinition des Servers angegeben ist, an den Requesterkanal zu senden. Wenn der Partnerkanal ein Senderkanal ist, akzeptiert der Senderkanal die Startanforderung, schließt aber dann die Verbindung mit dem Requester. Dann startet der Senderkanal, vereinbart eine Sitzung mit dem Partner-Requesterkanal und beginnt damit, Nachrichten aus der Übertragungswarteschlange zu senden, die in der Kanaldefinition des Senders angegeben ist. In diesem Fall wird im Wesentlichen ein Callback-Mechanismus bereitgestellt, indem der Requesterkanal eine Rückrufanforderung an den Senderkanal sendet.
Clustersender	Die Definition eines Clustersenderkanals (CLUSSDR) legt die Senderseite des Kanals fest, über den Clusterwarteschlangenmanager clusterspezifische Daten an eines der vollständigen Repositorys senden können. Der Clustersenderkanal wird dazu verwendet, das Repository über Änderungen am Status des Warteschlangenmanagers zu benachrichtigen, z. B. über das Hinzufügen oder Entfernen einer Warteschlange. Darüber hinaus wird dieser Kanal auch zur Nachrichtenübertragung verwendet. Die Warteschlangenmanager mit dem vollständigen Repository wiederum verfügen über Clustersenderkanäle, die auf den jeweils anderen Repository-Warteschlangenmanager verweisen. Über diese Kanäle informieren sie sich gegenseitig über Änderungen am Clusterstatus. Es ist unerheblich, auf welches vollständige Repository die Kanaldefinition 'CLUSSDR' eines Warteschlangenmanagers verweist. Nachdem der erste Kontakt hergestellt wurde, werden weitere Clusterwarteschlangenmanagerobjekte automatisch nach Bedarf definiert, sodass der Warteschlangenmanager Clusterinformationen an jedes vollständige Repository und Nachrichten an jeden Warteschlangenmanager senden kann.

Art der Nachrichtenkanaldefinition	Beschreibung
Clusterempfänger	Die Definition eines Clusterempfängerkanals (CLUSRCVR) legt die Empfängerseite des Kanals fest, über den Clusterwarteschlangenmanager Nachrichten von anderen Warteschlangenmanagern im Cluster empfangen können. Darüber hinaus können über Clusterempfängerkanäle auch Clusterdaten für die Repositorys übertragen werden. Indem der Warteschlangenmanager den Clusterempfängerkanal definiert, teilt er den anderen Warteschlangenmanagern dadurch mit, dass er für den Empfang von Nachrichten verfügbar ist. Für jeden Clusterwarteschlangenmanager ist mindestens ein Clusterempfängerkanal erforderlich.

Für jeden Kanal müssen beide Kanalenden definiert werden, sodass für jedes Kanalende eine eigene Definition vorhanden ist. Die beiden Kanalenden müssen vom Typ her miteinander kompatibel sein.

Folgende Kombinationen von Kanaldefinitionen sind möglich:

- Sender-Empfänger
- Server-Empfänger
- Requester-Server
- Requester-Sender (Callback)
- Clustersender-Clusterempfänger

Nachrichtenkanalagenten

Jede Kanaldefinition, die Sie erstellen, gehört zu einem bestimmten Warteschlangenmanager. Ein Warteschlangenmanager kann über mehrere Kanäle desselben Typs oder verschiedener Typen verfügen. An den beiden Kanalenden ist jeweils ein Programm installiert, der Nachrichtenkanalagent (MCA). Am einen Ende holt der aufrufende MCA Nachrichten aus der Übertragungswarteschlange und sendet diese über den Kanal. Am anderen Ende des Kanals empfängt der Responder-MCA die Nachrichten und leitet sie an den fernen Warteschlangenmanager weiter.

Ein aufrufender MCA kann einem Sende-, einem Server- oder einem Requesterkanal zugeordnet sein. Ein Responder-MCA kann jeder Nachrichtenkanalart zugeordnet werden.

In IBM MQ werden die folgenden Kombinationen von Kanaltypen an den beiden Verbindungsenden unterstützt:

Aufrufer		**Richtung des Nachrichtenflusses**	**Responder**	
Kanaltyp	**Empfangsprogramm erforderlich?**		**Empfangsprogramm erforderlich?**	**Kanaltyp**
Sender	Nein	Aufrufer an Responder	Ja	Empfänger
Server	Nein	Aufrufer an Responder	Ja	Empfänger
Server	Nein	Aufrufer an Responder	Ja	Requester
Requester	Nein	Responder an Aufrufer	Ja	Server

Aufrufer		Richtung des Nachrichtenflusses	Responder	
Kanaltyp	**Empfangsprogramm erforderlich?**		**Empfangsprogramm erforderlich?**	**Kanaltyp**
Requester	Ja	Responder an Aufrufer	Ja	Sender

MQI-Kanäle

MQI-Kanäle können zu einem der folgenden Kanaltypen gehören:

MQI-Kanaltyp	Beschreibung
Serververbindung	Ein Serververbindungskanal ist ein bidirektionaler MQI-Kanal, der eine Verbindung zwischen einem IBM MQ-Client und einem IBM MQ-Server herstellt. Dabei stellt der Serververbindungskanal das serverseitige Ende des Kanals dar.
Clientverbindung	Ein Clientverbindungskanal ist ein bidirektionaler MQI-Kanal, der eine Verbindung zwischen einem IBM MQ-Client und einem IBM MQ-Server herstellt. Im IBM MQ Explorer werden Clientverbindungen auch dazu verwendet, eine Verbindung zu fernen Warteschlangenmanagern herzustellen. Dabei stellt der Clientverbindungskanal das clientseitige Ende des Kanals dar. Beim Erstellen eines Clientverbindungskanals wird auf dem Computer, auf dem sich der Warteschlangenmanager befindet, eine Datei erstellt. Die Clientverbindungsdatei müssen Sie dann auf den IBM MQ-Clientcomputer kopieren.

Multi

Unterstützung mehrerer Threads — Pipelining

Sie haben die Option, einem Nachrichtenkanalagenten (MCA) die Übertragung von Nachrichten über mehrere Threads zu gestatten. Dieser Prozess, als *Pipelining* bezeichnet, ermöglicht dem MCA, Nachrichten effizienter mit weniger Wartestatus zu übertragen, wodurch sich die Kanalleistung verbessern lässt. Für jeden MCA können maximal zwei Threads ausgeführt werden.

Pipelining wird über den Parameter *PipeLineLength* in der Datei "qm.ini" gesteuert. Dieser Parameter wird der Zeilengruppe Channelshinzugefügt.

Anmerkung: Pipelining ist nur für TCP/IP-Kanäle effektiv.

Bei der Verwendung von Pipelining muss in der Konfiguration der Warteschlangenmanager an beiden Kanalenden der Parameter *PipeLineLength* auf einen größeren Wert als 1 festgelegt sein.

Überlegungen zu Kanalexits

Pipelining kann aus folgenden Gründen bei einigen Exitprogrammen zu Ausfällen führen:

- Exits wurden nicht seriell aufgerufen.
- Exits werden von unterschiedlichen Threads abwechselnd aufgerufen.

Überprüfen Sie die Struktur Ihres Exitprogramms, bevor Sie Pipelining nutzen:

- Exits müssen auf allen Ausführungsstufen wiedereintrittsfähig sein.
- Wenn Sie MQI-Aufrufe verwenden, achten Sie darauf, nicht die gleiche MQI-Kennung zu verwenden, wenn der Exit aus verschiedenen Threads aufgerufen wird.

Ziehen Sie einen Nachrichtenexit in Betracht, der eine Warteschlange öffnet und bei allen nachfolgenden Aufrufen des Exits dessen Kennung für MQPUT-Aufrufe angibt. Dies ist im Pipelining-Modus nicht möglich, da der Exit aus verschiedenen Threads aufgerufen wird. Um dies zu vermeiden, halten Sie eine Warteschlangenkennung für jeden Thread bereit, und überprüfen Sie bei jedem Aufruf des Exits die Thread-ID.

Kommunikation

IBM MQ MQI clients kommunizieren über MQI-Kanäle mit dem Server.

Auf der IBM MQ MQI client- und -Serverseite der Verbindung muss jeweils eine Kanaldefinition erstellt werden. Informationen zum Erstellen von Kanaldefinitionen finden Sie im Abschnitt MQI-Kanäle definieren.

In der folgenden Tabelle sind die möglichen Übertragungsprotokolle aufgeführt:

Tabelle 1. Übertragungsprotokolle für MQI-Kanäle

Clientplattform	LU 6.2	TCP/IP	NetBIOS	SPX
IBM i		Ja		
Linux AIX and Linux-Systeme	Ja[1]	Ja		
Windows	Ja	Ja	Ja	Ja

Anmerkung:

1. Linux LU6.2 wird auf folgenden Plattformen nicht unterstützt:
   - Linux (POWER-Plattform)
   - Linux (x86-64-Plattform)
   - Linux (zSeries s390x-Plattform)

In Übertragungsprotokolle - Kombination von IBM MQ MQI client- und Serverplattformen sind die möglichen Kombinationen von IBM MQ MQI client- und Serverplattformen mit diesen Übertragungsprotokollen aufgeführt.

Eine IBM MQ-Anwendung auf einem IBM MQ MQI client kann alle MQI-Aufrufe so verwenden, als ob ein lokaler Warteschlangenmanager verwendet wird. Der **MQCONN**- oder **MQCONNX**-Aufruf ordnet die IBM MQ-Anwendung dem ausgewählten Warteschlangenmanager zu und erstellt auf diese Weise eine *Verbindungskennung*. Alle weiteren Aufrufe, die diese Verbindungskennung verwenden, werden daraufhin von dem verbundenen Warteschlangenmanager verarbeitet. Im Unterschied zur Kommunikation der Warteschlangenmanager untereinander, die verbindungs- und zeitunabhängig erfolgt, ist für die IBM MQ MQI clientkommunikation eine aktive Verbindung zwischen Client und Server erforderlich.

Das Übertragungsprotokoll wird über die Verwendung der Kanaldefinition festgelegt und hat keine Auswirkung auf die Anwendung. So kann beispielsweise eine Windows-Anwendung zu einem Warteschlangenmanager eine Verbindung über TCP/IP und zu einem anderen über NetBIOS herstellen.

Leistungsaspekte

Das von Ihnen verwendete Übertragungsprotokoll kann sich auf die Leistung des IBM MQ-Client/Server-Systems auswirken. In bestimmten Situationen, in denen die Übertragung langsam ist, können Sie die IBM MQ -Kanalkomprimierung verwenden.

IBM MQ-Objekte benennen

Die Namenskonvention für die Benennung von IBM MQ-Objekten hängt vom jeweiligen Objekt ab. Auch für die Namen der Maschinen und die Benutzer-IDs für IBM MQ gelten einige Einschränkungen.

Jede Instanz eines Warteschlangenmanagers hat einen eigenen Namen. Dieser Name muss im Netz bzw. unter den verbundenen Warteschlangenmanagern eindeutig sein, damit ein Warteschlangenmanager eindeutig den Ziel-Warteschlangenmanager angeben kann, an den eine Nachricht gesendet werden soll.

Bei den anderen Objekttypen ist jedem Objekt ein Name zugewiesen, über den es identifiziert werden kann. Diese Namen müssen innerhalb eines Warteschlangenmanagers und innerhalb eines Objekttyps eindeutig sein. So können beispielsweise eine Warteschlange und ein Prozess denselben Namen haben, nicht jedoch zwei Warteschlangen.

In IBM MQ können Namen eine Länge von bis zu 48 Zeichen haben; ausgenommen hiervon sind *Kanalnamen*, die maximal eine Länge von 20 Zeichen haben können. Weitere Informationen zur Benennung von IBM MQ-Objekten finden Sie im Abschnitt „Regeln für die Benennung von IBM MQ-Objekten" auf Seite 40.

Für die Namen der Maschinen und die Benutzer-IDs für IBM MQ gelten ebenfalls einige Einschränkungen:

- Achten Sie darauf, dass der Maschinenname keine Leerzeichen enthält. Maschinennamen mit Leerzeichen werden von IBM MQ nicht unterstützt. Wenn Sie IBM MQ auf eine Maschine mit einem solchen Namen installieren, können Sie keine Warteschlangenmanager erstellen.
- Bei IBM MQ-Berechtigungen dürfen die Namen der Benutzer-IDs und -Gruppen nicht länger als 20 Zeichen sein (Leerzeichen sind nicht zulässig).
- Windows Von IBM MQ for Windows-Servern wird die Verbindung eines IBM MQ MQI clients nicht unterstützt, wenn der Client unter einer Benutzer-ID ausgeführt wird, die ein kommerzielles A (@) enthält (z. B. abc@d).

Zugehörige Konzepte

„IBM MQ-Dateinamen" auf Seite 44
Die einzelnen Warteschlangenmanager, Warteschlangen, Prozessdefinitionen, Namenslisten, Kanäle, Clientverbindungskanäle, Empfangsprogramme, Services und Authentifizierungsdatenobjekte in IBM MQ werden jeweils durch eine Datei dargestellt. Da Objektnamen nicht notwendigerweise auch gültige Dateinamen sind, wandelt der Warteschlangenmanager die Objektnamen bei Bedarf in gültige Dateinamen um.

Zugehörige Verweise

„Regeln für die Benennung von IBM MQ-Objekten" auf Seite 40
Für die Namen von IBM MQ-Objekten gelten Längeneinschränkungen. Außerdem wird bei diesen Namen die Groß-/Kleinschreibung beachtet. Nicht alle Zeichen werden für jeden Objekttyp unterstützt, und bei vielen Objekten gelten Regeln hinsichtlich der Eindeutigkeit der Namen.

Regeln für die Benennung von IBM MQ-Objekten

Für die Namen von IBM MQ-Objekten gelten Längeneinschränkungen. Außerdem wird bei diesen Namen die Groß-/Kleinschreibung beachtet. Nicht alle Zeichen werden für jeden Objekttyp unterstützt, und bei vielen Objekten gelten Regeln hinsichtlich der Eindeutigkeit der Namen.

Es gibt viele verschiedene Arten von IBM MQ-Objekten, wobei Objekte unterschiedlicher Typen gleiche Namen haben können, da sie in getrennten Namensbereichen verwaltet werden. Eine lokale Warteschlange und ein Senderkanal können zum Beispiel den gleichen Namen haben. Ein Objekt darf jedoch nicht den Namen eines anderen Objekts des gleichen Namensbereichs haben. Eine lokale Warteschlange darf zum Beispiel nicht den gleichen Namen wie eine Modellwarteschlange haben, und ein Senderkanal darf nicht den gleichen Namen wie ein Empfängerkanal haben.

Die folgenden IBM MQ-Objekte liegen in separaten Namensbereichen vor:

- Authentifizierungsdaten
- Kanal
- Clientkanal
- Empfangsprogramm
- Namensliste
- Prozess

- Warteschlange
- Service
- Speicherklasse
- Abonnement
- Thema

Zeichenlänge von Objektnamen

Im Allgemeinen dürfen die Namen von IBM MQ-Objekten bis zu 48 Zeichen lang sein. Diese Regel gilt für die folgenden Objekte:

- Authentifizierungsdaten
- Cluster
- Empfangsprogramm
- Namensliste
- Prozessdefinition
- Warteschlange
- Warteschlangenmanager
- Service
- Abonnement
- Thema

Allerdings gibt es hierzu auch Einschränkungen:

1. z/OS Unter z/OS müssen die Namen von Warteschlangenmanagern mindestens 4 Zeichen lang sein. Sie dürfen nur Großbuchstaben und numerische Zeichen enthalten.
2. Die Namen von Kanalobjekten und Clientverbindungskanälen dürfen maximal 20 Zeichen lang sein. Weitere Informationen zu Kanälen finden Sie im Abschnitt Kanäle definieren.
3. Themenzeichenfolgen dürfen maximal 10240 Byte lang sein. Bei allen Namen von IBM MQ-Objekten wird die Groß-/Kleinschreibung beachtet.
4. Subskriptionsnamen dürfen maximal 10240 Byte lang sein und können Leerzeichen enthalten.
5. Die Namen von Speicherklassen dürfen maximal 8 Zeichen lang sein.
6. Die Namen von Coupling-Facility-(CF-)Strukturen dürfen maximal 12 Zeichen lang sein.

Zeichen in Objektnamen

Für die Namen von IBM MQ-Objekten sind folgende Zeichen gültig:

Zeichen	Einschränkungen
Großbuchstaben (A - Z)	• --

Zeichen	Einschränkungen
Kleinbuchstaben (a - z)	• In MQSC-Scripts müssen Namen mit Kleinbuchstaben in einfache Anführungszeichen eingeschlossen werden, da die Kleinbuchstaben ansonsten in Großbuchstaben umgewandelt werden. • Systeme, auf denen EBCDIC Katakana verwendet wird, unterstützen in Objektnamen keine Kleinbuchstaben. • z/OS Unter z/OS gelten für Kleinbuchstaben gewisse Einschränkungen. Die Namen von Warteschlangenmanagern dürfen zum Beispiel keine Kleinbuchstaben enthalten. • IBM i Auf IBM i-Systemen müssen in CL-Befehlen Namen mit Kleinbuchstaben in einfache Anführungszeichen eingeschlossen werden, da die Kleinbuchstaben ansonsten in Großbuchstaben umgewandelt werden.
Numerische Zeichen (0 - 9)	• --
Punkt (.)	• --
Unterstrich (_)	• Multi -- • z/OS Vermeiden Sie Namen mit führenden oder abschließenden Unterstrichen, da diese von den Bedien- und Steuerkonsolen von IBM MQ for z/OS nicht verarbeitet werden können.
Schrägstrich (/)	• Windows Unter Windows darf das erste Zeichen eines Warteschlangenmanagernamens kein Schrägstrich sein. • IBM i Auf IBM i-Systemen müssen in CL-Befehlen Namen mit Schrägstrichen in einfache Anführungszeichen eingeschlossen werden. • z/OS --
Prozentzeichen (%)	• ALW -- • z/OS Wenn Sie RACF als externen Sicherheitsmanager für IBM MQ for z/OS verwenden, geben Sie in Objekten nicht das Prozentzeichen an, da Objektnamen mit Prozentzeichen bei Verwenden der generischen Profile von RACF nicht in Sicherheitsprüfungen eingeschlossen werden. • IBM i Auf IBM i-Systemen müssen in CL-Befehlen Namen mit Prozentzeichen in einfache Anführungszeichen eingeschlossen werden.

Darüber hinaus gelten für die in Objektnamen verwendeten Zeichen einige generelle Regeln:

1. Führende oder eingebettete Leerzeichen sind nicht erlaubt.
2. Landessprachliche Zeichen sind nicht erlaubt.
3. Namen, die kürzer als die Feldlänge sind, können am Ende mit Leerzeichen aufgefüllt werden. Alle zu kurzen Namen, die vom Warteschlangenmanager zurückgegeben werden, sind am Ende mit Leerzeichen aufgefüllt.

Warteschlangennamen

Der Name einer Warteschlange besteht aus zwei Teilen:

- Der Name eines Warteschlangenmanagers
- Aus dem lokalen Namen der Warteschlange, wie er dem Warteschlangenmanager bekannt ist

Jeder Teil des Warteschlangennamens ist 48 Zeichen lang.

Bei der Angabe einer lokalen Warteschlange können Sie den Namen des Warteschlangenmanagers weglassen (durch Ersetzen seines Namens durch Leerzeichen oder durch eine führende Null). Alle Warteschlangennamen, die von IBM MQ an ein Programm zurückgegeben werden, enthalten jedoch den Namen des Warteschlangenmanagers.

z/OS Eine gemeinsam genutzte Warteschlange, die für jeden Warteschlangenmanager der jeweiligen Gruppe mit gemeinsamer Warteschlange zugänglich ist, darf nicht den gleichen Namen haben wie eine nicht gemeinsam genutzte lokale Warteschlange der gleichen Gruppe mit gemeinsamer Warteschlange. Dadurch wird verhindert, dass eine Anwendung versehentlich eine gemeinsam genutzte Warteschlange öffnet, wenn sie eigentlich eine lokale Warteschlange öffnen sollte, und umgekehrt. Gemeinsam genutzte Warteschlangen und Gruppen mit gemeinsamer Warteschlange sind nur in IBM MQ for z/OS verfügbar.

Zur Angabe einer fernen Warteschlange muss ein Programm im vollständigen Warteschlangennamen den Namen des Warteschlangenmanagers angeben (es sei denn, es liegt eine lokale Definition der fernen Warteschlange vor).

Der von einer Anwendung angegebene Warteschlangenname kann der Name (oder der Aliasname) einer lokalen Warteschlange oder der Name einer lokalen Definition einer fernen Warteschlange sein. Die Anwendung selbst muss nicht wissen, um welche Art von Warteschlange es sich handelt, es sei denn, sie erwartet eine Nachricht aus dieser Warteschlange (in diesem Fall muss es sich um eine lokale Warteschlange handeln). Wenn die Anwendung das Warteschlangenobjekt öffnet, löst der Aufruf MQOPEN den Namen auf, wodurch festgestellt wird, an welcher Warteschlange die nachfolgenden Operationen durchgeführt werden sollen. Dieser Mechanismus ist insofern signifikant, als dass die Anwendung inhärent nicht von bestimmten Warteschlangen abhängig ist, die an bestimmten Orten im Netz der Warteschlangenmanager definiert sind. Falls ein Systemadministrator die Warteschlangen im Netz umstellt und deren Definitionen ändert, hat dies den Vorteil, dass die Anwendungen, die auf diese Warteschlangen zugreifen, nicht geändert werden müssen.

Reservierte Objektnamen

Objektnamen, die mit SYSTEM. beginnen, sind für vom Warteschlangenmanager definierte Objekte reserviert. Mit den Befehlen **Alter**, **Define** und **Replace** können Sie diese Objektdefinitionen an Ihre Installation anpassen. Eine vollständige Liste der für IBM MQ definierten Namen finden Sie im Abschnitt Warteschlangennamen.

z/OS Unter IBM MQ for z/OS ist der Name der Coupling-Facility-Anwendungsstruktur CSQSYSAPPL reserviert.

Zugehörige Konzepte

Installationsname unter AIX, Linux, and Windows

IBM MQ-Dateinamen

Die einzelnen Warteschlangenmanager, Warteschlangen, Prozessdefinitionen, Namenslisten, Kanäle, Clientverbindungskanäle, Empfangsprogramme, Services und Authentifizierungsdatenobjekte in IBM MQ werden jeweils durch eine Datei dargestellt. Da Objektnamen nicht notwendigerweise auch gültige Dateinamen sind, wandelt der Warteschlangenmanager die Objektnamen bei Bedarf in gültige Dateinamen um.

Der Standardpfad eines Warteschlangenmanager-Verzeichnisses setzt sich wie folgt zusammen:

- Ein Präfix, das in den IBM MQ-Konfigurationsdaten definiert wird:
  - Linux AIX Unter AIX and Linux ist `/var/mqm` das Standardpräfix. Es wird in der Zeilengruppe `DefaultPrefix` der Konfigurationsdatei 'mqs.ini' konfiguriert.
  - Windows Auf Windows-32-Bit-Systemen ist `C:\Programme (x86)\IBM\WebSphere MQ` das Standardpräfix. Auf Windows-64-Bit-Systemen ist `C:\Programme\IBM\MQ` das Standardpräfix. Sowohl bei 32-Bit- als auch bei 64-Bit-Installationen befinden sich die Datenverzeichnisse unter `C:\ProgramData \IBM \MQ`. Es wird in der Zeilengruppe `DefaultPrefix` der Konfigurationsdatei 'mqs.ini' konfiguriert.

  Sofern verfügbar, kann das Präfix über die IBM MQ-Eigenschaftenseite in IBM MQ Explorer geändert werden. Andernfalls müssen Sie die Konfigurationsdatei `mqs.ini` manuell bearbeiten.
- Der Name des Warteschlangenmanagers wird in einen gültigen Verzeichnisnamen umgesetzt. Der Warteschlangenmanager

  ```
  queue.manager
  ```

  beispielsweise wird dargestellt als

  ```
  queue!manager
  ```

  Dieser Vorgang wird als *Namensumsetzung* bezeichnet.

In IBM MQ können Sie Warteschlangenmanagern Namen mit bis zu 48 Zeichen zuweisen.

Beispielsweise kann ein Warteschlangenmanager den folgenden Namen erhalten:

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

Jeder Warteschlangenmanager wird jedoch anhand einer Datei dargestellt; für Dateinamen bestehen sowohl Längenbeschränkungen als auch Vorgaben für die Zeichen, die verwendet werden können. Daher werden die Namen der Dateien für Objekte automatisch entsprechend den Anforderungen des Dateisystems umgewandelt.

Für die Umsetzung von Warteschlangenmanager-Namen gelten die folgenden Regeln:

1. Umsetzung einzelner Zeichen:
   - Von . !
   - Von / in &
2. Ist das Ergebnis immer noch kein gültiger Name:
   a. Begrenzen Sie die Länge des Namens auf acht Zeichen.
   b. Hängen Sie ein dreistelliges numerisches Suffix an.

Wird beispielsweise das Standardpräfix verwendet und lautet der Warteschlangenmanager-Name `queue.manager`, gilt Folgendes:

- Windows Unter Windows mit NTFS oder FAT32 wird der Warteschlangenmanagername wie folgt umgesetzt:

```
C:\Programme\IBM\MQ\qmgrs\queue!manager
```

- Windows Unter Windows mit FAT wird der Warteschlangenmanagername wie folgt umgesetzt:

```
C:\Programme\IBM\MQ\qmgrs\queue!ma
```

- Linux AIX Unter AIX and Linux wird der Warteschlangenmanagername wie folgt umgesetzt:

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

Der Umsetzungsalgorithmus unterscheidet auch in Dateisystemen, bei denen die Groß-/Kleinschreibung keine Rolle spielt, zwischen Dateinamen, die sich nur in der Groß-/Kleinschreibung unterscheiden.

Objektnamensumsetzung

Bei Objektnamen handelt es sich nicht notwendigerweise um gültige Dateisystemnamen. Daher müssen Sie Objektnamen unter Umständen umwandeln. Die Vorgehensweise unterscheidet sich von der Umsetzung von Warteschlangenmanager-Namen, da auf jeder Maschine nur wenige Warteschlangenmanager-Namen, jedoch unter Umständen eine große Anzahl von Objekten pro Warteschlangenmanager vorhanden sind. Warteschlangen, Prozessdefinitionen, Namenslisten, Kanäle, Clientverbindungskanäle, Empfangsprogramme, Services und Authentifizierungsdatenobjekte sind im Dateisystem dargestellt.

Die vom Umsetzungsprozess generierten neuen Namen sind keine direkte Entsprechung des ursprünglichen Objektnamens. Mit dem Befehl **dspmqfls** können die ursprünglichen Namen umgesetzter Objektnamen angezeigt werden.

Zugehörige Verweise

dspmqfls (Dateinamen anzeigen)

Zugehörige Informationen

Zeilengruppe 'AllQueueManagers' in der Datei 'mqs.ini'

IBM i Objektnamen unter IBM i

Einem Warteschlangenmanager ist eine Warteschlangenmanagerbibliothek mit einem eindeutigen Namen zugeordnet. Warteschlangenmanagernamen und Objektnamen müssen gegebenenfalls umgewandelt werden, um den Anforderungen des IBM i Integrated File System (IFS) zu entsprechen.

Wenn ein Warteschlangenmanager erstellt wird, ordnet IBM MQ ihm eine Warteschlangenmanagerbibliothek zu. Diese Warteschlangenmanagerbibliothek erhält einen eindeutigen Namen aus maximal zehn Zeichen, der überwiegend auf dem benutzerdefinierten Namen des Warteschlangenmanagers basiert. Sowohl der Warteschlangenmanager als auch die Warteschlangenmanagerbibliothek werden in einem Verzeichnis abgelegt, das ebenfalls auf dem Namen des Warteschlangenmanagers basiert und das Präfix `/QIBM/UserData/mqm` besitzt. Es folgt ein Beispiel für einen Warteschlangenmanager, eine Warteschlangenmanagerbibliothek und ein Verzeichnis:

Name des Warteschlangenmanagers	ORANGE
Name der Warteschlangenmanagerbibliothek	QMORANGE
Directory	`/QIBM/UserData/mqm/ORANGE`

Alle Namen von Warteschlangenmanagern und Warteschlangenmanagerbibliotheken werden in Zeilengruppen in der Datei `/QIBM/UserData/mqm/mqs.ini` geschrieben.

IBM MQ-IFS-Verzeichnisse und -Dateien

Das integrierte Dateisystem (IFS) von IBM i wird von IBM MQ ausgiebig zum Speichern von Daten verwendet. Weitere Informationen zum integrierten Dateisystem finden Sie in der *Einführung zum integrierten Dateisystem*.

Jedes IBM MQ-Objekt, z. B. ein Kanal oder Warteschlangenmanager, wird durch eine Datei dargestellt. Da Objektnamen nicht notwendigerweise auch gültige Dateinamen sind, wandelt der Warteschlangenmanager die Objektnamen bei Bedarf in gültige Dateinamen um.

Der Pfad zu einem Warteschlangenmanager-Verzeichnis wird aus folgenden Teilen gebildet:

- Einem Präfix, das in der Konfigurationsdatei des Warteschlangenmanagers `qm.ini` definiert ist. Das Standardpräfix lautet `/QIBM/UserData/mqm`.
- Einem Literal (`qmgrs`).
- Einem codierten Warteschlangenmanager-Namen, bei dem es sich um den Warteschlangenmanager-Namen handelt, der in einen gültigen Verzeichnisnamen umgesetzt wurde. Beispielsweise wird der Warteschlangenmanager `queue/manager` durch `queue&manager` dargestellt.

Dieser Prozess wird als Namensumsetzung bezeichnet.

Warteschlangenmanager-Namensumsetzung im integrierten Dateisystem

In IBM MQ können Sie Warteschlangenmanagern Namen mit bis zu 48 Zeichen zuweisen.

Sie können einen Warteschlangenmanager beispielsweise `QUEUE/MANAGER/ACCOUNTING/SERVICES` nennen. Ebenso wie für jeden Warteschlangenmanager eine Bibliothek erstellt wird, wird jeder Warteschlangenmanager auch durch eine Datei dargestellt. Aufgrund unterschiedlicher Codepunkte in EBCDIC gibt es Einschränkungen hinsichtlich der Zeichen, die im Namen verwendet werden können. Dies hat zur Folge, dass Namen von Dateien des integrierten Dateisystems, die Objekte darstellen, automatisch umgesetzt werden, damit die Anforderungen des Dateisystems erfüllt werden.

Wenn Sie das Beispiel eines Warteschlangenmanagers mit dem Namen `queue/manager`verwenden, das Zeichen / in &umwandeln und das Standardpräfix annehmen, wird der Warteschlangenmanagername in IBM MQ for IBM i zu `/QIBM/UserData/mqm/qmgrs/queue&manager`.

Objektnamensumsetzung

Objektnamen sind nicht zwangsläufig auch gültige Dateisystemnamen, sodass sie möglicherweise umgesetzt werden müssen. Die zu diesem Zweck verwendete Methode unterscheidet sich von der für Warteschlangenmanager-Namen, weil es trotz der Tatsache, dass es für jedes System nur wenige Warteschlangenmanager-Namen gibt, für jeden Warteschlangenmanager eine große Zahl von anderen Objekten geben kann. Es werden nur Prozessdefinitionen, Warteschlangen und Namenslisten im Dateisystem dargestellt; für Kanäle gelten diese Hinweise nicht.

Die vom Umsetzungsprozess generierten neuen Namen sind keine direkte Entsprechung des ursprünglichen Objektnamens. Mit dem Befehl DSPMQMOBJN können Sie die umgesetzten Namen für IBM MQ-Objekte anzeigen.

Verteilte Warteschlangen und Cluster

Durch die verteilte Steuerung von Warteschlangen können Nachrichten von einem Warteschlangenmanager an einen anderen gesendet werden. Der empfangende Warteschlangenmanager kann sich auf demselben System oder auf einem anderen System ganz in der Nähe oder am anderen Ende der Welt befinden. Er kann auf derselben Plattform wie der lokale Warteschlangenmanager oder jeder anderen von IBM MQ unterstützten Plattform aktiv sein. In einer Umgebung mit verteilter Steuerung von Warteschlangen können Sie alle Verbindungen manuell definieren oder Sie können einen Cluster erstellen und IBM MQ den größten Teil der Verbindungskonfiguration überlassen.

Verteilte Steuerung von Warteschlangen

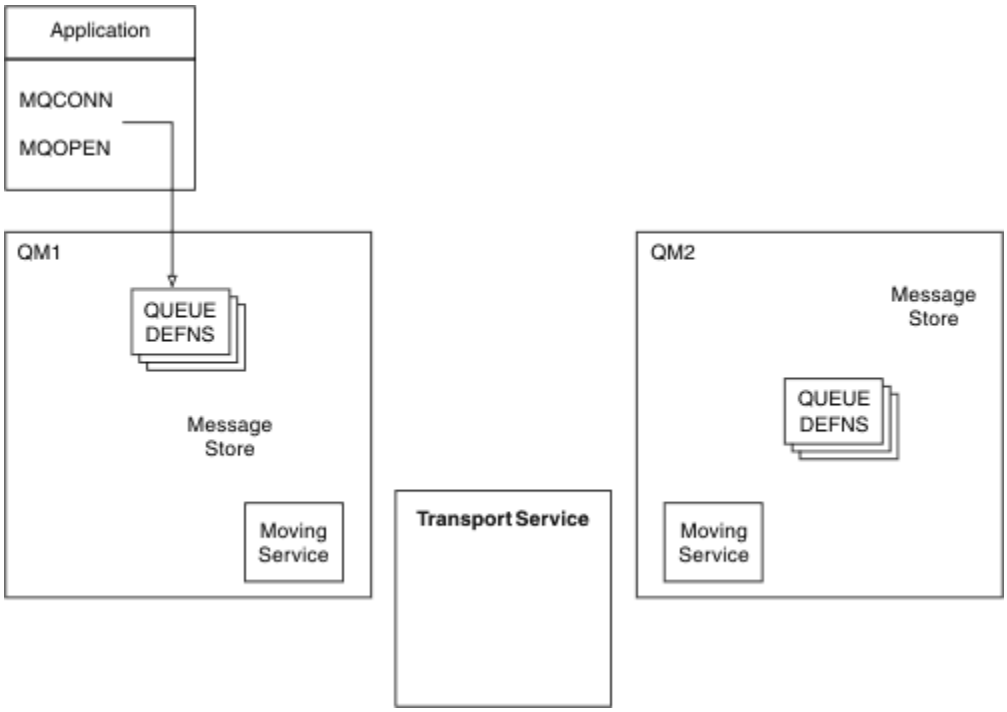


Abbildung 4. Übersicht über die Komponenten der verteilten Steuerung von Warteschlangen

In der vorherigen Abbildung:

- Eine Anwendung stellt mit dem Aufruf MQCONN eine Verbindung mit einem Warteschlangenmanager her. Danach öffnet die Anwendung mit dem Aufruf MQOPEN eine Warteschlange, sodass sie Nachrichten darin einreihen kann.
- Jeder Warteschlangenmanager besitzt für jede seiner Warteschlangen eine Definition. Dies können Definitionen *lokaler Warteschlangen* (von diesem Warteschlangenmanager bereitgestellt) und Definitionen *ferner Warteschlangen* (von anderen Warteschlangenmanagern bereitgestellt) sein.
- Wenn die Nachrichten für eine ferne Warteschlange bestimmt sind, behält der lokale Warteschlangenmanager sie in einer *Übertragungswarteschlange*, die die Nachrichten in einem Nachrichtenspeicher speichert, bis sie an den fernen Warteschlangenmanager weitergeleitet werden können.
- Jeder Warteschlangenmanager enthält eine Datenübertragungssoftware, den sogenannten *Übertragungsservice*, über den der Warteschlangenmanager mit anderen Warteschlangenmanagern kommuniziert.
- Der *Transportservice* ist vom Warteschlangenmanager unabhängig, sodass es sich um jeden der folgenden Services (je nach Plattform) handeln kann:
  - Systems Network Architecture Advanced Program-to Program Communication (SNA APPC)
  - Transmission Control Protocol/Internet Protocol (TCP/IP)
  - Network Basic Input/Output System (NetBIOS)
  - Sequenced Packet Exchange (SPX)

Komponenten, die zum Senden einer Nachricht erforderlich sind

Um eine Nachricht an einen fernen Warteschlangenmanager senden zu können, benötigt der lokale Warteschlangenmanager Definitionen für eine *Übertragungswarteschlange* und einen *Kanal*. Ein Kanal ist eine einseitige Übertragungsverbindung zwischen zwei Warteschlangenmanagern. In ihm können Nachrichten, die für beliebig viele Warteschlangen des fernen Warteschlangenmanagers bestimmt sind, übertragen werden.

Es gibt für jedes Ende eines Kanals eine separate Definition, in der es zum Beispiel als Sendeseite oder Empfangsseite definiert ist. Ein einfacher Kanal besteht aus einer *Sender*kanaldefinition beim lokalen

Warteschlangenmanager und einer *Empfänger*kanaldefinition beim fernen Warteschlangenmanager. Diese beiden Definitionen müssen den gleichen Namen haben und gemeinsam einen einzigen Kanal bilden.

Die Software, die das Senden und Empfangen von Nachrichten ausführt, wird als *Nachrichtenkanalagent* bezeichnet. An jedem Ende eines Kanals befindet sich ein *Nachrichtenkanalagent*.

Jeder Warteschlangenmanager muss eine *Warteschlange für nicht zustellbare Nachrichten* (auch *Warteschlange für nicht zugestellte Nachrichten* genannt) besitzen. In diese Warteschlange werden Nachrichten gestellt, die nicht an ihre Zieladresse übermittelt werden können.

Die folgende Abbildung zeigt die Beziehung zwischen Warteschlangenmanagern, Übertragungswarteschlangen, Kanäle und Nachrichtenkanalagenten:

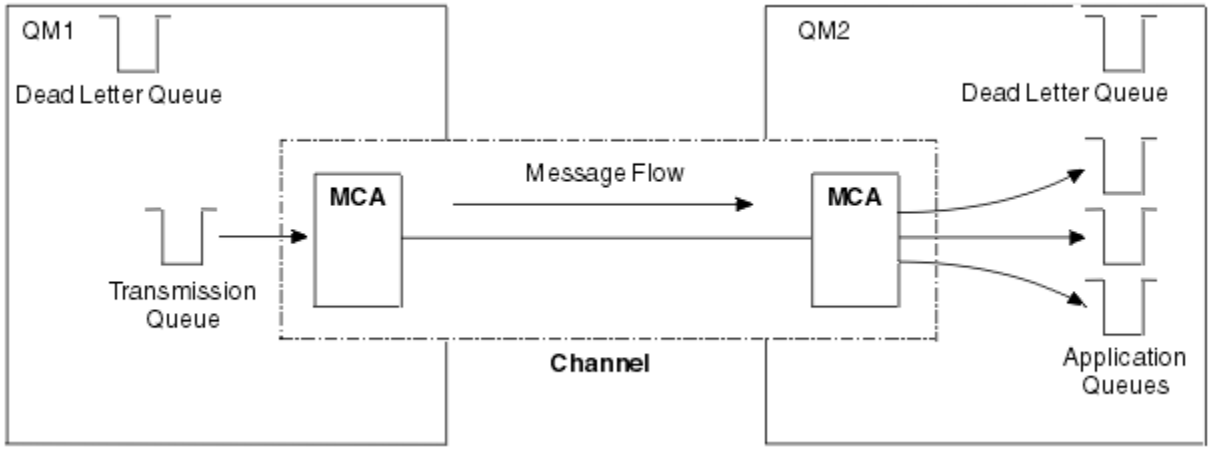


Abbildung 5. Nachrichten senden

Komponenten, die zur Rückgabe einer Nachricht erforderlich sind

Wenn Ihre Anwendung erfordert, dass Nachrichten vom fernen Warteschlangenmanager zurückgegeben werden, müssen Sie einen weiteren Kanal definieren, der, wie in der Abbildung gezeigt, in entgegengesetzter Richtung zwischen den Warteschlangenmanagern verläuft:

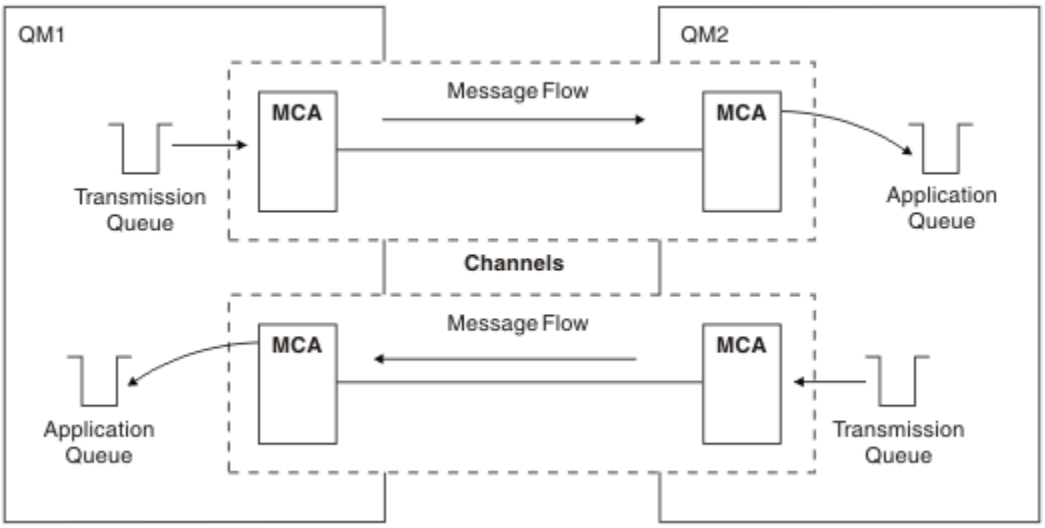


Abbildung 6. Nachrichten in beiden Richtungen senden

Cluster

Anstatt die Verbindungen in einer Umgebung mit verteilter Steuerung von Warteschlangen manuell zu definieren, können Sie auch mehrere Warteschlangenmanager zu einem Cluster zusammenfassen. In diesem Fall können die Warteschlangenmanager die von ihnen bereitgestellten Warteschlangen auch

anderen Warteschlangenmanagern im Cluster zur Verfügung stellen, ohne dass für jedes Ziel explizite Kanaldefinitionen, Definitionen ferner Warteschlangen oder Übertragungswarteschlangen vorhanden sein müssen. Jeder Warteschlangenmanager in einem Cluster besitzt eine einzelne Übertragungswarteschlange, um Nachrichten an einen anderen Warteschlangenmanager im Cluster zu übertragen. Sie müssen dann für jeden Warteschlangenmanager nur einen Clusterempfängerkanal und einen Clustersenderkanal definieren; sämtliche weiteren Kanäle werden automatisch vom Cluster verwaltet.

Die Verbindung eines IBM MQ-Clients mit einem Warteschlangenmanager eines Clusters erfolgt auf die gleiche Weise wie bei jedem anderen Warteschlangenmanager. Wie bei der manuell konfigurierten verteilten Steuerung von Warteschlangen wird der Aufruf MQPUT verwendet, um eine Nachricht in eine Warteschlange eines anderen Warteschlangenmanagers zu stellen. Mit dem Aufruf MQGET können Nachrichten aus einer lokalen Warteschlange abgerufen werden.

Warteschlangenmanager auf Plattformen, die Cluster unterstützen, müssen jedoch kein Teil eines Clusters sein. Neben oder anstatt der Verwendung von Clustern können Sie nach wie vor manuell Techniken für die verteilte Steuerung von Warteschlangen konfigurieren und verwenden.

Vorteile bei der Verwendung von Clustern

Cluster bieten zwei wichtige Vorteile:

- Cluster vereinfachen die Verwaltung von IBM MQ-Netzen, für die normalerweise unzählige Objektdefinitionen für Kanäle, Übertragungswarteschlangen und ferne Warteschlangen konfiguriert werden müssen. Besonders gravierend ist dieser Aufwand in großen Netzen, die sich häufig ändern und in denen die Verbindung vieler Warteschlangenmanager erforderlich ist. Bei einer solchen Architektur ist die Konfiguration und Verwaltung ohne Cluster besonders anspruchsvoll.
- In Clustern kann die Arbeitslast der Nachrichtenübertragung über mehrere Warteschlangen und Warteschlangenmanager verteilt werden. Eine solche Verteilung sieht die Verteilung der Arbeitslast einer einzelnen Warteschlange auf identische Instanzen dieser Warteschlange auf mehreren Warteschlangenmanagern vor. Durch diese Verteilung erreichen Sie zum Einen eine größere Fehlersicherheit des Systems, zum Anderen verbessern Sie aber auch die Skalierleistung besonders aktiver Nachrichtenflüsse im System. In einer solchen Umgebung verfügt jede Instanz der verteilten Warteschlangen über konsumierende Anwendungen, die die Nachrichten verarbeiten. Weitere Informationen finden Sie im Abschnitt Cluster für Workload-Management verwenden.

Weiterleitung von Nachrichten in einem Cluster

Sie können sich einen Cluster wie ein Netz aus Warteschlangen vorstellen, das von einem gewissenhaften Systemadministrator verwaltet wird. Sobald Sie eine Clusterwarteschlange definieren, erstellt der Systemadministrator automatisch auf den anderen Warteschlangenmanagern die benötigten Definitionen für die ferne Warteschlange.

Definitionen für Übertragungswarteschlangen müssen nicht erstellt werden, da IBM MQ auf jedem Warteschlangenmanager des Clusters eine Übertragungswarteschlange bereitstellt. Über diese Übertragungswarteschlange können Nachrichten an jeden anderen Warteschlangenmanager des Clusters übertragen werden. Sie müssen sich nicht auf die Verwendung nur einer einzelnen Übertragungswarteschlange beschränken. Ein Warteschlangenmanager kann mehrere Übertragungswarteschlangen verwenden, um die Nachrichten, die an die einzelnen Warteschlangenmanager in einem Cluster gesendet werden, voneinander zu trennen. In der Regel verwendet ein Warteschlangenmanager eine einzelne Clusterübertragungswarteschlange. Durch die Änderung des Warteschlangenmanagerattributs DEFCLXQ können Sie festlegen, dass ein Warteschlangenmanager für jeden Warteschlangenmanager in einem Cluster eine andere Clusterübertragungswarteschlange verwenden soll. Cluster-Übertragungs-WS können auch manuell definiert werden.

Alle Warteschlangenmanager, die einem Cluster beitreten, sind sich über diese Art der Zusammenarbeit einig. Sie geben Informationen über sich selbst und die von ihnen bereitgestellten Warteschlangen nach außen und erhalten ihrerseits Informationen über die anderen Mitglieder des Clusters.

Um sicherzustellen, dass keine Informationen verloren gehen, wenn ein Warteschlangenmanager ausfällt, sollten Sie zwei Warteschlangenmanager im Cluster mit *vollständigem Repository* definieren. Diese Warteschlangenmanager speichern einen vollständigen Satz aller Informationen über die Warteschlangenmanager und Warteschlangen im Cluster. Alle anderen Warteschlangenmanager im Cluster speichern nur

Informationen über diejenigen Warteschlangenmanager und Warteschlangen, mit denen sie Nachrichten austauschen. Diese Warteschlangenmanager werden als Warteschlangenmanager mit *Teilrepositorys* bezeichnet. Weitere Informationen finden Sie in „Cluster-Repository" auf Seite 61.

Für den Anschluss an einen Cluster benötigt ein Warteschlangenmanager zwei Kanäle, einen Clustersenderkanal und einen Clusterempfängerkanal:

- Ein Clustersenderkanal ist ein einem Senderkanal entsprechender Kommunikationskanal. Den Clustersenderkanal müssen Sie auf jedem Warteschlangenmanager manuell erstellen, um den Warteschlangenmanager mit einem vollständigen Repository des Clusters zu verbinden.
- Ein Clusterempfängerkanal ist ein einem Empfängerkanal entsprechender Kommunikationskanal. Sie müssen manuell einen Clusterempfängerkanal erstellen. Dieser Kanal wird als Mechanismus für den Empfang der Clusterkommunikation durch den Warteschlangenmanager genutzt.

Alle anderen Kanäle, die zusätzlich für die Kommunikation zwischen diesem Warteschlangenmanager und den anderen Mitgliedern des Clusters erforderlich sind, werden dann automatisch erstellt.

Die folgende Abbildung zeigt die Komponenten eines Clusters mit dem Namen CLUSTER:

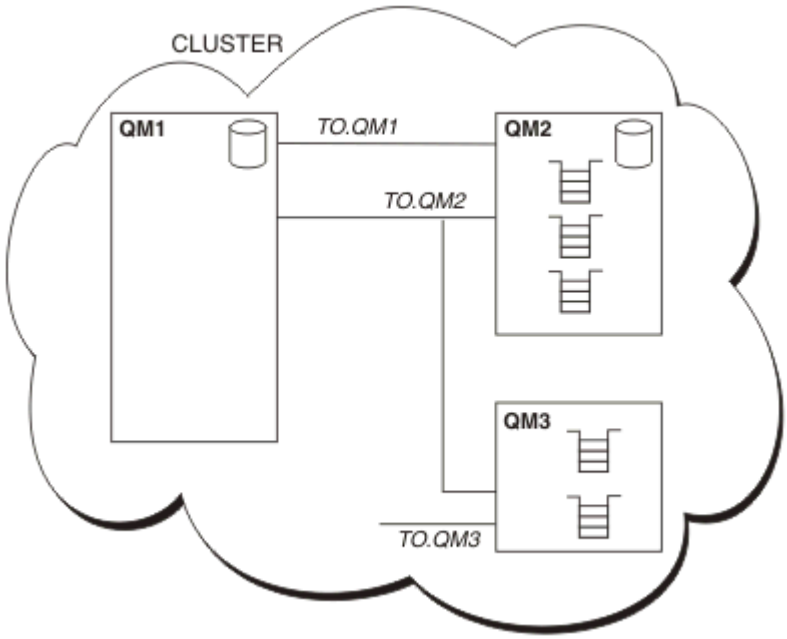


Abbildung 7. Warteschlangenmanagercluster

- CLUSTER enthält die drei Warteschlangenmanager QM1, QM2 und QM3.
- QM1 und QM2 verwalten vollständige Repositorys mit Informationen über die Warteschlangenmanager und Warteschlangen im Cluster.
- QM2 und QM3 verwalten einige Clusterwarteschlangen, d. h. Warteschlangen, auf die jeder andere Warteschlangenmanager im Cluster zugreifen kann.
- Jeder Warteschlangenmanager besitzt einen Clusterempfängerkanal mit dem Namen TO.qmgr, über den er Nachrichten empfangen kann.
- Außerdem besitzt jeder Warteschlangenmanager einen Clustersenderkanal, über den er Informationen an einen der Repository-Warteschlangenmanager senden kann.
- QM1 und QM3 senden Informationen an das Repository von QM2 und QM2 sendet Informationen an das Repository von QM1.

Komponenten der verteilten Steuerung von Warteschlangen

Die verteilte Steuerung von Warteschlangen besteht aus folgenden Komponenten: Nachrichtenkanälen, Nachrichtenkanalagenten, Übertragungswarteschlangen, Kanalinitiatoren, Empfangsprogrammen und Ka-

nalexitprogrammen. Bei der Definition der beiden Enden eines Nachrichtenkanals kann es sich um verschiedene Typen handeln.

Nachrichtenkanäle sind die Kanäle, über die Nachrichten von einem Warteschlangenmanager an einen anderen übertragen werden. Verwechseln Sie Nachrichtenkanäle nicht mit MQI-Kanälen. Es gibt zwei Typen von MQI-Kanälen, Serververbindung (SVRCONN) und Clientverbindung (CLNTCONN). Weitere Informationen finden Sie im Abschnitt Kanäle.

Bei der Definition der beiden Enden eines Nachrichtenkanals kann es sich um einen der folgenden Typen handeln:

- Sender (SDR)
- Empfänger (RCVR)
- Server (SVR)
- Requester (RQSTR)
- Clustersender (CLUSSDR)
- Clusterempfänger (CLUSRCVR)

Ein Nachrichtenkanal wird definiert, indem einer dieser Typen an einem Ende und ein kompatibler Typ am anderen Ende definiert werden. Folgende Kombinationen sind möglich:

- Sender-Empfänger
- Requester-Server
- Requester-Sender (Callback)
- Server-Empfänger
- Clustersender-Clusterempfänger

Detaillierte Anweisungen zum Erstellen eines Sender-Empfänger-Kanals finden Sie im Abschnitt Kanäle definieren. Beispiele für die erforderlichen Parameter zur Konfiguration von Sender-Empfänger-Kanälen finden Sie im betreffenden Abschnitt Beispielkonfigurationsinformationen für Ihre Plattform. Informationen zu den Parametern, die zur Definition eines Kanals eines beliebigen Typs benötigt werden, finden Sie in der Beschreibung von DEFINE CHANNEL.

Sender-Empfänger-Kanäle

Ein Sender in einem System startet den Kanal, damit er Nachrichten an das andere System senden kann. Der Sender fordert den Empfänger am anderen Ende des Kanals zum Starten auf. Der Sender sendet Nachrichten aus seiner Übertragungswarteschlange an den Empfänger. Der Empfänger stellt die Nachrichten in die Zielwarteschlange. Abbildung 8 auf Seite 51 zeigt diesen Vorgang.

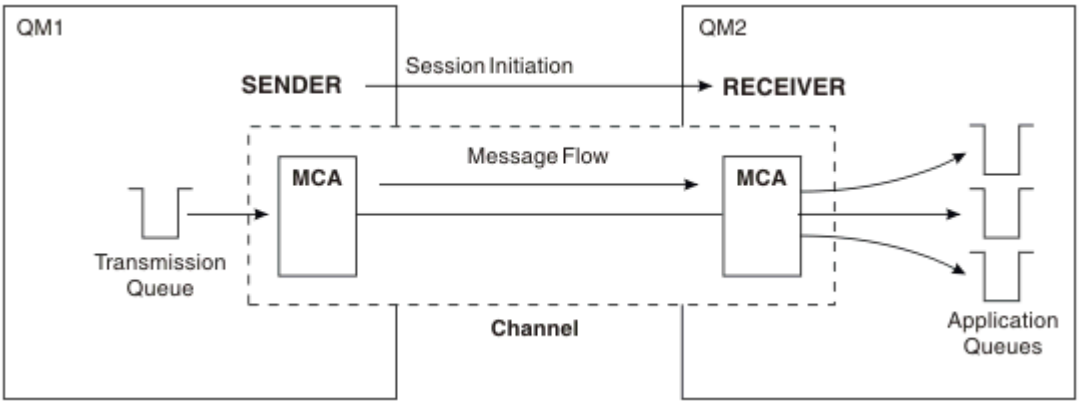


Abbildung 8. Sender-Empfänger-Kanal

Anforderer-Server-Kanäle

Ein Requester in einem System startet den Kanal, damit er Nachrichten vom anderen System empfangen kann. Der Requester fordert den Server am anderen Ende des Kanals zum Starten auf. Der Server sendet Nachrichten aus der Übertragungswarteschlange, die in seiner Kanaldefinition angegeben ist, an den Requester.

Ein Serverkanal kann auch selbst die Kommunikation einleiten und Nachrichten an einen Requester senden. Dies gilt nur für *vollständig qualifizierte* Server, d. h. für Serverkanäle, in deren Kanaldefinition der Verbindungsname des Partners angegeben ist. Ein vollständig qualifizierter Server kann entweder von einem Requester gestartet werden oder selbst eine Kommunikation mit einem Requester einleiten.

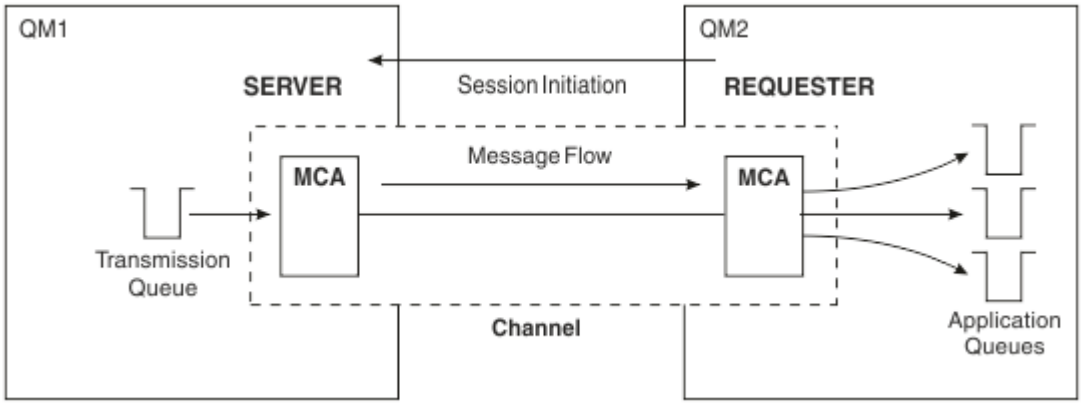


Abbildung 9. Requester-Server-Kanal

Anforderer-Sender-Kanäle

Der Requester startet den Kanal und der Sender beendet den Aufruf. Der Sender startet die Kommunikation anschließend erneut anhand der Informationen in seiner Kanaldefinition (bekannt als *Callback*). Er sendet Nachrichten aus der Übertragungswarteschlange an den Requester.

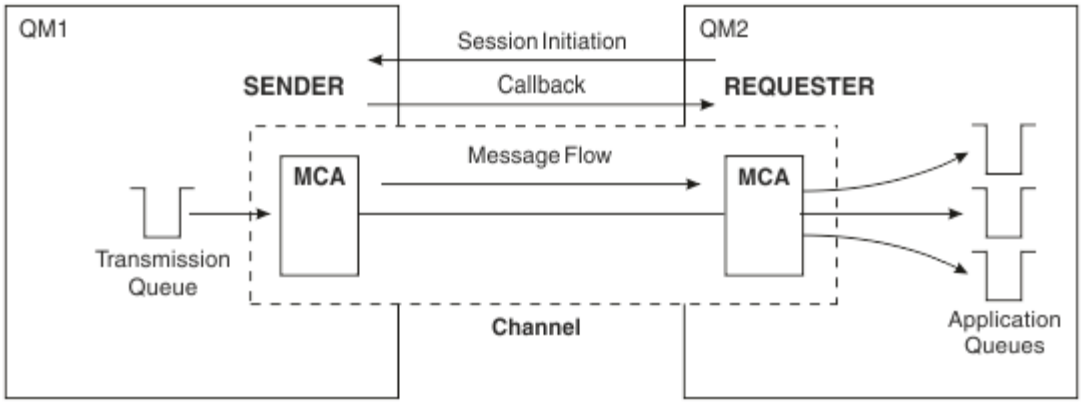


Abbildung 10. Requester-Sender-Kanal

Server-Empfänger-Kanäle

Dies entspricht einem Sender-Empfänger-Kanal, gilt aber nur für *vollständig qualifizierte* Server, d. h. für Serverkanäle, in deren Kanaldefinition der Verbindungsname des Partners angegeben ist. Der Start des Kanals muss auf der Serverseite der Verbindung eingeleitet werden. Dieser Vorgang wird in Abbildung 8 auf Seite 51 dargestellt.

Clustersenderkanäle

In einem Cluster besitzt jeder Warteschlangenmanager einen Clustersenderkanal, über den er Clusterinformationen an eines der vollständigen Warteschlangenmanager-Repositorys senden kann. Warteschlangenmanager können über Clustersenderkanäle auch Nachrichten an andere Warteschlangenmanager senden.

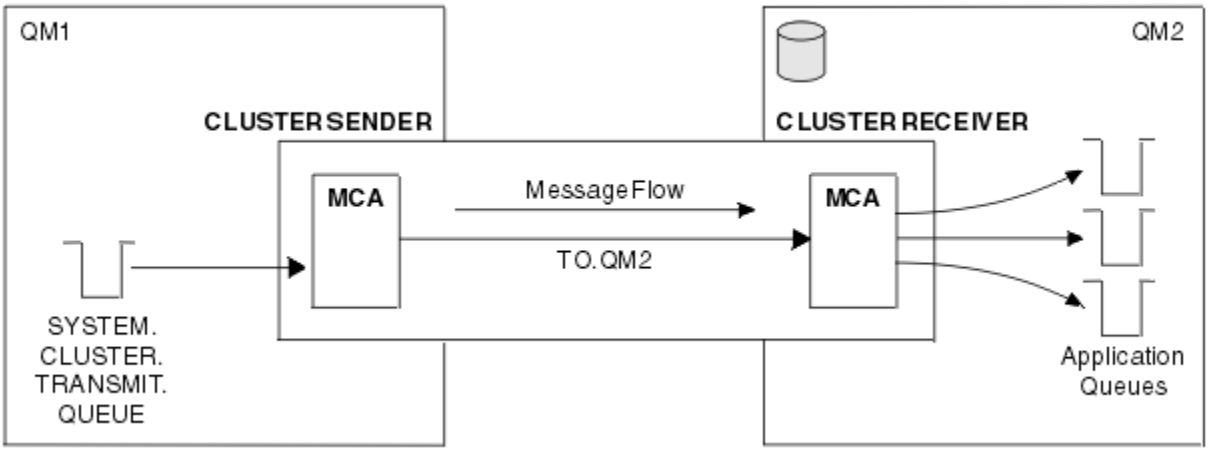


Abbildung 11. Ein Clustersenderkanal

Clusterempfängerkanäle

In einem Cluster besitzt jeder Warteschlangenmanager einen Clusterempfängerkanal, über den er Nachrichten sowie Informationen über den Cluster empfangen kann. Dieser Vorgang wird in Abbildung 11 auf Seite 53 dargestellt.

Warteschlangen für nicht zustellbare Nachrichten

Die Warteschlange für nicht zustellbare Nachrichten (oder Warteschlange für nicht zugestellte Nachrichten) ist die Warteschlange, an die Nachrichten gesendet werden, wenn sie nicht an ihr eigentliches Ziel weitergeleitet werden können. In der Regel hat jeder Warteschlangenmanager eine solche Warteschlange.

Eine *Warteschlange für nicht zustellbare Nachrichten* (DLQ), manchmal auch als *Warteschlange für nicht zugestellte Nachrichten*bezeichnet, ist eine Haltewarteschlange für Nachrichten, die nicht an ihre Zielwarteschlangen zugestellt werden können, z. B. weil die Warteschlange nicht vorhanden oder voll ist. Warteschlangen für nicht zustellbare Nachrichten werden auch auf der Sendeseite eines Kanals bei Datenkonvertierungsfehlern verwendet. Jeder Warteschlangenmanager im Netz hat üblicherweise eine lokale Warteschlange, die als Warteschlange für nicht zustellbare Nachrichten verwendet wird, sodass Nachrichten, die ihrem Ziel momentan nicht zugestellt werden können, für den späteren Abruf gespeichert werden können.

Nachrichten können von Warteschlangenmanagern, Nachrichtenkanalagenten (MCA; Message Channel Agent) und Anwendungen in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden. Alle Nachrichten in der DLQ müssen mit einer *Headerstruktur einer nicht zustellbaren Nachricht* (MQDLH) als Präfix vorangesetzt werden. Das Feld *Reason* der MQDLH-Struktur enthält einen Ursachencode, der angibt, warum sich die Nachricht in der DLQ befindet.

Sie sollten jedem Warteschlangenmanager eine Warteschlange für nicht zustellbare Nachrichten zuweisen. Wenn keine vorhanden ist und der Nachrichtenkanalagent eine Nachricht nicht einreihen kann, bleibt sie in der Übertragungswarteschlange stehen und der Kanal wird gestoppt. Auch werden schnelle, nicht persistente Nachrichten (siehe Schnelle, nicht persistente Nachrichten), die nicht zugestellt werden können, verworfen, wenn auf dem Zielsystem keine Warteschlange für nicht zustellbare Nachrichten vorhanden ist.

Allerdings kann sich die Verwendung von Warteschlangen für nicht zustellbare Nachrichten auf die Reihenfolge auswirken, in der Nachrichten zugestellt werden, sodass Sie möglicherweise darauf verzichten möchten.

Zugehörige Tasks

Mit dead-letter-Warteschlangen arbeiten

Fehlerbehebung bei nicht zugestellten Nachrichten

Zugehörige Verweise

runmqdlq (Steuerroutine der Warteschlange für nicht zustellbare Nachrichten ausführen)

Definitionen ferner Warteschlangen

Definitionen ferner Warteschlangen sind Definitionen für Warteschlangen, die Eigentum eines anderen Warteschlangenmanagers sind.

Abrufen können Anwendungen Nachrichten zwar nur aus lokalen Warteschlangen, aber Einreihen können Sie Nachrichten sowohl in lokale als auch in ferne Warteschlangen. Deshalb kann ein Warteschlangenmanager neben einer Definition für jede seiner lokalen Warteschlangen auch *Definitionen ferner Warteschlangen* besitzen. Der Vorteil von Definitionen ferner Warteschlangen besteht darin, dass sie es einer Anwendung ermöglichen, eine Nachricht in eine ferne Warteschlange zu stellen, ohne den Namen der fernen Warteschlange oder des fernen Warteschlangenmanagers bzw. den Namen der Übertragungswarteschlange kennen zu müssen. Definitionen ferner Warteschlangen bieten Standortunabhängigkeit.

Es gibt weitere Verwendungsmöglichkeiten für Definitionen ferner Warteschlangen, die später beschrieben werden.

Ferne Warteschlangenmanager abrufen

Sie verfügen möglicherweise nicht immer über einen separaten Kanal zwischen jedem Quellen- und Ziel-Warteschlangenmanager. Es gibt verschiedene Möglichkeiten, eine Verbindung zwischen den beiden herzustellen, einschließlich Multihopping, gemeinsame Nutzung von Kanälen, Nutzung verschiedener Kanäle und Clustering.

Multihopping

Wenn keine direkte Kommunikationsverbindung zwischen dem Quellen-Warteschlangenmanager und dem Ziel-Warteschlangenmanager besteht, ist es möglich, einen oder mehrere *zwischengeschaltete Warteschlangenmanager* auf dem Weg zum Ziel-Warteschlangenmanager zu durchlaufen. Dies ist als *Multihopping* bekannt.

Sie müssen Kanäle zwischen allen Warteschlangenmanagern und Übertragungswarteschlangen auf den zwischengeschalteten Warteschlangenmanagern definieren. Dies wird in Abbildung 12 auf Seite 55 gezeigt.

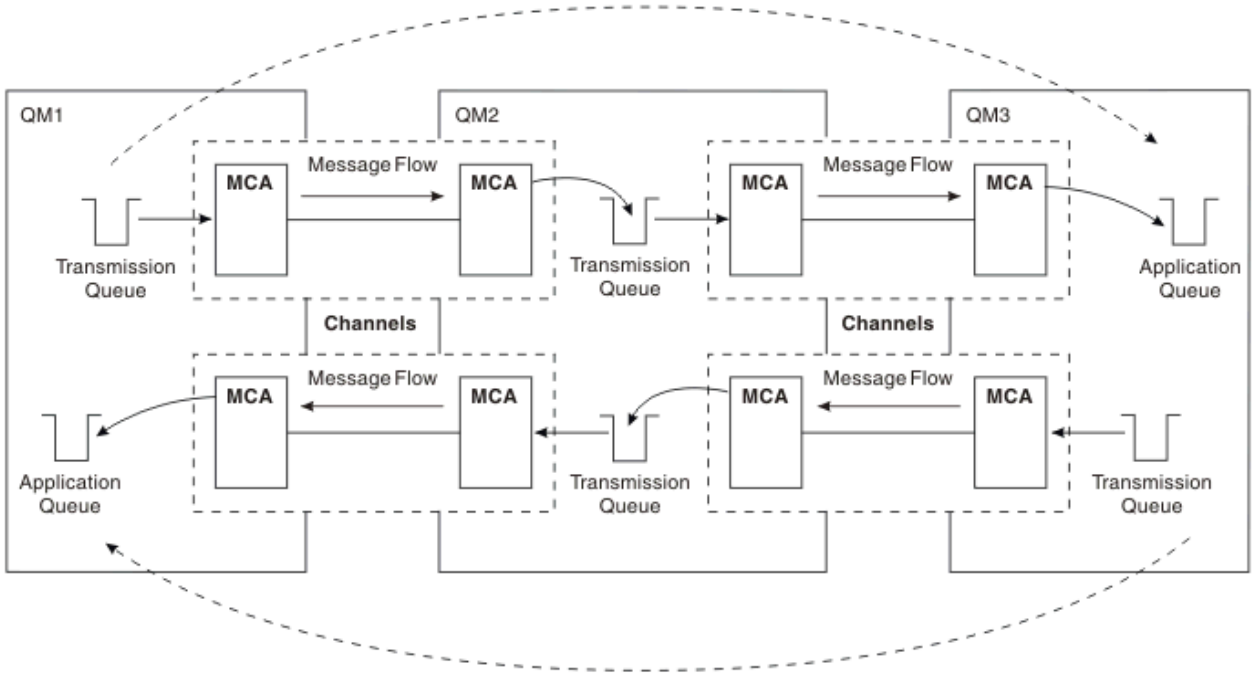


Abbildung 12. Übertragung über zwischengeschaltete Warteschlangenmanager

Gemeinsame Nutzung von Kanälen

Als Anwendungsentwickler können Sie wählen, ob Sie Ihre Anwendungen zwingen, neben dem Namen der Warteschlange auch den Namen des fernen Warteschlangenmanagers anzugeben, oder ob Sie für jede ferne Warteschlange eine *Definition einer fernen Warteschlange* erstellen. Diese Definition enthält den Namen des fernen Warteschlangenmanagers, den Namen der Warteschlange und den Namen der Übertragungswarteschlange. In jedem Fall werden alle Nachrichten von allen Anwendungen, die Warteschlangen an denselben fernen Standort adressieren, über dieselbe Übertragungswarteschlange gesendet. Dies wird in Abbildung 13 auf Seite 55 gezeigt.

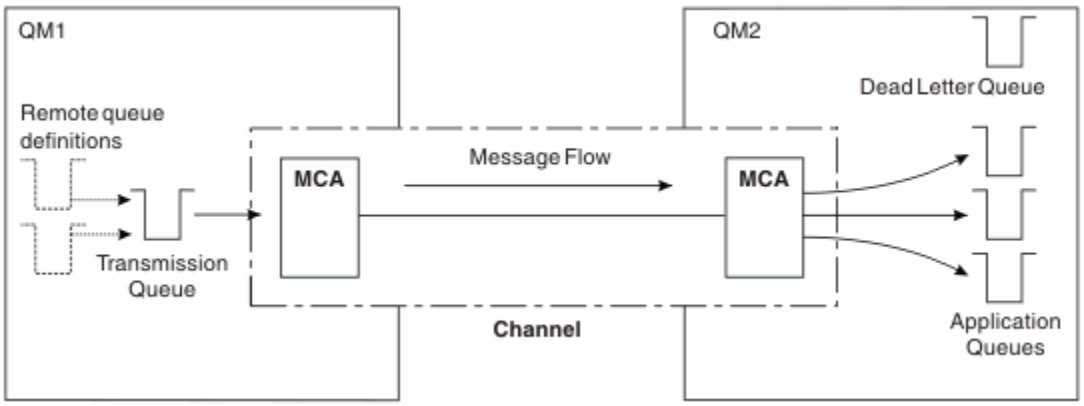


Abbildung 13. Gemeinsame Nutzung einer Übertragungswarteschlange

Abbildung 13 auf Seite 55 zeigt, dass für Nachrichten von mehreren Anwendungen an mehrere ferne Warteschlangen derselbe Kanal verwendet werden kann.

Nutzung verschiedener Kanäle

Wenn Nachrichten unterschiedlichen Typs zwischen zwei Warteschlangenmanagern ausgetauscht werden müssen, können Sie mehrere Kanäle zwischen den beiden definieren. Es kann Situationen geben, in

denen Sie alternative Kanäle benötigen, vielleicht aus Sicherheitsgründen oder um Zustellungsgeschwindigkeit durch reine Massenübertragung von Nachrichten auszugleichen.

Um einen zweiten Kanal zu konfigurieren, müssen Sie einen weiteren Kanal und eine weitere Übertragungswarteschlange definieren sowie eine Definition einer fernen Warteschlange erstellen, in der Sie den Standort und den Namen der Übertragungswarteschlange angeben. Ihre Anwendung kann anschließend jeden der beiden Kanäle verwenden, aber die Nachrichten werden weiterhin an dieselbe Zielwarteschlange zugestellt. Dies wird in Abbildung 14 auf Seite 56 gezeigt.

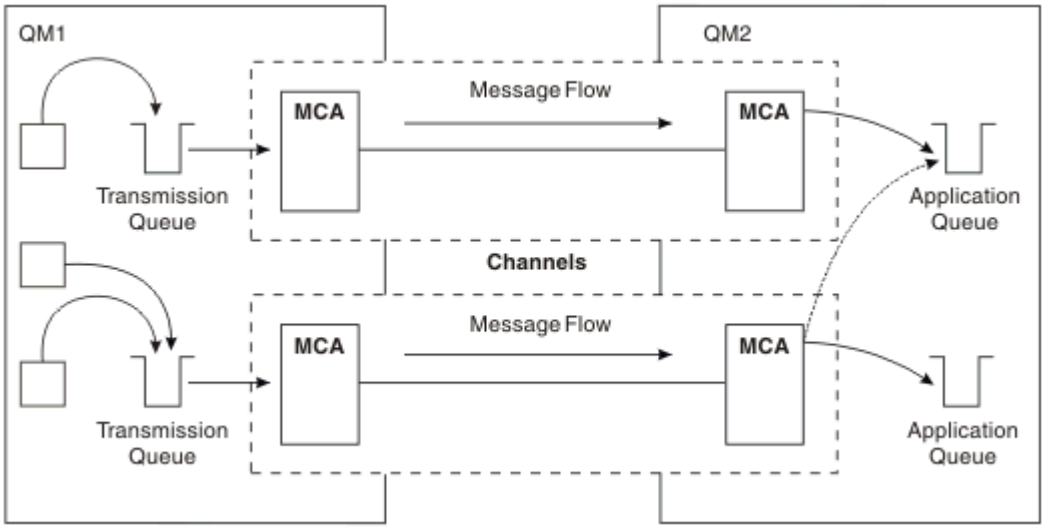


Abbildung 14. Nutzung mehrerer Kanäle

Wenn Sie eine Übertragungswarteschlange in Definitionen ferner Warteschlangen angeben, darf der Standort (d. h. der Ziel-Warteschlangenmanager) **nicht** in den Anwendungen selbst angegeben werden. Denn wenn er dort angegeben ist, verwendet der Warteschlangenmanager nicht die Definitionen ferner Warteschlangen. Definitionen ferner Warteschlangen bieten Standortunabhängigkeit. Anwendungen können Nachrichten in eine *logische* Warteschlange stellen, ohne zu wissen, wo sich die Warteschlange befindet, und Sie können die *physische* Warteschlange wechseln, ohne Ihre Anwendung ändern zu müssen.

Clustering verwenden

Jeder Warteschlangenmanager in einem Cluster definiert einen Clusterempfängerkanal. Wenn ein Warteschlangenmanager eine Nachricht an einen anderen Warteschlangenmanager senden möchte, definiert er automatisch den zugehörigen Clustersenderkanal. Wenn es beispielsweise mehrere Instanzen einer Warteschlange in einem Cluster gibt, kann der Clustersenderkanal für jeden der Warteschlangenmanager, die die Warteschlange betreiben, definiert werden. IBM MQ verwendet einen Workload-Management-Algorithmus mit einer Umlaufroutine, um einen verfügbaren Warteschlangenmanager zur Weiterleitung einer Nachricht auszuwählen. Weitere Informationen finden Sie im Abschnitt Cluster.

Adressinformation

Wenn eine Anwendung Nachrichten einreiht, die für einen fernen Warteschlangenmanager bestimmt sind, fügt der lokale Warteschlangenmanager einen Übertragungsheader hinzu, bevor er die Nachrichten in die Übertragungswarteschlange stellt. Dieser Header enthält die Namen der Zielwarteschlange und des Warteschlangenmanagers, also die *Adressinformationen*.

In einer Umgebung mit nur einem Warteschlangenmanager wird die Adresse einer Zielwarteschlange erstellt, wenn eine Anwendung eine Warteschlange zum Einreihen von Nachrichten öffnet. Da sich die Zielwarteschlange auf demselben Warteschlangenmanager befindet, werden keine Adressinformationen benötigt.

In einer Umgebung mit verteilter Steuerung von Warteschlangen muss der Warteschlangenmanager nicht nur den Namen der Zielwarteschlange kennen, sondern auch deren Standort (d. h. den Namen des Warteschlangenmanagers) und die Route zu diesem Standort (d. h. die Übertragungswarteschlange).

Diese Adressinformationen sind im Übertragungsheader enthalten. Der empfangende Kanal entfernt den Übertragungsheader und lokalisiert anhand der darin enthaltenen Informationen die Zielwarteschlange.

Sie können vermeiden, dass in Anwendungen der Name des Zielwarteschlangenmanagers angegeben werden muss, indem Sie eine Definition einer fernen Warteschlange verwenden. In dieser Definition werden der Name der fernen Warteschlange, der Name des fernen Warteschlangenmanagers, für den Nachrichten bestimmt sind, und der Name der Übertragungswarteschlange für den Transport der Nachrichten angegeben.

Was sind Aliasnamen?

Aliasnamen dienen dazu, eine Servicequalität für Nachrichten bereitzustellen. Mithilfe des Warteschlangenmanager-Aliasnamens kann ein Systemadministrator den Namen eines Ziel-Warteschlangenmanagers ändern, ohne dass Sie daraufhin Ihre Anwendungen ändern müssen. Außerdem erhält der Systemadministrator die Möglichkeit, die Route zu einem Zielwarteschlangenmanager zu ändern oder eine Route einzurichten, die durch mehrere andere Warteschlangenmanager führt (Multihopping). Der Aliasname für Empfangswarteschlangen für Antworten bietet eine Servicequalität für Antworten.

Warteschlangenmanager-Aliasnamen und Aliasnamen für Empfangswarteschlangen für Antworten werden mithilfe einer Definition einer fernen Warteschlange, die ein leeres RNAME-Feld enthält, erstellt. Solche Definitionen definieren keine realen Warteschlangen; sie werden vom Warteschlangenmanager verwendet, um Namen von physischen Warteschlangen, Warteschlangenmanager-Namen und Namen von Übertragungswarteschlangen aufzulösen.

Aliasnamensdefinitionen sind durch ein leeres RNAME-Feld gekennzeichnet.

Auflösung des Warteschlangennamens

Eine Auflösung von Warteschlangennamen findet auf jedem Warteschlangenmanager bei jedem Öffnen einer Warteschlange statt. Sie hat den Zweck, die Zielwarteschlange, den Ziel-Warteschlangenmanager (kann ein lokaler sein) und die Route zu diesem Warteschlangenmanager (kann null sein) zu ermitteln. Der aufgelöste Name besteht aus drei Teilen: den Namen des Warteschlangenmanagers, der Warteschlange und, falls es sich um einen fernen Warteschlangenmanager handelt, der Übertragungswarteschlange.

Wenn eine Definition einer fernen Warteschlange vorhanden ist, wird nicht auf Aliasnamensdefinitionen verwiesen. Der in der Anwendung angegebene Warteschlangenname wird in die Namen der Zielwarteschlange, des fernen Warteschlangenmanagers und der Übertragungswarteschlange, die in der Definition einer fernen Warteschlange angegeben sind, aufgelöst. Ausführlichere Informationen zur Auflösung von Warteschlangennamen finden Sie im Abschnitt Auflösung von Warteschlangennamen.

Wenn keine Definition einer fernen Warteschlange vorhanden und ein Warteschlangenmanager-Name angegeben ist (oder vom Namensservice aufgelöst wurde), überprüft der Warteschlangenmanager, ob für den übergebenen Warteschlangenmanager-Namen eine zugehörige Warteschlangenmanager-Aliasnamensdefinition vorhanden ist. Wenn ja, wird der Warteschlangenmanager-Name anhand der darin enthaltenen Informationen in den Namen des Zielwarteschlangenmanagers aufgelöst. Mithilfe der Warteschlangenmanager-Aliasnamensdefinition kann auch die Übertragungswarteschlange für den Zielwarteschlangenmanager ermittelt werden.

Wenn der aufgelöste Warteschlangenname nicht der Name einer lokalen Warteschlange ist, werden sowohl der Warteschlangenmanager-Name als auch der Warteschlangenname in den Übertragungsheader jeder Nachricht aufgenommen, die von der Anwendung in die Übertragungswarteschlange gestellt wird.

Der Name der verwendeten Übertragungswarteschlange entspricht normalerweise dem aufgelösten Warteschlangenmanager-Namen, sofern er nicht durch eine Definition einer fernen Warteschlange oder eine Warteschlangenmanager-Aliasnamensdefinition geändert wird. Wenn Sie keine Übertragungswarteschlange, aber eine Standardübertragungswarteschlange definiert haben, wird diese verwendet.

z/OS Namen von Warteschlangenmanagern, die unter z/OS ausgeführt werden, sind auf eine Länge von vier Zeichen begrenzt.

WS-Manager-Aliasdefinitionen

Warteschlangenmanager-Aliasnamensdefinitionen werden wirksam, wenn eine Anwendung, die eine Warteschlange zum Einreihen einer Nachricht öffnet, den Warteschlangennamen **und** den Warteschlangenmanager-Namen angibt.

Warteschlangenmanager-Aliasnamensdefinitionen werden in drei Situationen verwendet:

- Beim Senden von Nachrichten, um den Warteschlangenmanager-Namen neu zuzuordnen
- Beim Senden von Nachrichten, um die Übertragungswarteschlange zu ändern oder anzugeben
- Beim Empfangen von Nachrichten, um zu ermitteln, ob der lokale Warteschlangenmanager das gewünschte Ziel für die Nachrichten ist

Abgehende Nachrichten - Neuzuordnung des Warteschlangenmanager-Namens

Mithilfe von Warteschlangenmanager-Aliasnamensdefinitionen kann der in einem MQOPEN-Aufruf angegebene Warteschlangenmanager-Name neu zugeordnet werden. Ein Beispiel: In einem MQOPEN-Aufruf wird der Warteschlangenname THISQ und der Warteschlangenmanager-Name YOURQM angegeben. Der lokale Warteschlangenmanager verfügt über folgende Warteschlangenmanager-Aliasnamensdefinition:

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

Daraus geht hervor, dass tatsächlich der Warteschlangenmanager REALQM verwendet werden soll, wenn eine Anwendung Nachrichten an den Warteschlangenmanager YOURQM übergibt. Handelt es sich bei REALQM um den lokalen Warteschlangenmanager, stellt dieser die Nachrichten in die Warteschlange THISQ, eine lokale Warteschlange. Trägt der lokale Warteschlangenmanager nicht die Bezeichnung REALQM, wird die Nachricht an ein Übertragungswarteschlange namens REALQM weitergeleitet. Der Warteschlangenmanager ändert den Übertragungsheader von YOURQM in REALQM.

Abgehende Nachrichten - Änderung oder Angabe der Übertragungswarteschlange

Abbildung 15 auf Seite 58 zeigt ein Szenario, in dem bei Warteschlangenmanager QM1 Nachrichten ankommen, in deren Übertragungsheadern Namen von Warteschlangen auf Warteschlangenmanager QM3 angegeben sind. In diesem Szenario ist QM3 durch Multihopping über QM2 erreichbar.

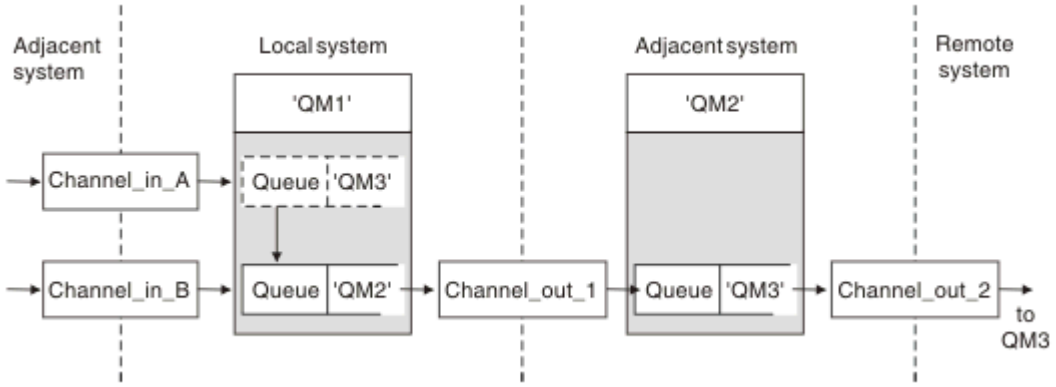


Abbildung 15. Aliasname des Warteschlangenmanagers

Alle Nachrichten für QM3 werden auf QM1 mit einem Warteschlangenmanager-Aliasnamen erfasst. Der Warteschlangenmanager-Aliasname lautet QM3 und enthält die Definition QM3 über die Übertragungswarteschlange QM2. Das folgende Beispiel zeigt die entsprechende Definition:

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

Der Warteschlangenmanager stellt die Nachrichten in Übertragungswarteschlange QM2, ändert jedoch den Header der Übertragungswarteschlange nicht, da sich der Name des Zielwarteschlangenmanagers QM3 nicht ändert.

Alle an QM1 ankommenden Nachrichten, in deren Übertragungsheader der Name einer Warteschlange von QM2 angegeben ist, werden auch in die Übertragungswarteschlange QM2 eingereiht. Auf diese Weise werden Nachrichten mit unterschiedlichen Zielen in einer gemeinsamen Übertragungswarteschlange auf einem geeigneten Nachbarsystem gesammelt, um sie dann an ihre jeweiligen Ziele zu übertragen.

Eingehende Nachrichten - Ermittlung des Ziels

Ein empfangender Nachrichtenkanalagent öffnet die Warteschlange, die im Übertragungsheader angegeben ist. Wenn eine Warteschlangenmanager-Aliasnamensdefinition mit demselben Namen wie dem des angegebenen Warteschlangenmanagers vorhanden ist, wird der im Übertragungsheader empfangene Name des Warteschlangenmanagers aus der Definition ersetzt.

Für diesen Prozess gibt es zwei Anwendungsfälle:

- Weiterleitung von Nachrichten an einen anderen Warteschlangenmanager
- Änderung des Warteschlangenmanager-Namens in den Namen des lokalen Warteschlangenmanagers

Aliasnamendefinitionen für Antwortwarteschlange

In einer Aliasnamensdefinition für eine Empfangswarteschlange für Antworten werden alternative Namen für die Antwortinformationen im Nachrichtendeskriptor angegeben. Dies bietet den Vorteil, dass Sie den Namen einer Warteschlange oder eines Warteschlangenmanagers ändern können, ohne Ihre Anwendungen ändern zu müssen.

Auflösung des Warteschlangennamens

Wenn eine Anwendung auf eine Nachricht antwortet, ermittelt sie anhand der Daten im *Nachrichtendeskriptor* der empfangenen Nachricht die Warteschlange, an die die Antwort gehen soll. Die sendende Anwendung gibt an, wohin Antworten gesendet werden sollen, indem sie entsprechende Informationen an ihre Nachrichten anhängt. Dieses Konzept muss im Rahmen des Anwendungsentwurfs koordiniert werden.

Die Auflösung des Warteschlangennamens finden auf der Sendeseite Ihrer Anwendung statt, bevor die Nachricht in eine Warteschlangen gestellt wird. Die Auflösung des Warteschlangennamens finden daher vor der Interaktion mit der fernen Anwendung statt, an die die Nachricht gesendet wird. Dies ist die einzige Situation, in der die Namensauflösung zu einem Zeitpunkt erfolgt, an dem eine Warteschlange nicht geöffnet ist.

Auflösung des Warteschlangennamens mithilfe eines Warteschlangenmanager-Aliasnamens

Normalerweise gibt eine Anwendung eine Empfangswarteschlange für Antworten an und lässt das Feld für den Namen des Managers der Empfangswarteschlange für Antworten leer. Der Warteschlangenmanager trägt seinen Namen zum Zeitpunkt des Einreihens der Nachricht ein. Dieses Verfahren funktioniert gut, es sei denn, für Antworten soll ein anderer Kanal verwendet werden, also beispielsweise ein Kanal, der die Übertragungswarteschlange QM1_relief verwendet, und nicht der Standardrückgabekanal, der die Übertragungswarteschlange QM1 verwendet. In dieser Situation handelt es sich bei den Warteschlangenmanager-Namen, die in den Übertragungswarteschlangenheadern angegeben sind, nicht um “echte” Warteschlangenmanager-Namen, sondern um Namen, die mithilfe von Warteschlangenmanager-Aliasnamensdefinitionen neu festgelegt werden. Um Antworten über alternative Routen zurückgeben zu können, müssen auch Daten der Empfangswarteschlange für Antworten mithilfe von Aliasnamensdefinitionen für Empfangswarteschlangen für Antworten zugeordnet werden.

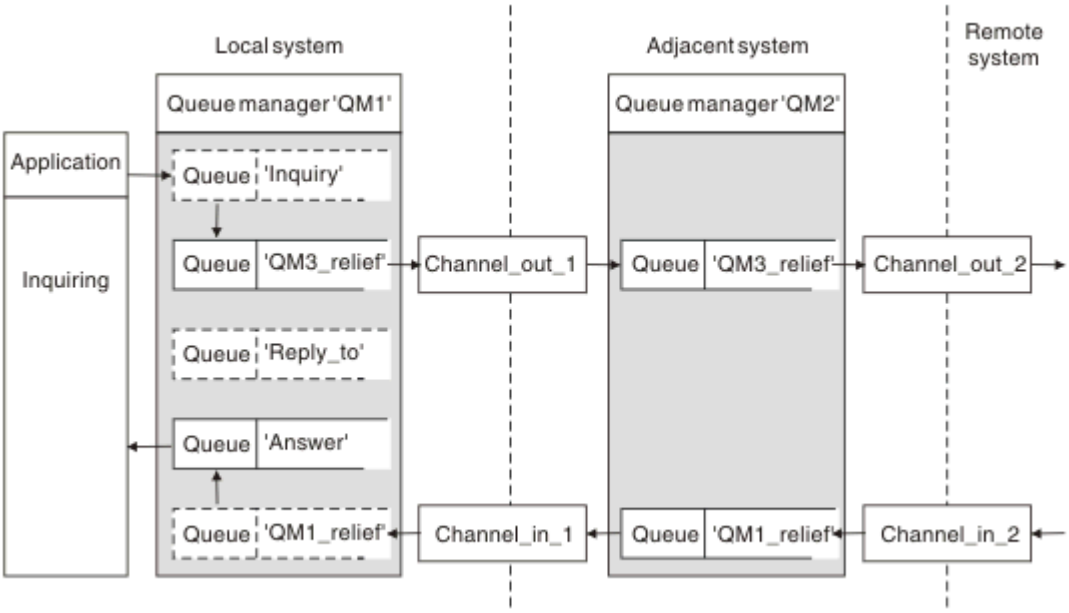


Abbildung 16. Aliasnamen für Empfangswarteschlange für Antworten zum Ändern der Antwortposition verwenden

Das Beispiel in Abbildung 16 auf Seite 60 zeigt Folgendes:

1. Die Anwendung reiht mit dem Aufruf MQPUT eine Nachricht ein und gibt folgende Informationen im Nachrichtendeskriptor an:

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

   Die Angabe für ReplyToQMgr muss ein Leerzeichen sein, damit der Aliasname der Empfangswarteschlange für Antworten verwendet wird.
2. Sie erstellen eine Aliasdefinition `Reply_to` für die Empfangswarteschlange für Antworten, in welcher der Name `Answer` und der Name des Warteschlangenmanagers `QM1_relief` enthalten sind.

   ```
   DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
   RQMNAME ('QM1_relief')
   ```
3. Die Nachrichten werden mit einem Nachrichtendeskriptor gesendet, in dem `ReplyToQ='Answer'` und `ReplyToQMgr='QM1_relief'` angegeben ist.
4. Die Anwendungsspezifikation muss die Information enthalten, dass Antworten in der Warteschlange `Answer` und nicht in `Reply_to` zu finden sind.

Um Vorbereitungen für den Empfang der Antworten zu treffen, müssen Sie den parallelen Rückgabekanal erstellen, indem Sie Folgendes definieren:

- Auf QM2 die Übertragungswarteschlange `QM1_relief`

  ```
  DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
  ```
- Auf QM1 den Warteschlangenmanager-Aliasnamen `QM1_relief`

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

Dieser Warteschlangenmanager-Aliasname beendet die Kette aus parallelen Rückgabekanälen und erfasst die Nachrichten für QM1.

Wenn Sie glauben, dass Sie dieses Verfahren irgendwann in der Zukunft einsetzen werden, stellen Sie sicher, dass der Aliasname von Anfang an in Anwendungen verwendet wird. Im Moment handelt es sich um einen normalen Aliasnamen für die Empfangswarteschlange für Antworten, der aber später in einen Warteschlangenmanager-Aliasnamen geändert werden kann.

Name der Empfangswarteschlange für Antworten

Bei der Benennung von Empfangswarteschlangen für Antworten ist sorgfältig vorzugehen. Dass eine Anwendung den Namen einer Empfangswarteschlange für Antworten in der Nachricht angibt, hat den Grund, dass sie auf diese Weise die Warteschlange angeben kann, an die die Antworten gesendet werden sollen. Wenn Sie eine Aliasnamensdefinition für eine Empfangswarteschlange für Antworten mit diesem Namen erstellen, darf die eigentliche Empfangswarteschlange für Antworten (d. h. die Definition einer lokalen Warteschlange) nicht denselben Namen haben. Deshalb muss die Aliasnamensdefinition der Empfangswarteschlange für Antworten einen neuen Warteschlangennamen und den Warteschlangenmanager-Namen enthalten und die Anwendungsspezifikation muss die Information enthalten, dass die Antworten in dieser anderen Warteschlange eingehen werden.

Die Anwendungen müssen die Nachrichten jetzt aus einer anderen Warteschlange abrufen, als sie beim Einreihen der ursprünglichen Nachricht als Empfangswarteschlange für Antworten angegeben haben.

Clusterkomponenten

Cluster bestehen aus Warteschlangenmanagern, Clusterrepositorys, Clusterkanälen und Clusterwarteschlangen.

In den folgenden Abschnitten finden Sie Informationen zu den einzelnen Clusterkomponenten:

Zugehörige Konzepte

Vergleich von Clustering und verteilter Steuerung von Warteschlangen

Zugehörige Tasks

WS-Manager-Cluster konfigurieren

Neuen Cluster einrichten

Cluster-Repository

Ein Repository ist eine Zusammenstellung von Informationen über die Warteschlangenmanager, die zu einem Cluster gehören.

Zu den Repositoryinformationen gehören die Namen der Warteschlangenmanager, ihre Standorte, ihre Kanäle, welche Warteschlangen von ihnen betrieben werden und weitere Informationen. Die Informationen werden in Form von Nachrichten in einer Warteschlange mit dem Namen `SYSTEM.CLUSTER.REPOSITORY.QUEUE` gespeichert. Diese Warteschlange ist eines der Standardobjekte. Multi Bei Multiplatforms wird sie definiert, wenn Sie einen IBM MQ-Warteschlangenmanager erstellen. z/OS Unter IBM MQ for z/OS wird sie als Teil der Warteschlangenmanageranpassung definiert.

Vollständiges Repository und Teilrepository

Üblicherweise verfügen zwei Warteschlangenmanager in einem Cluster über ein vollständiges Repository. Die übrigen Warteschlangenmanager verfügen alle über ein Teilrepository.

Ein Warteschlangenmanager, der über einen vollständigen Satz von Informationen über jeden Warteschlangenmanager im Cluster verfügt, besitzt ein vollständiges Repository. Andere Warteschlangenmanager im Cluster besitzen Teilrepositorys mit einer Untermenge der Informationen in den vollständigen Repositorys.

Ein Teilrepository enthält nur Informationen über die Warteschlangenmanager, mit denen der Warteschlangenmanager Nachrichten austauschen muss. Die Warteschlangenmanager fordern Aktualisierun-

gen zu den von Ihnen benötigten Informationen an, d. h., wenn sich die Informationen ändern, sendet ihnen der Warteschlangenmanager mit dem vollständigen Repository die neuen Informationen. Die meiste Zeit enthält ein Teilrepository alle Informationen, die ein Warteschlangenmanager zur Ausführung innerhalb des Clusters benötigt. Falls zusätzliche Informationen benötigt werden, ruft er diese aus dem vollständigen Repository ab und nimmt eine entsprechende Aktualisierung des Teilrepositorys vor. Für die Anforderung und den Empfang von Aktualisierungen der Repositorys verwenden die Warteschlangenmanager die Warteschlange `SYSTEM.CLUSTER.COMMAND.QUEUE`.

Migrieren Sie bei der Migration von Warteschlangenmanagern, die zu einem Cluster gehören, die vollständigen Repositorys vor den Teilrepositorys. Der Grund dafür ist, dass ein älteres Repository neuere Attribute, die in einem neueren Release eingeführt wurden, nicht speichern kann. Es toleriert sie, speichert sie aber nicht.

Clusterwarteschlangenmanager

Ein Clusterwarteschlangenmanager ist ein Warteschlangenmanager, der Mitglied eines Clusters ist.

Ein Warteschlangenmanager kann mehreren Clustern angehören. Jeder Clusterwarteschlangenmanager muss einen Namen haben, der für alle Cluster, denen er angehört, eindeutig ist.

Ein Clusterwarteschlangenmanager kann Warteschlangen betreiben, die er den anderen Warteschlangenmanagern im Cluster zugänglich macht. Dies muss er jedoch nicht tun. Stattdessen kann er Nachrichten in Warteschlangen stellen, die anderswo im Cluster gehostet werden, und nur Antworten empfangen, die explizit an ihn gerichtet sind.

z/OS In IBM MQ for z/OS kann ein Clusterwarteschlangenmanager Mitglied einer Gruppe mit gemeinsamer Warteschlange sein. In diesem Fall teilt er seine Warteschlangendefinitionen mit anderen Warteschlangenmanagern in derselben Gruppe mit gemeinsamer Warteschlange.

Clusterwarteschlangenmanager sind autonom. Sie besitzen die vollständige Kontrolle über die von ihnen definierten Warteschlangen und Kanäle. Ihre Definitionen können nicht von anderen Warteschlangenmanagern geändert werden (im Gegensatz zu Warteschlangenmanagern in derselben Gruppe mit gemeinsamer Warteschlange). Repository-Warteschlangenmanager kontrollieren nicht die Definitionen der anderen Warteschlangenmanager im Cluster. Sie verfügen über einen vollständigen Satz von Definitionen, der bei Bedarf verwendet werden kann. Ein Cluster ist eine Verknüpfung von Warteschlangenmanagern.

Nachdem Sie auf einem Clusterwarteschlangenmanager eine Definition erstellt oder geändert haben, werden die Informationen an den Warteschlangenmanager mit dem vollständigen Repository gesendet. Andere Repositorys im Cluster werden später aktualisiert.

Warteschlangenmanager mit vollständigem Repository

Ein Warteschlangenmanager mit vollständigem Repository ist ein Clusterwarteschlangenmanager, der über eine vollständige Darstellung der Ressourcen des Clusters verfügt. Um Verfügbarkeit sicherzustellen, sollten in jedem Cluster mindestens zwei Warteschlangenmanager mit vollständigem Repository eingerichtet werden. Warteschlangenmanager mit vollständigem Repository empfangen Informationen, die von den anderen Warteschlangenmanagern im Cluster gesendet werden, und aktualisieren ihre Repositorys. Sie tauschen Nachrichten untereinander aus, um sicherzustellen, dass beide immer auf dem neuesten Stand der Informationen über den Cluster sind.

Warteschlangenmanager und Repositorys

Jeder Cluster hat mindestens einen (besser zwei) Warteschlangenmanager mit vollständigen Repositorys, die alle Informationen über die Warteschlangenmanager, Warteschlangen und Kanäle in einem Cluster enthalten. Diese Repositorys enthalten auch Anforderungen von den anderen Warteschlangenmanagern im Cluster zur Aktualisierung der Informationen.

Jeder der anderen Warteschlangenmanager verfügt über ein Teilrepository, das Informationen über die Untergruppe von Warteschlangen und Warteschlangenmanagern enthält, mit denen er kommunizieren muss. Die Warteschlangenmanager erstellen ihre Teilrepositorys, indem sie Anfragen stellen, wenn sie zum ersten Mal auf eine andere Warteschlange oder einen anderen Warteschlangenmanager zugreifen

müssen. Dabei stellen sie die Anforderung, dass sie alle neuen Informationen erhalten, die die jeweilige Warteschlange oder den jeweiligen Warteschlangenmanager betreffen.

Jeder Warteschlangenmanager speichert seine Repositoryinformationen in Form von Nachrichten in der Warteschlange SYSTEM.CLUSTER.REPOSITORY.QUEUE. Die Warteschlangenmanager tauschen Repositoryinformationen in Form von Nachrichten in der Warteschlange SYSTEM.CLUSTER.COMMAND.QUEUE aus.

Jeder Warteschlangenmanager, der einem Cluster beitritt, definiert einen Clustersenderkanal (CLUSSDR) zu einem der Repositorys. Er erfährt sofort, welche anderen Warteschlangenmanager im Cluster über vollständige Repositorys verfügen. Von dem Moment an kann der Warteschlangenmanager Informationen aus den Repositorys anfordern. Wenn der Warteschlangenmanager Informationen an das ausgewählte Repository sendet, sendet er gleichzeitig Informationen an ein anderes Repository (falls vorhanden).

Ein vollständiges Repository wird aktualisiert, wenn der Warteschlangenmanager mit dem Repository Informationen von einem der Warteschlangenmanager empfängt, die mit ihm verbunden sind. Die neuen Informationen werden auch an ein anderes Repository gesendet, um das Risiko einer Verzögerung zu verringern, falls der Repository-Warteschlangenmanager außer Betrieb ist. Da alle Informationen doppelt gesendet werden, müssen Duplikate in den Repositorys gelöscht werden. Jedes Informationselement besitzt eine Folgenummer, die von den Repositorys zur Erkennung von Duplikaten genutzt wird. Alle Repositorys tauschen Nachrichten untereinander aus, um immer auf demselben Stand zu sein.

Clusterwarteschlangen

Eine Clusterwarteschlange wird von einem Clusterwarteschlangenmanager anderen Warteschlangenmanagern im Cluster zur Verfügung gestellt.

Eine Clusterwarteschlangendefinition wird den anderen Warteschlangenmanagern im Cluster zugänglich gemacht. Die anderen Warteschlangenmanager im Cluster können ohne entsprechende Definition einer fernen Warteschlange Nachrichten in eine Clusterwarteschlange einreihen. Über eine Clusternamensliste kann eine Clusterwarteschlange in mehreren Clustern zugänglich gemacht werden.

Wenn eine Warteschlange zugänglich gemacht wird, können alle Warteschlangenmanager im Cluster Nachrichten in diese Warteschlange einreihen. Um eine Nachricht einzureihen, muss der Warteschlangenmanager anhand der vollständigen Repositorys ermitteln, wo sich die Warteschlange befindet. Anschließend fügt der Warteschlangenmanager der Nachricht einige Routing-Informationen hinzu und stellt sie dann in eine Clusterübertragungswarteschlange.

Bei einer Clusterwarteschlange kann es sich um eine Warteschlange handeln, die von Mitgliedern einer Gruppe mit gemeinsamer Warteschlange in IBM MQ for z/OS gemeinsam genutzt wird.

Zugehörige Tasks

Clusterwarteschlangen definieren

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.

Clusterkanäle

In jedem vollständigen Repository definieren Sie einen Clusterempfängerkanal sowie eine Gruppe von Clustersenderkanälen für die Verbindung mit allen anderen vollständigen Repositorys im Cluster manuell. Wenn Sie ein Teilrepository hinzufügen, definieren Sie einen Clusterempfängerkanal und einen einzelnen Clustersenderkanal manuell, der sich mit einem der vollständigen Repositorys verbindet. Weitere Clustersenderkanäle werden bei Bedarf automatisch vom Cluster definiert. Automatisch definierte Clustersenderkanäle erhalten ihre Attribute von der jeweiligen Definition für den entsprechenden Clusterempfängerkanal im Empfangswarteschlangenmanager.

Clusterempfängerkanal: CLUSRCVR

Eine CLUSRCVR-Kanaldefinition definiert das Ende eines Kanals, an dem ein Clusterwarteschlangenmanager Nachrichten von anderen Warteschlangenmanagern im Cluster empfangen kann.

Sie müssen für jeden Cluster-Warteschlangenmanager mindestens einen CLUSRCVR-Kanal definieren. Durch die Definition des CLUSRCVR-Kanals zeigt der Warteschlangenmanager den anderen Cluster-Warteschlangenmanagern an, dass er für Nachrichten empfangsbereit ist.

Eine CLUSRCVR-Kanaldefinition ermöglicht anderen Warteschlangenmanagern außerdem die automatische Erstellung entsprechender Clustersenderkanaldefinitionen. Weitere Informationen hierzu finden Sie im Abschnitt „Automatisch definierte Clustersenderkanäle” auf Seite 65 dieses Artikels.

Clustersenderkanal: CLUSSDR

Sie definieren manuell einen CLUSSDR -Kanal von jedem vollständigen Repository-WS-Manager zu jedem anderen Repository-WS-Manager im Cluster, der vollständig in Repository enthalten ist. Alle Aktualisierungen, die von den vollständigen Repositorys ausgetauscht werden, fließen exklusiv über diese Kanäle. Durch die manuelle Definition dieser Kanäle steuern Sie explizit das Netz aus vollständigen Repositorys.

Wenn Sie einem Cluster ein Teilrepository hinzufügen, definieren Sie einen einzelnen CLUSSDR-Kanal manuell, der sich mit einem der vollständigen Repositorys verbindet. Welches vollständige Repository Sie wählen, macht kaum einen Unterschied, da nach der Herstellung des ersten Kontakts bei Bedarf weitere Cluster-Warteschlangenmanager-Objekte (einschließlich der CLUSSDR-Kanäle) automatisch für Ihren Warteschlangenmanager definiert werden. Dadurch kann Ihr Warteschlangenmanager Clusterinformationen an jedes beliebige vollständige Repository und Nachrichten an jeden beliebigen Warteschlangenmanager im Cluster senden.

Wie im Abschnitt dieses Artikels erläutert, basieren automatisch definierte Senderkanäle auf der Konfiguration des Clusterempfängerkanals. Daher sollten alle Kanaleigenschaften, die Sie für Clusterkanäle festlegen, auf den entsprechenden CLUSSDR- und Clusterempfängerkanälen identisch sein oder nur für Clusterempfängerkanäle festgelegt werden.

Sie sollten CLUSSDR-Kanäle nur aus den zuvor beschriebenen Gründen manuell definieren. Definieren Sie diese also nur, um eine Erstverbindung zwischen einem Teilrepository und einem vollständigen Repository herzustellen, oder um zwei vollständige Repositorys miteinander zu verbinden. Eine manuelle Konfiguration eines CLUSSDR-Kanals, der sich mit einem Teilrepository oder einem Warteschlangenmanager verbindet, der nicht zum Cluster gehört, führt zur Ausgabe von Fehlernachrichten, z. B. AMQ9427 und AMQ9428. In bestimmten Situationen ist dies jedoch als temporäre Lösung unvermeidlich (z. B. bei der Änderung des Standorts eines vollständigen Repositorys). Allerdings sollte die manuelle Definition so bald wie möglich gelöscht werden.

Automatisch definierte Clustersenderkanäle

Wenn Sie einem Cluster ein Teilrepository hinzufügen, definieren Sie normalerweise nur zwei Clusterkanäle auf dem Warteschlangenmanager manuell:

- Einen Clustersenderkanal (CLUSSDR) zu einem vollständigen Warteschlangenmanager-Repository für den Cluster.
- Einen Clusterempfängerkanal (CLUSRCVR).

Der von Ihnen definierte CLUSSDR-Kanal ermöglicht es dem Warteschlangenmanager, den ersten Kontakt mit dem Cluster herzustellen. Nachdem der erste Kontakt hergestellt wurde, werden weitere CLUSSDR-Kanäle bei Bedarf automatisch vom Cluster definiert.

Ein automatisch definierter CLUSSDR-Kanal erhält seine Attribute von der entsprechenden CLUSRCVR-Kanaldefinition auf dem empfangenden Warteschlangenmanager. Selbst wenn ein manuell definierter CLUSSDR-Kanal vorhanden ist, werden die Attribute aus dem automatisch definierten CLUSSDR-Kanal verwendet. Angenommen, Sie definieren beispielsweise einen CLUSRCVR-Kanal ohne Angabe einer Portnummer im Parameter **CONNAME** und definieren manuell einen CLUSSDR-Kanal, der eine Portnummer angibt. Wenn der automatisch definierte CLUSSDR-Kanal den manuell definierten Kanal ersetzt, ist die Portnummer (die dem Kanal CLUSRCVR entnommen wird) nicht mehr belegt. Es wird die Standardportnummer verwendet und der Kanal fällt aus.

Falls Konfigurationsunterschiede zwischen einem manuell definierten CLUSSDR-Kanal und der entsprechenden CLUSRCVR-Kanaldefinition bestehen, werden manche Unterschiede sofort wirksam (zum Beispiel die Parameter für den Lastausgleich), während andere erst beim Neustart des Kanals wirksam werden (zum Beispiel die TLS-Konfiguration).

Um Unklarheiten zu vermeiden, beachten Sie möglichst die folgenden Richtlinien:

- Definieren Sie nur CLUSSDR-Kanäle, die auf vollständige Repositorys verweisen, manuell.
- Wenn Sie über manuell definierte CLUSSDR-Kanäle verfügen, konfigurieren Sie diese so, dass sie genau mit der entsprechenden CLUSRCVR-Kanaldefinition auf dem empfangenden Warteschlangenmanager übereinstimmen.

Siehe auch Mit automatisch definierten Kanälen arbeiten.

Zugehörige Konzepte

Mit automatisch definierten Kanälen arbeiten
Arbeiten mit Clusterübertragungswarteschlangen und Clustersenderkanälen

Zugehörige Tasks
Neuen Cluster einrichten
WS-Manager zu einem Cluster hinzufügen

Clusterthemen

Clusterthemen sind Verwaltungsthemen, für die das Attribut **`cluster`** definiert ist. Informationen zu Clusterthemen werden an alle Clustermitglieder übertragen und mit lokalen Themen zu warteschlangenmanagerübergreifenden Thementeilbereichen verbunden. Damit können Nachrichten, die auf einem Warteschlangenmanager zu einem Thema veröffentlicht werden, an die Subskriptionen anderer Warteschlangenmanager im Cluster übermittelt werden.

Wenn Sie ein Clusterthema für einen Warteschlangenmanager definieren, wird diese Clusterthemendefinition an die Warteschlangenmanager mit vollständigem Repository gesendet. Anschließend leiten die vollständigen Repositorys die Clusterthemendefinition an alle Warteschlangenmanager im Cluster weiter, sodass das Clusterthema für alle Bereitsteller und Subskribenten verfügbar ist, die in einem Warteschlangenmanager im Cluster vorhanden sind. Der Warteschlangenmanager, in dem ein Clusterthema erstellt wird, wird als Clusterthemenhost bezeichnet. Das Clusterthema kann von allen Warteschlangenmanagern im Cluster verwendet werden; alle Änderungen an einem Clusterthema müssen jedoch in dem Warteschlangenmanager vorgenommen werden, in dem das Thema definiert ist (d. h. im Clusterthemenhost); anschließend wird die Änderung über die vollständigen Repositorys an alle Clustermitglieder weitergegeben.

Sie finden Informationen zur Konfiguration von Cluster-Topics für das *direkte Routing* oder *Topic-Host-Routing* sowie zur Cluster-Topic-Vererbung und Platzhaltersubskriptionen im Abschnitt Cluster-Topics definieren.

Informationen zu den Befehlen, mit denen Clusterthemen angezeigt werden, finden Sie in den zugehörigen Informationen.

Zugehörige Konzepte
Mit Verwaltungsthemen arbeiten
Mit Subskriptionen arbeiten
Zugehörige Verweise
ANZEIGETHEMA
ANZEIGETPSTATUS
ANZEIGEUNTERGEORDNET

Standardclusterobjekte

Multi Auf Multiplatforms-Systemen gehören die Standardclusterobjekte zur Gruppe der Standardobjekte, die beim Definieren eines Warteschlangenmanagers automatisch erstellt werden. z/OS Unter z/OS sind die Definitionen zu Standardclusterobjekten in den Beispielen für die Anpassung enthalten.

Anmerkung: Sie können die Standardkanaldefinitionen auf dieselbe Weise wie jede andere Kanaldefinition durch die Ausführung von WebSphere MQ-Scriptbefehlen oder PCF-Befehlen ändern. Ändern Sie die Standardwarteschlangendefinitionen mit Ausnahme von `SYSTEM.CLUSTER.HISTORY.QUEUE` nicht.

`SYSTEM.CLUSTER.COMMAND.QUEUE`
Jeder Warteschlangenmanager in einem Cluster besitzt eine lokale Warteschlange namens `SYSTEM.CLUSTER.COMMAND.QUEUE`, über die Nachrichten an das vollständige Repository übertragen werden. Die Nachrichten enthalten neue oder geänderte Informationen über den Warteschlangenmanager selbst oder Anforderungen für Informationen über andere Warteschlangenmanager. `SYSTEM.CLUSTER.COMMAND.QUEUE` ist normalerweise leer.

`SYSTEM.CLUSTER.HISTORY.QUEUE`
Jeder Warteschlangenmanager in einem Cluster verfügt über eine lokale Warteschlange namens `SYSTEM.CLUSTER.HISTORY.QUEUE`. `SYSTEM.CLUSTER.HISTORY.QUEUE` wird verwendet, um das Protokoll der Clusterstatusinformationen zu Servicezwecken zu speichern.

In den Standardobjekteinstellungen ist SYSTEM.CLUSTER.HISTORY.QUEUE auf PUT (ENABLED) gesetzt. Um die Erfassung von Protokolldaten zu unterdrücken, ändern Sie die Einstellung in PUT (DISABLED).

SYSTEM.CLUSTER.REPOSITORY.QUEUE
Jeder Warteschlangenmanager in einem Cluster verfügt über eine lokale Warteschlange namens SYSTEM.CLUSTER.REPOSITORY.QUEUE. In dieser Warteschlange werden alle Informationen über das vollständige Repository gespeichert. Sie ist normalerweise nicht leer.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
Jeder Warteschlangenmanager verfügt über eine Definition für eine lokale Warteschlange namens SYSTEM.CLUSTER.TRANSMIT.QUEUE. SYSTEM.CLUSTER.TRANSMIT.QUEUE ist die Standardübertragungswarteschlange für alle Nachrichten an alle Warteschlangen und Warteschlangenmanager innerhalb von Clustern. Sie können die Standardübertragungswarteschlange für jeden Clustersenderkanal in SYSTEM.CLUSTER.TRANSMIT. *ChannelName*ändern, indem Sie das Warteschlangenmanagerattribut DEFCLXQ ändern. Die Warteschlange SYSTEM.CLUSTER.TRANSMIT.QUEUE kann nicht gelöscht werden. Es wird auch verwendet, um Berechtigungsprüfungen festzulegen, ob die verwendete Standardübertragungswarteschlange SYSTEM.CLUSTER.TRANSMIT.QUEUE oder SYSTEM.CLUSTER.TRANSMIT. *ChannelName*ist.

SYSTEM.DEF.CLUSRCVR
In jedem Cluster gibt es eine Standarddefinition für den Clusterempfängerkanal (CLUSRCVR) mit dem Namen SYSTEM.DEF.CLUSRCVR. SYSTEM.DEF.CLUSRCVR liefert Standardwerte für alle Attribute, die Sie bei der Erstellung eines Clusterempfängerkanals für einen Warteschlangenmanager im Cluster nicht angeben.

SYSTEM.DEF.CLUSSDR
In jedem Cluster gibt es eine Standarddefinition für den Clustersenderkanal (CLUSSDR) mit dem Namen SYSTEM.DEF.CLUSSDR. SYSTEM.DEF.CLUSSDR liefert Standardwerte für alle Attribute, die Sie bei der Erstellung eines Clustersenderkanals für einen Warteschlangenmanager im Cluster nicht angeben.

Zugehörige Konzepte
Mit Standardclusterobjekten arbeiten

Publish/Subscribe-Messaging

Das Publish/Subscribe-Messaging ermöglicht die Entkopplung des Informationsanbieters von den Informationsempfängern. Die Sender- und Empfängeranwendungen müssen untereinander keine Informationen über die gesendeten und empfangenen Nachrichten austauschen.

Bevor eine Punkt-zu-Punkt-Anwendung in IBM MQ eine Nachricht an eine andere Anwendung senden kann, werden Informationen zu dieser Anwendung benötigt. Beispielsweise ist der Name der Warteschlange erforderlich, an die die Information gesendet werden soll, und möglicherweise muss auch ein Warteschlangenmanagername angegeben werden.

Durch IBM MQ Publish/Subscribe entfällt für eine Anwendung die Notwendigkeit, Informationen zu der Zielanwendung zu haben. Die sendende Anwendung muss lediglich Folgendes ausführen:

- *Einreihen* einer IBM MQ -Nachricht, die die Informationen enthält, die die Anwendung wünscht.
- Die Nachricht einem Thema zuweisen, das den Betreff der Informationen angibt
- IBM MQ die Verteilung dieser Informationen überlassen

Ebenso muss auch die Zielanwendung keine Angaben zur Quelle der empfangenen Informationen haben.

In folgender Abbildung wird das einfachste Publish/Subscribe-System gezeigt. Es gibt einen Publisher, einen Warteschlangenmanager und einen Subskribenten. Eine Subskription wird von einem Subskribenten auf einem Warteschlangenmanager vorgenommen, eine Veröffentlichung wird vom Publisher an den Warteschlangenmanager gesendet und die Veröffentlichung wird anschließend vom Warteschlangenmanager an den Subskribenten weitergeleitet.

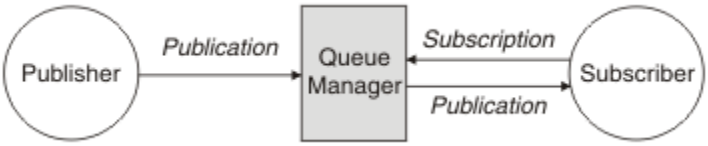


Abbildung 17. Einfache Publish/Subscribe-Konfiguration

Ein typisches Publish/Subscribe-System verfügt über mehrere Publisher und mehrere Subskribenten zu zahlreichen Themen und oft auch über mehr als einen Warteschlangenmanager. Eine Anwendung kann gleichzeitig Publisher und Subskribent sein.

Ein weiterer bedeutender Unterschied zwischen Publish/Subscribe- und Punkt-zu-Punkt-Messaging besteht darin, dass eine an eine Punkt-zu-Punkt-Warteschlange gesendete Nachricht nur von einer einzigen konsumierenden Anwendung verarbeitet wird. Eine für ein Publish/Subscribe-Thema veröffentlichte Nachricht, an der mehrere Subskribenten ihr Interesse gemeldet haben, wird von jedem interessierten Subskribenten verarbeitet.

Publish/Subscribe-Komponenten

Publish/Subscribe ist der Mechanismus, über den Subskribenten von Publishern Informationen in Form von Nachrichten erhalten. Die Interaktionen zwischen Publishern und Subskribenten werden von Warteschlangenmanagern über Standardfunktionen von IBM MQ gesteuert.

Ein typisches Publish/Subscribe-System verfügt über mehrere Publisher und mehrere Subskribenten zu zahlreichen Themen und oft auch über mehr als einen Warteschlangenmanager. Eine Anwendung kann gleichzeitig Publisher und Subskribent sein.

Der Bereitsteller von Informationen wird als *Publisher* bezeichnet. Publisher stellen Informationen zu einem Thema bereit, ohne irgendetwas über die Anwendungen, für die diese Informationen bestimmt sind, wissen zu müssen. Publisher erstellen diese Informationen in Form von Nachrichten, so genannten *Veröffentlichungen*, deren Thema sie definieren und die sie veröffentlichen.

Der Konsument der Informationen wird als *Subskribent* bezeichnet. Subskribenten erstellen *Subskriptionen*, die das Thema beschreiben, an dem sie interessiert sind. Eine Subskription bestimmt also, welche Veröffentlichungen an den Subskribenten weitergeleitet werden. Subskribenten können mehrere Subskriptionen erstellen und Informationen von vielen verschiedenen Publishern erhalten.

Veröffentlichte Informationen werden in einer IBM MQ-Nachricht gesendet, wobei der Inhalt dieser Informationen durch das *Thema* angedeutet wird. Das Thema wird bei der Veröffentlichung der Informationen vom Publisher festgelegt. Der Subskribent wiederum gibt die Themen an, zu denen er Veröffentlichungen erhalten möchte. Der Subskribent erhält nur Informationen zu den von ihm subskribierten Themen.

Durch Themen können beim Publish/Subscribe-Messaging Informationsbereitsteller von den Konsumenten der Informationen getrennt werden, denn hier ist es im Gegensatz zum Punkt-zu-Punkt-Messaging nicht erforderlich, das Ziel einer Nachricht anzugeben.

Die Interaktionen zwischen Publishern und Subskribenten werden von einem Warteschlangenmanager gesteuert. Der Warteschlangenmanager empfängt die Nachrichten der Publisher und die Subskriptionen (zu bestimmten Themen) der Subskribenten. Die Aufgabe des Warteschlangenmanagers ist es, die veröffentlichten Nachrichten an die Subskribenten weiterzuleiten, die ihr Interesse an den Themen der Nachrichten registriert haben.

Die Nachrichten werden mithilfe der Standardfunktionen von IBM MQ verteilt. Ihre Anwendungen können daher alle Funktionen verwenden, die vorhandenen IBM MQ-Anwendungen zur Verfügung stehen. Für die zuverlässige Nachrichtenübermittlung können Sie also auch persistente Nachrichten verwenden. Ebenso können Ihre Nachrichten Teil einer transaktionsorientierten Arbeitseinheit sein, durch die sichergestellt wird, dass Nachrichten nur dann dem Subskribenten zugestellt werden, wenn sie vom Publisher festgeschrieben wurden.

Veröffentlichungskomponenten und Veröffentlichungen

In der Publish/Subscribe-Funktion von IBM MQ bezeichnet eine Veröffentlichungskomponente eine Anwendung, die einem Warteschlangenmanager Informationen zu einem bestimmten Thema in Form einer IBM MQ-Standardnachricht zur Verfügung stellt, die "Veröffentlichung" genannt wird. Eine Veröffentlichungskomponente kann Informationen zu mehreren Themen veröffentlichen.

Veröffentlichungskomponenten verwenden das Verb MQPUT, um eine Nachricht in ein zuvor geöffnetes Thema einzureihen. Diese Nachricht ist eine Veröffentlichung. Der lokale Warteschlangenmanager leitet die Veröffentlichung dann an alle Subskribenten weiter, die Subskriptionen zum Thema der Veröffentlichung eingerichtet haben. Eine veröffentlichte Nachricht kann von mehreren Subskribenten gelesen werden.

Zusätzlich zum Verteilen von Veröffentlichungen an alle lokalen Subskribenten mit entsprechenden Subskriptionen kann ein Warteschlangenmanager eine Veröffentlichung auch an beliebige andere Warteschlangenmanager weiterleiten, die entweder direkt oder über ein Netz von Warteschlangenmanagern, die Subskribenten für dieses Thema verwalten, mit ihm verbunden sind.

In einem Publish/Subscribe-Netz von IBM MQ kann eine veröffentlichende Anwendung auch ein Subskribent sein.

Veröffentlichungen unter Synchronisationspunkt

Veröffentlichungskomponenten können MQPUT- oder MQPUT1-Aufrufe im Rahmen einer Synchronisationspunktverarbeitung ausgeben, um alle Nachrichten einzuschließen, die in einer Arbeitseinheit an Subskribenten geliefert werden. Wenn die Option MQPMO_RETAIN oder die Themabereitstellungsoptionen NPMSGDLV und PMSGDLV mit dem Wert ALL oder ALLDUR angegeben werden, verwendet der Warteschlangenmanager interne MQPUT- oder MQPUT1-Aufrufe unter Synchronisationspunktverarbeitung innerhalb des Geltungsbereichs des MQPUT- oder MQPUT1-Aufrufs der Veröffentlichungskomponente.

Status- und Ereignisinformationen

Bei Veröffentlichungen kann es sich um Statusveröffentlichungen (z. B. der aktuelle Aktienkurs) oder um Ereignisveröffentlichungen (z. B. ein Aktienverkauf) handeln.

Statusveröffentlichungen

Statusveröffentlichungen enthalten Informationen zum aktuellen Status oder Zustand von etwas, beispielsweise den Kurs einer Aktie oder den aktuellen Stand eines Fußballspiels. Sobald eine Änderung eintritt (sich also beispielsweise der Aktienpreis oder der Punktestand ändert), wird die vorherige Statusinformation nicht mehr benötigt, da sie durch die neue Information ersetzt wird.

Ein Subskribent möchte zu Beginn seiner Subskription die aktuelle Version der Statusinformationen erhalten. Diese möchte er aktualisiert haben, sobald sich der Status ändert.

Eine Veröffentlichung mit Statusinformationen wird meist als ständige Veröffentlichung veröffentlicht. Ein neuer Subskribent möchte die aktuellen Statusinformationen selbstverständlich sofort erhalten und nicht auf ein Ereignis warten müssen, das die erneute Veröffentlichung der Informationen auslöst. Subskribenten erhalten die ständige Veröffentlichung eines Themas automatisch bei der Subskription, es sei denn, der Subskribent verwendet die Option MQSO_PUBLICATIONS_ON_REQUEST oder MQSO_NEW_PUBLICATIONS_ONLY.

Ereignisveröffentlichungen

Ereignisveröffentlichungen enthalten Informationen zu einzelnen Ereignissen, beispielsweise zu einem Aktienhandel oder einem gefallenen Tor in einem Fußballspiel. Jedes Ereignis ist unabhängig von anderen Ereignissen.

Ein Subskribent möchte die Informationen zu einem Ereignis sofort erhalten, wenn es eintritt.

Ständige Veröffentlichungen

Eine Veröffentlichung wird nach dem Senden an alle interessierten Subskribenten standardmäßig gelöscht. Für den Fall, dass später weitere Subskribenten ihr Interesse an der Veröffentlichung anmelden, kann der Publisher aber auch festlegen, dass eine Kopie der Veröffentlichung aufbewahrt wird.

An alle interessierte Subskribenten gesendete Veröffentlichungen können im Falle von Ereignisdaten meist gelöscht werden, im Falle von Statusinformationen häufig aber nicht. Wird eine solche Nachricht aufbewahrt, müssen neue Subskribenten nicht auf eine erneute Veröffentlichung der Informationen warten, um die anfänglichen Statusinformationen zu erhalten. Ein Subskribent eines Aktienkurses erhielte den aktuellen Kurs dann sofort, ohne dass er auf eine Änderung (und damit erneute Veröffentlichung) des Aktienkurses warten müsste.

Der Warteschlangenmanager kann pro Thema nur eine Veröffentlichung aufbewahren. Die vorhandene ständige Veröffentlichung eines Themas wird daher gelöscht, sobald beim Warteschlangenmanager für das gleiche Thema eine neue ständige Veröffentlichung eingeht. Jedoch kann es passieren, dass die vorhandene Veröffentlichung nicht exakt gleichzeitig mit der Ankunft der neuen Veröffentlichung gelöscht wird. Daher sollten Sie ständige Veröffentlichungen zum gleichen Thema möglichst nur von einem Publisher aus senden.

Mit der Subskriptionsoption MQSO_NEW_PUBLICATIONS_ONLY können Subskribenten angeben, dass sie keine ständigen Veröffentlichungen erhalten wollen. Genauso ist es aber auch möglich, Duplikatkopien ständiger Veröffentlichungen anzufordern.

Eventuell möchten Sie in bestimmten Fällen keine ständigen Veröffentlichungen erstellen, nicht einmal für Statusinformationen. Dies kann zum Beispiel in den folgenden Situationen der Fall sein:

- Wenn alle Subskriptionen zu einem Thema abgeschlossen wurden, bevor zu diesem Thema Veröffentlichungen vorlagen, und Sie nicht davon ausgehen, dass neue Subskriptionen eingehen (bzw. keine neuen Subskriptionen zulassen), besteht kein Grund zur Aufbewahrung ständiger Veröffentlichungen, da alle Veröffentlichungen bereits bei ihrer ersten Veröffentlichung allen Subskribenten zugestellt wurden.
- Wenn Veröffentlichungen in einer sehr schnellen Abfolge versendet werden, zum Beispiel jede Sekunde, erhält ein neuer Subskribent bzw. ein Subskribent nach einem Recovery den aktuellen Status nahezu sofort nach dem Abschluss der Subskription. Es besteht für solche Veröffentlichungen daher kein Grund zur Aufbewahrung.
- Bei sehr großen Veröffentlichungen und/oder sehr vielen Themen benötigen Sie erheblichen Speicherplatz zur Aufbewahrung der ständigen Veröffentlichungen zu allen Themen. In einer Umgebung mit mehreren Warteschlangenmanagern werden ständige Veröffentlichungen auf allen Warteschlangenmanagern im Netz gespeichert, auf denen eine entsprechende Subskription vorliegt.

In die Überlegung, ob ständige Veröffentlichungen erstellt werden sollen, sollte auch einfließen, wie die abonnierenden Anwendungen nach einem Fehler wiederhergestellt werden. Wenn der Publisher keine ständigen Veröffentlichungen verwendet, muss die abonnierende Anwendung ihren aktuellen Status eventuell lokal speichern.

Zur Aufbewahrung einer ständigen Veröffentlichung verwenden Sie die Put-Option MQPMO_RETAIN. Wenn diese Option angegeben, aber eine ständige Veröffentlichung nicht möglich ist, wird die betreffende Nachricht nicht veröffentlicht und der Aufruf schlägt mit MQRC_PUT_NOT_RETAINED fehl.

Eine ständige Veröffentlichung ist durch die Nachrichteneigenschaft MQIsRetained gekennzeichnet. Die Persistenz einer Nachricht wird bei ihrer ursprünglichen Veröffentlichung festgelegt und bleibt danach unverändert.

Zugehörige Konzepte

Designüberlegungen zu ständigen Veröffentlichungen in Publish/Subscribe-Clustern

Veröffentlichungen unter Synchronisationspunkt

In der Publish/Subscribe-Funktion von IBM MQ kann der Synchronisationspunkt von Publishern oder intern vom Warteschlangenmanager verwendet werden.

Publisher verwenden den Synchronisationspunkt, wenn sie MQPUT/MQPUT1-Aufrufe mit der Option MQPMO_SYNCPOINT absetzen. Alle Nachrichten, die an Subskribenten zugestellt werden, werden bei der Berechnung der maximalen Anzahl von nicht festgeschriebenen Nachrichten in einer Arbeitseinheit berücksichtigt. Dieser Grenzwert wird über das Warteschlangenmanager-Attribut MAXUMSGS angegeben. Sobald der Grenzwert erreicht wird, empfängt der Publisher den Ursachencode 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED.

Wenn ein Publisher MQPUT/MQPUT1-Aufrufe unter Verwendung von MQPMO_NO_SYNCPOINT mit der Option MQPMO_RETAIN oder den Themenzustelloptionen NPMSGDLV/PMSGDLV mit den Werten ALL oder ALLDUR absetzt, verwendet der Warteschlangenmanager interne Synchronisationspunkte, um zu garantieren, dass die Nachrichten wie angefordert zugestellt werden. Der Publisher kann den Ursachencode 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED empfangen, wenn der Grenzwert im Bereich des Publisheraufrufs MQPUT/MQPUT1 erreicht wird.

Subskribenten und Subskriptionen

Beim Publish/Subscribe von IBM MQ ist ein Subskribent eine Anwendung, die von einem Warteschlangenmanager in einem Publish/Subscribe-Netz Informationen zu einem bestimmten Thema anfordert. Ein Subskribent kann Nachrichten zu einem oder mehreren Themen von einem oder mehreren Publishern erhalten.

Subskriptionen können manuell mit einem WebSphere MQ-Scriptbefehl oder durch eine Anwendung erstellt werden. Diese Subskriptionen werden dem lokalen Warteschlangenmanager mit Informationen zu den Veröffentlichungen übergeben, die der Subskribent erhalten möchte:

- Das Thema, an dem der Subskribent interessiert ist. Wenn Platzhalter verwendet werden, können dies auch mehrere Themen sein.
- Eine optionale Auswahlzeichenfolge, die auf die veröffentlichten Nachrichten angewendet wird.
- Eine Kennung für eine Warteschlange (die *Subskribentenwarteschlange*), in die die ausgewählten Veröffentlichungen eingereiht werden sollen, sowie die optionale CorrelId.

Der lokale Warteschlangenmanager speichert die Subskriptionsinformationen und untersucht diese, sobald er eine Veröffentlichung erhält, um festzustellen, ob ihm eine Subskription mit dem Thema und der Auswahlzeichenfolge dieser Veröffentlichung vorliegt. Bei einer übereinstimmenden Subskription leitet der Warteschlangenmanager die Veröffentlichung an die entsprechende Subskribentenwarteschlange weiter. Die Informationen, die ein Warteschlangenmanager zu Subskriptionen speichert, können mit den Befehlen DIS SUB und DIS SBSTATUS angezeigt werden.

Eine Subskription wird nur im Falle der folgenden Ereignisse gelöscht:

- Der Subskribent hebt die Subskription mit dem Aufruf MQCLOSE auf (nur bei nicht persistenten Subskriptionen).
- Die Subskription läuft ab.
- Die Subskription wird vom Systemadministrator mit dem Befehl DELETE SUB gelöscht.
- Die Subskribentenanwendung wird beendet (nur bei nicht persistenten Subskriptionen).
- Der Warteschlangenmanager wird gestoppt oder neu gestartet (nur bei nicht persistenten Subskriptionen).

Achten Sie darauf, dass Sie beim Abrufen von Nachrichten im MQGET-Aufruf die richtigen Optionen angeben. Wenn Ihre Anwendung nur Nachrichten für eine Subskription verarbeitet, sollten Sie mindestens `get-by-correlid` verwenden, wie im C-Beispielprogramm `amqssbxa.c` und unter Nicht verwalteter MQ-Subskribent veranschaulicht. Die zu verwendende **`CorrelId`** wird von MQSUB im MQSD zurückgegeben.**`SubCorrelId`** .

Zugehörige Konzepte

Klone und gemeinsam genutzte Subskriptionen

Zugehörige Verweise

Beispiele für Definition der Eigenschaft sharedSubscription

Verwaltete Warteschlangen und Publish/Subscribe

Wenn Sie eine Subskription erstellen, können Sie wählen, ob Sie die verwaltete Warteschlangensteuerung verwenden möchten. Falls Sie die verwaltete Warteschlangensteuerung verwenden, wird bei der Erstellung der Subskription automatisch eine Subskriptionswarteschlange erstellt. Verwaltete Warteschlangen werden entsprechend der Persistenz der Subskription automatisch bereinigt. Wenn Sie verwaltete Warteschlangen verwenden, brauchen Sie sich keine Gedanken mehr darüber zu machen, ob auch schon Warteschlangen für die Veröffentlichungen erstellt sind. Ebenso werden nicht konsumierte Veröffentlichungen automatisch aus den Subskribentenwarteschlangen gelöscht, sobald die Verbindung einer nicht persistenten Subskription geschlossen wird.

Muss für eine Anwendung keine bestimmte Warteschlange als Subskribentenwarteschlange verwendet werden (Ziel für die Veröffentlichungen, die die Anwendung subskribiert hat), können Sie für die Anwendung mit der Subskriptionsoption MQSO_MANAGED festlegen, dass sie *verwaltete Subskriptionen* verwendet. Bei der Erstellung einer verwalteten Subskription gibt der Warteschlangenmanager dem Subskribenten eine Objektkennung für eine Subskribentenwarteschlange zurück, die der Warteschlangenmanager erstellt und in die er die Veröffentlichungen für diese Subskription einreiht. Das ist darauf zurückzuführen, dass bei einer *verwalteten Subskription* IBM MQ die Subskription bearbeitet. Mithilfe dieser Objektkennung können Sie die Warteschlange suchen, ihren Inhalt abrufen und sie untersuchen. Deren Attribute können Sie allerdings nicht einstellen, es sei denn, Sie haben explizit Zugriff auf temporäre dynamische Warteschlangen.

Die Persistenz einer Subskription bestimmt, ob die verwaltete Warteschlange erhalten bleibt, wenn die Verbindung zwischen der abonnierenden Anwendung und dem Warteschlangenmanager getrennt wird.

Verwaltete Subskriptionen sind besonders praktisch in Verbindung mit nicht persistenten Subskriptionen, da andernfalls nach einer Beendigung der Verbindung der abonnierenden Anwendung nicht konsumierte Nachrichten in der Subskribentenwarteschlange verbleiben und auf dem Warteschlangenmanager für immer Speicherplatz belegen würden. Bei einer verwalteten Subskription ist die verwaltete Warteschlange hingegen eine temporäre dynamische Warteschlange und wird als solche zusammen mit nicht konsumierten Nachrichten gelöscht, wenn die Verbindung aus einer der folgenden Gründe getrennt wird:

- MQCLOSE wird mit MQCO_REMOVE_SUB ausgeführt und der verwaltete Hobj wird geschlossen.
- Eine Verbindung geht an eine Anwendung verloren, die eine nicht persistente Subskription (MQSO_NON_DURABLE) verwendet.
- Eine Subskription wird entfernt, da sie abgelaufen ist und der verwaltete Hobj geschlossen wurde.

Verwaltete Subskriptionen können auch bei persistenten Subskriptionen verwendet werden. In diesem Fall möchten Sie aber vermutlich nicht konsumierte Nachrichten in der Subskribentenwarteschlange belassen, sodass diese nach einer Wiederherstellung der Verbindung abgerufen werden können. Aus diesem Grund werden verwaltete Warteschlangen für persistente Subskriptionen als permanente dynamische Warteschlangen erstellt und bleiben erhalten, wenn die Verbindung der abonnierenden Anwendung zum Warteschlangenmanager getrennt wird.

Sie können für Ihre Subskription ein Ablaufdatum festlegen. Auch wenn sie eine Verbindungstrennung übersteht, bleibt sie in diesem Fall nicht für immer erhalten.

Wenn Sie eine verwaltete Warteschlange löschen, erhalten Sie eine Fehlernachricht.

An die Namen der verwalteten Warteschlangen wird am Ende eine Zeitmarke angefügt. Auf diese Weise erhält jede Warteschlange einen eindeutigen Namen.

Subskriptionspermanenz

Subskriptionen können persistent oder nicht persistent sein. Die Persistenz einer Subskription bestimmt, was mit der Subskription geschieht, wenn die abonnierende Anwendung vom Warteschlangenmanager getrennt wird.

Permanente Subskriptionen

Permanente Subskriptionen bleiben erhalten, wenn die Verbindung der subskribierenden Anwendung zum Warteschlangenmanager geschlossen wird. Die Subskription bleibt in diesem Fall auch bei einer Trennung der Verbindung erhalten und kann bei einer Wiederherstellung der Verbindung mit dem bei

der Erstellung der Subskription zurückgegebenen **SubName** wieder von der abonnierenden Anwendung angefordert werden.

Bei der Erstellung einer persistenten Subskription ist ein Subskriptionsname (**SubName**) erforderlich. Subskriptionsnamen müssen innerhalb eines Warteschlangenmanagers eindeutig sein, sodass diese eine Subskription zweifelsfrei identifizieren. Diese Identifikation ist bei der Angabe einer Subskription erforderlich, die Sie wiederaufnehmen möchten, nachdem Sie deren Verbindung entweder absichtlich (mit der Option MQCO_KEEP_SUB) getrennt haben oder nachdem deren Verbindung vom Warteschlangenmanager getrennt wurde. Eine vorhandene Subskription können Sie mit dem Aufruf MQSUB mit der Option MQSO_RESUME wiederaufnehmen. Subskriptionsnamen werden auch angezeigt, wenn Sie den Befehl DISPLAY SBSTATUS mit SUBTYPE ALL oder ADMIN verwenden.

Wird eine persistente Subskription nicht mehr von der abonnierenden Anwendung benötigt, so kann diese mit dem Funktionsaufruf MQCLOSE und der Option MQCO_REMOVE_SUB bzw. manuell mit dem WebSphere MQ-Scriptbefehl DELETE SUB gelöscht werden.

Mit dem Themenattribut **DURSUB** können Sie angeben, ob für ein Thema permanente Subskriptionen eingerichtet werden können.

Während der Rückgabe eines MQSUB-Aufrufs mit der Option MQSO_RESUME wird der Subskriptionsablauf auf die ursprüngliche Ablaufzeit der Subskription gesetzt, nicht auf die verbleibende Ablaufzeit.

Ein Warteschlangenmanager sendet einer persistenten Subskription weiterhin Veröffentlichungen, selbst wenn zur Subskribentenanwendung keine Verbindung mehr besteht. Dies führt in der Subskribentenwarteschlange zu einem Rückstau. Die einfachste Methode, dies zu vermeiden, ist, wann immer möglich, die Verwendung von nicht persistenten Subskriptionen. Ist eine persistente Subskription absolut erforderlich, so lässt sich ein solcher Rückstau aber auch vermeiden, wenn die Subskription mit der Option Ständige Veröffentlichungen erstellt wird. Ein Subskribent kann dann mit dem Aufruf MQSUBRQ steuern, wann er Veröffentlichungen empfängt.

Nicht persistente Subskriptionen

Nicht persistente Subskriptionen bleiben nur so lange bestehen, wie die Verbindung einer Subskribentenanwendung zu einem Warteschlangenmanager geöffnet bleibt. Die Subskription wird entfernt, wenn die abonnierende Anwendung absichtlich oder durch eine Verbindungsunterbrechung vom Warteschlangenmanager getrennt wird. Beim Schließen der Verbindung werden die Informationen zur Subskription vom Warteschlangenmanager gelöscht. Die Subskription tritt dann auch bei der Anzeige der Subskriptionen mit dem Befehl DISPLAY SBSTATUS nicht mehr in Erscheinung. Nach der Beendigung der Subskription werden keine weiteren Nachrichten mehr in die Subskribentenwarteschlange gestellt.

Was bei nicht persistenten Subskriptionen mit noch nicht konsumierten Veröffentlichungen in der Subskribentenwarteschlange geschieht, wird wie folgt bestimmt:

- Wenn eine abonnierende Anwendung ein verwaltetes Ziel verwendet, werden alle noch nicht konsumierten Veröffentlichungen automatisch entfernt.
- Hat die abonnierende Anwendung bei der Subskription eine Kennung ihrer eigenen Subskribentenwarteschlange bereitgestellt, so werden nicht konsumierte Veröffentlichungen nicht automatisch entfernt. In diesem Fall ist die Anwendung selbst dafür verantwortlich, die Warteschlange zu bereinigen. Falls die Warteschlange auch von anderen Subskribenten oder von Punkt-zu-Punkt-Anwendungen genutzt wird, darf die Warteschlange vermutlich nicht vollständig geleert werden.

Auch wenn dies für nicht persistente Subskriptionen nicht erforderlich ist, verwendet der Warteschlangenmanager den Subskriptionsnamen, sofern er bereitgestellt wird. Subskriptionsnamen müssen innerhalb eines Warteschlangenmanagers eindeutig sein, sodass diese eine Subskription zweifelsfrei identifizieren.

Zugehörige Konzepte

Klone und gemeinsam genutzte Subskriptionen

Zugehörige Tasks

Gemeinsam genutzte JMS 2.0-Subskriptionen verwenden

Zugehörige Verweise
Beispiele für Definition der Eigenschaft sharedSubscription

Auswahlzeichenfolgen
Eine *Auswahlzeichenfolge* ist ein Ausdruck, der auf eine Veröffentlichung angewendet wird, um festzustellen, ob sie mit einer Subskription übereinstimmt. Auswahlzeichenfolgen können Platzhalterzeichen einschließen.

Beim Einrichten einer Subskription können Sie neben einem Thema auch eine Auswahlzeichenfolge angeben, um Veröffentlichungen anhand ihrer Nachrichteneigenschaften auszuwählen.

Die Auswahlzeichenfolge wird mit der Nachricht, so wie sie vom Publisher bereitgestellt wurde, abgeglichen, bevor diese für die Zustellung an die einzelnen Subskribenten modifiziert wird. Seien Sie vorsichtig, wenn Sie Felder in der Auswahlzeichenfolge verwenden, die möglicherweise im Rahmen des Veröffentlichungsvorgangs modifiziert werden. Dies gilt beispielsweise für die MQMD-Felder `UserIdentifier` (Benutzer-ID), `MsgId` (Nachrichten-ID) und `CorrelId` (Korrelations-ID).

Auswahlzeichenfolgen sollten auf keins der Nachrichteneigenschaftsfelder verweisen, die im Rahmen des Veröffentlichungsvorgangs vom Warteschlangenmanager hinzugefügt werden (siehe Publish/Subscribe-Nachrichteneigenschaften), außer auf das Nachrichteneigenschaftsfeld `MQTopicString`, die die Themenzeichenfolge für die Veröffentlichung enthält.

Zugehörige Konzepte
Regeln und Einschränkungen für Auswahlzeichenfolgen

Themen
Ein Thema ist der Betreff der Informationen, die in einer Publish/Subscribe-Nachricht veröffentlicht werden.

Nachrichten in Punkt-zu-Punkt-Systemen werden an eine bestimmte Zieladresse gesendet. Nachrichten in betreffbasierten Publish/Subscribe-Systemen werden auf Basis des Betreffs, der den Inhalt der Nachricht beschreibt, an Subskribenten gesendet. Bei inhaltsbasierten Systemen werden Nachrichten auf Basis des Nachrichteninhalts an Subskribenten gesendet.

Das Publish/Subscribe-System von IBM MQ ist ein betreffbasiertes Publish/Subscribe-System. Ein Publisher erstellt eine Nachricht und veröffentlicht sie mit einer Themenzeichenfolge, die dem Betreff der Veröffentlichung am besten entspricht. Um Veröffentlichungen zu empfangen, erstellt eine Subskribent eine Subskription mit einer Zeichenfolge mit Mustererkennung, um Veröffentlichungsthemen auszuwählen. Der Warteschlangenmanager liefert Veröffentlichungen an Subskribenten, deren Subskriptionen dem Veröffentlichungsthema entsprechen und für den Empfang von Veröffentlichungen autorisiert sind. Der Artikel „Themenzeichenfolgen” auf Seite 75 beschreibt die Syntax von Themenzeichenfolgen, die den Betreff einer Veröffentlichung angeben. Subskribenten erstellen zudem auch Themazeichenfolgen, um die zu empfangenden Themen auszuwählen. Die Themazeichenfolgen, die Subskribenten erstellen, können eines von zwei Platzhalterschemas zur Musterübereinstimmung mit den Themazeichenfolgen in Veröffentlichungen enthalten. Informationen zur Musterübereinstimmung finden Sie in „Platzhalterschemas” auf Seite 76.

Beim betreffbasierten Publish/Subscribe sind Publisher oder Administratoren für die Klassifizierung von Betreffs in Artikeln verantwortlich. Normalerweise werden Betreffs hierarchisch in Themenstrukturen organisiert. Mit dem Zeichen ' / ' werden in der Themazeichenfolge Unterthemen erstellt. Beispiele zu Themenstrukturen finden Sie in „Themenstrukturen” auf Seite 82. Themen sind Knoten in einer Themenstruktur. Themen können Blattknoten ohne weitere Unterthemen oder Zwischenknoten mit Unterthemen sein.

Parallel zur Verwaltung von Betreffs in hierarchischen Themenstrukturen können Sie administrativen Themenobjekten Themen zuordnen. Sie weisen einem Thema Attribute zu, z. B., ob das Thema in einem Cluster verteilt wird, indem Sie es einem administrativen Themenobjekt zuordnen. Die Zuordnung findet über die Benennung des Themas unter Verwendung des Attributs `TOPICSTR` des administrativen Themenobjekts statt. Wenn Sie ein administratives Themenobjekt nicht explizit einem Thema zuordnen, übernimmt das Thema die Attribute seines nächsten Vorgängers in der Themenstruktur, den Sie einem

administrativen Themenobjekt *zugeordnet* haben. Wenn Sie keine übergeordneten Themen definiert haben, übernimmt das Thema die Attribute von SYSTEM.BASE.TOPIC. Eine Beschreibung der administrativen Themenobjekte finden Sie in „Verwaltungsthemenobjekte” auf Seite 83.

Anmerkung: Selbst wenn Sie alle Attribute eines Themas von SYSTEM.BASE.TOPIC übernehmen, müssen Sie ein Stammthema für Ihre Themen definieren, das die Attribute von SYSTEM.BASE.TOPIC direkt übernimmt. Beispielsweise ist im Themenbereich 'US-Bundesstaaten' (USA/Alabama, USA/Alaska usw.) USA das Stammthema. Hauptzweck eines Stammthemas ist es, getrennte, nicht überlappende Themenbereiche zu erstellen, um zu verhindern, dass Veröffentlichungen den falschen Subskriptionen entsprechen. Zudem können Sie auf diese Weise die Attribute Ihres Stammthemas ändern, um den gesamten Themenbereich entsprechend zu ändern. Sie können beispielsweise den Namen für das Attribut **CLUSTER** festlegen.

Wenn Sie als Publisher oder Subskribent auf ein Thema verweisen, können Sie eine Themazeichenfolge angeben oder auf ein Themenobjekt verweisen. Sie können auch beides tun, womit die von Ihnen angegebene Themazeichenfolge ein Unterthema des Themenobjekts definiert. Der Warteschlangenmanager identifiziert das Thema, indem er die Themazeichenfolge an das Präfix der im Themaobjekt genannten Themazeichenfolge anhängt. Dabei fügt er ein weiteres ' / ' -Zeichen zwischen die beiden Themazeichenfolgen ein, z. B. *Themazeichenfolge/Objektzeichenfolge*. Unter „Themenzeichenfolgen kombinieren” auf Seite 80 finden Sie weitere Informationen hierzu. Die daraus resultierende Themazeichenfolge dient zur Ermittlung des Themas und dessen Zuordnung zu einem administrativen Themenobjekt. Das administrative Themenobjekt ist nicht zwingend dasselbe Themenobjekt wie das Themenobjekt, das dem Masterthema entspricht.

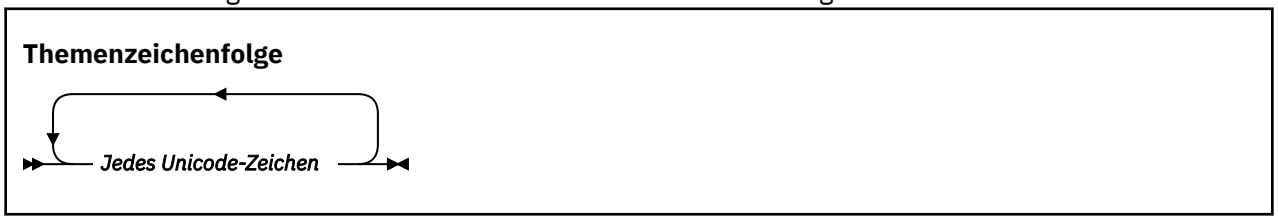
Beim inhaltsbasierten Publish/Subscribe definieren Sie, welche Nachrichten Sie empfangen möchten, indem Sie Auswahlzeichenfolgen angeben, die den Inhalt jeder Nachricht durchsuchen. IBM MQ stellt eine temporäre Form des inhaltsbasierten Publish/Subscribe bereit, bei dem Nachrichtenselektoren verwendet werden, die nicht den vollständigen Nachrichteninhalt, sondern die Nachrichteneigenschaften durchsuchen (siehe Selektoren). Normalerweise werden Nachrichtenselektoren so eingesetzt, dass zunächst ein Thema abonniert und die Auswahl dann mit einer numerischen Eigenschaft klassifiziert wird. Über den Selektor können Sie anzugeben, dass Sie nur an Werten in einem bestimmten Bereich interessiert sind. Dies ist mit Zeichen oder themenbasierten Platzhaltern nicht möglich. Wenn Sie auf Grundlage des gesamten Inhalts der Nachricht filtern möchten, müssen Sie IBM Integration Bus verwenden.

Themenzeichenfolgen

Kennsatzinformationen, die Sie mithilfe einer Themenzeichenfolge als ein Thema veröffentlichen. Abonnieren Sie Themengruppen, indem Sie entweder zeichenbasierte oder themenbasierte Platzhalterthemenzeichenfolgen verwenden.

Themen

Eine *Themenzeichenfolge* ist eine Zeichenfolge, die das Thema einer Publish/Subscribe-Nachricht angibt. Sie können beliebige Zeichen zum Erstellen einer Themenzeichenfolge verwenden.



Im Publish/Subscribe von IBM WebSphere MQ 7 haben drei Zeichen eine besondere Bedeutung. Sie sind zwar überall in einer Themenzeichenfolge zulässig, sollten aber mit Vorsicht verwendet werden. Die Verwendung der Sonderzeichen wird im Abschnitt „Themenbasiertes Platzhalterschema” auf Seite 77 erläutert.

Schrägstrich (/)

Dies ist das Trennzeichen für Themenebenen. Verwenden Sie das Zeichen ' / ', um eine Themenstruktur für das Thema zu erstellen.

Vermeiden Sie, wenn es möglich ist, leere Themenebenen (' / / '). Denn diese entsprechen Knoten in der Themenhierarchie ohne Themenzeichenfolge. Ein führender oder abschließender ' / ' in einer Themenzeichenfolge entspricht einem führenden oder abschließenden leeren Knoten und sollte ebenfalls vermieden werden.

Nummernzeichen (#)

Wird in Kombination mit ' / ' verwendet, um in Subskriptionen ein Platzhalterzeichen für mehrere Ebenen zu erstellen. Geben Sie acht bei der Verwendung einer Kombination aus '#' und ' / ' in Themenzeichenfolgen, die zum Benennen von veröffentlichten Themen dienen. „Beispiele für Themenzeichenfolgen" auf Seite 76 zeigt eine überlegte Verwendung von '#'.

Die Zeichenfolgen ' . . . /#/ . . . ', '#/ . . . ' und ' . . . /#' haben in Subskriptionsthemenzeichenfolgen eine besondere Bedeutung. Die Zeichenfolgen stimmen mit allen Themen auf einer oder mehreren Ebenen in der Themenhierarchie überein. Wenn Sie ein Thema mit einer dieser Folgen erstellt haben, könnten Sie dieses Thema deshalb nicht abonnieren, ohne auch alle Themen auf mehreren Ebenen in der Themenhierarchie zu abonnieren.

Pluszeichen (+)

Wird in Kombination mit ' / ' verwendet, um in Subskriptionen ein Platzhalterzeichen für eine einzelne Ebene zu erstellen. Geben Sie acht bei der Verwendung einer Kombination aus '+' und ' / ' in Themenzeichenfolgen, die zum Benennen von veröffentlichten Themen dienen.

Die Zeichenfolgen ' . . . /+/ . . . ', '+/ . . . ' und ' . . . /+' haben in Subskriptionsthemenzeichenfolgen eine besondere Bedeutung. Die Zeichenfolgen stimmen mit allen Themen auf einer einzelnen Ebene in der Themenhierarchie überein. Wenn Sie ein Thema mit einer dieser Folgen erstellt haben, könnten Sie dieses Thema deshalb nicht abonnieren, ohne auch alle Themen auf einer einzelnen Ebene in der Themenhierarchie zu abonnieren.

Beispiele für Themenzeichenfolgen

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

Zugehörige Verweise

TOPIC

Platzhalterschemas

Es gibt zwei Platzhalterschemas, die zum Abonnieren mehrerer Themen verwendet werden. Die Auswahl des Schemas ist eine Subskriptionsoption.

MQSO_WILDCARD_TOPIC
Wählen Sie zu abonnierende Themen mithilfe des themenbasierten Platzhalterschemas aus.

Dies ist der Standard, wenn kein Platzhalterschema explizit ausgewählt ist.

MQSO_WILDCARD_CHAR
Wählen Sie zu abonnierende Themen mithilfe des zeichenbasierten Platzhalterschemas aus.

Legen Sie eins der Schemas fest, indem Sie den Parameter **wschema** im Befehl DEFINE SUB angeben. Weitere Informationen finden Sie im Abschnitt DEFINE SUB.

Anmerkung: In Subskriptionen, die vor IBM WebSphere MQ 7.0 erstellt wurden, wird immer das zeichenbasierte Platzhalterschema verwendet.

Beispiele

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

Themenbasiertes Platzhalterschema
Themenbasierte Platzhalterzeichen ermöglichen es Subskribenten, mehrere Themen gleichzeitig zu abonnieren.

Themenbasierte Platzhalterzeichen sind eine leistungsfähige Funktion des Themensystems in IBM MQ Publish/Subscribe. Die Platzhalter für mehrere Ebenen bzw. für nur eine Ebene können für Subskriptionen, jedoch nicht innerhalb eines Themas durch den Publisher einer Nachricht verwendet werden.

Mithilfe des themenbasierten Platzhalterschemas können Sie Veröffentlichungen gruppiert nach Themenebene auswählen. Sie können für jede Ebene in der Themenhierarchie auswählen, ob die Zeichenfolge in der Subskription für die betreffende Themenebene mit der Zeichenfolge in der Veröffentlichung genau übereinstimmen muss oder nicht. Beispiel: Die Subskription `IBM/+/Results` wählt alle Themen aus.

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

Es gibt zwei Typen von Platzhalterzeichen:

Platzhalterzeichen für mehrere Ebenen

- Das Platzhalterzeichen für mehrere Ebenen wird in Subskriptionen verwendet. Bei Verwendung in einer Veröffentlichung wird es als Literal behandelt.
- Das Platzhalterzeichen `'#'` für mehrere Ebenen deckt alle Ebenen innerhalb eines Themas ab. Für das Themenstrukturbeispiel bedeutet dies, dass Sie bei einer Subskription von `'USA/Alaska/#'` Nachrichten zu den Themen `'USA/Alaska'` und `'USA/Alaska/Juneau'` empfangen.
- Das Platzhalterzeichen für mehrere Ebenen kann auf keine oder mehrere Ebenen zutreffen. Deshalb kann `'USA/#'` auch die einzelne Ebene `'USA'` abdecken, wobei `'#'` auf keine Ebene zutrifft. Das Trennzeichen für Themenebenen ist in diesem Kontext bedeutungslos, weil es keine zu trennenden Ebenen gibt.
- Das Platzhalterzeichen für mehrere Ebenen ist nur wirksam, wenn es allein steht oder neben dem Trennzeichen für Themenebenen. Deshalb sind `'#'` und `'USA/#'` gültige Themen, wobei das Zeichen `'#'` als Platzhalter behandelt wird. Aber obwohl `'USA#'` ebenfalls eine gültige Themenzeichenfolge ist, wird das Zeichen `'#'` nicht als Platzhalter angesehen und hat deshalb keine besondere Bedeutung. Weitere Informationen finden Sie im Abschnitt „Wenn themenbasierte Platzhalterzeichen unwirksam sind” auf Seite 79.

Platzhalterzeichen für einzelne Ebenen

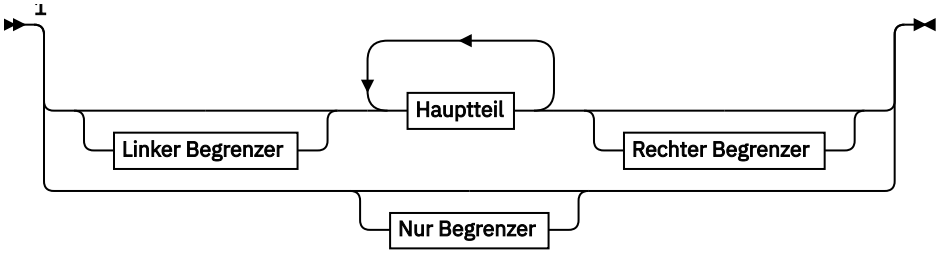
- Das Platzhalterzeichen für einzelne Ebenen wird in Subskriptionen verwendet. Bei Verwendung in einer Veröffentlichung wird es als Literal behandelt.
- Das Platzhalterzeichen `'+'` für einzelne Ebenen trifft nur auf eine einzige Themenebene zu. Beispielsweise deckt `'USA/+'` die Ebene `'USA/Alabama'` ab, aber nicht die Ebene `'USA/Alabama/Auburn'`. Da das Platzhalterzeichen für einzelne Ebenen nur für eine einzige Ebene zutrifft, deckt `'USA/+'` auch nicht die Ebene `'USA'` ab.
- Das Platzhalterzeichen für einzelne Ebenen kann auf jeder Ebene in der Themenstruktur und in Verbindung mit dem Platzhalterzeichen für mehrere Ebenen verwendet werden. Es muss neben dem Trennzeichen für Themenebenen angegeben werden, außer wenn es allein angegeben wird. Deshalb sind `'+'` und `'USA/+'` gültige Themen, wobei das Zeichen `'+'` als Platzhalter behandelt wird. Aber obwohl `'USA+'` ebenfalls eine gültige Themenzeichenfolge ist, wird das Zeichen `'+'` nicht als Platzhalter angesehen und hat deshalb keine besondere Bedeutung. Weitere Informationen finden Sie im Abschnitt „Wenn themenbasierte Platzhalterzeichen unwirksam sind” auf Seite 79.

Die Syntax für das themenbasierte Platzhalterschema umfasst keine Escapezeichen. Ob `'#'` und `'+'` als Platzhalterzeichen behandelt werden oder nicht, hängt von ihrem Kontext ab. Weitere Informationen finden Sie unter „Wenn themenbasierte Platzhalterzeichen unwirksam sind” auf Seite 79.

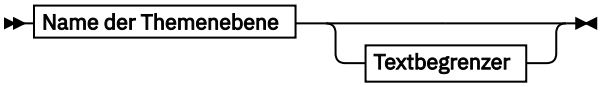
Anmerkung: Anfang und Ende einer Themenzeichenfolge werden auf besondere Weise behandelt. Wenn Sie `'$'` verwenden, um das Ende der Zeichenfolge anzugeben, ist `'$#/...'` ein Platzhalter für mehrere

Ebenen und '$/#/..'. ist ein leerer Knoten im Stammverzeichnis, gefolgt von einem Platzhalter für mehrere Ebenen.

Themenbasierte Platzhalterzeichenfolge

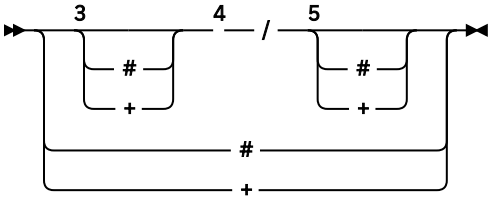


Hauptteil



Name der Themenebene

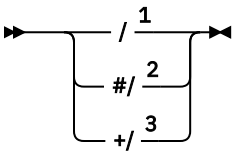
Nur Begrenzer



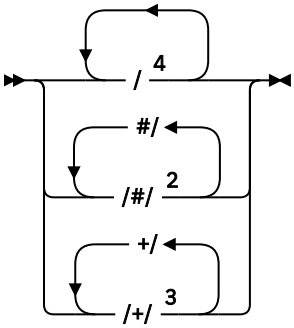
Anmerkungen:

[1] Eine Nullzeichenfolge oder leere Zeichenfolge als Themenzeichenfolge ist ungültig.
[2] Es wird empfohlen, keine *, ?, % in Ebenennamenzeichenfolgen zu verwenden, um die Kompatibilität zwischen zeichenbasierten und themenbasierten Platzhalterschemas zu gewährleisten.
[3] Diese Fälle entsprechen dem Muster von *Linker Begrenzer*.
[4] / ohne Platzhalterzeichen trifft auf ein einzelnes leeres Thema zu.
[5] Diese Fälle entsprechen dem Muster von *Rechter Begrenzer*.
[6] Trifft auf jedes Thema zu.
[7] Trifft auf jedes Thema zu, für das es nur eine Ebene gibt.

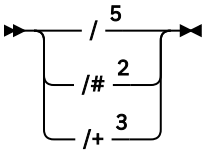
Linker Begrenzer



Textbegrenzer



Rechter Begrenzer



Anmerkungen:

[1] Die Themenzeichenfolge beginnt mit einem leeren Thema.
[2] Trifft auf keine oder mehrere Ebenen zu. Mehrere übereinstimmende Zeichenfolgen für mehrere Ebenen habe dieselbe Wirkung wie eine einzige übereinstimmende Zeichenfolge für mehrere Ebenen.
[3] Trifft auf genau eine Ebene zu.
[4] // ist ein leeres Thema - ein Themenobjekt ohne Themenzeichenfolge.
[5] Die Themenzeichenfolge endet mit einem leeren Thema.

Wenn themenbasierte Platzhalterzeichen unwirksam sind

Die Platzhalterzeichen '+' und '#' haben keine besondere Bedeutung, wenn sie in einer Themenebene mit anderen Zeichen (einschließlich sich selbst) gemischt werden.

Dies bedeutet, dass Themen, die in einer Themenebene das Zeichen '+' oder '#' zusammen mit anderen Zeichen enthalten, veröffentlicht werden können.

Betrachten Sie beispielsweise die beiden folgenden Themen:

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

Im ersten Beispiel werden die Zeichen '+' und '#' als Platzhalterzeichen behandelt und sind deshalb in einer Themenzeichenfolge, für die Veröffentlichungen erfolgen sollen, nicht gültig, wohingegen sie in einer Subskription gültig sind.

Im zweiten Beispiel werden die Zeichen '+' und '#' nicht als Platzhalterzeichen behandelt, weshalb für die Themenzeichenfolge sowohl Veröffentlichungen als auch Subskriptionen möglich sind.

Beispiele

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

Zeichenbasiertes Platzhalterschema
Mit dem zeichenbasierten Platzhalterschema können Sie Themen auf Grundlage des Abgleichs von konventionellen Zeichen auswählen.

Mit der Zeichenfolge '*' können Sie alle Themen auf mehreren Ebenen in einer Themenhierarchie auswählen. Die Verwendung von '*' im zeichenbasierten Platzhalterschema entspricht der Verwendung der themenbasierten Platzhalterzeichenfolge '#'

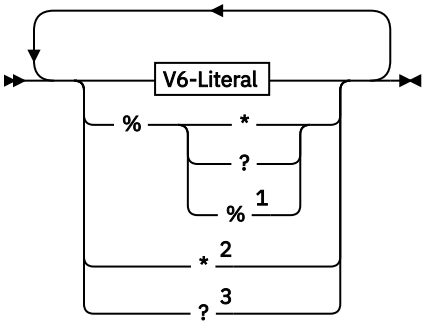
' *x*/*/*y* ' entspricht ' *x*/#/*y* ' im themenbasierten Schema und wählt alle Themen in der Themenhierarchie zwischen den Ebenen ' *x* und *y* 'aus, wobei ' *x* ' und ' *y* ' Themennamen sind, die nicht in der Gruppe der Ebenen enthalten sind, die vom Platzhalterzeichen zurückgegeben werden.

'/+/' im themenbasierten Schema hat keine exakte Entsprechung im zeichenbasierten Schema. 'IBM/*/Results' würde auch 'IBM/Patents/Software/Results'auswählen. Nur wenn die Gruppe der Themennamen in jeder Ebene der Hierarchie eindeutig ist, können Sie in jedem Fall Abfragen mit den beiden Schemas erstellen, die identische Übereinstimmungen ergeben.

Bei einer allgemeinen Verwendung haben '*' und '?' im zeichenbasierten Schema keine Entsprechungen im themenbasierten Schema. Das themenbasierte Schema führt keinen unvollständigen Abgleich mithilfe von Platzhaltern aus. Die zeichenbasierte Platzhaltersubskription 'IBM/*ware/Results' hat keine themenbasierte Entsprechung.

Anmerkung: Abgleiche mithilfe von Subskriptionen mit Platzhalterzeichen sind langsamer als Abgleiche mithilfe von themenbasierten Subskriptionen.

Zeichenbasierte Platzhalterzeichen



V6-Literal

Anmerkungen:

[1] Bedeutet "Folgendem Zeichen Escapezeichen voranstellen", damit es als Literal verwendet wird. Auf '%' muss entweder '*', '?' oder '%'folgen. Siehe „Beispiele für Themenzeichenfolgen” auf Seite 76.

[2] Bedeutet "Übereinstimmung mit null oder mehr Zeichen" in einer Subskription.

[3] Bedeutet "Übereinstimmung mit genau einem Zeichen" in einer Subskription.

Beispiele

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

Themenzeichenfolgen kombinieren

Wenn Sie Subskriptionen erstellen oder Themen öffnen, um Nachrichten für sie veröffentlichen zu können, kann die Themenzeichenfolge gebildet werden, indem zwei separate Unterthemenzeichenfolgen (oder "Unterthemen") kombiniert werden. Das eine Unterthema wird von der Anwendung oder vom Verwaltungsbefehl als Themenzeichenfolge bereitgestellt und das andere ist die Themenzeichenfolge, die einem Themenobjekt zugeordnet ist. Sie können eines der Unterthemen als Themenzeichenfolge verwenden oder die Unterthemen kombinieren, um einen neuen Themennamen zu bilden.

Wenn Sie beispielsweise eine Subskription mit dem MQSC-Befehl **DEFINE SUB**definieren, kann der Befehl entweder **TOPICSTR** (Themenzeichenfolge) oder **TOPICOBJ** (Themenobjekt) als Attribut oder beides zusammen verwenden. Wird nur **TOPICOBJ** angegeben, wird die Themenzeichenfolge, die diesem Themenobjekt zugeordnet ist, als Themenzeichenfolge verwendet. Wird nur **TOPICSTR** angegeben, wird eben diese als Themenzeichenfolge verwendet. Werden beide angegeben, werden sie zu einer einzigen Themenzeichenfolge im Format **TOPICOBJ** / **TOPICSTR** verkettet, wobei **TOPICOBJ** immer den ersten Teil bildet und beide Teile immer durch das Zeichen "/" getrennt sind.

Entsprechend wird in einem MQI-Programm von MQOPEN der vollständige Themenname erstellt. Er besteht aus zwei Feldern, die in Publish/Subscribe-MQI-Aufrufen verwendet werden, in der aufgeführten Reihenfolge:

1. Das Attribut **TOPICSTR** des Themenobjekts, das im Feld **ObjectName** benannt ist.
2. Der Parameter **ObjectString**, der das von der Anwendung bereitgestellte Unterthema definiert.

Die daraus resultierende Themenzeichenfolge wird im Parameter **ResObjectString** zurückgegeben.

Diese Felder gelten als vorhanden, wenn das erste Zeichen jedes Felds kein Leerzeichen oder Nullzeichen ist und die Feldlänge größer als null ist. Wenn nur eines der Felder vorhanden ist, wird es unverändert als Themenname verwendet. Wenn keines der beiden Felder einen Wert enthält, schlägt der Aufruf mit dem Ursachencode MQRC_UNKNOWN_OBJECT_NAME oder MQRC_TOPIC_STRING_ERROR fehl, falls der vollständige Themenname ungültig ist.

Wenn beide Felder vorhanden sind, wird das Zeichen "/" zwischen die beiden Elemente des entstandenen kombinierten Themennamens eingefügt.

Die folgende Tabelle enthält Beispiele für die Verkettung von Themenzeichenfolgen:

Tabelle 2. Beispiele für die Verkettung von Themenzeichenfolgen

TOPICSTR des Themenobjekts	**Themenzeichenfolge, die von der Anwendung oder dem Befehl DEFINE SUB bereitgestellt wird.**	**Vollständiger Themenname**	**Kommentar**
Fußball/Ergebnisse	''	Fußball/Ergebnisse	Es wird nur TOPICSTR des Themenobjekts verwendet.
''	Fußball/Ergebnisse	Fußball/Ergebnisse	Es wird nur ObjectString/TOPICSTR verwendet.
Fußball	Ergebnisse	Fußball/Ergebnisse	Am Verkettungspunkt wird das Zeichen "/" hinzugefügt.
Fußball	/Ergebnisse	Fußball//Ergebnisse	Zwischen den beiden Zeichenfolgen entsteht ein "leerer Knoten". Dies ist anders als bei "Fußball/Ergebnisse".
/Fußball	Ergebnisse	/Fußball/Ergebnisse	Das Thema beginnt mit einem "leeren Knoten". Dies ist anders als bei "Fußball/Ergebnisse".

Das Zeichen "/" wird als Sonderzeichen betrachtet, das eine Struktur für den vollständigen Themennamen in „Themenstrukturen” auf Seite 82 angibt. Das Zeichen "/" darf zu keinem anderen Zweck verwendet werden, da die Themenstruktur betroffen ist. Das Thema "/Football" entspricht nicht dem Thema "Football".

Anmerkung: Wenn Sie beim Erstellen einer Subskription ein Themenobjekt verwenden, wird der Wert der Themenzeichenfolge des Themenobjekts zum Zeitpunkt der Definition in der Subskription festgelegt. Nachfolgende Änderungen des Themenobjekts haben keine Auswirkung auf die Themenzeichenfolge, für die die Subskription definiert ist.

Platzhalterzeichen in Themenzeichenfolgen

Die folgenden Platzhalterzeichen sind Sonderzeichen:

- Pluszeichen (+)
- Nummernzeichen (#)
- Stern (*)
- Fragezeichen (?)

Platzhalterzeichen haben nur eine besondere Bedeutung, wenn sie von einer Subskription verwendet werden. Diese Zeichen werden nicht als ungültig betrachtet, wenn sie woanders verwendet werden, aber Sie müssen sehr genau wissen, wie sie verwendet werden, und verzichten eventuell lieber darauf, diese Zeichen bei der Veröffentlichung oder Definition von Themenobjekten in Ihren Themenzeichenfolgen zu verwenden.

Bei Veröffentlichungen für eine Themenzeichenfolge mit # oder + in Kombination mit anderen Zeichen (einschließlich sich selbst) innerhalb einer Themenebene kann die Themenzeichenfolge mit beiden Platzhalterschemas subskribiert werden.

Bei Veröffentlichungen für eine Themenzeichenfolge mit # oder + als dem einzigen Zeichen zwischen zwei /-Zeichen kann die Themenzeichenfolge nicht explizit von einer Anwendung subskribiert werden, die das Platzhalterschema MQSO_WILDCARD_TOPIC verwendet. Diese Situation führt dazu, dass die Anwendung mehr Veröffentlichungen abruft als erwartet.

Sie sollten kein Platzhalterzeichen in der Themenzeichenfolge eines definierten Themenobjekts verwenden. Wenn doch, wird das Zeichen bei Verwendung des Objekts durch eine Publisher als Literal und bei Verwendung durch eine Subskription als Platzhalterzeichen behandelt. Dies kann zu Unklarheiten führen.

Mustercodeausschnitt

Dieser Codeausschnitt, der dem Beispielprogramm Beispiel 2: Publisher eines variablen Themas entnommen wurde, kombiniert ein Themenobjekt mit einer variablen Themenzeichenfolge:

```
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);
```

Themenstrukturen

Jedes Thema, das Sie definieren, wird in der Themenstruktur durch ein Element oder einen Knoten dargestellt. Die Themenstruktur kann entweder leer sein, um ganz neu zu beginnen, oder Themen enthalten, die zuvor mithilfe von WebSphere MQ-Scriptbefehlen oder PCF-Befehlen definiert wurden. Sie können ein neues Thema definieren, indem Sie entweder die Befehle zum Erstellen von Themen verwenden oder das Thema zum ersten Mal in einer Veröffentlichung oder Subskription angeben.

Obwohl Sie zum Definieren der Themenzeichenfolge eines Themas eine beliebige Zeichenfolge verwenden können, sollten Sie eine Themenzeichenfolge auswählen, die in eine hierarchische Baumstruktur passt. Ein sorgfältiger Entwurf von Themenzeichenfolgen und Themenstrukturen erleichtert folgende Vorgänge:

- Abonnieren mehrerer Themen

- Erstellen von Sicherheitsrichtlinien

Obwohl Sie eine Themenstruktur auch als flache, lineare Struktur entwerfen können, ist es besser, eine Themenstruktur in einer hierarchische Struktur mit einem oder mehreren Stammthemen zu erstellen. Weitere Informationen zur Sicherheitsplanung und zu Themen finden Sie im Abschnitt Publish/Subscribe-Sicherheit.

In Abbildung 18 auf Seite 83 zeigt ein Beispiel für eine Themenstruktur mit einem einzigen Stammthema.

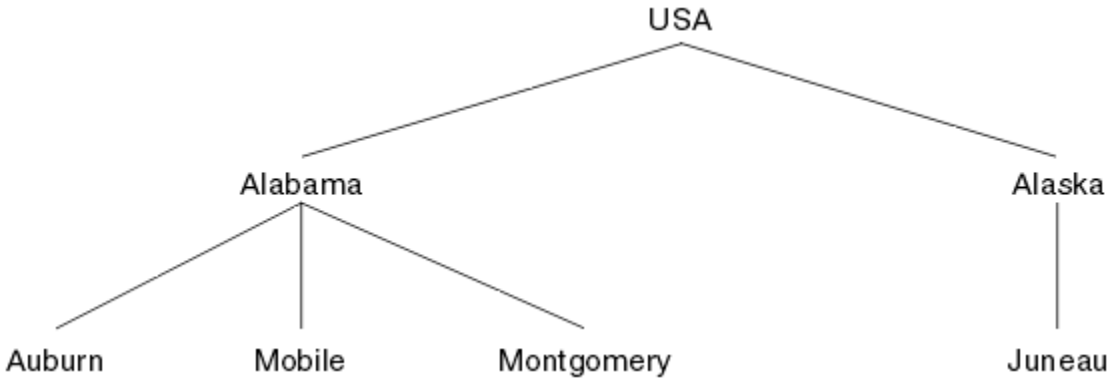


Abbildung 18. Beispiel für eine Themenstruktur

Jede Zeichenfolge in der Abbildung steht für einen Knoten in der Themenstruktur. Eine vollständige Themenzeichenfolge entsteht durch die Zusammenfassung von Knoten auf einer oder mehreren Ebenen in der Themenstruktur. Ebenen werden durch das Zeichen "/" getrennt. Eine vollständig angegebenen Themenzeichenfolge hat das Format "Stamm/Ebene2/Ebene3".

Die gültigen Themen in der Themenstruktur in Abbildung 18 auf Seite 83 lauten wie folgt:

"USA"
"USA/Alabama"
"USA/Alaska"
"USA/Alabama/Auburn"
"USA/Alabama/Mobile"
"USA/Alabama/Montgomery"
"USA/Alaska/Juneau"

Beachten Sie beim Entwerfen von Themenzeichenfolgen und Themenstrukturen, dass der Warteschlangenmanager nicht die Themenzeichenfolge selbst interpretiert oder auch nur versucht, eine Bedeutung daraus abzuleiten. Er verwendet die Themenzeichenfolge nur, um ausgewählte Nachrichten an Subskribenten des betreffenden Themas zu senden.

Für die Erstellung und den Inhalt einer Themenstruktur gelten folgende Prinzipien:

- Die Anzahl Ebenen in einer Themenstruktur ist nicht begrenzt.
- Der Länge des Namens einer Ebene in einer Themenstruktur ist nicht begrenzt.
- Es können beliebig viele Stammknoten vorhanden sein, d. h., es kann beliebig viele Themenstrukturen geben.

Zugehörige Tasks

Reduzieren der Anzahl unerwünschter Themen in der Themenstruktur

Verwaltungsthemenobjekte

Mit einem Verwaltungsthemenobjekt können Sie Themen bestimmte, nicht standardmäßige Attribute zuweisen.

Abbildung 19 auf Seite 84 zeigt das übergeordnete Thema `Sport` an, unterteilt in mehrere Unterthemen für verschiedene Sportarten und in Form einer Themenstruktur:

Abbildung 19. Visualisierung einer Themenstruktur

Abbildung 20 auf Seite 84 zeigt, wie die Themenstruktur weiter unterteilt werden kann, um unterschiedliche Informationsarten für jede Sportart zu trennen:

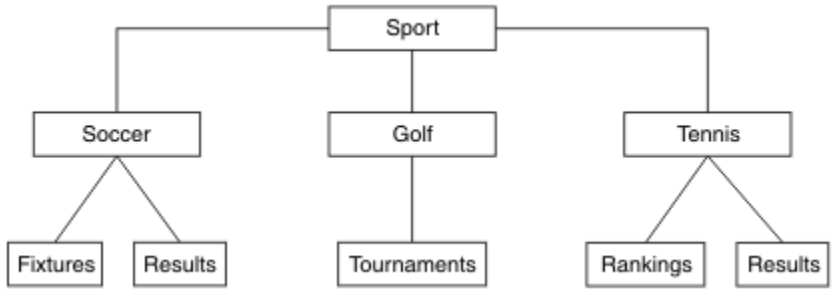


Abbildung 20. Erweiterte Themenstruktur

Zur Erstellung der gezeigten Themenstruktur müssen keine Objekte für Verwaltungsthemen definiert werden. Alle Knoten in dieser Baumstruktur sind durch eine Themenzeichenfolge definiert, die in einer Veröffentlichung oder Subskription erstellt wurde. Jedes Thema der Baumstruktur übernimmt seine Attribute von den ihm übergeordneten Elementen. Attribute werden von dem übergeordneten Themenobjekt übernommen, da standardmäßig alle Attribute auf ASPARENT eingestellt sind. In diesem Beispiel besitzt daher jedes Thema die gleichen Attribute wie das Thema `Sport`. Das Thema `Sport` hat kein Verwaltungsthemenobjekt und übernimmt seine Attribute aus SYSTEM.BASE.TOPIC.

Es wird dringend davon abgeraten, Berechtigungen für Benutzer, die nicht der Gruppe mqm angehören, auf Stammknotenebene der Themenstruktur (also auf der Ebene SYSTEM.BASE.TOPIC) zu erteilen, weil die Berechtigungen übernommen werden, aber nicht eingeschränkt werden können. Das heißt, wenn Sie Berechtigungen auf dieser Ebene erteilen, erteilen Sie damit Berechtigungen für die gesamte Struktur. Erteilen Sie die Berechtigung besser auf einer niedrigeren Themenebene in der Hierarchie.

Mit Verwaltungsthemenobjekten können bestimmte Attribute für bestimmte Knoten der Verzeichnisstruktur definiert werden. Im folgenden Beispiel wird das Verwaltungsthemenobjekt definiert, um die Eigenschaft der permanenten Subskription DURSUB des Themas 'Fußball' auf den Wert NO zu setzen:

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

Die Verzeichnisstruktur kann nun visuell dargestellt werden als:

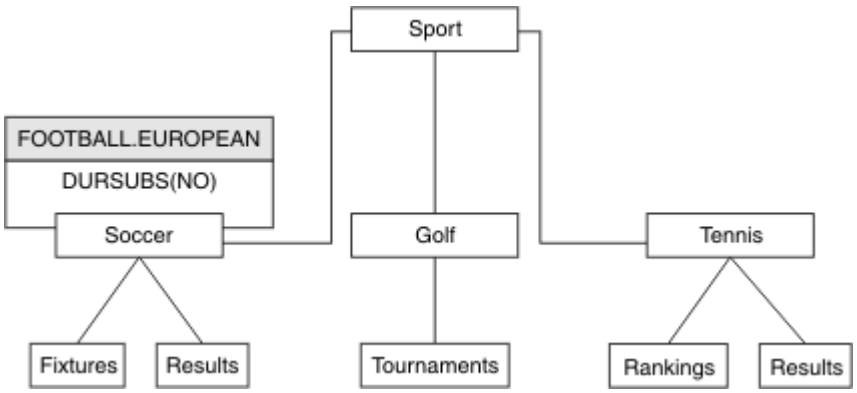


Abbildung 21. Visualisierung eines dem Thema 'Sport/Fußball' zugeordneten Verwaltungsthemenobjekts

Haben Anwendungen Themen abonniert, die in der Verzeichnisstruktur dem Thema 'Fußball' untergeordnet sind, können sie auch weiterhin die vor dem Hinzufügen des Verwaltungsthemenobjekts gültige Themenzeichenfolge verwenden. Eine Anwendung kann jetzt jedoch so geschrieben werden, dass sie den Objektnamen `FOOTBALL.EUROPEAN`anstelle der Zeichenfolge `/Sport/Soccer`subskribiert. Um beispielsweise `/Sport/Soccer/Results`zu subskribieren, kann eine Anwendung `MQSD.ObjectName` als `FOOTBALL.EUROPEAN` und `MQSD.ObjectString` als `Results`angeben.

Auf diese Weise können Sie einen Teil der Verzeichnisstruktur vor den Anwendungsentwicklern verbergen. Wenn Sie ein Verwaltungsthemenobjekt in einem bestimmten Knoten der Verzeichnisstruktur definieren, können Anwendungsentwickler somit ihre eigenen Themen diesem unterordnen. Die Entwickler müssen dabei das übergeordnete Thema, jedoch keine weiteren Knoten in der übergeordneten Baumstruktur kennen.

Attribute übernehmen

Sind in einer Themenstruktur viele Verwaltungsthemenobjekte vorhanden, übernimmt jedes Verwaltungsthemenobjekt standardmäßig die Attribute des ihm am nächsten übergeordneten Verwaltungsthemas. Das vorhergehende Beispiel wurde in Abbildung 22 auf Seite 85 erweitert:

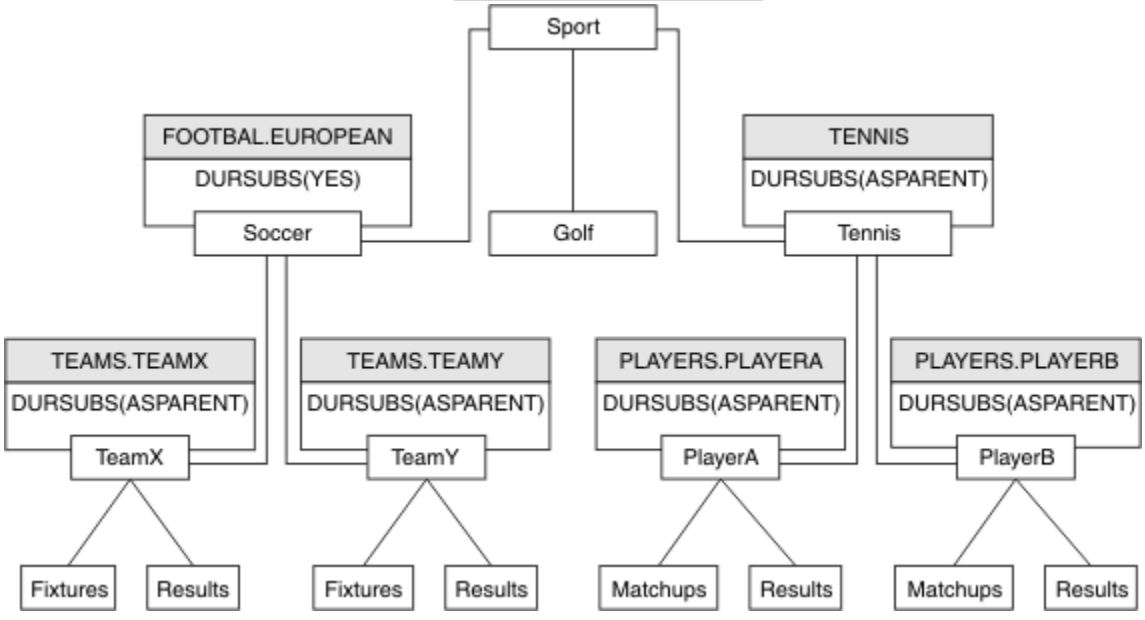


Abbildung 22. Themenstruktur mit mehreren Verwaltungsthemenobjekten

Verwenden Sie beispielsweise die Übernahme, um allen untergeordneten Themen von `/Sport/Soccer` die Eigenschaft zu geben, dass Subskriptionen nicht permanent sind. Ändern Sie das Attribut `DURSUB` von `FOOTBALL.EUROPEAN` in `NO`.

Dieses Attribut können Sie mit folgendem Befehl festlegen:

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

Für alle Verwaltungsthemenobjekte untergeordneter Themen von `Sport/Soccer` ist die Eigenschaft DURSUB auf den Standardwert ASPARENTgesetzt. Nachdem Sie den DURSUB -Eigenschaftswert von `FOOTBALL.EUROPEAN` in `NO`geändert haben, übernehmen die untergeordneten Themen von `Sport/Soccer` den DURSUB -Eigenschaftswert `NO`. Alle untergeordneten Themen von `Sport/Tennis` übernehmen den Wert von DURSUB aus dem Objekt `SYSTEM.BASE.TOPIC` . `SYSTEM.BASE.TOPIC` hat den Wert YES.

Der Versuch, eine permanente Subskription für das Thema `Sport/Soccer/TeamX/Results` zu erstellen, schlägt jetzt fehl. Der Versuch, eine permanente Subskription für `Sport/Tennis/PlayerB/Results` zu erstellen, wäre jedoch erfolgreich.

Verwendung von Platzhaltern mit der Eigenschaft WILDCARD steuern

Mit der MQSC-Eigenschaft **`Topic`** WILDCARD oder der entsprechenden PCF-Eigenschaft Topic `WildcardOperation` können Sie die Zustellung von Veröffentlichungen an Subskribentenanwendungen steuern, die Namen von Platzhalterthemenzeichenfolgen verwenden. Die Eigenschaft WILDCARD kann einen von zwei möglichen Werten haben:

WILDCARD
Aktionen von Subskriptionen mit Platzhaltern bezüglich dieses Themas.

PASSTHRU
Subskriptionen für ein Thema mit Platzhalter, das weniger spezifisch ist als die Themenzeichenfolge für dieses Themenobjekt, empfangen Veröffentlichungen zu diesem Thema und zu spezifischeren Themenzeichenfolge.

BLOCK
Subskriptionen für ein Thema mit Platzhalter, das weniger spezifisch ist als die Themenzeichenfolge für dieses Themenobjekt, empfangen keine Veröffentlichungen zu diesem Thema und zu spezifischeren Themenzeichenfolge.

Der Wert für dieses Attribut wird bei der Definition von Subskriptionen verwendet. Wenn Sie dieses Attribut ändern, ist die Gruppe von Themen, die bereits durch vorhandene Subskriptionen abgedeckt sind, nicht durch die Änderung betroffen. Dieses Szenario gilt auch, wenn sich durch die Erstellung oder das Löschen von Themenobjekten die Topologie ändert; die Themen mit Subskriptionen, die nach der Änderung des Attributs WILDCARD erstellt wurden, werden mit der geänderten Topologie erstellt. Wenn die Themen mit den vorhandenen Subskriptionen übereinstimmen sollen, müssen Sie den Warteschlangenmanager neu starten.

Führen Sie die Schritte im Beispiel „Beispiel: Publish/Subscribe-Cluster Sport erstellen” auf Seite 90 aus, um die in Abbildung 23 auf Seite 87 gezeigte Themenstruktur zu erstellen.

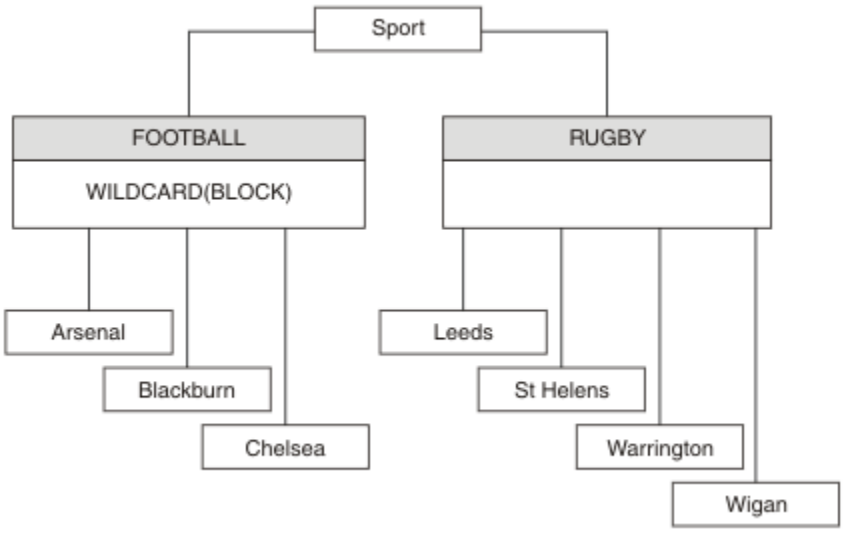


Abbildung 23. Themenstruktur, die die WILDCARD-Eigenschaft BLOCK verwendet

Ein Subskribent, der die Platzhalter-Themenzeichenfolge # verwendet, empfängt alle Veröffentlichungen für das Thema `Sport` und die Unterverzeichnisstruktur `Sport/Rugby` Der Subskribent empfängt keine Veröffentlichungen für die untergeordnete Baumstruktur `Sport/Football` , weil der Wert der Eigenschaft `WILDCARD` des Themas `Sport/Football` BLOCKist.

Die Standardeinstellung lautet PASSTHRU. Sie können den Wert PASSTHRU der Eigenschaft `WILDCARD` auf Knoten in der Baumstruktur `Sport` festlegen. Wenn die Knoten nicht über den Eigenschaftswert `WILDCARD` verfügen BLOCK, ändert die Einstellung PASSTHRU nicht das Verhalten, das von Subskribenten für Knoten in der `Sports` -Baumstruktur beobachtet wird.

Erstellen Sie im Beispiel Subskriptionen, um zu sehen, wie sich die Platzhaltereinstellung auf die bereitgestellten Veröffentlichungen auswirkt (siehe dazu Abbildung 27 auf Seite 92). Führen Sie den Publish-Befehl in Abbildung 30 auf Seite 93 aus, um einige Veröffentlichungen zu erstellen.

```
pub QMA
```

Abbildung 24. Für QMA publizieren

Die Ergebnisse werden in Tabelle 3 auf Seite 87 gezeigt. Beachten Sie, wie die Einstellung des `WILDCARD`-Eigenschaftswerts BLOCK verhindert, dass Subskriptionen mit Platzhaltern Veröffentlichungen für Themen empfangen, die innerhalb des Platzhalterumfangs liegen.

Tabelle 3. Auf QMA empfangene Veröffentlichungen

Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			Alle Veröffentlichungen zur Unterverzeichnisstruktur 'Football', die von `WILDCARD(BLOCK)` auf `Sports/Football` blockiert werden

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` on `Sports/Football` verhindert Platzhaltersubskription unter `Arsenal`
SLEEDS	`Sports/#/Leeds`	`Sports/Rugby/Leeds`	Standardwert `WILDCARD` für 'Sport/Rugby' verhindert keine Platzhaltersubskription auf `Leeds`.

Anmerkung:

Angenommen, eine Subskription hat einen Platzhalter, der einem Themenobjekt mit dem WILDCARD-Eigenschaftswert BLOCK entspricht. Wenn die Subskription auch eine Themenzeichenfolge rechts neben dem entsprechenden Platzhalter hat, empfängt die Subskription niemals Veröffentlichungen. Die nicht geblockten Veröffentlichungen sind Veröffentlichungen für Themen, die dem geblockten Platzhalter übergeordnet sind. Veröffentlichungen für Themen, die dem Thema mit dem Eigenschaftswert BLOCK untergeordnet sind, werden vom Platzhalter geblockt. Subskriptionsthemen-Zeichenfolgen, die ein Thema rechts neben dem Platzhalter einschließen, empfangen daher niemals passende Veröffentlichungen.

Wenn der Eigenschaftswert WILDCARD auf BLOCK gesetzt wird, ist keine Subskription mithilfe einer Themenzeichenfolge möglich, die Platzhalter beinhaltet. Eine solche Subskription ist normal. Die Subskription hat ein explizites Thema, das dem Thema mit einem Themenobjekt entspricht, das den WILDCARD-Eigenschaftswert BLOCK hat. Sie verwendet Platzhalter für Themen, die dem WILDCARD-Eigenschaftswert BLOCK über- oder untergeordnet sind. Im Beispiel in Abbildung 23 auf Seite 87kann eine Subskription wie Sports/Football/# Veröffentlichungen empfangen.

Platzhalter und Cluster-Topics

Definitionen von Cluster-Topics werden an jeden Warteschlangenmanager in einem Cluster weitergegeben. Die Subskription eines Cluster-Topics auf einem Warteschlangenmanager in einem Cluster führt dazu, dass der Warteschlangenmanager Proxy-Subskriptionen erstellt. Eine Proxy-Subskription wird in jedem Warteschlangenmanager im Cluster erstellt. Subskriptionen, die Themenzeichenfolgen mit Platzhaltern verwenden, kombiniert mit Cluster-Topics, lassen das Verhalten schwer vorhersagen. Das Verhalten wird im folgenden Beispiel erläutert.

In dem für das Beispiel verwendeten Cluster „Beispiel: Publish/Subscribe-Cluster Sport erstellen” auf Seite 90 hat QMB dieselben Subskriptionen wie QMA. QMB hat jedoch keine Veröffentlichungen empfangen, nachdem der Publisher an QMA veröffentlicht hat (siehe Abbildung 24 auf Seite 87). Obwohl die Themen Sports/Footbal und Sports/Rugby Clusterthemen sind, verweisen die in fullsubs.tst definierten Subskriptionen nicht auf ein Clusterthema. Es werden keine Proxy-Subskriptionen von QMB an QMA weitergegeben. Ohne Proxy-Subskriptionen werden keine Veröffentlichungen für QMA an QMB weitergeleitet.

Einige Subskriptionen, wie z. B. Sports/#/Leeds, scheinen möglicherweise auf ein Clusterthema zu verweisen, in diesem Fall Sports/Rugby. Die Sports/#/Leeds -Subskription wird tatsächlich in das Themenobjekt SYSTEM.BASE.TOPICaufgelöst.

Die Regel zum Auflösen des Themenobjekts, das von einer Subskription wie Sports/#/Leeds referenziert wird, lautet wie folgt. Schneiden Sie die Themenzeichenfolge bis zum ersten Platzhalter ab. Suchen Sie links über die Themenzeichenfolge nach dem ersten Thema mit einem zugehörigen Verwaltungsthemenobjekt. Das Themenobjekt kann einen Clusternamen angeben oder ein lokales Themenobjekt definieren. Im Beispiel Sports/#/Leedsist die Themenzeichenfolge nach dem Abschneiden Sports, die kein Themenobjekt hat, und Sports/#/Leeds übernimmt daher von SYSTEM.BASE.TOPIC, einem lokalen Themenobjekt.

Um zu sehen, wie die Subskription von Cluster-Topics die Weitergabe von Platzhaltern ändern kann, führen Sie das Stapelscript upsubs.bat aus. Das Script löscht die Subskriptionswarteschlangen und fügt die Subskriptionen des Cluster-Topics zu fullsubs.tst hinzu. Führen Sie puba.bat erneut aus, um einen Stapel von Veröffentlichungen zu erstellen (siehe Abbildung 24 auf Seite 87).

Tabelle 4 auf Seite 89 zeigt das Ergebnis, nachdem zwei neue Subskriptionen dem Warteschlangenmanager hinzugefügt wurden, in dem die Veröffentlichungen veröffentlicht wurden. Wie erwartet, empfangen die neuen Subskriptionen je eine Veröffentlichung, und die Anzahl der von den anderen Subskriptionen empfangenen Veröffentlichungen ist unverändert. Die unerwarteten Resultate treten im anderen Cluster-Warteschlangenmanager auf, siehe Tabelle 5 auf Seite 89.

Tabelle 4. Auf QMA empfangene Veröffentlichungen			
Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Alle Veröffentlichungen zur Unterverzeichnisstruktur 'Football', die von WILDCARD(BLOCK) auf Sports/Football blockiert werden
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football verhindert Platzhaltersubskription unter Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Standardwert WILDCARD für 'Sport/Rugby' verhindert keine Platzhaltersubskription auf Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal empfängt eine Veröffentlichung, da die Subskription keinen Platzhalter hat.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds würde für jedes Ereignis eine Veröffentlichung empfangen.

Tabelle 5 auf Seite 89 zeigt die Resultate, nachdem die beiden neuen Subskriptionen zu QMB hinzugefügt und auf QMA veröffentlicht wurden. QMB empfing keine Veröffentlichungen ohne diese beiden neuen Subskriptionen. Wie erwartet empfangen die beiden neuen Subskriptionen Veröffentlichungen, da Sports/FootBall und Sports/Rugby Clusterthemen sind. QMB weitergeleitete Proxy-Subskriptionen für Sports/Football/Arsenal und Sports/Rugby/Leeds an QMA, die dann die Veröffentlichungen an QMBgesendet haben

Das unerwartete Ergebnis ist, dass die beiden Subskriptionen Sports/# und Sports/#/Leeds , die zuvor keine Veröffentlichungen empfangen haben, jetzt Veröffentlichungen empfangen. Dies liegt daran, dass die Sports/Football/Arsenal -und Sports/Rugby/Leeds -Veröffentlichungen, die für die anderen Subskriptionen an QMB weitergeleitet werden, jetzt für jeden Subskribenten verfügbar sind, der QMBzugeordnet ist. Folglich empfangen die Subskriptionen für die lokalen Themen Sports/# und Sports/#/Leeds die Veröffentlichung Sports/Rugby/Leeds . Sports/#/Arsenal erhält weiterhin keine Veröffentlichung, weil für Sport/Fußball die Eigenschaft WILDCARD auf BLOCKgesetzt ist.

Tabelle 5. Auf QMB empfangene Veröffentlichungen			
Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football verhindert Platzhaltersubskription unter Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Der Standardwert WILDCARD für Sport/Rugby verhindert nicht das Platzhalterabonnement für Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal empfängt eine Veröffentlichung, da die Subskription keinen Platzhalter hat.

Tabelle 5. Auf QMB empfangene Veröffentlichungen (Forts.)			
Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds würde für jedes Ereignis eine Veröffentlichung empfangen.

Bei den meisten Anwendungen ist es nicht erwünscht, dass eine Subskription das Verhalten einer anderen Subskription beeinflusst. Eine wichtige Verwendung der Eigenschaft WILDCARD mit dem Wert BLOCK ist es, dass sich die Subskriptionen für dieselbe Themenzeichenfolge mit Platzhalterzeichen einheitlich verhalten. Unabhängig davon, ob sich die Subskription in demselben Warteschlangenmanager wie der Publisher oder in einem anderen Warteschlangenmanager befindet, sind die Resultate der Subskription dieselben.

Platzhalter und Datenströme

Wenn eine neue Anwendung für die Publish/Subscribe-API geschrieben wird, empfängt eine Subskription von * keine Veröffentlichungen. Um alle Sportveröffentlichungen zu erhalten, müssen Sie Sports/*oder Sports/#und ähnlich für Business -Veröffentlichungen subskribieren.

Das Verhalten einer vorhandenen Publish/Subscribe-Anwendung in der Warteschlange ändert sich nicht, wenn der Publish/Subscribe-Broker auf eine höhere Version von IBM MQmigriert wird. Die Eigenschaft **StreamName** in den Befehlen **Publish**, **Register Publisher** oder **Subscriber** wird dem Namen des Themas zugeordnet, auf das der Datenstrom migriert wurde.

Platzhalter und Subskriptionspunkte

Wenn eine neue Anwendung für die Publish/Subscribe-API geschrieben wird, empfängt eine Subskription von * keine Veröffentlichungen. Um alle Sportveröffentlichungen zu erhalten, müssen Sie Sports/*oder Sports/#und ähnlich für Business -Veröffentlichungen subskribieren.

Das Verhalten einer vorhandenen Publish/Subscribe-Anwendung in der Warteschlange ändert sich nicht, wenn der Publish/Subscribe-Broker auf eine höhere Version von IBM MQmigriert wird. Die Eigenschaft **SubPoint** in den Befehlen **Publish**, **Register Publisher** oder **Subscriber** wird dem Namen des Themas zugeordnet, auf das die Subskription migriert wurde.

Beispiel: Publish/Subscribe-Cluster Sport erstellen

In den nachfolgenden Schritten wird ein Cluster CL1 mit vier Warteschlangenmanagern erstellt: zwei vollständige Repositorys, CL1A und CL1B, und zwei Teilrepositorys, QMA und QMB. Die vollständigen Repositorys enthalten nur Clusterdefinitionen. QMA wird als Cluster-Topic-Host festgelegt. Permanente Subskriptionen sind in QMA und QMB definiert.

Anmerkung: Das Beispiel ist für Windows codiert. Wenn Sie das Beispiel auf anderen Plattformen konfigurieren und testen möchten, müssen Sie den Programmcode für Create qmgrs.bat und create pub.bat ändern.

1. Erstellen Sie die Scriptdateien.
   a. Create topics.tst
   b. Create wildsubs.tst
   c. Create fullsubs.tst
   d. Create qmgrs.bat
   e. create pub.bat
2. Führen Sie Create qmgrs.bat aus, um die Konfiguration zu erstellen.

```
qmgrs
```

Erstellen Sie die Themen in Abbildung 23 auf Seite 87. Das Script in Abbildung 5 erstellt die Clusterthemen `Sports/Football` und `Sports/Rugby`.

Anmerkung: Die Option REPLACE ersetzt nicht die TOPICSTR-Eigenschaften eines Themas. TOPICSTR ist eine Eigenschaft, die im Beispiel gezielt variiert wird, um verschiedene Themenstrukturen zu testen. Wenn Sie ein Thema ändern möchten, müssen Sie das Thema zuerst löschen.

```
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')
```

Abbildung 25. Themen löschen und erstellen: topics.tst

Anmerkung: Löschen Sie die Themen, da REPLACE keine Themenzeichenfolgen ersetzt.

Erstellen Sie Subskriptionen mit Platzhaltern. Die Platzhalter entsprechen den Themen mit Themenobjekten in Abbildung 23 auf Seite 87. Erstellen Sie für jede Subskription eine Warteschlange. Beim Ausführen bzw. erneuten Ausführen des Scripts werden die Warteschlangen geleert, und die Subskriptionen werden gelöscht.

Anmerkung: Die Option REPLACE ersetzt nicht die die Eigenschaften TOPICOBJ oder TOPICSTR einer Subskription. TOPICOBJ oder TOPICSTR sind die Eigenschaften, die im Beispiel sinnvoll variiert werden, um verschiedene Subskriptionen zu testen. Um sie zu ändern, löschen Sie zunächst die Subskription.

```
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)
```

Abbildung 26. Platzhaltersubskriptionen erstellen: wildsubs.tst

Erstellen Sie Subskriptionen, die die Cluster-Topic-Objekte referenzieren.

Anmerkung:

Das Begrenzungszeichen / wird automatisch zwischen die von TOPICOBJ referenzierte Themenzeichenfolge und die von TOPICSTR definierte Themenzeichenfolge eingefügt.

Die Definition `DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL)` erstellt dieselbe Subskription. Die Verwendung von TOPICOBJ ist eine zeiteffiziente Methode, bereits definierte Themenzeichenfolgen zu referenzieren. Die erstellte Subskription referenziert das Themenobjekt nicht länger.

```
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)
```

Abbildung 27. Subskriptionen löschen und erstellen: fullsubs.tst

Erstellen Sie einen Cluster mit zwei Repositorys. Erstellen Sie zwei Teilrepositorys zum Veröffentlichen und Abonnieren. Führen Sie das Script erneut aus, um alles zu löschen und neu zu starten. Das Script erstellt zudem die Themenhierarchie und die ursprünglichen Platzhaltersubskriptionen.

Anmerkung:

Schreiben Sie auf anderen Plattformen ein ähnliches Script, oder geben Sie alle Befehle ein. Mit einem Script lässt sich alles schneller löschen und mit einer identischen Konfiguration erneut starten.

```
@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%)') CLUS
TER(CL1) REPLACE | runmqsc %1
goto:eof
```

Abbildung 28. Warteschlangenmanager erstellen: qmgrs.bat

Aktualisieren Sie die Konfiguration durch Hinzufügen der Subskriptionen zu den Cluster-Topics.

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

Abbildung 29. Subskriptionen aktualisieren: upsubs.bat

Führen Sie `pub.bat` mit einem Warteschlangenmanager als Parameter aus, um Nachrichten zu veröffentlichen, die die Veröffentlichungsthema-Zeichenfolge enthalten. `Pub.bat` verwendet das Beispielprogramm **amqspub**.

```
@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
```

Abbildung 30. Veröffentlichen: pub.bat

Datenströme und Themen

Das eingereihte Publish/Subscribe basiert auf dem Konzept eines Veröffentlichungsdatenstroms, das es im Modell für integriertes Publish/Subscribe nicht gibt. Im eingereihten Publish/Subscribe bieten Datenströme die Möglichkeit, den Informationsfluss für unterschiedliche Themen zu trennen. Ein Datenstrom wird als Thema der höchsten Ebene implementiert, das administrativ einer anderen Themenkennung zugeordnet werden kann.

Der Standarddatenstrom SYSTEM.BROKER.DEFAULT.STREAM wird für alle Broker und Warteschlangenmanager in einem Netz automatisch eingerichtet und es ist keine zusätzliche Konfiguration erforderlich, um den Standarddatenstrom verwenden zu können. Sie können sich den Standarddatenstrom als nicht benannten Themenspeicherbereich vorstellen. Themen, die im Standarddatenstrom veröffentlicht werden, sind sofort für alle verbundenen Warteschlangenmanager mit aktiviertem eingereihtem Publish/Subscribe verfügbar. Benannte Datenströme gleichen separaten, benannten Themenspeicherbereichen. Der benannte Datenstrom muss auf jedem Broker definiert werden, auf dem er verwendet wird.

Wenn sich die Publisher und Subskribenten auf verschiedenen Warteschlangenmanagern befinden, ist für den Austausch der Veröffentlichungen und Subskriptionen zwischen den Brokern keine weitere Konfiguration erforderlich, sobald die Broker in derselben Brokerhierarchie miteinander verbunden sind. Diese Interoperabilität funktioniert auch umgekehrt.

Benannte Datenströme

Ein Lösungsentwickler, der mit dem Programmiermodell für eingereihtes Publish/Subscribe arbeitet, entscheidet sich möglicherweise dafür, alle Sportveröffentlichungen in einen benannten Datenstrom mit dem Namen Sport zu stellen. Damit der Datenstrom für einen Warteschlangenmanager verfügbar ist, der unter IBM MQ mit aktiviertem eingereihtem Publish/Subscribe ausgeführt wird, muss der Datenstrom manuell hinzugefügt werden.

Eingereihte Publish/Subscribe-Anwendungen, die Soccer/Results im Datenstrom Sport subskribieren, funktionieren unverändert. Integrierte Publish/Subscribe-Anwendungen, die das Thema Sport mit MQSUBsubskribieren und die Themenzeichenfolge Soccer/Results angeben, empfangen dieselben Veröffentlichungen ebenfalls.

Die Aufgabe zum Hinzufügen eines Datenstroms wird im Abschnitt Datenstrom hinzufügen beschrieben. Sie müssen Datenströme eventuell aus zwei Gründen manuell hinzufügen.

1. Sie entwickeln weiterhin Ihre Anwendungen mit eingereihtem Publish/Subscribe, die auf Warteschlangenmanagern einer höheren Version ausgeführt werden, statt die Anwendungen auf die MQI-Schnittstelle für integriertes Publish/Subscribe zu migrieren.
2. Die Standardzuordnung von Datenströmen zu Themen führt zu einer "Kollision" im Themenspeicherbereich und Veröffentlichungen in einem Datenstrom verfügen über dieselbe Themenzeichenfolge wie Veröffentlichungen aus anderen Quellen.

Berechtigungen

Standardmäßig befinden sich im Stammelement der Themenstruktur mehrere Themenobjekte: SYSTEM.BASE.TOPIC, SYSTEM.BROKER.DEFAULT.STREAM und SYSTEM.BROKER.DEFAULT.SUBPOINT. Berechtigungen (z. B. für die Veröffentlichung oder Subskription) werden von den Behörden auf dem SYSTEM.BASE.TOPIC festgelegt; Alle Berechtigungen unter SYSTEM.BROKER.DEFAULT.STREAM oder SYSTEM.BROKER.DEFAULT.SUBPOINT werden ignoriert. Wenn SYSTEM.BROKER.DEFAULT.STREAM oder SYSTEM.BROKER.DEFAULT.SUBPOINT gelöscht und mit einer nicht leeren Themenzeichenfolge neu

erstellt werden, werden die für diese Objekte definierten Berechtigungen wie ein normales Themenobjekt verwendet.

Zuordnung von Datenströmen zu Themen

Ein Datenstrom für eingereihtes Publish/Subscribe wird in IBM MQ nachgeahmt, indem eine Warteschlange erstellt und ihr der Name des Datenstroms zugeordnet wird. Manchmal wird die Warteschlange auch Datenstromwarteschlange genannt, weil sie von Anwendungen mit eingereihtem Publish/Subscribe so wahrgenommen wird. Die Warteschlange wird für die Publish/Subscribe-Steuerkomponente identifiziert, indem sie zu der speziellen Namensliste namens `SYSTEM.QPUBSUB.QUEUE.NAMELIST` hinzugefügt wird. Sie können so viele Datenströme wie nötig hinzufügen, indem Sie der Namensliste weitere spezielle Warteschlangen hinzufügen. Schließlich müssen Sie Themen hinzufügen, die dieselben Namen wie die Datenströme und dieselben Themenzeichenfolgen wie der Datenstromname aufweisen, damit Sie Inhalte in den Themen veröffentlichen und abonnieren können.

Unter außergewöhnlichen Umständen können Sie für die Themen, die den Datenströmen entsprechen, jedoch beliebige Themenzeichenfolgen angeben, die Sie bei der Auswahl der Themen definieren. Zweck der Themenzeichenfolge ist es, dem Thema einen eindeutigen Namen im Themenspeicherbereich zu geben. Normalerweise erfüllt der Datenstromname diesen Zweck perfekt. Manchmal kollidieren ein Datenstromname und ein vorhandener Themenname. Wählen Sie zur Lösung des Problems eine andere Themenzeichenfolge für das dem Datenstrom zugeordnete Thema aus. Stellen Sie bei der Auswahl einer Themenzeichenfolge sicher, dass sie eindeutig ist.

Die Themenzeichenfolge, die in der Themendefinition definiert ist, ist wie üblich im Vergleich zu der Themenzeichenfolge, die von Publishern und Subskribenten über die MQI-Aufrufe `MQOPEN` oder `MQSUB` bereitgestellt werden, mit einem Präfix versehen. Anwendungen, die auf Themen verweisen, die Themenobjekte verwenden, sind von der Auswahl der Präfixthemenzeichenfolge nicht betroffen. Deshalb können Sie eine beliebige Themenzeichenfolge auswählen, sofern die Veröffentlichungen im Themenspeicherbereich eindeutig bleiben.

Für die Neuzuordnung von unterschiedlichen Datenströmen zu anderen Themen ist es wichtig, dass die Präfixe, die für die Themenzeichenfolgen verwendet werden, eindeutig sind, damit eine Themengruppe komplett von einer anderen Themengruppe getrennt werden kann. Sie müssen eine allgemeine Benennungskonvention für Themen definieren, die strikt befolgt werden muss, damit die Zuordnung funktioniert.

In IBM MQverwenden Sie den Vorfixierungsmechanismus, um eine Topiczeichenfolge einem anderen Bereich im Topicbereich zuzuordnen.

Anmerkung: Wenn Sie einen Datenstrom löschen, löschen Sie zuerst alle Subskriptionen im Datenstrom. Diese Aktion ist dann besonders wichtig, wenn Subskriptionen von anderen Brokern in der Brokerhierarchie stammen.

Subskriptionspunkte und Themen

Benannte Subskriptionspunkte werden durch Themen und Themenobjekte emuliert.

Informationen zum manuellen Hinzufügen von Subskriptionspunkten finden Sie im Abschnitt Subskriptionspunkt hinzufügen.

Subskriptionspunkte in IBM MQ

IBM MQ ordnet Subskriptionspunkte verschiedenen Themenspeicherbereichen in der IBM MQ-Themenstruktur zu. Themen in Befehlsnachrichten ohne Subskriptionspunkt werden unverändert dem Stammelement der IBM MQ-Themenstruktur zugeordnet und übernehmen Eigenschaften von `SYSTEM.BASE.TOPIC`.

Befehlsnachrichten mit einem Subskriptionspunkt werden mithilfe der Liste der Themenobjekte in `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST` verarbeitet. Der Name des Subskriptionspunktes in der Befehlsnachricht wird für jedes Themenobjekt in der Liste mit der Themenzeichenfolge abgeglichen. Wenn eine Übereinstimmung gefunden wird, wird der Name des Subskriptionspunktes als Themenknoten der

Themenzeichenfolge vorangestellt. Das Thema übernimmt die Eigenschaften von dem in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST gefundenen zugehörigen Themenobjekt.

Die Verwendung von Subskriptionspunkten führt dazu, dass für jeden Subskriptionspunkt ein separater Themenspeicherbereich erstellt wird. Das Stammelement des Themenspeicherbereichs ist ein Thema, das denselben Namen wie der Subskriptionspunkt hat. Themen in den einzelnen Themenspeicherbereichen übernehmen die Eigenschaften von dem Themenobjekt, das denselben Namen wie der Subskriptionspunkt hat.

Alle Eigenschaften, die in dem übereinstimmenden Themenobjekt nicht festgelegt sind, werden wie üblich von SYSTEM.BASE.TOPIC übernommen.

Vorhandene Publish/Subscribe-Anwendungen in der Warteschlange, die MQRFH2 -Nachrichtenheader verwenden, funktionieren weiterhin, indem sie die Eigenschaft **SubPoint** in den Befehlsnachrichten Publish oder Register subscriber festlegen. Der Subskriptionspunkt wird in der Befehlsnachricht mit der Themenzeichenfolge kombiniert und das sich ergebende Thema wird wie alle anderen Themen verarbeitet.

IBM MQ -Anwendungen sind von Subskriptionspunkten nicht betroffen. Wenn eine Anwendung ein Thema verwendet, das Informationen von einem der passenden Themenobjekte übernimmt, interagiert sie mit einer eingereihten Anwendung, die den entsprechenden Subskriptionspunkt verwendet.

Beispiel

Von einer vorhandenen Publish/Subscribe-Anwendung von WebSphere Message Broker (jetzt "IBM Integration Bus"), die nach IBM MQ migriert wurde, wurden die beiden Themenobjekte GBP und USD mit den entsprechenden Themenzeichenfolgen 'GBP' und 'USD' erstellt.

Vorhandene Publisher zum Thema NYSE/IBM/SPOT, die migriert wurden, um in IBM MQ ausgeführt werden zu können und die den Subskriptionspunkt USD verwenden, erstellen Veröffentlichungen zum Thema USD/NYSE/IBM/SPOT. Ebenso erstellen vorhandene Subskribenten von NYSE/IBM/SPOT, die den Subskriptionspunkt USD verwenden, Subskriptionen in USD/NYSE/IBM/SPOT.

Abonnieren Sie den Spotpreis in Dollar in einem IBM MQ -Publish/Subscribe-Programm, indem Sie MQSUBaufrufen. Erstellen Sie eine Subskription mithilfe des Themenobjekts USD und der Themenzeichenfolge 'NYSE/IBM/SPOT', wie im 'C'-Codefragment dargestellt.

```
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. Legen Sie das Attribut CLUSTER der Themenobjekte USD und GBP auf dem Clusterthemenhost fest.
2. Löschen Sie alle Kopien der Themenobjekte USD und GBP in anderen Warteschlangenmanagern im Cluster.
3. Vergewissern Sie sich, dass USD und GBP in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST in jedem Warteschlangenmanager im Cluster definiert sind.

Beispiel für die Publish/Subscribe-Konfiguration für einen einzelnen Warteschlangenmanager

Abbildung 31 auf Seite 96 zeigt eine grundlegende Publish/Subscribe-Konfiguration für einen einzelnen Warteschlangenmanager. Das Beispiel zeigt die Konfiguration für einen Nachrichtenservice, der Informationen von verschiedenen Publishern zu mehreren Themen bereitstellt:

- Publisher 1 veröffentlicht Sportergebnisse unter dem Thema "Sport"
- Publisher 2 veröffentlicht Aktienkurse unter dem Thema "Stock" (Aktien)
- Publisher 3 veröffentlicht Filmkritiken unter dem Thema "Films" (Filme) und das Fernsehprogramm unter dem Thema "TV"

Drei Subskribenten haben ihr Interesse an verschiedenen Themen angemeldet, sodass ihnen der Warteschlangenmanager die entsprechenden Informationen zusendet:

- Subskribent 1 erhält die Sportergebnisse und Aktienkurse
- Subskribent 2 erhält die Filmkritiken
- Subskribent 3 erhält die Sportergebnisse

Keiner der Subskribenten ist am Thema "TV" interessiert, das Fernsehprogramm wird daher nicht verteilt.

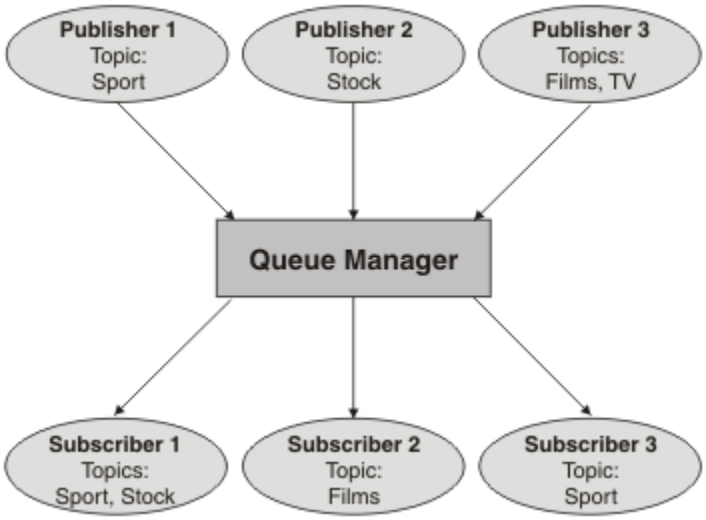


Abbildung 31. Beispiel für die Publish/Subscribe-Konfiguration für einen einzelnen Warteschlangenmanager

Verteilte Publish/Subscribe-Netzwerke

Jeder Warteschlangenmanager gleicht die für ein Thema veröffentlichten Nachrichten mit den lokal erstellten Subskriptionen für das Thema ab. Sie können ein Netz aus Warteschlangenmanagern konfigurieren, sodass Nachrichten, die von einer mit einem Warteschlangenmanager verbundenen Anwendung veröffentlicht werden, an alle übereinstimmenden Subskriptionen auf den anderen Warteschlangenmanagern des Netzes übertragen werden. Dazu müssen die Warteschlangenmanager zusätzlich über einfache Kanäle miteinander verbunden sein.

Bei einer verteilten Publish/Subscribe-Konfiguration handelt es sich um ein Netz aus miteinander verbundenen Warteschlangenmanagern. Die Warteschlangenmanager können sich alle auf dem gleichen physischen System befinden oder über mehrere physische Systeme verteilt sein. Wenn Sie Warteschlangenmanager miteinander verbinden, können Subskribenten eine Subskription für einen Warteschlangenmanager einrichten und über diesen Nachrichten empfangen, die auf einem anderen Warteschlangenmanager veröffentlicht wurden. Zur Veranschaulichung dieses Konzepts wurde der im Abschnitt „Beispiel für die Publish/Subscribe-Konfiguration für einen einzelnen Warteschlangenmanager” auf Seite 95 beschriebenen Konfiguration in der folgenden Abbildung ein zweiter Warteschlangenmanager hinzugefügt.

- Die veröffentlichende Stelle 4 veröffentlicht auf Warteschlangenmanager 2 Wettervorhersagen unter dem Thema 'Wetter' sowie Verkehrsmeldungen für wichtige Straßenverbindungen unter dem Thema 'Verkehr'.
- Subskribent 4 verwendet ebenfalls diesen Warteschlangenmanager und hat das Thema 'Verkehr' subskribiert.
- Subskribent 3 hat zusätzlich das Thema 'Wetter' subskribiert, obwohl er den Warteschlangenmanager einer anderen veröffentlichenden Stelle verwendet. Dies ist möglich, da die Warteschlangenmanager miteinander verbunden sind.

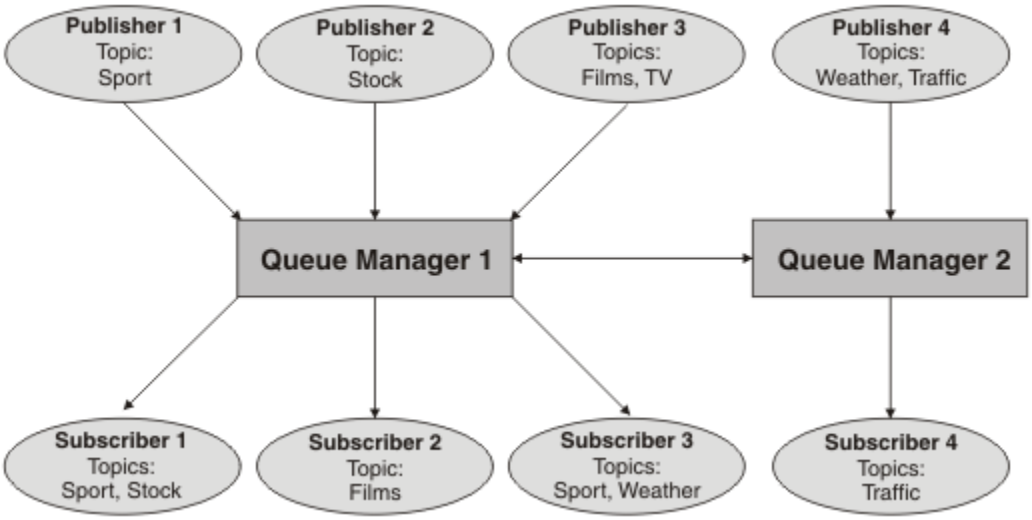


Abbildung 32. Beispiel für Publish/Subscribe mit zwei Warteschlangenmanagern

Sie können Warteschlangenmanager manuell in einer Parent-/Child-Hierarchie verbinden oder einen Publish/Subscribe-Cluster erstellen und IBM MQ den größten Teil der Verbindungskonfiguration überlassen. Sie können auch beide Topologien in Kombination verwenden, zum Beispiel indem sie mehrere Cluster zu einer Hierarchie zusammenfassen.

Übersicht über Publish/Subscribe-Cluster

Ein Publish/Subscribe-Cluster ist ein Standardcluster mit einem oder mehreren Themenobjekten. Wenn Sie auf einem beliebigen Warteschlangenmanager in einem Cluster ein administratives Themenobjekt definieren und aus diesem Themenobjekt durch Angabe eines Clusternamens ein Clusterobjekt machen, können die Publisher und Subskribenten dieses Themas eine Verbindung mit einem beliebigen Warteschlangenmanager des Clusters herstellen, um die veröffentlichten Nachrichten über die Clusterkanäle zwischen den Warteschlangenmanagern an die Subskribenten weiterzuleiten.

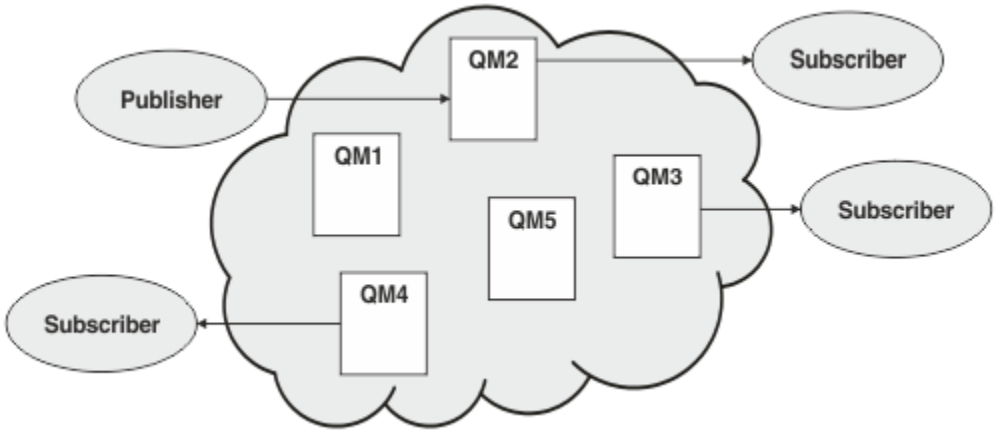


Abbildung 33. Publish/Subscribe-Cluster

Die Weiterleitung der Publish/Subscribe-Nachrichten in einem Cluster kann auf zwei Weisen erfolgen:

- Direktes Routing (DIRECT-Routing)
- Topic-Host-Routing (TOPICHOST-Routing)

Wenn Sie ein Cluster-Topic mit DIRECT-Routing konfigurieren, werden die auf einem Warteschlangenmanager veröffentlichten Nachrichten direkt von diesem Warteschlangenmanager an alle Subskriptionen auf den anderen Warteschlangenmanagern des Clusters weitergeleitet. Dadurch wird bei der Weiterleitung von Veröffentlichungen zwar der direkteste Weg eingeschlagen, allerdings führt diese Konfiguration auch

dazu, dass alle Warteschlangenmanager des Clusters alle anderen Warteschlangenmanager kennen und potenziell zwischen allen Warteschlangenmanager des Clusters Clusterkanäle eingerichtet werden.

Beim TOPICHOST-Routing hingegen werden die auf einem Warteschlangenmanager veröffentlichten Nachrichten von diesem Warteschlangenmanager an einen Warteschlangenmanager übertragen, der über eine Definition des verwalteten Themenobjekts verfügt. Der *Topic-Host-Warteschlangenmanager* leitet die Nachricht an jede Subskription in jedem anderen Warteschlangenmanager im Cluster weiter. Falls sich die Publisher und Subskribenten nicht auf den Topic-Host-Warteschlangenmanagern befinden, wird der Übertragungsweg für die Veröffentlichungen länger. Der Vorteil jedoch ist, dass nur die Topic-Host-Warteschlangenmanager die anderen Warteschlangenmanager im Cluster kennen müssen und nur zwischen diesen Clusterkanäle eingerichtet werden.

Weitere Informationen finden Sie unter „Publish/Subscribe-Cluster” auf Seite 99.

Übersicht über die Publish/Subscribe-Hierarchie

Bei einer Publish/Subscribe-Hierarchie handelt es sich um eine hierarchische Struktur aus Warteschlangenmanagern, die über Kanäle miteinander verbunden sind. Jeder Warteschlangenmanager erkennt seinen *übergeordneten* Warteschlangenmanager, wie im Abschnitt Warteschlangenmanager zu einer Publish/Subscribe-Hierarchie verbinden beschrieben.

Publisher und Subskribenten eines Themas können eine Verbindung mit jedem Warteschlangenmanager der Hierarchie herstellen und die Nachrichten werden in der hierarchischen Struktur der verbundenen Warteschlangenmanager übertragen.

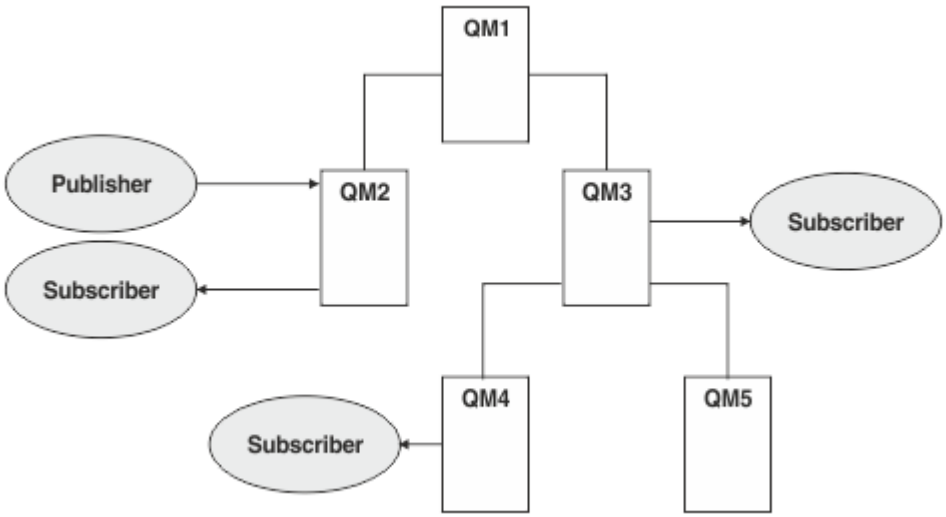


Abbildung 34. Publish/Subscribe-Hierarchie

In dieser Abbildung werden Veröffentlichungen, die für die Subskribenten auf QM3 und QM4 bestimmt sind, von QM2 an QM1 und von dort an QM3 und schließlich an QM4 übertragen.

Hierarchien geben Ihnen unmittelbare Kontrolle über die Beziehungen zwischen den einzelnen Warteschlangenmanagern der Hierarchie. Dies ermöglicht eine differenzierte Steuerung der Nachrichtenweiterleitung von Publishern zu Subskribenten, was besonders für das Routing zwischen Warteschlangenmanagernetzen mit eingeschränkter Konnektivität wichtig ist. Sie sollten hierbei besonders auf die Verfügbarkeit und die Möglichkeiten eines jeden Warteschlangenmanagers achten, über den eine Nachricht auf dem Weg vom Publisher zum Subskribenten weitergeleitet wird.

Weitere Informationen finden Sie unter „Publish/Subscribe-Hierarchien” auf Seite 102.

Verteilung der Veröffentlichungen zwischen den Warteschlangenmanagern

Neben den verschiedenen Routing-Möglichkeiten gibt es auch zwei Methoden der Verteilung der Veröffentlichungen im Netz der Warteschlangenmanager:

- Senden von Veröffentlichungen von einem Warteschlangenmanager nur an diejenigen Warteschlangenmanager, denen zur Zeit eine Subskription für diese Veröffentlichung vorliegt
- Senden aller Veröffentlichungen an alle Warteschlangenmanager - erst auf diesen erfolgt der Abgleich mit den eigenen Subskriptionen

Bei der ersten Methode werden Veröffentlichungen nur bei Bedarf gesendet. Hierzu müssen allerdings alle Warteschlangenmanager über ein gewisses Grad an Subskriptionskenntnissen verfügen. Bei der zweiten Methode sind keine Kenntnisse der Subskriptionen auf den anderen Warteschlangenmanagern erforderlich, allerdings kommt es bei dieser Methode, was die Veröffentlichungen anbelangt, vermutlich zu einem erhöhten und unnötigen Übertragungsaufwand.

IBM MQ verwendet standardmäßig die erste Methode, bei der Veröffentlichungen nur an Warteschlangenmanager gesendet werden, die auch die entsprechende Subskription aufweisen. Die Subskriptionsdaten werden zwischen den Warteschlangenmanagern in Form von *Proxy-Subskriptionen* übertragen. Welche Methode in einer Publish/Subscribe-Topologie die effizientere ist, hängt von der Verteilung und der Lebensdauer der Subskriptionen und dem Veröffentlichungsintervall ab. Weitere Informationen finden Sie im Abschnitt Subskriptionsleistung in Publish/Subscribe-Netzen.

Zugehörige Konzepte

„Themenstrukturen” auf Seite 82
Jedes Thema, das Sie definieren, wird in der Themenstruktur durch ein Element oder einen Knoten dargestellt. Die Themenstruktur kann entweder leer sein, um ganz neu zu beginnen, oder Themen enthalten, die zuvor mithilfe von WebSphere MQ-Scriptbefehlen oder PCF-Befehlen definiert wurden. Sie können ein neues Thema definieren, indem Sie entweder die Befehle zum Erstellen von Themen verwenden oder das Thema zum ersten Mal in einer Veröffentlichung oder Subskription angeben.

Publish/Subscribe-Hierarchieszenarios

Zugehörige Tasks

Publish/Subscribe-Cluster entwerfen

Publish/Subscribe-Cluster

Ein Publish/Subscribe-Cluster ist ein Standardcluster aus verbundenen Warteschlangenmanagern, in dem Veröffentlichungen automatisch aus den veröffentlichenden Anwendungen in Subskriptionen verschoben werden, die auf einem beliebigen der zu einem Cluster zusammengefassten Warteschlangenmanager vorliegen. Es gibt zwei Optionen zum Weiterleiten von Publikationen über einen Publish/Subscribe-Cluster: *direktes Routing* und *Topic-Host-Routing*. Für welches Verfahren Sie sich entscheiden, hängt von der Größe und dem erwarteten Aktivitätsmuster Ihres Clusters ab.

Ein Cluster für Publish/Subscribe-Messaging unterscheidet sich nicht von einem IBM MQ-Standardcluster. Das bedeutet, dass sich die Warteschlangenmanager innerhalb eines Publish/Subscribe-Clusters auf physisch getrennten Computern befinden können und bei Bedarf automatisch über Clusterkanäle miteinander verbunden werden. Weitere Informationen hierzu finden Sie im Abschnitt Clusters.

Zur Konfiguration eines Standard-Warteschlangenmanagerclusters für Publish/Subscribe-Messaging definieren Sie auf einem Warteschlangenmanager des Clusters ein oder mehrere verwaltete Themenobjekte. Um aus diesem Thema ein Cluster-Topic zu machen, geben Sie unter der Eigenschaft **CLUSTER** den Namen des Clusters an. Danach wird jedes Thema, das von einem Publisher oder Subskribenten an dieser Stelle in der Themenstruktur oder darunter verwendet wird, für alle Warteschlangenmanager des Clusters freigegeben und Nachrichten, die in einem geclusterten Zweig der Themenstruktur veröffentlicht werden, werden automatisch an die Subskriptionen auf anderen Warteschlangenmanagern des Clusters weitergeleitet.

Unabhängig von der Anzahl der Subskribenten der Nachricht auf dem Ziel-Warteschlangenmanager wird nur eine Kopie jeder Nachricht von dem Warteschlangenmanager des Publishers an jeden der anderen

Warteschlangenmanager übertragen. Erst beim Eingang auf einem Warteschlangenmanager mit mehreren Subskriptionen wird die Nachricht entsprechend der Anzahl der Subskriptionen dupliziert.

Jeder Warteschlangenmanager, der neu in den Cluster aufgenommen wird, wird automatisch über die Clusterthemen informiert, und die Publisher und Subskribenten auf diesem Warteschlangenmanager nehmen automatisch am Cluster teil.

In einem Publish/Subscribe-Cluster können auch nicht geclusterte Publish/Subscribe-Aktivitäten stattfinden, wenn Themenzeichenfolgen verwendet werden, die keinem geclusterten Themenobjekt angehören.

Es gibt zwei Optionen zum Weiterleiten von Publikationen über einen Publish/Subscribe-Cluster: *direktes Routing* und *Topic-Host-Routing*. Um das Nachrichtenrouting auszuwählen, das im Cluster verwendet werden soll, müssen Sie für das verwaltete Themenobjekt für die Eigenschaft **CLROUTE** einen der folgenden Werte festlegen:

- **DIRECT**
- **TOPICHOST**

Topic-Routing ist standardmäßig **DIRECT**. Wenn Sie ein direkt geroutetes Cluster-Topic in einem Warteschlangenmanager konfigurieren, werden sämtliche Warteschlangenmanager im Cluster aller anderen Warteschlangenmanager im Cluster gewahr. Bei der Ausführung von Publish- und Subscribe-Operationen kann jeder Warteschlangenmanager direkt eine Verbindung zu anderen Warteschlangenmanagern im Cluster herstellen.

Ab IBM MQ 8.0können Sie Topic-Routing stattdessen als **TOPICHOST**konfigurieren. Bei Verwendung der Routing-Methode TOPICHOST können alle Warteschlangenmanager im Cluster die Clusterwarteschlangenmanager erkennen, die die Definition des weitergeleiteten Themas enthalten (d. h. die Warteschlangenmanager, in denen Sie das Themenobjekt definiert haben). Beim Ausführen von Publish/Subscribe-Operationen werden Warteschlangenmanager im Cluster nur mit diesen Topic-Host-Warteschlangenmanagern und nicht direkt miteinander verbunden. Die Topic-Host-Warteschlangenmanager sind für das Routing von Publikationen aus Warteschlangenmanagern verantwortlich, in denen Publikationen für Warteschlangenmanager mit übereinstimmenden Subskriptionen veröffentlicht werden.

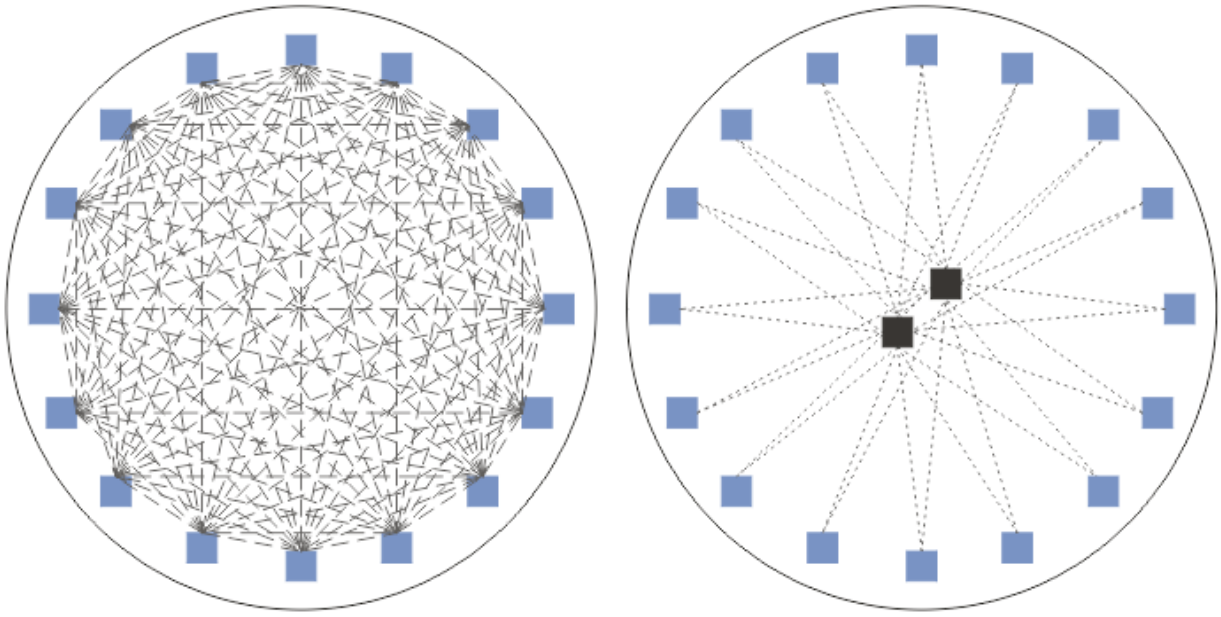


Abbildung 35. DIRECT- und TOPICHOST-Routing

Übersicht über DIRECT-Routing

Wenn ein verwaltetes Topic-Objekt für DIRECT-Routing konfiguriert wird, muss das Topic-Objekt nur auf einem der Warteschlangenmanager im Cluster definiert werden, damit alle Warteschlangenmanager

darüber erfahren. Auf welchem Warteschlangenmanager das Topic-Objekt definiert wird, wirkt sich nicht auf das Verhalten des Publish/Subscribe-Messaging für das Thema aus.

Jede Nachricht wird auf direktem Weg vom veröffentlichenden Warteschlangenmanager ohne Zwischenstation über andere Warteschlangenmanager an jede Subskription auf den anderen Warteschlangenmanagern des Clusters übertragen.

Standardmäßig werden Nachrichten nur an die Warteschlangenmanager im Cluster gesendet, auf denen eine oder mehrere Subskriptionen vorliegen.

- Es ist Aufgabe jedes Warteschlangenmanagers, alle anderen Warteschlangenmanager im Cluster direkt über alle Themen zu informieren, die deren Subskriptionen betreffen. Dies führt dazu, dass allen Warteschlangenmanagern im Cluster alle subskribierten Themen bekannt sind, und dass alle Warteschlangenmanager, die eine Subskription bereitstellen, einen Verbindungskanal mit allen anderen Warteschlangenmanagern aufbauen. Dieser Mechanismus erfolgt unabhängig davon, ob die einzelnen Warteschlangenmanager über Publisher verfügen.
- Die Informationen zu den subskribierten Themen der einzelnen Warteschlangenmanager können durch einen Wechsel zu einem Modell entfernt werden, bei dem alle Veröffentlichungen unabhängig von den auf den einzelnen Warteschlangenmanagern vorliegenden Subskriptionen an alle Warteschlangenmanager des Clusters gesendet werden. Dadurch reduziert sich der Austausch der Subskriptionsdaten, der durch Veröffentlichungen bewirkte Datenverkehr sowie möglicherweise die Anzahl der von den Warteschlangenmanagern eingerichteten Kanäle erhöht sich hingegen. Weitere Informationen finden Sie im Abschnitt Subskriptionsleistung in Publish/Subscribe-Netzen.

Publish/Subscribe-Nachrichtenflüsse mit direkt weitergeleiteten Cluster-Topics können über mehrere Publish/Subscribe-Cluster verlaufen, wenn aus jedem Cluster ein Warteschlangenmanager zu einer Publish/Subscribe-Hierarchie verbunden wird. Weitere Informationen hierzu finden Sie im Abschnitt Themenbereiche mehrerer Cluster zusammenfassen.

Nähere Ausführungen zum direkten Routing finden Sie im Abschnitt Direktes Routing in Publish/Subscribe-Clustern.

Übersicht über TOPICHOST-Routing

Wenn ein verwaltetes Topic-Objekt für TOPICHOST-Routing konfiguriert ist, werden die Veröffentlichungen eines Warteschlangenmanagers des Clusters über einen Warteschlangenmanager weitergeleitet, auf dem das Topic-Objekt konfiguriert ist (dies ist der "Topic-Host"). Von diesem Topic-Host werden die Veröffentlichungen dann an die Warteschlangenmanager weitergeleitet, auf denen die entsprechenden Subskriptionen vorliegen.

- Es ist Aufgabe jedes Warteschlangenmanagers, alle Topic-Hosts über alle Themen zu informieren, für die eine oder mehrere Subskriptionen vorliegen. Jeder Warteschlangenmanager, auf dem eine Subskription vorliegt, baut für die jeweilige Subskription einen Kanal mit dem zugehörigen Topic-Host auf.
- Warteschlangenmanager, die keine Themen bereitstellen, werden im Rahmen des Publish/Subscribe nicht über andere Warteschlangenmanager des Clusters informiert, die ebenfalls keine Topic-Hosts sind, und zwischen diesen Nicht-Topic-Hosts werden auch keine Kanäle eingerichtet.
- Wenn die veröffentlichende Anwendung mit einem Warteschlangenmanager verbunden ist, auf dem das Thema bereitgestellt wird, werden die veröffentlichten Nachrichten direkt - ohne zwischengeschaltete "Hops" - an die Warteschlangenmanager weitergeleitet, auf dem entsprechende Subskriptionen erstellt wurden. Ebenso werden für ein Thema veröffentlichte Nachrichten, wenn die passenden Subskriptionen auf dem einzigen Warteschlangenmanager erstellt wurden, auf dem das Thema bereitgestellt wird, direkt - ohne zwischengeschaltete "Hops" - an diesen Warteschlangenmanager weitergeleitet.
- Subskriptionen, die sich auf demselben Warteschlangenmanager wie der Publisher befinden, werden unmittelbar bedient, ohne dass die Veröffentlichungen zuerst an die Hosts des Topic-Objekts weitergeleitet werden.

Wie Clusterwarteschlangen können auch mehrere Warteschlangenmanager das gleiche Topic-Verwaltungsobjekt konfigurieren. Dadurch wird eine höhere Verfügbarkeit des Nachrichtenroutings sowie eine horizontale Skalierung durch Lastausgleich sichergestellt. Wenn auf mehreren Warteschlangenmanagern das gleiche benannte Thema für den gleichen Zweig der Themenstruktur konfiguriert wird, wird bei

Topic-Objekten, die über Topic-Hosts weitergeleitet werden, jeder Topic-Host über die von den einzelnen Warteschlangenmanagern subskribierten Themen informiert.

- Eine veröffentlichte Nachricht wird an einen der Topic-Host-Warteschlangenmanager weitergeleitet, von dem aus die Nachricht an die Warteschlangenmanager mit der entsprechenden Subskription weitergeleitet wird. Bei der Auswahl des Topic-Host-Warteschlangenmanagers wird nach denselben Standardlastausgleichsregeln vorgegangen wie bei Punkt-zu-Punkt-Clusterwarteschlangen.
- Können ein oder auch mehrere Topic-Host-Warteschlangenmanager vom veröffentlichenden Warteschlangenmanager nicht kontaktiert werden, so werden die Nachrichten an die übrigen verfügbaren Topic-Host-Warteschlangenmanager weitergeleitet.

Jede Veröffentlichung zu einem Thema in einem weitergeleiteten Zweig der Themenstruktur wird an einen der Topic-Hosts weitergeleitet, selbst wenn im gesamten Cluster keine Subskription zu diesem Thema vorliegt. Standardmäßig werden Nachrichten von dort aus nur an die Warteschlangenmanager im Cluster gesendet, auf denen eine oder mehrere Subskriptionen vorliegen.

- Erforderlich hierzu ist, dass jeder Topic-Host-Warteschlangenmanager über alle subskribierten Themenzeichenfolgen auf allen Warteschlangenmanagern im Cluster informiert wird.
- Die Informationen zu den subskribierten Themen können durch einen Wechsel zu einem Modell entfernt werden, bei dem alle Veröffentlichungen unabhängig von den auf den einzelnen Warteschlangenmanagern vorliegenden Subskriptionen an alle Warteschlangenmanager des Clusters gesendet werden. Dadurch reduziert sich der Austausch der Subskriptionsdaten, der durch Veröffentlichungen bewirkte Datenverkehr sowie möglicherweise die Anzahl der mit den Topic-Host-Warteschlangenmanagern eingerichteten Kanäle erhöht sich hingegen. Weitere Informationen finden Sie im Abschnitt Subskriptionsleistung in Publish/Subscribe-Netzen.

Publish/Subscribe-Nachrichtenflüsse mit via Topic-Host-Routing weitergeleiteten Cluster-Topics können **nicht** mittels einer Publish/Subscribe-Hierarchie über mehrere Publish/Subscribe-Cluster verlaufen.

Nähere Ausführungen zum Topic-Host-Routing finden Sie im Abschnitt Topic-Host-Routing in Publish/Subscribe-Clustern.

Publish/Subscribe-Hierarchien

Sie haben eine Publish/Subscribe-Hierarchie erstellt, indem Sie Warteschlangenmanager über Kanäle verbunden und dann jeweils zwischen Paaren dieser Warteschlangenmanager eine Child/Parent-Beziehung definiert haben. Eine Nachricht wird in dieser Hierarchie durch die direkten Beziehungen zwischen den Warteschlangenmanagern von einem Publisher zu den Subskriptionen übertragen. Allerdings können mehrere "Hops" erforderlich sein, bis eine Nachricht an ihr Ziel gelangt.

Unabhängig von der Anzahl der Subskribenten der Nachricht auf dem Ziel-Warteschlangenmanager wird jeweils zwischen den Warteschlangenmanagerpaaren nur eine Kopie der Nachricht übertragen. Erst beim Eingang auf einem Warteschlangenmanager mit mehreren Subskriptionen wird die Nachricht entsprechend der Anzahl der Subskriptionen dupliziert.

Standardmäßig werden Nachrichten nur an diejenigen Warteschlangenmanager in der Hierarchie gesendet, die direkt auf dem Weg zu einer Subskription auf einem anderen Warteschlangenmanager liegen:

- Es ist Aufgabe jedes Warteschlangenmanagers, jede direkte Beziehung über alle Themen zu informieren, für die zur Zeit auf dem gleichen oder dem anderen Warteschlangenmanager der jeweiligen Beziehung eine oder mehrere Subskriptionen vorliegen. Dies führt dazu, dass allen Warteschlangenmanagern in der Hierarchie alle subskribierten Themen bekannt sind.
- Dieses Verhalten kann jedoch auch geändert werden, sodass Veröffentlichungen generell an alle Warteschlangenmanager der Hierarchie gesendet werden, unabhängig davon, ob auf dem jeweiligen Warteschlangenmanager Subskriptionen vorliegen. In diesem Fall brauchen innerhalb der Hierarchie keine Subskriptionsdaten übertragen zu werden, dafür erhöht sich der Datenverkehr für die Veröffentlichungen.

Wenn Sie einen Cluster erstellen, müssen Sie darauf achten, dass keine Schleife entsteht, über die die Nachrichten endlos innerhalb des Netzes kreisen. In einer Hierarchie ist eine solche Schleife erst gar nicht möglich.

Jeder Warteschlangenmanager muss einen eindeutigen Namen haben.

Publish/Subscribe-Nachrichtenflüsse können über mehrere Publish/Subscribe-Cluster verlaufen. Dazu brauchen Sie lediglich einen Warteschlangenmanager aus jedem Cluster zu einer Hierarchie zusammenzufügen.

Nähere Ausführungen zum Routing innerhalb einer Hierarchie finden Sie im Abschnitt Routing in Publish/Subscribe-Hierarchien.

Proxy-Subskriptionen in einem Publish/Subscribe-Netz

Eine Proxy-Subskription ist eine Subskription, die von einem Warteschlangenmanager für Themen eingerichtet wird, die auf einem anderen Warteschlangenmanager veröffentlicht werden. Eine Proxy-Subskription fließt zwischen Warteschlangenmanagern für jede einzelne Themenzeichenfolge, für die eine Subskription eingerichtet wurde. Sie müssen Proxy-Subskriptionen nicht explizit erstellen; das macht der Warteschlangenmanager automatisch für Sie.

Sie können Warteschlangenmanager in einem Publish/Subscribe-Cluster oder in einer Publish/Subscribe-Hierarchie miteinander verbinden. Proxy-Subskriptionen fließen zwischen den verbundenen Warteschlangenmanagern. Proxy-Subskriptionen sorgen dafür, dass Veröffentlichungen für ein Thema, die von einem Publisher erstellt werden, der mit einem der Warteschlangenmanager verbunden ist, auch von Subskribenten des Themas empfangen werden, die mit anderen Warteschlangenmanagern verbunden sind. Siehe „Verteilte Publish/Subscribe-Netzwerke” auf Seite 96.

In Publish/Subscribe-Topologien, in denen es Tausende von Subskriptionen für einzelne Themenzeichenfolgen gibt oder in denen die Anzahl dieser Subskriptionen möglicherweise stark schwankt, muss der Aufwand für die Weitergabe der Proxy-Subskriptionen berücksichtigt werden. Neben der im verbleibenden Teil dieses Abschnitts beschriebenen Aggregation können Sie manuelle Konfigurationsänderungen vornehmen, die den Fluss der Proxy-Subskriptionen und Veröffentlichungen zwischen verbundenen Warteschlangenmanagern weiterhin einschränken und die Latenzzeit reduzieren, bis eine Proxy-Subskription an alle verbundenen Warteschlangenmanager weitergeleitet ist. Weitere Informationen finden Sie im Abschnitt Subskriptionsleistung in Publish/Subscribe-Netzen.

Proxy-Subskriptionen enthalten keine von lokalen Subskriptionen verwendeten Selektoren, und Subskriptions-Themenzeichenfolgen, die Platzhalterzeichen enthalten, könnten vereinfacht werden. Dies kann zu mit Proxy-Subskriptionen übereinstimmenden Veröffentlichungen führen, obwohl die eigentliche Subskription nicht übereinstimmt, was wiederum dazu führt, dass zwischen den Warteschlangenmanagern zusätzliche Veröffentlichungen übertragen werden. Der die Subskriptionen bereitstellende Warteschlangenmanager filtert solche Abweichungen aus, sodass zusätzliche Veröffentlichungen nicht an die Subskriptionen zurückgegeben werden.

Zusammenfassung von Proxy-Subskriptionen

Proxy-Subskriptionen werden mithilfe eines Duplikateliminierungssystems zusammengefasst. Für eine einzelne aufgelöste Themenzeichenfolge wird bei der ersten lokalen Subskription oder empfangenen Proxy-Subskription eine Proxy-Subskription gesendet. Nachfolgende Subskriptionen für dieselbe Themenzeichenfolge nutzen diese bestehende Proxy-Subskription.

Die Proxy-Subskription wird aufgehoben, nachdem die letzte lokale Subskription oder empfangene Proxy-Subskription aufgehoben wurde.

Zusammenfassung von Veröffentlichungen

Wenn es auf einem Warteschlangenmanager mehrere Subskriptionen für dieselbe Themenzeichenfolge gibt, senden andere Warteschlangenmanager in der Publish/Subscribe-Topologie nur eine einzige Kopie von jeder Veröffentlichung für die betreffende Themenzeichenfolge. Bei Eingang der Nachricht liefert der lokale Warteschlangenmanager eine Kopie der Nachricht an alle übereinstimmenden Subskriptionen.

Wenn die Proxy-Subskriptionen Platzhalterzeichen enthalten, ist es möglich, dass mehrere Proxy-Subskriptionen mit der Themenzeichenfolge einer einzelnen Veröffentlichung übereinstimmen. Wird auf einem Warteschlangenmanager eine Nachricht veröffentlicht, die mit mehreren, von einem einzelnen ver-

bundenen Warteschlangenmanager erstellte Proxy-Subskriptionen übereinstimmt, wird nur eine einzige Kopie der Veröffentlichung an den fernen Warteschlangenmanager weitergeleitet, um alle vorhandenen Proxy-Subskriptionen zu bedienen.

Zugehörige Konzepte

Schleifenermittlung in einem Netz mit verteiltem Publish/Subscribe

Platzhalter in Proxy-Subskriptionen

Zum Abgleich mehrerer Themenzeichenfolgen in Veröffentlichungen können Subskriptionen Platzhalterzeichen in Themenzeichenfolgen verwenden.

Für Subskriptionen sind zwei Schemas für Platzhalterzeichen möglich: *topic-based* und *character-based*. Siehe„Platzhalterschemas” auf Seite 76.

In IBM MQ werden alle Proxy-Subskriptionen für Platzhaltersubskriptionen für die Verwendung von themenbasierten Platzhalterzeichen konvertiert. Zeichenbasierte (character-based) Platzhalterzeichen werden bis zurück zum nächsten / durch das Zeichen # ersetzt. Zum Beispiel wird `/aaa/bbb/c*d` umgewandelt in `/aaa/bbb/#`. Die Umwandlung führt dazu, dass ferne Warteschlangenmanager etwas mehr Veröffentlichungen senden, als explizit subskribiert wurden. Die zusätzlichen Veröffentlichungen werden vom lokalen Warteschlangenmanager ausgefiltert, wenn er die Veröffentlichungen an seine lokalen Subskribenten weitergibt.

Verwendung von Platzhaltern mit der Eigenschaft WILDCARD steuern

Mit der MQSC-Eigenschaft **`Topic`** WILDCARD oder der entsprechenden PCF-Eigenschaft Topic `WildcardOperation` können Sie die Zustellung von Veröffentlichungen an Subskribentenanwendungen steuern, die Namen von Platzhalterthemenzeichenfolgen verwenden. Die Eigenschaft WILDCARD kann einen von zwei möglichen Werten haben:

WILDCARD
Aktionen von Subskriptionen mit Platzhaltern bezüglich dieses Themas.

PASSTHRU
Subskriptionen für ein Thema mit Platzhalter, das weniger spezifisch ist als die Themenzeichenfolge für dieses Themenobjekt, empfangen Veröffentlichungen zu diesem Thema und zu spezifischeren Themenzeichenfolge.

BLOCK
Subskriptionen für ein Thema mit Platzhalter, das weniger spezifisch ist als die Themenzeichenfolge für dieses Themenobjekt, empfangen keine Veröffentlichungen zu diesem Thema und zu spezifischeren Themenzeichenfolge.

Der Wert für dieses Attribut wird bei der Definition von Subskriptionen verwendet. Wenn Sie dieses Attribut ändern, ist die Gruppe von Themen, die bereits durch vorhandene Subskriptionen abgedeckt sind, nicht durch die Änderung betroffen. Dieses Szenario gilt auch, wenn sich durch die Erstellung oder das Löschen von Themenobjekten die Topologie ändert; die Themen mit Subskriptionen, die nach der Änderung des Attributs WILDCARD erstellt wurden, werden mit der geänderten Topologie erstellt. Wenn die Themen mit den vorhandenen Subskriptionen übereinstimmen sollen, müssen Sie den Warteschlangenmanager neu starten.

Führen Sie die Schritte im Beispiel „Beispiel: Publish/Subscribe-Cluster Sport erstellen” auf Seite 90 aus, um die in Abbildung 23 auf Seite 87 gezeigte Themenstruktur zu erstellen.

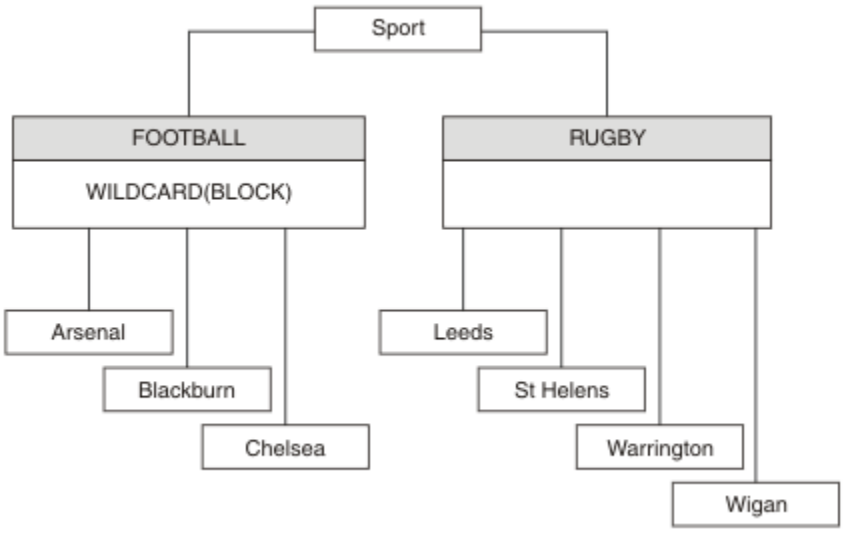


Abbildung 36. Themenstruktur, die die WILDCARD-Eigenschaft BLOCK verwendet

Ein Subskribent, der die Platzhalter-Themenzeichenfolge # verwendet, empfängt alle Veröffentlichungen für das Thema `Sport` und die Unterverzeichnisstruktur `Sport/Rugby` Der Subskribent empfängt keine Veröffentlichungen für die untergeordnete Baumstruktur `Sport/Football` , weil der Wert der Eigenschaft `WILDCARD` des Themas `Sport/Football` BLOCKist.

Die Standardeinstellung lautet PASSTHRU. Sie können den Wert PASSTHRU der Eigenschaft `WILDCARD` auf Knoten in der Baumstruktur `Sport` festlegen. Wenn die Knoten nicht über den Eigenschaftswert `WILDCARD` verfügen BLOCK, ändert die Einstellung PASSTHRU nicht das Verhalten, das von Subskribenten für Knoten in der `Sports` -Baumstruktur beobachtet wird.

Erstellen Sie im Beispiel Subskriptionen, um zu sehen, wie sich die Platzhaltereinstellung auf die bereitgestellten Veröffentlichungen auswirkt (siehe dazu Abbildung 27 auf Seite 92). Führen Sie den Publish-Befehl in Abbildung 30 auf Seite 93 aus, um einige Veröffentlichungen zu erstellen.

```
pub QMA
```

Abbildung 37. Für QMA publizieren

Die Ergebnisse werden in Tabelle 3 auf Seite 87 gezeigt. Beachten Sie, wie die Einstellung des `WILDCARD`-Eigenschaftswerts BLOCK verhindert, dass Subskriptionen mit Platzhaltern Veröffentlichungen für Themen empfangen, die innerhalb des Platzhalterumfangs liegen.

Tabelle 6. Auf QMA empfangene Veröffentlichungen

Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			Alle Veröffentlichungen zur Unterverzeichnisstruktur 'Football', die von `WILDCARD(BLOCK)` auf `Sports/Football` blockiert werden

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` on `Sports/Football` verhindert Platzhaltersubskription unter `Arsenal`
SLEEDS	`Sports/#/Leeds`	`Sports/Rugby/Leeds`	Standardwert `WILDCARD` für 'Sport/Rugby' verhindert keine Platzhaltersubskription auf `Leeds`.

Anmerkung:

Angenommen, eine Subskription hat einen Platzhalter, der einem Themenobjekt mit dem WILDCARD-Eigenschaftswert BLOCK entspricht. Wenn die Subskription auch eine Themenzeichenfolge rechts neben dem entsprechenden Platzhalter hat, empfängt die Subskription niemals Veröffentlichungen. Die nicht geblockten Veröffentlichungen sind Veröffentlichungen für Themen, die dem geblockten Platzhalter übergeordnet sind. Veröffentlichungen für Themen, die dem Thema mit dem Eigenschaftswert BLOCK untergeordnet sind, werden vom Platzhalter geblockt. Subskriptionsthemen-Zeichenfolgen, die ein Thema rechts neben dem Platzhalter einschließen, empfangen daher niemals passende Veröffentlichungen.

Wenn der Eigenschaftswert WILDCARD auf BLOCK gesetzt wird, ist keine Subskription mithilfe einer Themenzeichenfolge möglich, die Platzhalter beinhaltet. Eine solche Subskription ist normal. Die Subskription hat ein explizites Thema, das dem Thema mit einem Themenobjekt entspricht, das den WILDCARD-Eigenschaftswert BLOCK hat. Sie verwendet Platzhalter für Themen, die dem WILDCARD-Eigenschaftswert BLOCK über- oder untergeordnet sind. Im Beispiel in Abbildung 23 auf Seite 87kann eine Subskription wie Sports/Football/# Veröffentlichungen empfangen.

Platzhalter und Cluster-Topics

Definitionen von Cluster-Topics werden an jeden Warteschlangenmanager in einem Cluster weitergegeben. Die Subskription eines Cluster-Topics auf einem Warteschlangenmanager in einem Cluster führt dazu, dass der Warteschlangenmanager Proxy-Subskriptionen erstellt. Eine Proxy-Subskription wird in jedem Warteschlangenmanager im Cluster erstellt. Subskriptionen, die Themenzeichenfolgen mit Platzhaltern verwenden, kombiniert mit Cluster-Topics, lassen das Verhalten schwer vorhersagen. Das Verhalten wird im folgenden Beispiel erläutert.

In dem für das Beispiel verwendeten Cluster „Beispiel: Publish/Subscribe-Cluster Sport erstellen” auf Seite 90 hat QMB dieselben Subskriptionen wie QMA. QMB hat jedoch keine Veröffentlichungen empfangen, nachdem der Publisher an QMA veröffentlicht hat (siehe Abbildung 24 auf Seite 87). Obwohl die Themen Sports/Football und Sports/Rugby Clusterthemen sind, verweisen die in fullsubs.tst definierten Subskriptionen nicht auf ein Clusterthema. Es werden keine Proxy-Subskriptionen von QMB an QMA weitergegeben. Ohne Proxy-Subskriptionen werden keine Veröffentlichungen für QMA an QMB weitergeleitet.

Einige Subskriptionen, wie z. B. Sports/#/Leeds, scheinen möglicherweise auf ein Clusterthema zu verweisen, in diesem Fall Sports/Rugby. Die Sports/#/Leeds -Subskription wird tatsächlich in das Themenobjekt SYSTEM.BASE.TOPICaufgelöst.

Die Regel zum Auflösen des Themenobjekts, das von einer Subskription wie Sports/#/Leeds referenziert wird, lautet wie folgt. Schneiden Sie die Themenzeichenfolge bis zum ersten Platzhalter ab. Suchen Sie links über die Themenzeichenfolge nach dem ersten Thema mit einem zugehörigen Verwaltungsthemenobjekt. Das Themenobjekt kann einen Clusternamen angeben oder ein lokales Themenobjekt definieren. Im Beispiel Sports/#/Leedsist die Themenzeichenfolge nach dem Abschneiden Sports, die kein Themenobjekt hat, und Sports/#/Leeds übernimmt daher von SYSTEM.BASE.TOPIC, einem lokalen Themenobjekt.

Um zu sehen, wie die Subskription von Cluster-Topics die Weitergabe von Platzhaltern ändern kann, führen Sie das Stapelscript upsubs.bat aus. Das Script löscht die Subskriptionswarteschlangen und fügt die Subskriptionen des Cluster-Topics zu fullsubs.tst hinzu. Führen Sie puba.bat erneut aus, um einen Stapel von Veröffentlichungen zu erstellen (siehe Abbildung 24 auf Seite 87).

Tabelle 4 auf Seite 89 zeigt das Ergebnis, nachdem zwei neue Subskriptionen dem Warteschlangenmanager hinzugefügt wurden, in dem die Veröffentlichungen veröffentlicht wurden. Wie erwartet, empfangen die neuen Subskriptionen je eine Veröffentlichung, und die Anzahl der von den anderen Subskriptionen empfangenen Veröffentlichungen ist unverändert. Die unerwarteten Resultate treten im anderen Cluster-Warteschlangenmanager auf, siehe Tabelle 5 auf Seite 89.

Tabelle 7. Auf QMA empfangene Veröffentlichungen			
Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Alle Veröffentlichungen zur Unterverzeichnisstruktur 'Football', die von WILDCARD(BLOCK) auf Sports/Football blockiert werden
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football verhindert Platzhaltersubskription unter Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Standardwert WILDCARD für 'Sport/Rugby' verhindert keine Platzhaltersubskription auf Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal empfängt eine Veröffentlichung, da die Subskription keinen Platzhalter hat.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds würde für jedes Ereignis eine Veröffentlichung empfangen.

Tabelle 5 auf Seite 89 zeigt die Resultate, nachdem die beiden neuen Subskriptionen zu QMB hinzugefügt und auf QMA veröffentlicht wurden. QMB empfing keine Veröffentlichungen ohne diese beiden neuen Subskriptionen. Wie erwartet empfangen die beiden neuen Subskriptionen Veröffentlichungen, da Sports/FootBall und Sports/Rugby Clusterthemen sind. QMB weitergeleitete Proxy-Subskriptionen für Sports/Football/Arsenal und Sports/Rugby/Leeds an QMA, die dann die Veröffentlichungen an QMBgesendet haben

Das unerwartete Ergebnis ist, dass die beiden Subskriptionen Sports/# und Sports/#/Leeds , die zuvor keine Veröffentlichungen empfangen haben, jetzt Veröffentlichungen empfangen. Dies liegt daran, dass die Sports/Football/Arsenal -und Sports/Rugby/Leeds -Veröffentlichungen, die für die anderen Subskriptionen an QMB weitergeleitet werden, jetzt für jeden Subskribenten verfügbar sind, der QMBzugeordnet ist. Folglich empfangen die Subskriptionen für die lokalen Themen Sports/# und Sports/#/Leeds die Veröffentlichung Sports/Rugby/Leeds . Sports/#/Arsenal erhält weiterhin keine Veröffentlichung, weil für Sport/Fußball die Eigenschaft WILDCARD auf BLOCKgesetzt ist.

Tabelle 8. Auf QMB empfangene Veröffentlichungen			
Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football verhindert Platzhaltersubskription unter Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Der Standardwert WILDCARD für Sport/Rugby verhindert nicht das Platzhalterabonnement für Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal empfängt eine Veröffentlichung, da die Subskription keinen Platzhalter hat.

Tabelle 8. Auf QMB empfangene Veröffentlichungen (Forts.)			
Abonnement	**Themenzeichenfolge**	**Empfangene Veröffentlichungen**	**Anmerkungen**
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds würde für jedes Ereignis eine Veröffentlichung empfangen.

Bei den meisten Anwendungen ist es nicht erwünscht, dass eine Subskription das Verhalten einer anderen Subskription beeinflusst. Eine wichtige Verwendung der Eigenschaft WILDCARD mit dem Wert BLOCK ist es, dass sich die Subskriptionen für dieselbe Themenzeichenfolge mit Platzhalterzeichen einheitlich verhalten. Unabhängig davon, ob sich die Subskription in demselben Warteschlangenmanager wie der Publisher oder in einem anderen Warteschlangenmanager befindet, sind die Resultate der Subskription dieselben.

Platzhalter und Datenströme

Wenn eine neue Anwendung für die Publish/Subscribe-API geschrieben wird, empfängt eine Subskription von * keine Veröffentlichungen. Um alle Sportveröffentlichungen zu erhalten, müssen Sie Sports/*oder Sports/#und ähnlich für Business -Veröffentlichungen subskribieren.

Das Verhalten einer vorhandenen Publish/Subscribe-Anwendung in der Warteschlange ändert sich nicht, wenn der Publish/Subscribe-Broker auf eine höhere Version von IBM MQmigriert wird. Die Eigenschaft **StreamName** in den Befehlen **Publish**, **Register Publisher** oder **Subscriber** wird dem Namen des Themas zugeordnet, auf das der Datenstrom migriert wurde.

Platzhalter und Subskriptionspunkte

Wenn eine neue Anwendung für die Publish/Subscribe-API geschrieben wird, empfängt eine Subskription von * keine Veröffentlichungen. Um alle Sportveröffentlichungen zu erhalten, müssen Sie Sports/*oder Sports/#und ähnlich für Business -Veröffentlichungen subskribieren.

Das Verhalten einer vorhandenen Publish/Subscribe-Anwendung in der Warteschlange ändert sich nicht, wenn der Publish/Subscribe-Broker auf eine höhere Version von IBM MQmigriert wird. Die Eigenschaft **SubPoint** in den Befehlen **Publish**, **Register Publisher** oder **Subscriber** wird dem Namen des Themas zugeordnet, auf das die Subskription migriert wurde.

Beispiel: Publish/Subscribe-Cluster Sport erstellen

In den nachfolgenden Schritten wird ein Cluster CL1 mit vier Warteschlangenmanagern erstellt: zwei vollständige Repositorys, CL1A und CL1B, und zwei Teilrepositorys, QMA und QMB. Die vollständigen Repositorys enthalten nur Clusterdefinitionen. QMA wird als Cluster-Topic-Host festgelegt. Permanente Subskriptionen sind in QMA und QMB definiert.

Anmerkung: Das Beispiel ist für Windows codiert. Wenn Sie das Beispiel auf anderen Plattformen konfigurieren und testen möchten, müssen Sie den Programmcode für Create qmgrs.bat und create pub.bat ändern.

1. Erstellen Sie die Scriptdateien.
   a. Create topics.tst
   b. Create wildsubs.tst
   c. Create fullsubs.tst
   d. Create qmgrs.bat
   e. create pub.bat
2. Führen Sie Create qmgrs.bat aus, um die Konfiguration zu erstellen.

```
qmgrs
```

Erstellen Sie die Themen in Abbildung 23 auf Seite 87. Das Script in Abbildung 5 erstellt die Clusterthemen `Sports/Football` und `Sports/Rugby`.

Anmerkung: Die Option REPLACE ersetzt nicht die TOPICSTR-Eigenschaften eines Themas. TOPICSTR ist eine Eigenschaft, die im Beispiel gezielt variiert wird, um verschiedene Themenstrukturen zu testen. Wenn Sie ein Thema ändern möchten, müssen Sie das Thema zuerst löschen.

```
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')
```

Abbildung 38. Themen löschen und erstellen: topics.tst

Anmerkung: Löschen Sie die Themen, da REPLACE keine Themenzeichenfolgen ersetzt.

Erstellen Sie Subskriptionen mit Platzhaltern. Die Platzhalter entsprechen den Themen mit Themenobjekten in Abbildung 23 auf Seite 87. Erstellen Sie für jede Subskription eine Warteschlange. Beim Ausführen bzw. erneuten Ausführen des Scripts werden die Warteschlangen geleert, und die Subskriptionen werden gelöscht.

Anmerkung: Die Option REPLACE ersetzt nicht die die Eigenschaften TOPICOBJ oder TOPICSTR einer Subskription. TOPICOBJ oder TOPICSTR sind die Eigenschaften, die im Beispiel sinnvoll variiert werden, um verschiedene Subskriptionen zu testen. Um sie zu ändern, löschen Sie zunächst die Subskription.

```
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)
```

Abbildung 39. Platzhaltersubskriptionen erstellen: wildsubs.tst

Erstellen Sie Subskriptionen, die die Cluster-Topic-Objekte referenzieren.

Anmerkung:

Das Begrenzungszeichen / wird automatisch zwischen die von TOPICOBJ referenzierte Themenzeichenfolge und die von TOPICSTR definierte Themenzeichenfolge eingefügt.

Die Definition `DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL)` erstellt dieselbe Subskription. Die Verwendung von TOPICOBJ ist eine zeiteffiziente Methode, bereits definierte Themenzeichenfolgen zu referenzieren. Die erstellte Subskription referenziert das Themenobjekt nicht länger.

```
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)
```

Abbildung 40. Subskriptionen löschen und erstellen: fullsubs.tst

Erstellen Sie einen Cluster mit zwei Repositorys. Erstellen Sie zwei Teilrepositorys zum Veröffentlichen und Abonnieren. Führen Sie das Script erneut aus, um alles zu löschen und neu zu starten. Das Script erstellt zudem die Themenhierarchie und die ursprünglichen Platzhaltersubskriptionen.

Anmerkung:

Schreiben Sie auf anderen Plattformen ein ähnliches Script, oder geben Sie alle Befehle ein. Mit einem Script lässt sich alles schneller löschen und mit einer identischen Konfiguration erneut starten.

```
@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%)') CLUS
TER(CL1) REPLACE | runmqsc %1
goto:eof
```

Abbildung 41. Warteschlangenmanager erstellen: qmgrs.bat

Aktualisieren Sie die Konfiguration durch Hinzufügen der Subskriptionen zu den Cluster-Topics.

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

Abbildung 42. Subskriptionen aktualisieren: upsubs.bat

Führen Sie `pub.bat` mit einem Warteschlangenmanager als Parameter aus, um Nachrichten zu veröffentlichen, die die Veröffentlichungsthema-Zeichenfolge enthalten. `Pub.bat` verwendet das Beispielprogramm **amqspub**.

```
@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
```

Abbildung 43. Veröffentlichen: pub.bat

Zugehörige Konzepte

Subskriptionen mit Platzhalterzeichen und ständige Veröffentlichungen

Veröffentlichungsumfang

In einem Publish/Subscribe-Cluster bzw. einer Publish/Subscribe-Hierarchie bestimmt auch der Bereich einer Veröffentlichung, ob Warteschlangenmanager die Veröffentlichung an ferne Warteschlangenmanager weiterleiten. Verwalten Sie den Bereich von Veröffentlichungen mit dem Themenattribut **PUBSCOPE**.

Wenn eine Veröffentlichung nicht an ferne Warteschlangenmanager weitergeleitet wird, empfangen nur lokale Subskribenten die Veröffentlichung.

In einem Publish/Subscribe-Cluster wird der Veröffentlichungsumfang weitgehend durch die an bestimmten Stellen der Themenstruktur definierten Cluster-Themenobjekten gesteuert. Der Veröffentlichungsumfang muss festgelegt werden, damit Veröffentlichungen an andere Warteschlangenmanager im Cluster übertragen werden können. Sie sollten den Veröffentlichungsumfang eines Cluster-Topics nur dann einschränken, wenn Sie auf bestimmten Warteschlangenmanagern feinste Kontrolle über bestimmte Themen benötigen.

In einer Publish/Subscribe-Hierarchie wird der Veröffentlichungsumfang weitgehend durch dieses Attribut in Verbindung mit dem Attribut subscription scope (Veröffentlichungsumfang) gesteuert.

Das Attribut **PUBSCOPE** bestimmt den Veröffentlichungsumfang eines bestimmten Themas. Sie können das Attribut auf einen der folgenden Werte setzen:

QMGR
: Die Veröffentlichung wird nur an lokale Subskribenten übermittelt. Diese Veröffentlichungen werden als *lokale Veröffentlichungen* bezeichnet. Lokale Veröffentlichungen werden nicht an ferne Warteschlangenmanager weitergeleitet und deshalb nicht von Subskribenten empfangen, die mit fernen Warteschlangenmanagern verbunden sind.

ALLE
: Die Veröffentlichung wird an lokale Subskribenten und an Subskribenten, die in einem Publish/Subscribe-Cluster mit fernen Warteschlangenmanagern verbunden sind, übermittelt. Diese Veröffentlichungen werden als *globale Veröffentlichungen* bezeichnet.

ASPARENT
: Verwenden Sie die Einstellung **PUBSCOPE** des übergeordneten Themas in der Themenstruktur.

Publisher können auch mit der Nachrichteneinreihungsoption `MQPMO_SCOPE_QMGR` angeben, ob es sich um eine lokale oder globale Veröffentlichung handelt. Wird diese Option verwendet, setzt sie das Verhalten außer Kraft, das mit dem Themenattribut **PUBSCOPE** festgelegt wurde.

Zugehörige Konzepte

„Verwaltungsthemenobjekte” auf Seite 83
Mit einem Verwaltungsthemenobjekt können Sie Themen bestimmte, nicht standardmäßige Attribute zuweisen.

Zugehörige Tasks

Verteilte Publish/Subscribe-Netze konfigurieren

Subskriptionsumfang

Der Bereich einer Subskription bestimmt, ob eine Subskription auf einem Warteschlangenmanager nur Veröffentlichungen von lokalen Publishern empfängt oder auch Veröffentlichungen, die auf einem ande-

ren Warteschlangenmanager in einem Publish/Subscribe-Cluster oder einer Publish/Subscribe-Hierarchie erfolgen.

Eine Begrenzung des Subskriptionsbereich auf einen Warteschlangenmanager verhindert, dass Proxy-Subskriptionen an andere Warteschlangenmanager in der Publish/Subscribe-Topologie weitergeleitet werden. Dadurch wird der Publish/Subscribe-Nachrichtenübertragungsverkehr zwischen den Warteschlangenmanagern reduziert.

In einem Publish/Subscribe-Cluster wird der Subskriptionsumfang weitgehend durch die an bestimmten Stellen der Themenstruktur definierten Cluster-Themenobjekten gesteuert. Der Subskriptionsumfang muss festgelegt werden, damit Proxy-Subskriptionen an andere Warteschlangenmanager im Cluster übertragen werden können. Sie sollten den Subskriptionsumfang eines Cluster-Topics nur dann einschränken, wenn Sie auf bestimmten Warteschlangenmanagern feinste Kontrolle über bestimmte Themen benötigen.

In einer Publish/Subscribe-Hierarchie wird der Subskriptionsumfang weitgehend durch dieses Attribut in Verbindung mit dem Attribut publication scope (Veröffentlichungsumfang) gesteuert.

Mithilfe des Topic-Attributs **SUBSCOPE** wird der Umfang der Subskriptionen für ein bestimmtes Thema ermittelt. Sie können das Attribut auf einen der folgenden Werte setzen:

QMGR
Eine Subskription empfängt nur lokale Veröffentlichungen und Proxy-Subskriptionen werden nicht an ferne Warteschlangenmanager weitergegeben.

ALLE
Eine Proxy-Subskription wird an ferne Warteschlangenmanager in einem Publish/Subscribe-Cluster oder einer Publish/Subscribe-Hierarchie weitergegeben und der Subskribent empfängt lokale und ferne Veröffentlichungen.

ASPARENT
Verwenden Sie die Einstellung **SUBSCOPE** des übergeordneten Themas in der Themenstruktur.

Wenn der Subskriptionsbereich für ein Thema auf ALLE gesetzt ist (entweder direkt oder über ASPARENT aufgelöst), können einzelne Subskriptionen für dieses Thema ihren Geltungsbereich auf QMGR beschränken, indem sie beim Erstellen der Subskription MQSO_SCOPE_QMGR angeben. Für eine Themensubskription, deren Umfang auf QMGR gesetzt ist, kann der Umfang jedoch nicht umgekehrt auf ALL erweitert werden.

Zugehörige Konzepte
„Verwaltungsthemenobjekte” auf Seite 83
Mit einem Verwaltungsthemenobjekt können Sie Themen bestimmte, nicht standardmäßige Attribute zuweisen.

Zugehörige Tasks
Verteilte Publish/Subscribe-Netze konfigurieren

Themenbereiche

Ein Themenbereich ist eine Gruppe von Themen, die Sie abonnieren oder veröffentlichen können. Ein Warteschlangenmanager in einer Topologie mit verteiltem Publish/Subscribe verfügt über einen Themenbereich, der potenziell Themen einschließt, die auf verbundenen Warteschlangenmanagern in dieser Topologie subskribiert oder veröffentlicht wurden.

Anmerkung: Eine Übersicht über die Themen innerhalb eines Warteschlangenmanagers wie administrative Themenobjekte, Themenzeichenfolgen und Themenstrukturen finden Sie im Abschnitt „Themen” auf Seite 74. Weitere Verweise auf Themen (*topics*) im aktuellen Artikel beziehen sich, wenn nicht anders angegeben, auf Themenzeichenfolgen (*topic strings*).

Themen werden zunächst auf eine der folgenden Arten erstellt:

- administrativ, wenn Sie ein Themenobjekt oder eine permanente Subskription definieren
- dynamisch, wenn eine Anwendung eine Veröffentlichung oder Subskription für ein neues Thema erstellt

Themen werden sowohl über Proxy-Subskriptionen als auch durch die Erstellung administrativer Cluster-Topic-Objekte an andere Warteschlangenmanager weitergegeben. Proxy-Subskriptionen bewirken, dass

Veröffentlichungen von dem Warteschlangenmanager, mit dem ein Publisher verbunden ist, an die Warteschlangenmanager von Subskribenten weitergeleitet werden.

Proxy-Subskriptionen werden zwischen allen Warteschlangenmanagern weitergegeben, die durch Beziehungen zwischen übergeordneten und untergeordneten Elementen innerhalb einer Warteschlangenmanagerhierarchie miteinander verbunden sind. Daraus folgt, dass Sie auf jedem der Warteschlangenmanager ein Thema abonnieren können, das auf einem der anderen Warteschlangenmanager in der Hierarchie definiert ist. Solange es einen verbundenen Pfad zwischen den Warteschlangenmanagern gibt, spielt es keine Rolle, wie die Warteschlangenmanager verbunden sind.

Proxy-Subskriptionen werden auch für Subskriptionen für Cluster-Topics in einem Publish/Subscribe-Cluster weitergegeben. Ein Cluster-Topic ist ein Thema, das an ein Themenobjekt angehängt ist, das entweder selbst das Attribut **CLUSTER** besitzt oder das Attribut von seinem übergeordneten Element übernimmt. Themen, die keine Cluster-Topics sind, sind als lokale Themen bekannt und werden nicht im Cluster repliziert. Subskriptionen für lokale Themen geben keine Proxy-Subskriptionen an den Cluster weiter.

Zusammenfassend gesagt, werden also in zwei Situationen Proxy-Subskriptionen für Subskribenten erstellt:

1. Ein Warteschlangenmanager ist Mitglied einer Hierarchie und eine Proxy-Subskription wird an das übergeordnete Element und an die untergeordneten Elemente des Warteschlangenmanagers weitergeleitet.
2. Ein Warteschlangenmanager ist Mitglied eines Clusters und die Subskriptionsthemenzeichenfolge wird in ein Thema aufgelöst, das einem Cluster-Topic-Objekt zugeordnet ist. Bei einem *direkt weitergeleiteten* Cluster-Topic werden Proxy-Subskriptionen an alle Mitglieder des Clusters weitergeleitet. Bei einem über einen *Topic-Host weitergeleiteten* Cluster-Topic werden Proxy-Subskriptionen nur an die Warteschlangenmanager im Cluster weitergeleitet, auf denen das Cluster-Topic-Objekt definiert ist. Weitere Informationen hierzu finden Sie unter „Publish/Subscribe-Cluster” auf Seite 99.

Wenn ein Warteschlangenmanager Mitglied eines Clusters und einer Hierarchie ist, werden Proxy-Subskriptionen durch beide Mechanismen weitergeleitet, ohne das doppelte Veröffentlichungen an den Subskribenten übergeben werden.

In der folgenden Liste werden die Themenbereiche von drei Publish/Subscribe-Topologien beschrieben:

- „Fall 1. Publish/Subscribe-Cluster” auf Seite 113.
- „Fall 2. Publish/Subscribe-Hierarchien” auf Seite 114.

In separaten Abschnitten wird anhand der folgenden Konfigurationsaufgaben beschrieben, wie Themenbereiche zusammengefasst werden:

- Einzelnen Themenbereich in einem Publish/Subscribe-Cluster erstellen
- Themenbereiche mehrerer Cluster zusammenfassen
- Themenbereiche in mehreren Clustern zusammenfassen und isolieren
- Veröffentlichungen und Subskriptionen für Themenbereiche in mehreren Clustern durchführen

Fall 1. Publish/Subscribe-Cluster

Im Beispiel wird vorausgesetzt, dass der Warteschlangenmanager nicht mit einer Publish/Subscribe-Hierarchie verbunden ist.

Wenn ein Warteschlangenmanager Mitglied eines Publish/Subscribe-Clusters ist, besteht sein Themenbereich aus lokalen Themen und Cluster-Topics. Lokale Themen sind Themenobjekten ohne das Attribut **CLUSTER** zugeordnet. Wenn ein Warteschlangenmanager über lokale Themenobjektdefinitionen verfügt, unterscheidet sich sein Themenbereich von dem eines anderen Warteschlangenmanagers im Cluster, der ebenfalls über eigene, lokal definierte Themenobjekte verfügt.

In einem Publish/Subscribe-Cluster können Sie kein Thema abonnieren, das auf einem anderen Warteschlangenmanager definiert ist, außer wenn das Thema, das Sie abonnieren, in ein Cluster-Topic-Objekt aufgelöst wird.

Wenn die gleichen benannten Definitionen eines Cluster-Topic-Objekts auf mehreren Warteschlangenmanagern erforderlich sind (z. B. bei Verwendung des *Topic-Host-Routing*, müssen alle Definitionen an den erforderlichen Stellen übereinstimmen. Weitere Informationen hierzu finden Sie im Abschnitt Einzelnen Themenbereich in einem Publish/Subscribe-Cluster erstellen.

Eine lokale Definition eines Themenobjekts, sei sie nun für ein Cluster-Topic oder ein lokales Thema, hat Vorrang vor demselben Themenobjekt, das anderswo im Cluster definiert ist. Es wird das lokal definierte Thema verwendet, auch wenn das anderswo definierte Thema jünger ist.

Es ist wichtig, dass ein Cluster-Topic-Objekt überall im Cluster derselben Themenzeichenfolge zugeordnet ist. Sie können die Themenzeichenfolge, der ein Themenobjekt zugeordnet ist, nicht ändern. Um dasselbe Themenobjekt einer anderen Themenzeichenfolge zuzuordnen, müssen Sie das Themenobjekt löschen und es mit der neuen Themenzeichenfolge erneut erstellen. Wenn das Thema zu einem Cluster gehört, bedeutet dies, dass die Kopien des Themenobjekts, die auf den anderen Mitgliedern des Clusters gespeichert sind, gelöscht und dann überall im Cluster Kopien des neuen Themenobjekts erstellt werden müssen. Die Kopien des Themenobjekts verweisen alle auf dieselbe Themenzeichenfolge.

Es kann vorkommen, dass versehentlich zwei Definitionen eines gleichnamigen Themenobjekts auf unterschiedlichen Warteschlangenmanagern im Cluster mit zwei unterschiedlichen Themenzeichenfolgen erstellt werden. Dies kann zu einem verwirrenden Verhalten führen, da mehrere Definitionen desselben Themenobjekts mit unterschiedlichen Themenzeichenfolgen je nachdem, wie und wo das Thema referenziert wird, zu unterschiedlichen Ergebnissen führen können. Weitere Informationen zu diesem wichtigen Punkt finden Sie im Abschnitt Mehrere Cluster-Topic-Definitionen mit gleichen Namen.

Fall 2. Publish/Subscribe-Hierarchien

Im Beispiel wird vorausgesetzt, dass der Warteschlangenmanager nicht Mitglied eines Publish/Subscribe-Clusters ist.

Wenn in IBM MQein Warteschlangenmanager Mitglied einer Publish/Subscribe-Hierarchie ist, besteht sein Topicbereich aus allen Themen, die lokal und auf verbundenen Warteschlangenmanagern definiert sind. Der Themenbereich ist bei allen Warteschlangenmanagern in einer Hierarchie derselbe. Es gibt keine Unterscheidung der Themen in lokale Themen und globale Themen.

Setzen Sie entweder die Option **PUBSCOPE** oder die Option **SUBSCOPE** auf `QMGR`, um zu verhindern, dass eine Veröffentlichung für ein Thema von einem Publisher an einen Subskribenten, der mit anderen Warteschlangenmanagern in der Hierarchie verbunden ist, übergeben wird.

Angenommen, Sie definieren ein Themenobjekt namens `Alabama` mit der Themenzeichenfolge `USA/Alabama` in Warteschlangenmanager `QMA`. Das würde zu folgenden Ergebnissen führen:

1. Der Themenbereich in `QMA` enthält jetzt auch das Themenobjekt `Alabama` und die Themenzeichenfolge `USA/Alabama`.
2. Eine Anwendung oder ein Administrator kann in `QMA` eine Subskription mit dem Themenobjektnamen `Alabama` erstellen.
3. Eine Anwendung kann in jedem Warteschlangenmanager in der Hierarchie eine Subskription für jedes Thema, einschließlich `USA/Alabama`, erstellen. Wenn `QMA` nicht lokal definiert wurde, wird das Thema `USA/Alabama` in das Themenobjekt `SYSTEM.BASE.TOPIC` aufgelöst.

Zugehörige Konzepte

„Veröffentlichungsumfang” auf Seite 111
In einem Publish/Subscribe-Cluster bzw. einer Publish/Subscribe-Hierarchie bestimmt auch der Bereich einer Veröffentlichung, ob Warteschlangenmanager die Veröffentlichung an ferne Warteschlangenmanager weiterleiten. Verwalten Sie den Bereich von Veröffentlichungen mit dem Themenattribut **PUBSCOPE**.

„Subskriptionsumfang” auf Seite 111
Der Bereich einer Subskription bestimmt, ob eine Subskription auf einem Warteschlangenmanager nur Veröffentlichungen von lokalen Publishern empfängt oder auch Veröffentlichungen, die auf einem anderen Warteschlangenmanager in einem Publish/Subscribe-Cluster oder einer Publish/Subscribe-Hierarchie erfolgen.

Zugehörige Tasks

Verteilte Publish/Subscribe-Netze konfigurieren

IBM MQ Multicasting

IBM MQ Multicast bietet eine geringe Latenzzeit, eine hohe Ausgabefächerung und eine zuverlässige Multicasting-Nachrichtenübertragung.

Bei Multicast handelt es sich um eine effiziente Form der Publish/Subscribe-Nachrichtenübermittlung, da eine große Zahl von Subskribenten zugeordnet werden kann, ohne dass es zu negativen Auswirkungen beim Leistungsverhalten kommt. IBM MQ ermöglicht eine zuverlässige Multicasting-Nachrichtenübertragung, indem mithilfe von Empfangsbestätigungen, negativen Rückmeldungen und Folgenummern eine Nachrichtenübertragung mit geringer Latenzzeit und hoher Ausgabefächerung erzielt wird.

Die ausreichende Übermittlung von IBM MQ ermöglicht eine fast gleichzeitige Übermittlung, wodurch sichergestellt wird, dass kein Empfänger einen Vorteil hat. Da IBM MQ Multicast für die Übermittlung von Nachrichten ein Netz verwendet, wird für die Ausgabefächerung der Daten keine Publish/Subscribe-Engine benötigt. Nach der Zuordnung eines Themas zu einer Gruppenadresse ist kein Warteschlangenmanager erforderlich, da Publisher und Subskribenten in einem Peer-to-Peer-Modus arbeiten können. Dies ermöglicht die Reduzierung der Arbeitslast auf Warteschlangenmanager-Servern und der Warteschlangenmanager-Server ist keine potenzielle Fehlerquelle mehr.

Ursprüngliche Multicasting-Konzepte

IBM MQ Multicast kann unter Verwendung des Kommunikationsinformationsobjekts COMMINFO problemlos in bestehende Systeme und Anwendungen integriert werden. Zwei TOPIC-Objektfelder ermöglichen die schnelle Konfiguration bestehender TOPIC-Objekte zur Unterstützung bzw. zum Ignorieren von Multicasting-Datenverkehr.

Für Multicasting erforderliche Objekte

Nachfolgend finden Sie eine kurze Übersicht über die beiden Objekte, die für IBM MQ Multicast erforderlich sind:

COMMINFO-Objekt

Das COMMINFO-Objekt enthält die Attribute, die der Multicasting-Übertragung zugeordnet sind. Weitere Informationen zu den Parametern des Befehls COMMINFO finden Sie in DEFINE COMMINFO .

Das einzige COMMINFO-Feld, das angegeben werden MUSS, ist das Feld mit dem Namen des COMMINFO-Objekts. Mit diesem Namen wird das COMMINFO-Objekts für ein Thema ermittelt. Prüfen Sie das Feld **GRPADDR** des COMMINFO-Objekts, um sicherzustellen, dass der Wert eine gültige Multicastgruppenadresse ist.

Themenobjekt

Ein Thema ist der Gegenstand der Informationen, die in einer Publish/Subscribe-Nachricht veröffentlicht werden, und ein Thema wird definiert, indem ein TOPIC-Objekt erstellt wird. Weitere Informationen zu den Parametern des TOPIC-Objekts finden Sie unter DEFINE TOPIC.

Bestehende Themen können für das Multicasting verwendet werden, indem Sie die Werte der folgenden TOPIC-Objektparameter ändern: **COMMINFO** und **MCAST**.

- **COMMINFO** Dieser Parameter gibt den Namen des Multicastkommunikationsinformationsobjekts an.
- **MCAST** Dieser Parameter gibt an, ob Multicasting an dieser Position in der Themenstruktur zulässig ist. Standardmäßig ist **MCAST** auf ASPARENT gesetzt, d. h., das Multicasting-Attribut des Themas wird vom übergeordneten Element übernommen. Wenn Sie **MCAST** auf ENABLED setzen, ist auf diesem Knoten Multicasting-Datenverkehr möglich.

Multicasting-Netze und -Themen

Nachfolgend finden Sie eine Übersicht darüber, was mit Subskriptionen mit unterschiedlichen Subskriptionstypen und Themendefinitionen geschieht. Alle folgenden Beispiele setzen voraus, dass der Parameter **COMMINFO** des TOPIC-Objekts auf den Namen eines gültigen COMMINFO-Objekts gesetzt wurde:

Thema für Multicasting aktiviert (MCAST ENABLED)
Wenn der Parameter der Themenzeichenfolge **MCAST** auf ENABLED gesetzt ist, sind Subskriptionen Multicasting-fähiger Clients zugelassen und eine Multicasting-Subskription wird vorgenommen – mit folgender Ausnahme:

- Es handelt sich um die permanente Subskription eines Multicasting-fähigen Clients.
- Es handelt sich um eine nicht verwaltete Subskription eines Multicasting-fähigen Clients.
- Es handelt sich um die Subskription eines nicht Multicasting-fähigen Clients.

In diesen Fällen wird eine andere Subskription als eine Multicast-Subskription vorgenommen und Subskriptionen werden in ein normales Publish/Subscribe herabgestuft.

Thema für Multicasting inaktiviert
Wenn der Parameter der Themenzeichenfolge **MCAST** auf DISABLED gesetzt ist, wird immer eine Nicht-Multicasting-Subskription vorgenommen und Subskriptionen werden in ein normales Publish/Subscribe herabgestuft.

Thema nur für Multicasting aktiviert
Wenn der Parameter der Themenzeichenfolge **MCAST** auf ONLY gesetzt ist, sind Subskriptionen Multicasting-fähiger Clients zugelassen und eine Multicasting-Subskription wird vorgenommen – mit folgender Ausnahme:

- Es handelt sich um eine permanente Subskription: Permanente Subskriptionen werden mit dem Ursachencode 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED abgelehnt.
- Es handelt sich um eine nicht verwaltete Subskription: Nicht verwaltete Subskriptionen werden mit dem Ursachencode 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR abgelehnt.
- Es handelt sich um eine Subskription von einem nicht Multicast-fähigen Client: Diese Subskriptionen werden mit dem Ursachencode 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY abgelehnt.
- Es handelt sich um eine Subskription aus einer lokal gebundenen Anwendung: Diese Subskriptionen werden mit dem Ursachencode 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY abgelehnt.

Windows Linux AIX Die Übersicht MQ Telemetry

MQ Telemetry umfasst einen Telemetrieservice (MQXR), der Bestandteil eines Warteschlangenmanagers ist, und Telemetrieclients, die Sie selbst schreiben oder kostenlos herunterladen können. Außerdem werden Befehlszeilenschnittstellen und Explorer-Verwaltungsschnittstellen zur Verfügung gestellt. Die Telemetrie bezeichnet die Erfassung von Daten aus zahlreichen fernen Einheiten und deren Verwaltung. Mit MQ Telemetry können Sie die Datensammlung und die Einheitensteuerung in Webanwendungen integrieren.

MQ Telemetry ist eine Komponente von IBM MQ. Ein Upgrade dieser Versionen besteht im Prinzip darin, eine höhere Version von IBM MQ zu installieren.

Beispielanwendungen sind über Eclipse Paho und MQTT.orgfrei verfügbar. Weitere Informationen finden Sie im Abschnitt IBM MQ Telemetry Transport-Beispielprogramme.

Da MQ Telemetry eine Komponente von IBM MQist, kann MQ Telemetry mit dem Hauptprodukt oder nach der Installation des Hauptprodukts installiert werden. Informationen zur Migration finden Sie unter Migration von MQ Telemetry unter Windows und Migration von MQ Telemetry unter Linux.

MQ Telemetry enthält folgende Komponenten:

Telemetriekanäle
Mithilfe von Telemetriekanälen können Sie die Verbindung zwischen MQTT-Clients und IBM MQ verwalten. Telemetriekanäle verwenden neue IBM MQ -Objekte, z. B. SYSTEM.MQTT.TRANSMIT.QUEUE, für die Interaktion mit IBM MQ.

Telemetrieservice (MQXR)
MQTT -Clients verwenden den Telemetrieservice `SYSTEM.MQXR.SERVICE` , um eine Verbindung zu Telemetriekanälen herzustellen.

IBM MQ Explorer-Unterstützung für MQ Telemetry
MQ Telemetry kann unter Verwendung von IBM MQ Explorer verwaltet werden.

Dokumentation
Die MQ Telemetry -Dokumentation ist in der IBM MQ -Standardproduktdokumentation enthalten. Die SDK-Dokumentation für Java und C-Clients wird in der Produktdokumentation sowie als Javadoc und im HTML-Format bereitgestellt.

Telemetriekonzepte

Sie sammeln Informationen aus Ihrer Umgebung und entscheiden dann über Ihre Vorgehensweise. Als Verbraucher überprüfen Sie Ihre Vorräte und entscheiden dann, welche Lebensmittel Sie kaufen. Vor der Buchung einer Anschlussverbindung möchten Sie wissen, wie lange die Reise dauert, wenn Sie jetzt aufbrechen. Sie überprüfen Ihre Symptome und entscheiden dann, ob Sie einen Arzt konsultieren. Sie prüfen, wann ein Bus ankommen wird, und entscheiden dann, ob Sie warten. Die Informationen für diese Entscheidungen stammen direkt aus Zählern und Geräten, aus schriftlichen Unterlagen oder einem Bildschirm oder von Ihnen selbst. Sie sammeln Informationen, kombinieren diese, analysieren sie und handeln auf deren Grundlage - immer und überall.

Wenn die Informationsquellen weit verstreut oder nicht zugänglich sind, wird die Erfassung präzisester Informationen schwierig und aufwändig. Wenn Sie viele Änderungen vornehmen möchten oder es schwierig ist, diese Änderungen vorzunehmen, erfolgen keine Änderungen oder sie erfolgen an Stellen, an denen sie nicht so effektiv sind.

Was wäre, wenn sich der Aufwand der Erfassung von Informationen von den weit verstreuten Geräten und deren Steuerung erheblich verringern ließe, indem die Geräte mithilfe einer digitalen Technologie mit dem Internet verbunden würden? Die Informationen können mithilfe der Ressourcen des Internets und Unternehmens analysiert werden. Sie haben bessere Möglichkeiten für fundierte Entscheidungen und die Ergreifung der entsprechenden Maßnahmen.

Diese Änderungen werden durch technologische Trends und durch den umweltpolitischen und wirtschaftlichen Druck vorangetrieben:

1. Der Aufwand und die Kosten für die Verbindung und Steuerung von Sensoren und Aktuatoren können aufgrund der Standardisierung und Verbindung mit kostengünstigen digitalen Prozessoren reduziert werden.
2. Zur Verbindung von Geräten werden immer häufiger das Internet und Internettechnologien genutzt. In manchen Ländern werden für die Verbindung mit Internetanwendungen mehr Mobiltelefone als Personal Computer verwendet. Mit Sicherheit werden andere Geräte folgen.
3. Das Internet und die Internettechnologien erleichtern einer Anwendung den Abruf von Daten erheblich. Der schnelle Zugriff auf Daten fördert den Einsatz der Datenanalyse, damit Daten von Sensoren in Informationen umgewandelt werden können, die in zahlreichen anderen Lösungen hilfreich sind.
4. Der intelligente Einsatz von Ressourcen ist häufig der schnellere und kostengünstigere Weg zur Reduzierung von CO2-Emissionen und Kosten. Die Alternativen: Die Suche neuer Ressourcen oder die Entwicklung neuer Technologien zur Nutzung bestehender Ressourcen stellt vermutlich die langfristige Lösung dar. Im kurzfristigen Einsatz ist die Entwicklung neuer Technologien oder die Suche nach neuen Ressourcen oft riskanter, langsamer und teurer als die Verbesserung vorhandener Lösungen.

Beispiel

Anhand eines Beispiels soll veranschaulicht werden, wie diese Trends neue Gelegenheiten für eine intelligente Interaktion mit der Umwelt bieten.

Das internationale Übereinkommen zum Schutz des menschlichen Lebens auf See (International Convention for the Safety of Life at Sea (SOLAS)) verlangt auf vielen Schiffen den Einsatz eines automatischen Identifikationssystems (AIS). Dieses System ist auf Handelsschiffen mit über 300 Tonnen und auf Passa-

gierschiffen erforderlich. AIS ist in erster Linie ein System zur Kollisionsvermeidung in der Küstenschifffahrt. Es wird von Schifffahrtsbehörden zur Überwachung und Kontrolle von Küstengewässern eingesetzt.

Enthusiasten auf der ganzen Welt implementieren kostengünstige AIS-Überwachungsstationen und stellen Informationen der Küstenschifffahrt in das Internet. Andere begeisterte Anhänger schreiben Anwendungen, die die Informationen aus dem AIS mit anderen Informationen aus dem Internet kombinieren. Die Ergebnisse werden auf Websites gestellt und mit Twitter und SMS veröffentlicht.

In einer Anwendung werden beispielsweise Informationen der AIS-Stationen in der Nähe von Southampton mit Informationen zum Schiffseigner und geografischen Daten kombiniert. Die Anwendung speist die Echtzeitdaten zu Ankunfts- und Abfahrtszeiten von Fähren in Twitter ein. Regelmäßige Pendler, die die Fähren zwischen Southampton und der Isle of Wight benutzen, abonnieren den Newsfeed mittels Twitter oder SMS. Wenn der Feed eine Verspätung der Fähre anzeigt, können sich Pendler entsprechend später auf den Weg machen und die Fähre pünktlich erreichen, obwohl diese nicht zur planmäßigen Ankunftszeit anlegt.

Weitere Beispiele finden Sie im Abschnitt „Telemetrieanwendungsfälle” auf Seite 119.

Zugehörige Tasks

Installieren von MQ Telemetry
MQ Telemetry verwalten
MQ Telemetry unter Windows migrieren
MQ Telemetry unter Linux migrieren
Anwendungen für MQ Telemetry entwickeln
Fehlerbehebung bei Problemen mit MQ Telemetry

Zugehörige Verweise

MQ Telemetry Referenz

Windows Linux AIX

Einführung in MQ Telemetry

Privatpersonen, Unternehmen und Regierungen interessieren sich mehr und mehr für die Verwendung von MQ Telemetry, um schnell mit dem Lebens- und Arbeitsumfeld interagieren zu können. MQ Telemetry verbindet alle Arten von Geräten mit dem Internet und Unternehmen, wodurch sich die Kosten für die Erstellung von Anwendungen für Smart Devices verringern.

Was ist MQ Telemetry?

- Diese Funktion von IBM MQ erweitert die universelle Messaging-Zentralverbindung, die IBM MQ einer Vielzahl von fernen Sensoren, Aktuatoren und Telemetriegeräten bereitstellt. MQ Telemetry erweitert IBM MQ, sodass intelligente Unternehmensanwendungen, Services und Entscheidungsträger mit Netzen aus instrumentierten Geräten verbunden werden können.
- Die Kernkomponenten von MQ Telemetry sind:

  MQ Telemetry-Service (MQXR)
  Dieser Service läuft auf dem IBM MQ-Server; er verwendet zur Kommunikation mit Telemetriegeräten das IBM MQ Telemetry Transport-(MQTT-)Protokoll.

  MQTT-Anwendungen, die Sie schreiben
  Diese Anwendungen steuern die Informationen, die zwischen den Telemetriegeräten und dem IBM MQ-Warteschlangenmanager übertragen werden, sowie alle Aktionen, die als Reaktion auf diese Informationen durchgeführt werden. Bei der Erstellung dieser Anwendungen helfen Ihnen die MQTT-Clientbibliotheken.

Was kann die Software für mich tun?

- MQTT ist ein offenes Messaging-Protokoll, mit dem MQTT-Implementierungen für eine Vielzahl von Geräten erstellt werden können.
- MQTT-Clients können auch auf Geräten mit geringem Speicher und begrenzten Ressourcen ausgeführt werden.

- MQTT kann auf effiziente Weise in Netzen mit niedriger Bandbreite eingesetzt werden, bei denen das Versenden von Daten kostenintensiv ist oder die störanfällig sind.
- Die Nachrichtenübermittlung wird von der Anwendung sichergestellt und voneinander entkoppelt.
- Anwendungsprogrammierer müssen dazu keine Kenntnisse über das Programmieren von Kommunikationsroutinen besitzen.
- Nachrichten können mit anderen Messaging-Anwendungen ausgetauscht werden, zum Beispiel mit anderen Telemetrieanwendungen oder einer MQI-, JMS- oder Unternehmens-Messaging-Anwendung.

Wozu wende ich die Software an?

- Es werden Beispielscripts bereitgestellt, die mit einem Beispiel für eine Clientanwendung von IBM MQ Telemetry Transport der Version 3 arbeiten (`mqttv3app.jar`). Siehe IBM MQ Telemetry Transport-Beispielprogramme.
- Mit dem IBM MQ Explorer und seinen Tools können Sie die Telemetriefunktion von IBM MQ verwalten.
- Mit den Clientbibliotheken können Sie MQTT-Anwendungen erstellen, die eine Verbindung zu einem Warteschlangenmanager herstellen und Publish/Subscribe-Messaging verwenden.
- Stellen Sie Ihre Anwendung und die Clientbibliothek auf dem Gerät bereit, auf dem die Anwendung ausgeführt werden soll.

Wie funktioniert die Software?

- MQTT ist ein Publish/Subscribe-Protokoll. Eine MQTT-Clientanwendung kann Nachrichten an einen MQTT-Server veröffentlichen oder Nachrichten abonnieren, die von Anwendungen gesendet werden, die mit einem MQTT-Server verbunden sind.
- MQTT-Clientanwendungen verwenden Clientbibliotheken, die das MQTT-Messaging-Protokoll implementieren.
- Eine einfache MQTT-Clientanwendung funktioniert wie ein MQ-Standardclient, kann jedoch auf wesentlich mehr verschiedenen Plattformen und in deutlich mehr Netzen ausgeführt werden.
- Mit dem MQ Telemetry-Service (MQXR) wird aus einem IBM MQ-Warteschlangenmanager ein MQTT-Server.
- Wenn ein IBM MQ-Warteschlangenmanager als MQTT-Server agiert, können andere Anwendungen, die sich mit dem Warteschlangenmanager verbinden, die Nachrichten vom MQTT-Client abonnieren und empfangen.
- Der Warteschlangenmanager agiert als Router, der Nachrichten von veröffentlichenden Anwendungen an abonnierende Anwendungen verteilt.
- Die Nachrichten können zwischen verschiedenen Arten von Clientanwendungen verteilt werden. Dies ist beispielsweise zwischen Telemetrieclients und JMS-Clients möglich.

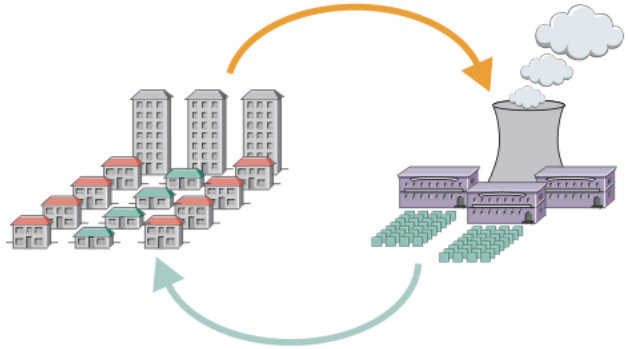
Anmerkung: MQ Telemetry ersetzt die SCADA-Knoten, die aus Version 7 von WebSphere Message Broker (jetzt "IBM Integration Bus") entfernt wurden, und wird unter Windows, Linux und AIX ausgeführt.

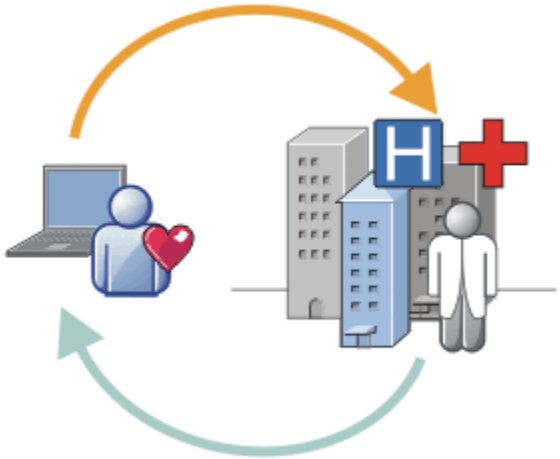
Windows Linux AIX Telemetrieanwendungsfälle

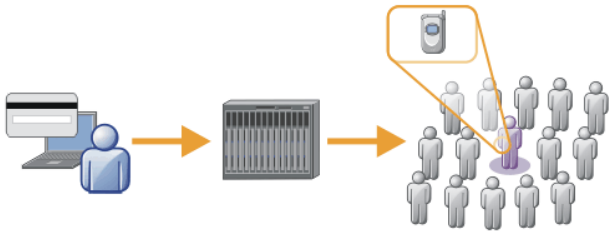
Telemetrie ist die automatisierte Erkennung und Messung von Daten ferner Geräte sowie die Steuerung dieser fernen Geräte. Der Schwerpunkt liegt in der Datenübertragung von fernen Geräten zu einem zentralen Kontrollpunkt. Die Telemetrie beinhaltet auch das Senden von Konfigurations- und Steuerinformationen an Einheiten.

MQ Telemetry verbindet kompakte Endgeräte über das MQTT protocol. Die Geräte werden unter Verwendung von IBM MQ mit anderen Anwendungen verbunden. MQ Telemetry bildet eine Brücke zwischen den Geräten und dem Internet und vereinfacht so die Erstellung von "smarten Lösungen". Smarte Lösungen ermöglichen Anwendungen, die Geräte überwachen und steuern, den Zugriff auf die Fülle von Informationen im Internet und in Unternehmensanwendungen.

Die folgenden Diagramme zeigen Beispiele zur typischen Verwendung von MQ Telemetry:

Telemetry: Smart Electricity	
	• MQTT-Nachricht mit den Energieverbrauchsdaten, die an den Service-Provider gesendet werden • MQ Telemetry sendet Steuerbefehle auf Grundlage der analysierten Energieverbrauchsdaten. • Weitere Informationen finden Sie im folgenden Anwendungsfall: „Anwendungsfall für Telemetry: Überwachung und Steuerung des Energieverbrauchs in Privathaushalten" auf Seite 122

Telemetry: Smart Health Services	
• MQ Telemetry sendet Gesundheitsdaten an Ihr Krankenhaus & Ihren Arzt. • MQTT-Rückmeldungen mit Warnungen oder Feedback können auf Basis der analysierten medizinischen Daten gesendet werden. • Weitere Informationen finden Sie im folgenden Anwendungsfall: „Anwendungsfall für Telemetry: Überwachung nicht stationärer Patienten" auf Seite 120	

Telemetry: One in a Crowd	
	• Eine einfache Kartentransaktion wird an den Server einer Bank gesendet. • MQ Telemetry ermittelt die betreffende Kontaktperson und warnt den Kunden, dass seine Karte verwendet wurde. • MQ Telemetry kann mithilfe einfachster Eingabeinformationen die betreffende Kontaktperson ausfindig machen.

Die nachfolgenden Beispiele, die auf realen Anwendungsfällen beruhen, veranschaulichen einige Einsatzmöglichkeiten von Telemetry und zeigen einige der häufig auftretenden Probleme auf, die von der Telemetrietechnologie gelöst werden müssen.

Windows Linux AIX Anwendungsfall für Telemetry: Überwachung nicht stationärer Patienten

Im Rahmen der Zusammenarbeit zwischen IBM und einem Anbieter des Gesundheitswesens bei einem Betreuungssystem für Herzpatienten kommuniziert ein implantierter Kardioverter-Defillibrator mit einem Krankenhaus. Die Daten des Patienten und implantierten Geräts werden mithilfe der RF-Telemetrie (RF = Radiofrequenz) an das MQTT-Gerät in der Wohnung des Patienten übertragen.

Für gewöhnlich erfolgt die Übertragung nachts an einen Sender, der sich neben dem Bett befindet. Der Sender überträgt die Daten auf sichere Weise über die Telefonanlage an das Krankenhaus, wo die Daten analysiert werden.

Durch dieses System verringert sich die Anzahl der Besuche des Patienten beim Arzt. Es erkennt, wenn Probleme beim Patienten oder Gerät auftreten, und benachrichtigt bei einem Notfall den diensthabenden Arzt.

Die Zusammenarbeit zwischen IBM und dem Anbieter des Gesundheitswesens weist Merkmale auf, die in verschiedenen Anwendungsfällen für Telemetry typisch sind:

Unsichtbarkeit

Für die Nutzung des Geräts muss der Anwender lediglich Strom und eine Telefonverbindung bereitstellen und sich zu bestimmten Tageszeiten in der Nähe des Geräts befinden. Ansonsten ist keinerlei Benutzereingriff erforderlich. Der Betrieb ist zuverlässig und die Bedienung einfach.

Damit der Patient das Gerät nicht einrichten muss, wird es vom Gerätelieferanten bereits vorkonfiguriert. Der Patient muss das Gerät lediglich einstecken. Dadurch, dass die Konfiguration nicht durch den Patienten erfolgt, wird der Gerätebetrieb vereinfacht und die Gefahr einer falschen Konfiguration verringert sich.

Der MQTT-Client ist in das Gerät integriert. Der Geräteentwickler integriert die MQTT-Clientimplementierung in das Gerät und der Entwickler bzw. Anbieter konfiguriert den MQTT-Client im Rahmen der Vorkonfiguration.

Der MQTT-Client wird als Java SE-JAR-Datei ausgeliefert. Diese JAR-Datei wird vom Entwickler in die zugehörige Java-Anwendung aufgenommen. In Umgebungen ohne Java (wie in diesem Fall) kann der Geräteentwickler einen Client unter Verwendung der veröffentlichten MQTT-Formate und des MQTT-Protokolls in einer anderen Sprache implementieren. Der Entwickler kann auch einen der Clients in der Programmiersprache C verwenden, die als gemeinsam genutzte Bibliotheken für Windows-, Linux- und ARM-Plattformen ausgeliefert werden.

Ungleichmäßige Konnektivität

Die Kommunikation zwischen dem Defibrillator und dem Krankenhaus ist mit den Herausforderungen einer ungleichmäßigen Konnektivität konfrontiert. Zwei verschiedene Netze werden zur Lösung der unterschiedlichen Problemstellung der Datenerfassung beim Patienten und der Datenübertragung an das Krankenhaus genutzt. Zwischen dem Patienten und dem MQTT-Gerät wird ein RF-Nahbereichsnetz mit geringer Leistung verwendet. Der Sender verbindet sich unter Verwendung einer TCP/IP-Verbindung im virtuellen privaten Netz über eine Telefonleitung mit geringer Bandbreite mit dem Krankenhaus.

Die direkte Verbindung jedes Geräts mit einem IP-Netz gestaltet sich häufig als aufwändig. Die Verwendung von zwei Netzen, die über einen Hub miteinander verbunden sind, ist eine bewährte Lösung für dieses Problem. Das MQTT-Gerät ist ein einfacher Hub, der Informationen des Patienten speichert und diese an das Krankenhaus weiterleitet.

Sicherheit

Der Arzt muss sich auf die Authentizität der Patientendaten verlassen können, während der Patient die Vertraulichkeit seiner Daten wünscht.

In einigen Anwendungsfällen reicht es aus, die Verbindung über ein virtuelles privates Netz (VPN) oder TLS zu verschlüsseln. In anderen Anwendungsfällen müssen die Daten auch nach ihrer Abspeicherung sicher aufbewahrt werden.

Gelegentlich lässt die Sicherheit des Telemetriegeräts zu wünschen übrig. Es könnte sich beispielsweise in einer Gemeinschaftsunterkunft befinden. Der Benutzer des Geräts muss authentifiziert werden, damit die Daten auf jeden Fall vom richtigen Patienten stammen. Das Gerät selbst kann über TLS beim Server authentifiziert werden. Die Authentifizierung in umgekehrter Richtung erfolgt ebenso.

Der Telemetriekanal zwischen dem Gerät und dem Warteschlangenmanager unterstützt JAAS (Java Authentication and Authorization Service) für die Benutzerauthentifizierung und TLS für die Verschlüs-

selung der Kommunikation sowie die Geräteauthentifizierung. Der Zugriff auf eine Veröffentlichung wird durch den Objektberechtigungsmanager in IBM MQ gesteuert.

Die Kennung, die für die Authentifizierung des Benutzers verwendet wird, kann einer anderen Kennung zugeordnet werden - beispielsweise einer allgemeinen Patientenidentität. Eine allgemeine Kennung vereinfacht die Konfiguration der Autorisierung für Veröffentlichungsthemen in IBM MQ.

Konnektivität

Der Verbindungsaufbau zwischen dem MQTT-Gerät und dem Krankenhaus erfolgt über eine Anwahl, wobei eine so niedrige Bandbreite wie 300 Baud genutzt werden kann.

Damit auch bei 300 Baud ein effektiver Betrieb möglich ist, fügt das MQTT protocol neben den TCP/IP-Headern lediglich eine geringe Anzahl an zusätzlichen Bytes zu einer Nachricht hinzu.

Das MQTT protocol bietet eine Nachrichtenübermittlung auf Basis des *fire-and-forget*-Prinzips mit einer Einzelübertragung, was die Latenzzeiten niedrig hält. Es kann aber auch mehrere Übertragungen verwenden, um die Zustellung des Typs *mindestens einmal* und *genau einmal* zu ermöglichen. Dies empfiehlt sich in Fällen, bei denen eine garantierte Übermittlung wichtiger als die Reaktionszeit ist. Zur Gewährleistung der Übermittlung werden die Nachrichten im Gerät gespeichert, bis sie erfolgreich zugestellt wurden. Wenn ein Gerät drahtlos verbunden ist, ist die garantierte Übermittlung besonders sinnvoll.

Skalierbarkeit

Telemetriegeräte werden für gewöhnlich in einer hohen Zahl implementiert - dies reicht von Zehntausenden bis hin zu Millionen.

Die Verbindung vieler Geräte mit einem System stellt hohe Anforderungen an die Lösung. Dies sind wirtschaftliche Anforderungen wie die Kosten für Geräte und die zugehörige Software und administrative Anforderungen wie die Verwaltung der Lizenzen, Geräte und Benutzer. Technische Anforderungen umfassen die Netz- und Serverauslastung.

Das Öffnen von Verbindungen belegt mehr Serverressourcen, als die Verbindungen offen zu halten. In einem Anwendungsfall wie dem vorliegenden, bei dem Telefonleitungen genutzt werden, bringen die Ausgaben für die Telefonverbindungen mit sich, dass die Verbindungen nur so lange offen bleiben, wie sie benötigt werden. Die Datenübertragungen erfolgen vorwiegend im Stapelbetrieb. Die Verbindungen können für die gesamte Nacht terminiert werden, um eine plötzliche Verbindungsspitze vor dem Schlafengehen zu vermeiden.

Auf dem Client wird die Skalierbarkeit von Clients durch den minimalen Aufwand für die Clientkonfiguration gefördert. Der MQTT-Client ist in das Gerät integriert. Es müssen keine Konfigurationsschritte oder Schritte zum Akzeptieren der MQTT-Clientlizenz zur Bereitstellung der Geräte für Patienten eingebunden werden.

Auf dem Server hat MQ Telemetry ein Anfangsziel von 50.000 offenen Verbindungen je Warteschlangenmanager.

Die Verbindungen werden mithilfe von IBM MQ Explorer verwaltet. Der IBM MQ Explorer filtert die anzuzeigenden Verbindungen auf eine Anzahl, deren Verwaltung bewältigt werden kann. Durch die geeignete Wahl eines Schemas, bei dem den Clients Kennungen zugeordnet werden, können Sie Verbindungen auf Basis der geografischen Lage oder alphabetisch nach Patientennamen filtern.

Windows Linux AIX Anwendungsfall für Telemetry: Überwachung und Steuerung des Energieverbrauchs in Privathaushalten

Intelligente Zähler erfassen mehr Einzeldaten zum Energieverbrauch als herkömmliche Messgeräte.

Intelligente Zähler sind häufig mit einem lokalen Telemetrienetz verbunden, um einzelne Geräte in einem Privathaushalt zu überwachen und zu steuern. Einige davon sind auch remote zur Fernüberwachung und -steuerung verbunden.

Die Fernverbindung kann von einer Einzelperson, einem Elektrizitätswerk oder einer zentralen Kontrollstelle eingerichtet werden. Die ferne Kontrollstelle kann den Leistungsverbrauch ablesen und Nutzungs-

daten bereitstellen. Sie kann Daten bereitstellen, die die Nutzung beeinflussen, beispielsweise fortlaufende Preis- und Wetterdaten. Durch eine Lastbegrenzung kann die gesamte Stromerzeugungseffizienz gesteigert werden.

Die Implementierung intelligenter Zähler wird immer beliebter. Die britische Regierung steht beispielsweise in Verhandlungen über die Implementierung intelligenter Zähler in allen britischen Haushalten bis zum Jahr 2020.

Die Anwendungsfälle für die Messung in Privathaushalten weisen eine Reihe allgemeiner Merkmale auf:

Unsichtbarkeit

Das Messgerät erfordert keinen Benutzereingriff, es sei denn, der Benutzer möchte aktiv in die Energieersparnis eingreifen, die durch die Verwendung des Messgeräts ermöglicht wird. Es darf die Zuverlässigkeit der Energieversorgung einzelner Systeme nicht beeinträchtigen.

Ein MQTT-Client kann in die Software eingebettet werden, die mit dem Messgerät implementiert wird, und erfordert keine separate Installation oder Konfiguration.

Ungleichmäßige Konnektivität

Die Datenübertragung zwischen Geräten und dem intelligenten Zähler erfordert andere Konnektivitätsstandards als die Datenübertragung zwischen dem Zähler und dem Fernverbindungspunkt.

Die Verbindung vom intelligenten Zähler zu den Geräten muss hoch verfügbar sein und den Netzstandards für ein Heimnetz entsprechen.

Das ferne Netz nutzt wahrscheinlich verschiedene physische Verbindungen. Einige davon, beispielsweise Mobiltelefone, weisen hohe Übertragungskosten auf und sind möglicherweise nicht unterbrechungsfrei. Die MQTT v3-Spezifikation wurde für Fernverbindungen und Verbindungen zwischen lokalen Adaptern und dem intelligenten Zähler konzipiert.

Für die Verbindung zwischen den Netzsteckdosen und Geräten und dem Messgerät wird ein Heimnetz wie beispielsweise Zigbee verwendet. MQTT für Sensorennetze (MQTT-S) wurde für die Arbeit mit Zigbee und anderen Netzprotokollen mit geringer Bandbreite entwickelt. MQ Telemetry unterstützt MQTT-S nicht direkt. Es wird ein Gateway für die Verbindung von MQTT-S mit MQTT v3 benötigt.

Wie bei der Überwachung nicht stationärer Patienten erfordern auch die Lösungen für die Überwachung und Steuerung des Energieverbrauchs in Privathaushalten mehrere Netze, die unter Verwendung des intelligenten Zählers als Hub miteinander verbunden sind.

Sicherheit

Im Zusammenhang mit intelligenten Zählern bestehen einige Sicherheitsprobleme. Dazu gehören der fälschungssichere Herkunftsnachweis von Transaktionen, die Autorisierung eingeleiteter Steuerungsvorgänge und der Datenschutz der Energieverbrauchswerte.

Zur Gewährleistung des Datenschutzes können die Daten, die von MQTT zwischen dem Messgerät und der fernen Kontrollstelle übertragen werden, mithilfe von Transport Layer Security (TLS) verschlüsselt werden. Zur Sicherstellung, dass Steuerungsvorgänge tatsächlich autorisiert sind, kann die MQTT-Verbindung zwischen dem Messgerät und der fernen Kontrollstelle gegenseitig unter Verwendung von TLS authentifiziert werden.

Konnektivität

Die physische Spezifik des fernen Netzes kann beträchtlich variieren. Sie kann beispielsweise eine bestehende Breitbandverbindung oder ein Mobilnetz mit hohen Verbindungskosten und einer nicht unterbrechungsfreien Verfügbarkeit nutzen. Bei teuren, nicht unterbrechungsfreien Verbindungen ist MQTT ein effizientes und zuverlässiges Protokoll; siehe Abschnitt „Anwendungsfall für Telemetry: Überwachung nicht stationärer Patienten” auf Seite 120.

Skalierbarkeit

Energieversorgungsunternehmen oder zentrale Kontrollstellen planen die Implementierung von letztendlich Zig-Millionen intelligenter Zähler. Anfänglich bewegt sich die Anzahl der Messgeräte je Implementierung zwischen Zehn- und Hunderttausenden. Diese Zahl ist vergleichbar mit dem anfänglichen MQTT-Ziel von 50.000 offenen Clientverbindungen je Warteschlangenmanager.

Ein kritischer Aspekt in der Architektur bei der Überwachung und Steuerung des Energieverbrauchs in Privathaushalten ist die Verwendung des intelligenten Zählers als Netzkonzentrator. Jeder Geräteadapter ist ein separater Sensor. Durch deren Verbindung mit einem lokalen Hub unter Verwendung von MQTT kann der Hub die Datenflüsse in einer einzelnen TCP/IP-Sitzung mit der zentralen Kontrollstelle konzentrieren und außerdem die Nachrichten für einen kurzen Zeitraum speichern, um Sitzungsausfallzeiten entgegenzuwirken.

Fernverbindungen müssen in den Anwendungsfällen für die Energieverbrauchssteuerung in Privathaushalten aus zwei Gründen offen gehalten werden. Der erste Grund ist, dass das Öffnen von Verbindungen im Vergleich mit dem Senden von Anforderungen viel Zeit in Anspruch nimmt. Der zeitliche Aufwand für das Öffnen vieler Verbindungen zum Senden von Anforderungen zur "Lastbegrenzung" in einem kurzen Zeitraum ist zu hoch. Der zweite Grund ist, dass die Verbindung für den Empfang von Anforderungen zur Lastbegrenzung vom Energieversorgungsunternehmen zuerst vom Client geöffnet werden muss. Bei MQTT werden Verbindungen immer durch den Client aufgebaut und für den Empfang von Anforderungen zur Lastbegrenzung vom Energieversorgungsunternehmen muss die Verbindung offen gehalten werden.

Wenn die Geschwindigkeit des Öffnens von Verbindungen kritisch ist oder der Server zeitkritische Anforderungen initialisiert, unterhält die Lösung für gewöhnlich viele offene Verbindungen.

Windows Linux AIX Anwendungsfälle für Telemetry: Radiofrequenzidentifikation (RFID)

RFID ist die Verwendung eines eingebetteten RFID-Tags zur Identifizierung und drahtlosen Überwachung eines Objekts. RFID-Tags können auf eine Entfernung von bis zu mehreren Metern gelesen werden und können sich außerhalb der Sichtweite der RFID-Leseeinheit befinden. Passive Tags werden von einer RFID-Leseeinheit aktiviert. Aktive Tags übertragen Daten ohne externe Aktivierung. Aktive Tags müssen über einen Versorgungsstromkreis verfügen. Passive Tags können einen Versorgungsstromkreis nutzen, um ihre Reichweite zu erhöhen.

RFID wird in vielen Anwendungen verwendet und die Anwendungsfälle sind vielfältig. RFID-Anwendungsfälle und die Anwendungsfälle für die Überwachung nicht stationärer Patienten sowie die Überwachung und Steuerung des Energieverbrauchs in Privathaushalten weisen einige Gemeinsamkeiten, aber auch Unterschiede auf.

Unsichtbarkeit

In vielen Anwendungsfällen werden die RFID-Leseeinheiten in hoher Zahl implementiert und müssen ohne Benutzereingriff funktionieren. Die Leseeinheit enthält einen integrierten MQTT-Client für die Kommunikation mit einem zentralen Kontrollpunkt.

In einem Vertriebslager verwendet die Leseeinheit beispielsweise einen Bewegungssensor zur Ortung einer Palette. Sie aktiviert die RFID-Tags der Artikel auf der Palette und sendet Daten und Anforderungen an zentrale Anwendungen. Mithilfe der Daten wird die Position des Warenbestands aktualisiert. Die Anforderungen steuern, was mit der Palette im nächsten Schritt passieren soll; sie könnte beispielsweise an eine bestimmte Position verschoben werden. Fluglinien und Gepäcksysteme in Flughäfen nutzen RFID beispielsweise auf diese Weise.

In einigen RFID-Anwendungsfällen verfügt die Leseeinheit über eine standardmäßige Datenverarbeitungsumgebung wie Java Platform, Micro Edition (Java ME). In diesen Fällen kann der MQTT-Client nach der Herstellung in einem eigenen Konfigurationsschritt implementiert werden.

Ungleichmäßige Konnektivität

Die RFID-Leseeinheiten können getrennt vom lokalen Steuergerät sein, das einen MQTT-Client enthält, oder jede Leseeinheit kann über einen integrierten MQTT-Client verfügen. Für gewöhnlich wird die gewählte Topologie von geografischen oder kommunikationstechnischen Faktoren bestimmt.

Sicherheit

Bei der Anlage von RFID-Tags stellen der Datenschutz und die Authentizität ein Sicherheitsproblem dar. RFID-Tags sind unauffällig und können heimlich überwacht, verfälscht oder manipuliert werden.

Die Lösung der RFID-Sicherheitsprobleme schafft Raum für die Implementierung neuer RFID-Lösungen. Auch wenn das Sicherheitsrisiko im RFID-Tag und in der lokalen Leseeinheit liegt, kann die Verarbeitung zentraler Daten Möglichkeiten für die Reaktion auf verschiedene Sicherheitsbedrohungen bieten. Eine Tag-Manipulation kann beispielsweise erkannt werden, indem der Warenbestand dynamisch mit Lieferungen und dem Versand korreliert wird.

Konnektivität

In RFID-Anwendungen waren für gewöhnlich sowohl stapelorientierte Store-and-forward-Verfahren für erfasste Informationen, die aus den RFID-Leseeinheiten stammen, als auch sofortige Abfragen involviert. Im Anwendungsfall des Vertriebslagers ist die RFID-Leseeinheit ständig verbunden. Wenn ein Tag gelesen wird, wird er gemeinsam mit Informationen zur Leseeinheit veröffentlicht. Die Warehousing-Anwendung veröffentlicht die Antwort wiederum bei der Leseeinheit.

In der Warehousing-Anwendung ist das Netz normalerweise zuverlässig und die sofortigen Anforderungen verwenden unter Umständen zur Verkürzung der Latenzzeit Nachrichten des Typs *fire and forget*. Die Daten des stapelorientierten Store-and-forward-Verfahrens nutzen möglicherweise eine Nachrichtenübertragung nach dem Prinzip *genau einmal*, um den Verwaltungsaufwand zu minimieren, der mit einem Datenverlust verbunden ist.

Skalierbarkeit

Wenn die RFID-Anwendung sofortige Antworten benötigt, also innerhalb von ein oder zwei Sekunden, müssen die RFID-Leseeinheiten verbunden bleiben.

Windows Linux AIX Anwendungsfälle für Telemetry: Erfassung von Umweltdaten

Bei der Erfassung von Umweltdaten werden mithilfe der Telemetrie Informationen zum Wasserstand und zur Wasserqualität von Flüssen, zu Luftschadstoffen und zu sonstigen Umweltdaten gesammelt.

Die Sensoren befinden sich häufig an fernen Orten ohne Zugriff auf eine festnetzgebundene Kommunikation. Die drahtlose Bandbreite ist kostenintensiv und die Zuverlässigkeit kann gering sein. Für gewöhnlich sind mehrere Umgebungssensoren in einem kleinen geografischen Bereich mit einem lokalen Überwachungsgerät an einem sicheren Ort verbunden. Die lokalen Verbindungen können festnetzgebunden oder drahtlos sein.

Unsichtbarkeit

Die Sensoreinheiten sind in der Regel weniger zugänglich, haben eine niedrigere Leistung und werden in größerer Zahl implementiert als das zentrale Überwachungsgerät. Die Sensoren selbst sind gelegentlich "unintelligent" und das lokale Überwachungsgerät enthält Adapter für die Umwandlung und Speicherung der Sensordaten. Das Überwachungsgerät umfasst häufig einen Mehrzweckcomputer, der Java Platform, Standard Edition (Java SE) oder Java Platform, Micro Edition (Java ME) unterstützt. Bei der Konfiguration des MQTT-Clients spielt die Unsichtbarkeit für gewöhnlich eine untergeordnete Rolle.

Ungleichmäßige Konnektivität

Das Leistungsspektrum von Sensoren sowie die Kosten und Bandbreite der Fernverbindung führen für gewöhnlich dazu, dass ein lokaler Überwachungshub mit einem zentralen Server verbunden wird.

Sicherheit

Die Sicherheit spielt eine untergeordnete Rolle, es sei denn, die Lösung wird in einem militärischen Umfeld oder Verteidigungsszenario verwendet.

Konnektivität

Viele Einsatzgebiete erfordern keine kontinuierliche Überwachung oder sofortige Datenverfügbarkeit. Ausnahmedaten wie beispielsweise eine Warnung vor hohem Wasserstand müssen hingegen sofort weitergeleitet werden. Die Sensordaten werden beim lokalen Überwachungsprogramm gesammelt, um Verbindungs- und Kommunikationskosten zu reduzieren, und dann in einem festgelegten Zeitrahmen über Verbindungen übertragen. Ausnahmedaten werden direkt nach ihrer Ermittlung beim Überwachungsprogramm weitergeleitet.

Skalierbarkeit

Sensoren konzentrieren sich um lokale Hubs und die Sensordaten werden in Paketen gesammelt, die nach einem festgelegten Zeitplan übertragen werden. Diese Faktoren reduzieren die Arbeitslast auf dem zentralen Server, die bei der Verwendung von direkt verbundenen Sensoren bestehen würde.

Windows Linux AIX Anwendungsfälle für Telemetry: Mobile Anwendungen

Mobile Anwendungen sind Anwendungen, die auf mobilen Endgeräten ausgeführt werden. Bei den Geräten handelt es sich entweder um generische Anwendungsplattformen oder kundenspezifische Geräte.

Allgemeine Plattformen umfassen mobile Endgeräte wie Telefone und elektronische Organizer sowie mobile Geräte wie Notebook-Computer. Kundenspezifische Geräte verwenden spezielle Hardware, die auf bestimmte Anwendungen zugeschnitten ist. Ein Gerät für die Erfassung einer Paketzustellung "mit Unterschrift" ist ein Beispiel für ein solches kundenspezifisches mobiles Gerät. Anwendungen auf kundenspezifischen mobilen Geräten basieren häufig auf einer generischen Softwareplattform.

Unsichtbarkeit

Die Implementierung kundenspezifischer mobiler Anwendungen wird verwaltet und kann die Konfiguration der MQTT-Clientanwendung beinhalten. Bei der Konfiguration des MQTT-Clients spielt die Unsichtbarkeit für gewöhnlich eine untergeordnete Rolle.

Ungleichmäßige Konnektivität

Im Gegensatz zu der lokalen Hub-Topologie der vorherigen Anwendungsfälle verbinden sich mobile Clients über Fernzugriff. Die Clientanwendungsschicht verbindet sich direkt mit einer Anwendung am zentralen Hub.

Sicherheit

Da nur wenig physische Sicherheit besteht, müssen das mobile Gerät und der mobile Benutzer authentifiziert werden. Zur Überprüfung der Identität des Geräts wird TLS verwendet, während die Benutzerauthentifizierung über JAAS erfolgt.

Konnektivität

Wenn die mobile Anwendung von einer drahtlosen Versorgung abhängt, muss sie in der Lage sein, offline zu agieren und auf effiziente Weise mit einer unterbrochenen Verbindung umgehen zu können. In dieser Umgebung besteht das Ziel darin, verbunden zu bleiben, die Anwendung muss jedoch Nachrichten im Store-and-forward-Verfahren verarbeiten können. Bei den Nachrichten handelt es sich häufig um Bestellungen oder Zustellnachweise, die von wichtigem geschäftlichen Nutzen sind. Sie müssen zuverlässig im Store-and-forward-Verfahren gespeichert und weitergeleitet werden.

Skalierbarkeit

Die Skalierbarkeit spielt in diesem Fall eine untergeordnete Rolle. In angepassten Anwendungsfällen mit mobilen Anwendungen ist es unwahrscheinlich, dass die Anzahl der Anwendungsclients Tausende oder Zehntausende von Clients überschreitet.

Windows Linux AIX Telemetriegeräte mit einem Warteschlangenmanager verbinden

Telemetriegeräte verbinden sich unter Verwendung eines MQTT v3-Clients mit einem Warteschlangenmanager. Der MQTT v3-Client nutzt TCP/IP, um eine Verbindung zu einem TCP/IP-Empfangsprogramm, dem sogenannten Telemetrieservice (MQXR), herzustellen.

Wenn Sie ein Telemetriegerät mit einem Warteschlangenmanager verbinden, leitet der MQTT-Client mit der Methode `MqttClient.connect` eine TCP/IP-Verbindung ein. Wie IBM MQ-Clients muss auch ein MQTT-Client mit dem Warteschlangenmanager verbunden sein, damit Nachrichten gesendet und empfangen werden können. Die Verbindung wird auf dem Server unter Verwendung eines TCP/IP-Empfangsprogramms hergestellt, das im Rahmen von MQ Telemetry installiert und als Telemetrieservice (MQXR)

bezeichnet wird. Auf jedem Warteschlangenmanager wird höchstens ein Telemetrieservice (MQXR) ausgeführt.

Der Telemetrieservice (MQXR) nutzt zur Zuordnung der Verbindung zu einem Telemetriekanal die Adresse des fernen Sockets, die durch jeden Client in der Methode `MqttClient.connect` festgelegt wird. Eine Socketadresse ist eine Kombination aus TCP/IP-Hostnamen und Portnummer. Mehrere Clients, die dieselbe Adresse eines fernen Sockets verwenden, werden vom Telemetrieservice (MQXR) mit demselben Telemetriekanal verbunden.

Falls sich auf einem Server mehrere Warteschlangenmanager befinden, teilen Sie die Telemetriekanäle zwischen den Warteschlangenmanagern auf. Ordnen Sie die Adressen ferner Sockets den verschiedenen Warteschlangenmanagern zu. Definieren Sie jeden Telemetriekanal mit einer eindeutigen Adresse eines fernen Sockets. Eine Socketadresse kann nicht von zwei Telemetriekanälen verwendet werden.

Wenn dieselbe Adresse eines fernen Sockets für Telemetriekanäle auf mehreren Warteschlangenmanagern konfiguriert ist, wird der erste Telemetriekanal für die Verbindung genutzt. Nachfolgende Kanäle, die sich unter derselben Adresse verbinden, schlagen fehl.

Falls sich auf dem Server mehrere Netzadapter befinden, teilen Sie die Adressen ferner Sockets zwischen den Telemetriekanälen auf. Die Zuordnung der Socketadressen erfolgt völlig beliebig, sofern eine bestimmte Socketadresse in nur einem Telemetriekanal konfiguriert ist.

Konfigurieren Sie IBM MQ für die Verbindung von MQTT-Clients unter Verwendung der Assistenten, die in der MQ Telemetry-Ergänzung für IBM MQ Explorer bereitgestellt werden. Alternativ können Sie die Telemetrie auch manuell konfigurieren. Folgen Sie dazu den Anweisungen in den Abschnitten Warteschlangenmanager für Telemetrie unter Linux und AIX konfigurieren und Warteschlangenmanager für Telemetrie unter Windows konfigurieren.

Zugehörige Verweise

MQXR-Eigenschaften

Windows Linux AIX Telemetry-Verbindungsprotokolle

MQ Telemetry unterstützt TCP/IP IPv4 und IPv6 sowie TLS.

Windows Linux AIX Telemetrieservice (MQXR)

Beim Telemetrieservice (MQXR) handelt es sich um ein TCP/IP-Empfangsprogramm, das als IBM MQ-Service verwaltet wird. Erstellen Sie den Service mithilfe eines IBM MQ Explorer-Assistenten oder mit einem **runmqsc**-Befehl.

Der MQ Telemetry -Service (MQXR) heißt SYSTEM.MQXR.SERVICE

Der Assistent für die Telemetry-Beispielkonfiguration, der mit der MQ Telemetry-Funktion für IBM MQ Explorer bereitgestellt wird, erstellt den Telemetrieservice und einen Beispieltelemetriekanal; siehe Installation von MQ Telemetry mithilfe von IBM MQ Explorerüberprüfen.

Erstellen Sie die Beispielkonfiguration über die Befehlszeile; siehe Installation von MQ Telemetry über die Befehlszeile überprüfen.

Der Telemetrieservice (MQXR) wird gemeinsam mit dem Warteschlangenmanager automatisch gestartet und gestoppt. Steuern Sie den Service über den Ordner 'Services' in IBM MQ Explorer. Zum Anzeigen des Service müssen Sie auf das Symbol klicken, um das IBM MQ Explorer Herausfiltern von SYSTEM-Objekten aus der Anzeige zu stoppen.

Ein Beispiel für die manuelle Erstellung des Service finden Sie unter

- Linux AIX `SYSTEM.MQXR.SERVICE` unter Linuxerstellen.
- Windows `SYSTEM.MQXR.SERVICE` unter Windowserstellen.

In diesen Abschnitten wird auch der Standardschlüssel angegeben, damit Kennphrasen für MQTT-TLS-Kanäle verschlüsselt werden müssen. Weitere Informationen finden Sie unter Verschlüsselung von Passphrasen für MQTT TLS-Kanäle.

Windows Linux AIX Telemetriekanäle

Erstellen Sie Telemetriekanäle, um Verbindungen mit unterschiedlichen Eigenschaften wie einer JAAS- oder TLS-Authentifizierung (JAAS = Java Authentication and Authorization Service) zu erstellen oder Clientgruppen zu verwalten.

Erstellen Sie Telemetriekanäle mit dem Assistenten **New Telemetry Channel** , der in der Funktion MQ Telemetry für IBM MQ Explorerbereitgestellt wird. Konfigurieren Sie einen Kanal unter Verwendung des Assistenten für die Annahme von Verbindungen aus MQTT-Clients an einem bestimmten TCP/IP-Port. Seit IBM WebSphere MQ 7.1 kann MQ Telemetry mit dem Befehlszeilenprogramm **runmqsc** konfiguriert werden.

Erstellen Sie mehrere Telemetriekanäle an verschiedenen Ports, um die Verwaltung sehr vieler Clienttransaktionen durch die Aufteilung von Clients in Gruppen zu vereinfachen. Jeder Telemetriekanal hat einen anderen Namen.

Sie können Telemetriekanäle mit unterschiedlichen Sicherheitsattributen konfigurieren und so verschiedene Verbindungsarten erstellen. Erstellen Sie mehrere Kanäle, damit Clientverbindungen an verschiedenen TCP/IP-Adressen akzeptiert werden können. Verschlüsseln Sie Nachrichten mithilfe von TLS und authentifizieren Sie den Telemetriekanal und Client; siehe TLS-Konfiguration von MQTT-Clients und Telemetriekanälen. Geben Sie die Benutzer-ID zur einfacheren Berechtigung des Zugriffs auf IBM MQ-Objekte. Geben Sie eine JAAS-Konfiguration an, um den MQTT-Benutzer mit JAAS zu authentifizieren; siehe MQTT-Clientidentifikation, Autorisierung und Authentifizierung.

Windows Linux AIX IBM MQ Telemetry Transport-Protokoll

Das IBM MQ Telemetry Transport (MQTT) v3 wurde für den Nachrichtenaustausch zwischen kompakten Endgeräten mit geringer Bandbreite oder kostenintensiven Verbindungen und für eine zuverlässige Nachrichtenübertragung entwickelt. Es verwendet TCP/IP.

Das MQTT protocol ist veröffentlicht (siehe IBM MQ Telemetry Transport-Format und -Protokoll). Version 3 des Protokolls nutzt Publish/Subscribe und unterstützt drei Servicequalitäten: *Fire and Forget*, *mindestens einmal* und *genau einmal*.

Die geringe Größe der Protokollheader und der Bytefeldgruppe der Nachrichtennutzdaten sorgt für Nachrichten mit geringem Umfang. Die Header umfassen einen festgelegten Header mit 2 Bytes und bis zu 12 Bytes zusätzliche variable Header. Das Protokoll verwendet variable Header mit 12 Bytes für Subskriptionen und Verbindungen und variable Header mit nur 2 Bytes für die meisten Veröffentlichungen.

Mit drei Servicequalitäten haben Sie Spielraum für einen guten Ausgleich zwischen niedrigen Latenzzeiten und Zuverlässigkeit; siehe Von einem MQTT-Client bereitgestellte Servicequalitäten. *Fire and Forget* verwendet keinen permanenten Gerätespeicher und nur eine Übertragung für das Senden oder Empfangen einer Veröffentlichung. *Mindestens einmal* und *genau einmal* erfordern einen permanenten Speicher im Gerät, um den Protokollstatus aufrechtzuerhalten und eine Nachricht bis zu ihrer Bestätigung zu speichern.

Windows Linux AIX MQTT-Clients

Eine MQTT-Client-App ist für die Erfassung von Daten aus dem Telemetriegerät, die Verbindung zum Server und die Veröffentlichung der Informationen auf dem Server verantwortlich. Sie kann außerdem Themen abonnieren, Veröffentlichungen empfangen und das Telemetriegerät steuern.

Im Gegensatz zu IBM MQ-Clientanwendungen handelt es sich bei MQTT-Client-Apps um keine IBM MQ-Anwendungen. Sie geben keinen Warteschlangenmanager für die Verbindungsherstellung an. Sie sind nicht auf die Verwendung bestimmter IBM MQ-Programmierschnittstellen begrenzt. Stattdessen implementieren MQTT-Clients das MQTT 3-Protokoll. Sie können Ihre eigene Clientbibliothek als Schnittstelle zum MQTT protocol in der Programmiersprache und auf der Plattform Ihrer Wahl schreiben. Weitere Informationen finden Sie im Abschnitt IBM MQ Telemetry Transport-Format und -Protokoll.

Die Entwicklung von MQTT-Client-Apps erleichtern Sie sich durch die C-, Java- und JavaScript-Clientbibliotheken, in denen das MQTT protocol für eine Reihe von Plattformen enthalten ist. Wenn Sie diese Bibliotheken in Ihre MQTT-Apps einbinden, können Sie auch einen MQTT-Client mit vollem Funktionsum-

fang schreiben, der nicht länger als 15 Codezeilen ist. MQTT-Clientbibliotheken sind bei Eclipse Paho und MQTT.org kostenlos erhältlich. Weitere Informationen finden Sie im Abschnitt IBM MQ Telemetry Transport-Beispielprogramme.

Die MQTT-Client-App ist immer für den Aufbau einer Verbindung mit einem Telemetriekanal verantwortlich. Sobald die Verbindung hergestellt ist, kann entweder die MQTT-Client-App oder eine IBM MQ-Anwendung den Nachrichtenaustausch starten.

MQTT-Client-Apps und IBM MQ-Anwendungen veröffentlichen und abonnieren dieselbe Themengruppe. Eine IBM MQ-Anwendung kann auch eine Nachricht direkt an eine MQTT-Client-App senden, ohne dass die Client-App eine Subskription erstellen muss. Weitere Informationen finden Sie im Abschnitt Verteilte Steuerung von Warteschlangen zum Senden von Nachrichten an MQTT-Clients verwenden.

MQTT-Client-Apps werden unter Verwendung eines Telemetriekanals mit IBM MQ verbunden. Der Telemetriekanal agiert als Brücke zwischen den unterschiedlichen Nachrichtentypen, die von MQTT und IBM MQ verwendet werden. Er erstellt für die MQTT-Client-App Veröffentlichungen und Subskriptionen im Warteschlangenmanager. Der Telemetriekanal sendet Veröffentlichungen, die den Subskriptionen einer MQTT-Client-App entsprechen, vom Warteschlangenmanager an die MQTT-Client-App.

Windows Linux AIX Nachricht an einen MQTT-Client senden

IBM MQ-Anwendungen können Nachrichten an MQTT v3-Clients senden, indem Sie Informationen in Subskriptionen veröffentlichen, die von Clients erstellt wurden, oder indem Sie die Nachrichten direkt senden. MQTT-Clients können sich gegenseitig Nachrichten senden, indem sie Informationen in Themen veröffentlichen, die von anderen Clients abonniert wurden.

Ein MQTT-Client abonniert eine Veröffentlichung, die er von IBM MQ erhält

Führen Sie die Task „Nachricht über IBM MQ Explorer im MQTT -Clientdienstprogramm veröffentlichen” auf Seite 131 aus, um eine Veröffentlichung aus IBM MQ an einen MQTT-Client zu senden.

Üblicherweise erhält ein MQTT v3-Client Nachrichten, indem er eine Subskription für ein Thema oder eine Themengruppe erstellt. Im Beispielcode-Snippet Abbildung 44 auf Seite 130subskribiert der MQTT -Client die Topiczeichenfolge `"MQTT Examples"`. Eine IBM MQ C-Anwendung, Abbildung 45 auf Seite 130, veröffentlicht unter Verwendung der Themenzeichenfolge `"MQTT Examples"`im Thema. Im Codeausschnitt Abbildung 46 auf Seite 131 empfängt der MQTT-Client die Veröffentlichung in der Callback-Methode `messageArrived`.

Weitere Informationen zum Konfigurieren von IBM MQ für das Senden von Veröffentlichungen als Antwort auf Subskriptionen von MQTT -Clients finden Sie unter Nachricht als Antwort auf eine MQTT -Clientsubskription veröffentlichen.

Eine IBM MQ-Anwendung sendet eine Nachricht direkt an einen MQTT-Client

Führen Sie die Task „Nachricht mithilfe von IBM MQ Explorer an einen MQTT -Client senden” auf Seite 136 aus, um eine Nachricht aus IBM MQ direkt an einen MQTT-Client zu senden.

Eine Nachricht, die auf diese Weise an einen MQTT-Client gesendet wird, wird als "nicht erwartete Nachricht" bezeichnet. MQTT v3-Clients empfangen nicht erwartete Nachrichten als Veröffentlichungen mit einem festgelegten Themennamen. Der Telemetrieservice (MQXR) setzt den Themennamen auf den Namen der fernen Warteschlange.

Weitere Informationen zur Konfiguration von IBM MQ für das direkte Senden von Nachrichten an MQTT -Clients finden Sie unter Nachricht direkt an einen Client senden.

Ein MQTT-Client veröffentlicht eine Nachricht

Ein MQTT v3-Client kann eine Nachricht veröffentlichen, die von einem anderen MQTT v3-Client empfangen wird, er kann jedoch keine nicht erwartete Nachricht senden. Der Codeausschnitt Abbildung 47 auf Seite 131 zeigt, wie ein MQTT v3-Client, der in der Programmiersprache Java geschrieben wurde, eine Nachricht veröffentlicht.

Das typische Beispiel für das Senden einer Nachricht an einen bestimmten MQTT v3-Client sieht so aus, dass jeder Client eine Subskription für seinen eigenen `ClientIdentifier`-Eintrag erstellt. Führen Sie die Task „Nachricht an einen bestimmten MQTT v3-Client veröffentlichen" auf Seite 137 aus, um eine Nachricht von einem MQTT-Client für einen anderen MQTT-Client unter Verwendung von `ClientIdentifier` als Topic-Zeichenfolge zu veröffentlichen.

Beispielcodeausschnitte

Das Code-Snippet in Abbildung 44 auf Seite 130 zeigt, wie ein in Java geschriebener MQTT -Client eine Subskription erstellt. Er benötigt außerdem eine Callback-Methode (`messageArrived`), um Veröffentlichungen für die Subskription zu empfangen.

```
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);
```

Abbildung 44. MQTT v3-Client-Subskribent

Der Codeausschnitt in Abbildung 45 auf Seite 130 zeigt, wie eine IBM MQ-Anwendung, die in der Programmiersprache C geschrieben wurde, eine Veröffentlichung sendet. Der Codeausschnitt wird aus der Task Veröffentlichungskomponente für ein variables Thema erstellen extrahiert.

```
/* 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);
```

Abbildung 45. IBM MQ-Publisher

Wenn die Veröffentlichung ankommt, ruft der MQTT -Client die Methode `messageArrived` der Klasse `MqttCallback` des MQTT Anwendungsclients auf.

```
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
}
```

Abbildung 46. `messageArrived`-Methode

Abbildung 47 auf Seite 131 zeigt einen MQTT v3-Client, der eine Nachricht in der Subskription veröffentlicht, die in Abbildung 44 auf Seite 130 erstellt wurde.

```
      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);
```

Abbildung 47. MQTT v3-Client-Publisher

Windows Linux AIX Nachricht über IBM MQ Explorer im MQTT -Clientdienstprogramm veröffentlichen

Führen Sie die Schritte in dieser Task aus, um eine Nachricht unter Verwendung von IBM MQ Explorer zu veröffentlichen und diese mit dem MQTT-Clientdienstprogramm zu abonnieren. In einer weiteren Task wird veranschaulicht, wie die Standardübertragungswarteschlange nicht auf `SYSTEM.MQTT.TRANSMIT.QUEUE` gesetzt, sondern ein Warteschlangenmanager-Aliasname konfiguriert wird.

Vorbereitende Schritte

Bei der Task wird vorausgesetzt, dass Sie mit IBM MQ und dem IBM MQ Explorer vertraut sind, und dass IBM MQ und die Komponente MQ Telemetry installiert sind.

Der Benutzer, der die Warteschlangenmanagerressourcen für diese Task erstellt, muss über ausreichende Berechtigungen für diesen Vorgang verfügen. Zu Demonstrationszwecken wird davon ausgegangen, dass die IBM MQ Explorer-Benutzer-ID der Gruppe `mqm` angehört.

Informationen zu diesem Vorgang

In dieser Task erstellen Sie ein Thema in IBM MQ und abonnieren das Thema unter Verwendung des MQTT-Clientdienstprogramms. Wenn Sie unter Verwendung von IBM MQ Explorer Informationen in dem Thema veröffentlichen, empfängt der MQTT-Client die Veröffentlichung.

Vorgehensweise

Führen Sie eine der folgenden Tasks aus:

- Sie haben MQ Telemetry zwar installiert, aber noch nicht gestartet. Führen Sie diese Task aus: „Task ohne definierten Telemetrieservice (MQXR) starten” auf Seite 132.

- Sie haben IBM MQ Telemetry zuvor bereits ausgeführt, möchten für die Demonstration aber einen neuen Warteschlangenmanager verwenden. Führen Sie diese Task aus: „Task ohne definierten Telemetrieservice (MQXR) starten" auf Seite 132.
- Sie möchten die Task mit einem vorhandenen Warteschlangenmanager ausführen, auf dem keine Telemetrieressourcen definiert sind. Sie möchten den Assistenten **Beispielkonfiguration definieren** nicht ausführen.

  a. Führen Sie eine der folgenden Tasks aus, um die Telemetrie einzurichten:

     – Warteschlangenmanager für Telemetrie unter Linux und AIX konfigurieren
     – Warteschlangenmanager für Telemetrie unter Windows konfigurieren

  b. Führen Sie die folgende Task aus: „Task mit einem aktiven Telemetrieservice (MQXR) starten" auf Seite 133
- Wenn Sie die Task mit einem vorhandenen Warteschlangenmanager ausführen möchten, auf dem bereits Telemetrieressourcen definiert sind, führen Sie diese Task aus: „Task mit einem aktiven Telemetrieservice (MQXR) starten" auf Seite 133.

Nächste Schritte

Führen Sie die Task „Nachricht mithilfe von IBM MQ Explorer an einen MQTT -Client senden" auf Seite 136 aus, um eine Nachricht direkt an das Clientdienstprogramm zu senden.

Task ohne definierten Telemetrieservice (MQXR) starten

Erstellen Sie einen Warteschlangenmanager und führen Sie den Assistenten **Beispielkonfiguration definieren** aus, um Beispieltelemetrieressourcen für den Warteschlangenmanager zu definieren. Veröffentlichen Sie eine Nachricht unter Verwendung von IBM MQ Explorer und abonnieren Sie diese mit dem MQTT-Clientdienstprogramm.

Informationen zu diesem Vorgang

Wenn Sie Beispieltelemetrieressourcen mithilfe des Assistenten **Beispielkonfiguration definieren** einrichten, legt der Assistent die Berechtigungen für die Gastbenutzer-ID fest. Überlegen Sie genau, ob Sie eine derartige Berechtigung der Gastbenutzer-ID wünschen. `guest` unter Windowsund `nobody` unter Linuxerhalten die Berechtigung zum Veröffentlichen und Subskribieren im Stammverzeichnis der Themenstruktur und zum Einreihen von Nachrichten in `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Der Assistent setzt außerdem die Standardübertragungswarteschlange auf `SYSTEM.MQTT.TRANSMIT.QUEUE`, was zu Konflikten bei Anwendungen führen kann, die auf einem bestehenden Warteschlangenmanager ausgeführt werden. Es ist zwar aufwändig, jedoch möglich, die Telemetrie zu konfigurieren und nicht die Standardübertragungswarteschlange zu verwenden; führen Sie diese Folgetask aus: „Warteschlangenmanager-Aliasnamen verwenden" auf Seite 134. In dieser Task erstellen Sie einen Warteschlangenmanager, um einen möglichen Konflikt mit vorhandenen Standardübertragungswarteschlangen zu vermeiden.

Vorgehensweise

1. Erstellen Sie unter Verwendung von IBM MQ Explorer einen neuen Warteschlangenmanager und starten Sie diesen.

   a) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers` und klicken Sie auf **Neu** > **Warteschlangenmanager ...**. Geben Sie einen Warteschlangenmanagername ein und klicken Sie auf **Fertigstellen**.

   Denken Sie sich einen Warteschlangenmanagernamen aus, z. B. `MQTTQMGR`.
2. Erstellen und starten Sie den Telemetrieservice (MQXR) und erstellen Sie einen Beispieltelemetriekanal.

   a) Öffnen Sie den Ordner `Queue Managers\`*`QmgrName`*`\Telemetry`.

   b) Klicken Sie auf **Beispielkonfiguration definieren ...** > **Fertigstellen**.

Lassen Sie das Kontrollkästchen **MQTT-Clientdienstprogramm starten** aktiviert.

3. Erstellen Sie mit dem MQTT -Clientdienstprogramm eine Subskription für `MQTT Example` .
   a) Klicken Sie auf **Verbinden**.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbunden).

   b) Geben Sie `MQTT Example` in das Feld **Subscription \ Topic** ein > **Subscribe**.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Subscribed` (Verbunden).
4. Erstellen Sie `MQTTExampleTopic` in IBM MQ.
   a) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers\`*`QmgrName`*`\Topics` in **MQ Explorer** und klicken Sie auf **Neu** > **Thema**.
   b) Geben Sie `MQTTExampleTopic` als **Name** ein und klicken Sie auf **Weiter**.
   c) Geben Sie `MQTT Example` als **Themenzeichenfolge** > **Fertigstellen**ein.
   d) Klicken Sie auf **OK**, um das Bestätigungsfenster zu schließen.
5. Veröffentlichen Sie `Hello World!` mithilfe von IBM MQ Explorerim Thema `MQTT Example` .
   a) Klicken Sie im IBM MQ Explorer auf den Ordner `Queue Managers\`*`QmgrName`*`\Topics`.
   b) Klicken Sie mit der rechten Maustaste auf `MQTTExampleTopic` > **Testveröffentlichung...**
   c) Geben Sie `Hello World!` in das Feld **Nachrichtendaten** ein und klicken Sie auf **Nachricht veröffentlichen**. Wechseln zum Fenster des MQTT-Clientdienstprogramms.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Received` (Verbunden).

Task mit einem aktiven Telemetrieservice (MQXR) starten

Erstellen Sie einen Telemetriekanal und ein Thema. Berechtigen Sie den Benutzer für die Verwendung des Themas und der Übertragungswarteschlange der Telemetrie. Veröffentlichen Sie eine Nachricht unter Verwendung von IBM MQ Explorer und abonnieren Sie diese mit dem MQTT-Clientdienstprogramm.

Vorbereitende Schritte

In dieser Version der Task ist der Warteschlangenmanager *QmgrName*definiert und aktiv. Ein Telemetrieservice (MQXR) ist definiert und aktiv. Der Telemetrieservice (MQXR) kann manuell oder durch Ausführung des Assistenten **Beispielkonfiguration definieren** erstellt worden sein.

Informationen zu diesem Vorgang

In dieser Task konfigurieren Sie einen vorhandenen Warteschlangenmanager für das Senden einer Veröffentlichung an das MQTT-Clientdienstprogramm.

In Schritt „1” auf Seite 133 der Task wird außerdem die Standardübertragungswarteschlange auf `SYSTEM.MQTT.TRANSMIT.QUEUE` gesetzt, was zu Konflikten bei Anwendungen führen kann, die auf einem bestehenden Warteschlangenmanager ausgeführt werden. Es ist zwar aufwändig, jedoch möglich, die Telemetrie zu konfigurieren und nicht die Standardübertragungswarteschlange zu verwenden; führen Sie diese Folgetask aus: „Warteschlangenmanager-Aliasnamen verwenden” auf Seite 134.

Vorgehensweise

1. Legen Sie `SYSTEM.MQTT.TRANSMIT.QUEUE` als Standardübertragungswarteschlange fest.
   a) Klicken Sie mit der rechten Maustaste auf `Queue Managers\`*`QmgrName`* `folder` und klicken Sie auf **Eigenschaften ...**
   b) Klicken Sie im Navigator auf **Kommunikation**.
   c) Klicken Sie auf **Auswählen ...** > Auswählen `SYSTEM.MQTT.TRANSMIT.QUEUE` > **OK** > **OK**.
2. Erstellen Sie den Telemetriekanal `MQTTExampleChannel`, um das MQTT-Clientdienstprogramm mit IBM MQ zu verbinden, und starten Sie das MQTT-Clientdienstprogramm.

a) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers\`*`QmgrName`*` \Telemetry\Channels` im **MQ Explorer** und anschließend auf **Neu** > **Telemetriekanal ...**.

b) Geben Sie `MQTTExampleChannel` in das Feld **Kanalname** ein und klicken Sie auf **Weiter** > **Weiter**.

c) Ändern Sie den Eintrag unter **Festgelegte Benutzer-ID** im Fenster für die Clientberechtigung in die Benutzer-ID, die für `MQTTExample` Veröffentlichungen und Subskriptionen vornehmen wird, und klicken Sie auf **Weiter**.

d) Lassen Sie die Option **Clientdienstprogramm starten** aktiviert und klicken Sie auf **Fertigstellen**.

3. Erstellen Sie mit dem MQTT -Clientdienstprogramm eine Subskription für `MQTT Example` .

a) Klicken Sie auf **Verbinden**.

Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbunden).

b) Geben Sie `MQTT Example` in das Feld **Subscription \ Topic** ein > **Subscribe**.

Das **Clientprotokoll** enthält jetzt das Ereignis `Subscribed` (Verbunden).

4. Erstellen Sie `MQTTExampleTopic` in IBM MQ.

a) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers\`*`QmgrName`*`\Topics` in **MQ Explorer** und klicken Sie auf **Neu** > **Thema**.

b) Geben Sie `MQTTExampleTopic` als **Name** ein und klicken Sie auf **Weiter**.

c) Geben Sie `MQTT Example` als **Themenzeichenfolge** > **Fertigstellen**ein.

d) Klicken Sie auf **OK**, um das Bestätigungsfenster zu schließen.

5. Wenn Sie möchten, dass ein Benutzer, der nicht zur Gruppe `mqm` gehört, Veröffentlichungen und Subskriptionen für das Thema `MQTTExample` vornimmt, gehen Sie wie folgt vor:

a) Berechtigen Sie den Benutzer für die Veröffentlichung und Subskription im Zusammenhang mit dem Thema `MQTTExampleTopic`:

```
setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
```

b) Berechtigen Sie den Benutzer zur Einreihung einer Nachricht in die Warteschlange `SYSTEM.MQTT.TRANSMIT.QUEUE`:

```
setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
```

6. Veröffentlichen Sie `Hello World!` mithilfe von IBM MQ Explorerim Thema `MQTT Example` .

a) Klicken Sie im IBM MQ Explorer auf den Ordner `Queue Managers\`*`QmgrName`*`\Topics`.

b) Klicken Sie mit der rechten Maustaste auf `MQTTExampleTopic` > **Testveröffentlichung...**

c) Geben Sie `Hello World!` in das Feld **Nachrichtendaten** ein und klicken Sie auf **Nachricht veröffentlichen**. Wechseln zum Fenster des MQTT-Clientdienstprogramms.

Das **Clientprotokoll** enthält jetzt das Ereignis `Received` (Verbunden).

Warteschlangenmanager-Aliasnamen verwenden

Veröffentlichen Sie eine Nachricht im MQTT -Clientdienstprogramm unter Verwendung von IBM MQ Explorer , ohne die Standardübertragungswarteschlange auf `SYSTEM.MQTT.TRANSMIT.QUEUE`zu setzen.

Diese Task ist eine Fortsetzung der vorherigen Task; dabei wird ein Warteschlangenmanager-Aliasname verwendet, damit die Standardübertragungswarteschlange nicht auf `SYSTEM.MQTT.TRANSMIT.QUEUE` gesetzt werden muss.

Vorbereitende Schritte

Die Task „Task ohne definierten Telemetrieservice (MQXR) starten” auf Seite 132 oder „Task mit einem aktiven Telemetrieservice (MQXR) starten” auf Seite 133 muss vollständig ausgeführt worden sein.

Informationen zu diesem Vorgang

Wenn ein MQTT-Client eine Subskription erstellt, sendet IBM MQ seine Antwort unter Verwendung von `ClientIdentifier` für den Namen des fernen Warteschlangenmanagers. In dieser Task wird `MyClient` als Client-ID (`ClientIdentifier`) verwendet.

Falls keine Übertragungswarteschlange oder kein Warteschlangenmanager-Aliasname namens `MyClient` vorhanden ist, wird die Antwort in die Standardübertragungswarteschlange gestellt. Wenn Sie die Standardübertragungswarteschlange auf `SYSTEM.MQTT.TRANSMIT.QUEUE`setzen, erhält der MQTT -Client die Antwort.

Wenn Sie Warteschlangenmanager-Aliasnamen verwenden, müssen Sie die Standardübertragungswarteschlange nicht auf `SYSTEM.MQTT.TRANSMIT.QUEUE` setzen. Sie müssen für jede Instanz von `ClientIdentifier` einen Warteschlangenmanager-Aliasnamen definieren. Für gewöhnlich können Warteschlangenmanager-Aliasnamen aufgrund der Vielzahl an Clients nicht verwendet werden. Häufig ist der `ClientIdentifier`-Eintrag unvorhersehbar, was eine derartige Konfiguration der Telemetrie unmöglich macht.

In manchen Situationen kann es jedoch vorkommen, dass Sie die Standardübertragungswarteschlange mit einem anderen Wert als `SYSTEM.MQTT.TRANSMIT.QUEUE` konfigurieren müssen. Mit den Schritten unter Vorgehensweise wird ein Warteschlangenmanager-Alias konfiguriert, anstatt die Standardübertragungswarteschlange auf `SYSTEM.MQTT.TRANSMIT.QUEUE`zu setzen.

Vorgehensweise

1. Entfernen Sie `SYSTEM.MQTT.TRANSMIT.QUEUE` als Standardübertragungswarteschlange.
   a) Klicken Sie mit der rechten Maustaste auf `Queue Managers\`*`QmgrName`* `folder` und klicken Sie auf **Eigenschaften ...**
   b) Klicken Sie im Navigator auf **Kommunikation**.
   c) Entfernen Sie `SYSTEM.MQTT.TRANSMIT.QUEUE` aus dem Feld **Standardübertragungswarteschlange** und klicken Sie auf **OK**.
2. Vergewissern Sie sich, dass mit dem MQTT-Clientdienstprogramm keine Erstellung einer Subskription mehr möglich ist:
   a) Klicken Sie auf **Verbinden**.

      Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbunden).
   b) Geben Sie `MQTT Example` in das Feld **Subscription \ Topic** ein > **Subscribe**.

   Im **Clientprotokoll** werden die Ereignisse `Subscribe failed` (Subskription fehlgeschlagen) und `Connection lost` (Verbindung nicht mehr vorhanden) aufgezeichnet.
3. Erstellen Sie einen Warteschlangenmanager-Aliasnamen für den `ClientIdentifier`-Eintrag `MyClient`.
   a) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers\`*`QmgrName`*`\Queues` und klicken Sie anschießend auf **Neu** > **Definition der fernen Warteschlange**.
   b) Geben Sie der Definition den Namen `MyClient` > und klicken Sie auf **Weiter**.
   c) Geben Sie `MyClient` in das Feld **Ferner Warteschlangenmanager** ein.
   d) Geben Sie `SYSTEM.MQTT.TRANSMIT.QUEUE` im Feld **Übertragungswarteschlange** und klicken Sie auf **Fertigstellen**.
4. Stellen Sie wieder eine Verbindung zum MQTT-Clientdienstprogramm her.
   a) Vergewissern Sie sich, dass die **Client-ID** auf `MyClient` gesetzt ist.
   b) **Verbindung herstellen**

   Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbunden).
5. Erstellen Sie mit dem MQTT -Clientdienstprogramm eine Subskription für `MQTT Example` .
   a) Klicken Sie auf **Verbinden**.

Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbunden).

b) Geben Sie `MQTT Example` in das Feld **Subscription \ Topic** ein > **Subscribe**.

Das **Clientprotokoll** enthält jetzt das Ereignis `Subscribed` (Verbunden).

6. Veröffentlichen Sie `Hello World!` mithilfe von IBM MQ Explorerim Thema `MQTT Example` .
   a) Klicken Sie im IBM MQ Explorer auf den Ordner `Queue Managers\`*`QmgrName`*`\Topics`.
   b) Klicken Sie mit der rechten Maustaste auf `MQTTExampleTopic` > **Testveröffentlichung...**
   c) Geben Sie `Hello World!` in das Feld **Nachrichtendaten** ein und klicken Sie auf **Nachricht veröffentlichen**. Wechseln zum Fenster des MQTT-Clientdienstprogramms.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Received` (Verbunden).

Windows Linux AIX Nachricht mithilfe von IBM MQ Explorer an einen MQTT -Client senden

Senden Sie eine Nachricht an das MQTT -Clientdienstprogramm, indem Sie eine Nachricht mithilfe von IBM MQ Explorerin eine IBM MQ -Warteschlange stellen. Diese Task veranschaulicht die Konfiguration einer Definition einer fernen Warteschlange zum direkten Senden einer Nachricht an einen MQTT-Client.

Vorbereitende Schritte

Führen Sie den im Abschnitt „Nachricht über IBM MQ Explorer im MQTT -Clientdienstprogramm veröffentlichen” auf Seite 131 beschriebenen Schritt aus. Behalten Sie die Verbindung des MQTT-Clientdienstprogramms bei.

Informationen zu diesem Vorgang

Die Task veranschaulicht das Senden einer Nachricht an einen MQTT-Client, indem sie nicht in einem Thema veröffentlicht, sondern in eine Warteschlange gestellt wird. Sie erstellen keine Subskription im Client. Schritt „2” auf Seite 136 in der Task veranschaulicht, dass die vorherige Subskription gelöscht wurde.

Vorgehensweise

1. Verwerfen Sie alle vorhandenen Subskriptionen, indem Sie die Verbindung des MQTT-Clientdienstprogramms trennen und wiederherstellen.

   Sofern Sie die Standardeinstellungen nicht geändert haben, wird die Subskription verworfen, da das MQTT-Clientdienstprogramm mit einer bereinigten Sitzung eine Verbindung herstellt (siehe Sitzungen bereinigen).

   Geben Sie zur Vereinfachung der Task Ihren eigenen `ClientIdentifier`-Eintrag ein, statt den generierten `ClientIdentifier`-Eintrag zu verwenden, der vom MQTT-Clientdienstprogramm erstellt wurde.

   a) Klicken Sie auf **Verbindung trennen**, um die Verbindung zwischen dem MQTT-Clientdienstprogramm und dem Telemetriekanal zu trennen.

      Das **Clientprotokoll** enthält jetzt das Ereignis `Disconnected` (Verbindung getrennt).

   b) Ändern Sie die **Client-ID** in `MyClient`.

   c) Klicken Sie auf **Verbinden**.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbindung getrennt).
2. Stellen Sie sicher, dass das MQTT-Clientdienstprogramm keine Veröffentlichungen für das Thema `MQTTExampleTopic` mehr empfängt.
   a) Klicken Sie im IBM MQ Explorer auf den Ordner `Queue Managers\`*`QmgrName`*`\Topics`.
   b) Klicken Sie mit der rechten Maustaste auf `MQTTExampleTopic` > **Testveröffentlichung...**

c) Geben Sie `Hello World!` in das Feld **Nachrichtendaten** ein und klicken Sie auf **Nachricht veröffentlichen**. Wechseln zum Fenster des MQTT-Clientdienstprogramms.

Im **Clientprotokoll** wird kein Ereignis erfasst.

3. Erstellen Sie eine Definition einer fernen Warteschlange für den Client.

   Legen Sie den `ClientIdentifier`-Eintrag `MyClient` für den Namen des fernen Warteschlangenmanagers in der Definition einer fernen Warteschlange fest. Für den Namen der fernen Warteschlange kann ein beliebiger Name verwendet werden. Der Name der fernen Warteschlange wird als Themenname an einen MQTT-Client übergeben.

   a) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers\`*`QmgrName`*`\Queues` und klicken Sie anschießend auf **Neu** > **Definition der fernen Warteschlange**.

   b) Geben Sie der Definition den Namen `MyClientRemoteQueue` und klicken Sie auf **Weiter**.

   c) Geben Sie `MQTTExampleQueue` in das Feld **Ferne Warteschlange** ein.

   d) Geben Sie `MyClient` in das Feld **Ferner Warteschlangenmanager** ein.

   e) Geben Sie `SYSTEM.MQTT.TRANSMIT.QUEUE` im Feld **Übertragungswarteschlange** und klicken Sie auf **Fertigstellen**.

4. Reihen Sie eine Testnachricht in `MyClientRemoteQueue` ein.

   a) Klicken Sie mit der rechten Maustaste auf **MyClientRemoteQueue** > **Testnachricht einreihen...**

   b) Geben Sie `Hello queue!` in das Feld für Nachrichtendaten ein > **Nachricht einreihen** > **Schließen**

   Das **Clientprotokoll** enthält jetzt das Ereignis `Received` (Empfangen).

5. Entfernen Sie `SYSTEM.MQTT.TRANSMIT.QUEUE` als Standardübertragungswarteschlange.

   a) Klicken Sie mit der rechten Maustaste auf `Queue Managers\`*`QmgrName`* `folder` und klicken Sie auf **Eigenschaften ...**

   b) Klicken Sie im Navigator auf **Kommunikation**.

   c) Entfernen Sie `SYSTEM.MQTT.TRANSMIT.QUEUE` aus dem Feld **Standardübertragungswarteschlange** und klicken Sie auf **OK**.

6. Wiederholen Sie den Schritt „4” auf Seite 137.

   `MyClientRemoteQueue` ist eine Definition einer fernen Warteschlange, in der die Übertragungswarteschlange explizit genannt wird. Sie müssen keine Standardübertragungswarteschlange definieren, um eine Nachricht an `MyClient` zu senden.

Nächste Schritte

Wenn die Standardübertragungswarteschlange nicht mehr auf `SYSTEM.MQTT.TRANSMIT.QUEUE`gesetzt ist, kann das MQTT -Clientdienstprogramm keine neue Subskription mehr erstellen, es sei denn, für `ClientIdentifier` `MyClient`ist ein Warteschlangenmanager-Aliasname definiert. Setzen Sie die Standardübertragungswarteschlange wieder auf `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Windows Linux AIX Nachricht an einen bestimmten MQTT v3-Client veröffentlichen

Veröffentlichen Sie eine Nachricht von einem MQTT v3-Client an einen anderen und verwenden Sie dabei `ClientIdentifier` als Themennamen und IBM MQ als Publish/Subscribe-Broker.

Vorbereitende Schritte

Führen Sie den im Abschnitt „Nachricht über IBM MQ Explorer im MQTT -Clientdienstprogramm veröffentlichen” auf Seite 131 beschriebenen Schritt aus. Behalten Sie die Verbindung des MQTT-Clientdienstprogramms bei.

Informationen zu diesem Vorgang

Die Task veranschaulicht zwei Punkte:

1. Die Subskription eines Themas in dem einen MQTT-Client und dem Empfang einer Veröffentlichung von einem anderen MQTT-Client.
2. Die Einrichtung von "Punkt-zu-Punkt"-Subskriptionen unter Verwendung von `ClientIdentifier` als Themenzeichenfolge.

Vorgehensweise

1. Verwerfen Sie alle vorhandenen Subskriptionen, indem Sie die Verbindung des MQTT-Clientdienstprogramms trennen und wiederherstellen.

   Sofern Sie die Standardeinstellungen nicht geändert haben, wird die Subskription verworfen, da das MQTT-Clientdienstprogramm mit einer bereinigten Sitzung eine Verbindung herstellt (siehe Sitzungen bereinigen).

   Geben Sie zur Vereinfachung der Task Ihren eigenen `ClientIdentifier`-Eintrag ein, statt den generierten `ClientIdentifier`-Eintrag zu verwenden, der vom MQTT-Clientdienstprogramm erstellt wurde.

   a) Klicken Sie auf **Verbindung trennen**, um die Verbindung zwischen dem MQTT-Clientdienstprogramm und dem Telemetriekanal zu trennen.

      Das **Clientprotokoll** enthält jetzt das Ereignis `Disconnected` (Verbindung getrennt).

   b) Ändern Sie die **Client-ID** in `MyClient`.

   c) Klicken Sie auf **Verbinden**.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbindung getrennt).
2. Erstellen Sie eine Subskription für das Thema `MyClient`.

   `MyClient` ist der `ClientIdentifier`-Eintrag dieses Clients.

   a) Geben Sie `MyClient` in das Feld **Subscription\Topic** ein und klicken Sie auf **Subskribieren**.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Subscribed` (Verbunden).
3. Starten Sie ein anderes MQTT-Clientdienstprogramm.

   a) Öffnen Sie den Ordner `Queue Managers\`*`QmgrName`*`\Telemetry\channels`.

   b) Klicken Sie mit der rechten Maustaste auf den Kanal **PlainText** und klicken Sie auf **MQTT-Clientdienstprogramm ausführen...**

   c) Klicken Sie auf **Verbinden**.

   Das **Clientprotokoll** enthält jetzt das Ereignis `Connected` (Verbindung getrennt).
4. Veröffentlichen Sie `Hello MyClient!` im Thema `MyClient`.

   a) Kopieren Sie das Subskriptionsthema `MyClient` aus dem MQTT-Clientdienstprogramm, das mit dem `ClientIdentifier`-Eintrag `MyClient` ausgeführt wird.

   b) Fügen Sie `MyClient` in das Feld **Publication\Topic** aller MQTT-Clientdienstprogramminstanzen ein.

   c) Geben Sie `Hello MyClient!` in das Feld **Publication\message** ein.

   d) Klicken Sie in beiden Instanzen auf **Veröffentlichen**.

Ergebnisse

Das **Clientprotokoll** im MQTT-Clientdienstprogramm mit dem `ClientIdentifier`-Eintrag `MyClient` enthält zwei Ereignisse des Typs **Received** (Empfangen) und ein Ereignis des Typs **Published** (Veröffentlicht). Die andere Instanz des MQTT-Clientdienstprogramms dokumentiert ein Ereignis des Typs **Published** (Veröffentlicht).

Wenn Sie nur ein Ereignis des Typs **Received** (Empfangen) sehen, prüfen Sie folgende mögliche Ursachen:

1. Ist die Standardübertragungswarteschlange für den Warteschlangenmanager auf SYSTEM.MQTT.TRANSMIT.QUEUE gesetzt?
2. Haben Sie bei der Durchführung der anderen Übungen Warteschlangenmanager-Aliasnamen oder Definitionen einer fernen Warteschlange erstellt, die auf `MyClient` verweisen? Falls ein Konfigurationsproblem vorliegt, löschen Sie alle Ressourcen, die auf `MyClient` verweisen (beispielsweise Warteschlangenmanager-Aliasnamen oder Übertragungswarteschlangen). Trennen Sie die Verbindung der Clientdienstprogramme, stoppen Sie den Telemetrieservice (MQXR) und starten Sie ihn erneut.

Windows Linux AIX Nachricht von einem MQTT -Client an eine IBM MQ -Anwendung senden

Eine IBM MQ-Anwendung kann eine Nachricht von einem MQTT v3-Client empfangen, indem sie ein Thema abonniert. Der MQTT-Client verbindet sich mit IBM MQ über einen Telemetriekanal und sendet eine Nachricht an die IBM MQ-Anwendung, indem er Informationen in demselben Thema veröffentlicht.

Führen Sie die Task „Nachricht von einem MQTT -Client in IBM MQ veröffentlichen” auf Seite 139 aus, um zu lernen, wie eine Veröffentlichung von einem MQTT-Client an eine Subskription gesendet wird, die in IBM MQ definiert ist.

Falls das Thema in Gruppen zusammengefasst ist oder unter Verwendung der Publish/Subscribe-Hierarchie verteilt wird, kann sich die Subskription auf einem anderen Warteschlangenmanager befinden als dem Warteschlangenmanager, mit dem der MQTT-Client verbunden ist.

Windows Linux AIX Nachricht von einem MQTT -Client in IBM MQ veröffentlichen

Erstellen Sie eine Subskription für ein Thema unter Verwendung von IBM MQ Explorer und veröffentlichen Sie Informationen im Thema unter Verwendung des MQTT-Clientdienstprogramms.

Vorbereitende Schritte

Führen Sie den im Abschnitt „Nachricht über IBM MQ Explorer im MQTT -Clientdienstprogramm veröffentlichen” auf Seite 131 beschriebenen Schritt aus. Behalten Sie die Verbindung des MQTT-Clientdienstprogramms bei.

Informationen zu diesem Vorgang

Die Task veranschaulicht die Veröffentlichung einer Nachricht mit einem MQTT-Client und den Empfang der Veröffentlichung über eine nicht verwaltete permanente Subskription, die unter Verwendung von IBM MQ Explorer erstellt wird.

Vorgehensweise

1. Erstellen Sie eine permanente Subskription für die Themenzeichenfolge `MQTT Example`.

   Führen Sie die folgenden Schritte zur Erstellung der Warteschlange und Subskription unter Verwendung von IBM MQ Explorer aus.

   a) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers\`*`QmgrName`*`\Queues` unter IBM MQ Explorer > **Neu** > **Lokale Warteschlange...**.

   b) Geben Sie `MQTTExampleQueue` als Warteschlangenname ein > **Fertigstellen**.

   c) Klicken Sie mit der rechten Maustaste auf den Ordner `Queue Managers\`*`QmgrName`*`\Subscriptions` unter IBM MQ Explorer > **Neu** > **Subskription...**.

   d) Geben Sie `MQTTExampleSubscription` als Warteschlangenname ein und klicken Sie auf **Weiter**.

   e) Klicken Sie auf **Auswählen...** > `MQTTExampleTopic` > **OK**.

   Sie haben das Thema `MQTTExampleTopic` bereits in Schritt „4” auf Seite 133 von „Nachricht über IBM MQ Explorer im MQTT -Clientdienstprogramm veröffentlichen” auf Seite 131 erstellt.

   f) Geben Sie `MQTTExampleQueue` als Bestimmungsname ein und klicken Sie auf **Fertigstellen**.

2. Sie können wahlweise die Warteschlange zur Verwendung eines anderen Benutzers ohne mqm-Berechtigung einrichten.

   Wenn Sie die Konfiguration für Benutzer mit einer niedrigeren Berechtigung als mqm einrichten, müssen Sie für MQTTExampleQueue die Berechtigungen put und get erteilen. Der Zugriff auf das Thema und die Übertragungswarteschlange wurde in „Nachricht über IBM MQ Explorer im MQTT -Clientdienstprogramm veröffentlichen" auf Seite 131 konfiguriert.

   a) Berechtigen Sie einen Benutzer für die Einreihung in die Warteschlange MQTTExampleQueue sowie für den Abruf aus dieser Warteschlange:

   ```
   setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
   ```

3. Veröffentlichen Sie Hello IBM MQ! mithilfe des MQTT -Clientdienstprogramms im Thema MQTT Example .

   Wenn die Verbindung zum MQTT-Clientdienstprogramm nicht mehr besteht, klicken Sie mit der rechten Maustaste auf den **PlainText**-Kanal und wählen **MQTT-Clientdienstprogramm ausführen...** > **Verbinden** aus.

   a) Geben Sie MQTT Example in das Feld **Publication\Topic** ein.

   b) Geben Sie Hello IBM MQ! in das Feld **Publication\Message** ein und klicken Sie auf **Veröffentlichen**.

4. Öffnen Sie den Ordner Queue Managers*QmgrName*\Queues und suchen Sie den Eintrag MQTTExampleQueue.

   Das Feld **Aktuelle Warteschlangenlänge** hat den Wert 1

5. Klicken Sie mit der rechten Maustaste auf MQTTExampleQueue > **Nachrichten durchsuchen ...** und sehen Sie sich die Veröffentlichung an.

Windows Linux AIX MQTT-Publish/Subscribe-Anwendungen

Verwenden Sie die themenbasierte Publish/Subscribe-Funktion zum Schreiben von MQTT-Anwendungen.

Wenn der MQTT-Client verbunden ist, fließen Veröffentlichungen zwischen dem Client und Server in beide Richtungen. Die Veröffentlichungen werden vom Client gesendet, sobald Informationen auf dem Client veröffentlicht werden. Veröffentlichungen werden beim Client empfangen, sobald eine Nachricht in einem Thema veröffentlicht wird, das einer vom Client erstellten Subskription entspricht.

Der Publish/Subscribe-Broker von IBM MQ verwaltet die Themen und Subskriptionen, die von MQTT -Clients erstellt wurden. Die von MQTT-Clients erstellten Themen nutzen denselben Themenbereich wie Themen, die von IBM MQ-Anwendungen erstellt werden.

Veröffentlichungen, die der Themenzeichenfolge in einer MQTT -Clientsubskription entsprechen, werden in SYSTEM.MQTT.TRANSMIT.QUEUE gestellt, wobei der Name des fernen Warteschlangenmanagers auf die ClientIdentifier des Clients gesetzt ist. Der Telemetrieservice (MQXR) leitet die Veröffentlichungen an den Client weiter, von dem die Subskription erstellt wurde. Er verwendet den ClientIdentifier-Eintrag, der als Name des fernen Warteschlangenmanagerrs festgelegt wurde, zur Identifizierung des Clients.

Für gewöhnlich muss SYSTEM.MQTT.TRANSMIT.QUEUE als Standardübertragungswarteschlange definiert werden. Es ist zwar aufwendig, aber möglich, MQTT so zu konfigurieren, dass nicht die Standardübertragungswarteschlange verwendet wird (siehe Verteilte Steuerung von Warteschlangen zum Senden von Nachrichten an MQTT-Clients verwenden).

Ein MQTT-Client kann eine persistente Sitzung erstellen (siehe „Statusunabhängige und statusbehaftete MQTT-Sitzungen" auf Seite 144). Subskriptionen, die in einer persistenten Sitzung erstellt werden, sind permanent. Veröffentlichungen, die für einen Client mit einer persistenten Sitzung eingehen, werden in der Warteschlange SYSTEM.MQTT.TRANSMIT.QUEUE gespeichert und an den Client weitergeleitet, wenn er die Verbindung wiederherstellt.

Ein MQTT-Client kann auch in ständigen Veröffentlichungen Informationen veröffentlichen und abonnieren (siehe Ständige Veröffentlichungen und MQTT-Clients). Ein Subskribent eines Themas in einer ständigen Veröffentlichung empfängt die neueste Veröffentlichung des Themas. Der Subskribent empfängt die ständige Veröffentlichung, wenn er eine Subskription erstellt oder die Verbindung zu seiner früheren Sitzung wiederherstellt.

Windows Linux AIX Telemetrieanwendungen

Schreiben Sie Telemetrieanwendungen unter Verwendung von IBM MQ- oder IBM Integration Bus-Nachrichtenflüssen.

Verwenden Sie JMS, MQI oder sonstige IBM MQ-Programmierschnittstellen, um Telemetrieanwendungen in IBM MQ zu programmieren.

Der Telemetrieservice (MQXR) nimmt eine Konvertierung zwischen MQTT v3-Nachrichten und IBM MQ-Nachrichten vor. Er erstellt Subskriptionen und Veröffentlichungen für MQTT-Clients und leitet Veröffentlichungen an MQTT-Clients weiter. Eine Veröffentlichung sind die Nutzdaten einer MQTT v3-Nachricht. Die Nutzdaten umfassen Nachrichtenheader und eine Bytefeldgruppe im Format `jms-bytes`. Der Telemetrieserver ordnet die Header zwischen einer MQTT v3-Nachricht und einer IBM MQ-Nachricht zu (siehe „Integration von MQ Telemetry mit Warteschlangenmanagern” auf Seite 141).

Verwenden Sie die Publication-, MQInput- und JMSInput-Knoten, um Veröffentlichungen zwischen IBM Integration Bus und MQTT-Clients zu senden und zu empfangen.

Mithilfe von Nachrichtenflüssen können Sie die Telemetrie mithilfe von HTTP mit Websites und mithilfe von IBM MQ und WebSphere Adapters mit anderen Anwendungen integrieren.

Windows Linux AIX Integration von MQ Telemetry mit Warteschlangenmanagern

Der MQTT-Client ist als Publish/Subscribe-Anwendung in IBM MQ integriert. Er kann Themen in IBM MQ entweder veröffentlichen oder abonnieren, neue Themen erstellen oder vorhandene Themen verwenden. Er empfängt Veröffentlichungen von IBM MQ, wenn MQTT-Clients (ihn selbst eingeschlossen) oder andere IBM MQ-Anwendungen Veröffentlichungen zu den Themen seiner Subskriptionen veröffentlichen. Bei der Festlegung der Attribute einer Veröffentlichung gelten bestimmte Regeln.

Viele der Attribute, die den von IBM MQ bereitgestellten Themen, Veröffentlichungen, Subskriptionen und Nachrichten zugeordnet sind, werden nicht unterstützt. Unter „MQTT-Client an IBM MQ-Publish/Subscribe-Broker” auf Seite 141 und „IBM MQ an einen MQTT-Client” auf Seite 143 wird beschrieben, wie Attribute von Veröffentlichungen festgelegt werden. Die Einstellungen hängen davon ab, ob die Veröffentlichung an den IBM MQ-Publish/Subscribe-Broker geleitet wird oder von diesem stammt.

In IBM MQ-Publish/Subscribe sind Themen mit administrativen Themenobjekten verknüpft. Dies gilt auch für Themen, die von MQTT-Clients erstellt werden. Wenn ein MQTT-Client eine Themenzeichenfolge für eine Veröffentlichung erstellt, ordnet der IBM MQ-Publish/Subscribe-Broker diese einem administrativen Themenobjekt zu. Der Broker ordnet die Themenzeichenfolge in der Veröffentlichung dem nächsten übergeordneten administrativen Themenobjekt zu. Die Zuordnung erfolgt auf die gleiche Weise wie bei IBM MQ-Anwendungen. Wenn kein Thema vorhanden ist, das von einem Benutzer erstellt wurde, wird das Veröffentlichungsthema `SYSTEM.BASE.TOPIC` zugeordnet. Die Attribute der Veröffentlichung leiten sich aus dem Themenobjekt ab.

Wenn eine IBM MQ-Anwendung oder ein Administrator eine Subskription erstellt, erhält die Subskription einen Namen. Abonnements auflisten mitIBM MQ Explorer oder mithilfe von**runmqsc** oder PCF-Befehle. Alle MQTT-Clientsubskriptionen haben einen Namen. Sie erhalten einen Namen der Form:*ClientIdentifier*:*Topic name*

MQTT-Client an IBM MQ-Publish/Subscribe-Broker

Ein MQTT-Client hat eine Veröffentlichung an IBM MQ gesendet. Der Telemetrieservice (MQXR) konvertiert die Veröffentlichung in eine IBM MQ-Nachricht. Die IBM MQ-Nachricht enthält drei Teile:

1. MQMD
2. RFH2
3. Nachricht

MQMD-Eigenschaften werden auf ihre Standardwerte gesetzt, es sei denn, dies ist in Tabelle 9 auf Seite 142 anders angegeben.

Tabelle 9. Einstellungen für MQMD-Eigenschaften

MQMD-Feld	**Typ**	**Wert**
Format	MQCHAR8	MQFMT_RF_HEADER_2
UserIdentifier	MQCHAR12	Legen Sie einen der folgenden Werte fest: MqttClient.ClientIdentifier MqttConnectOptions.UserName Eine Benutzer-ID, die vom IBM MQ-Administrator für den Telemetriekanal festgelegt wird
Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (Standardeinstellung bei IBM MQ, während JMS den Standardwert 4 hat)
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

Der Header RFH2 enthält keinen <msd>-Ordner für die Definition des Typs der JMS-Nachricht. Der Telemetrieservice (MQXR) erstellt die IBM MQ-Nachricht als JMS-Standardnachricht. Der JMS-Standardnachrichtentyp ist eine jms-bytes-Nachricht. Eine Anwendung kann in Form von Nachrichteneigenschaften auf die zusätzlichen Kopfzeileninformationen zugreifen; siehe Nachrichteneigenschaften.

RFH2-Werte werden wie in Tabelle 10 auf Seite 142 dargestellt festgelegt. Die Eigenschaft 'Format' wird im festen RFH2-Header festgelegt, während die übrigen Werte in RFH2-Ordnern festgelegt werden.

Tabelle 10. Einstellungen für RFH2-Eigenschaften

RFH2-Eigenschaft	**Typ/Ordner**	**Header**
Format	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	Kopieren Sie MqttClient.ClientIdentifier mit einer Länge von 1...23 Bytes.
QoS	mqtt/qos	Kopieren Sie QoS aus der eingehenden MQTT-Nachricht.
Nachrichten-ID	mqtt/msgid	Kopieren Sie die Nachrichten-ID (Message ID) aus der eingehenden MQTT-Nachricht, wenn QoS den Wert 1 oder 2 hat.
MQIsRetained	mqps/Ret	Festgelegt, wenn die ursprüngliche MQTT-Veröffentlichung mit der Einstellung der Eigenschaft RETAIN gesendet wurde und die Nachricht als ständige Veröffentlichung empfangen wird.
MQTopicString	mqps/Top	Das Thema, in dem die MQTT-Nachricht veröffentlicht wurde.

Die Nutzdaten in einer MQTT-Veröffentlichung werden dem Inhalt einer IBM MQ-Nachricht zugeordnet:

Tabelle 11. Zuordnung der Nutzdaten in einer MQTT-Veröffentlichung zu den IBM MQ-Nachrichteninhalten		
Nachrichteninhalt(e)	**Typ**	**Inhalt der IBM MQ-Nachricht**
`Buffer`	`MQBYTE` *n*	Kopie der Bytes der eingehenden MQTT-Nachricht. Die Länge kann null sein.

IBM MQ an einen MQTT-Client

Ein Client hat ein Veröffentlichungsthema abonniert. Eine IBM MQ -Anwendung hat zu dem Thema veröffentlicht, sodass eine Veröffentlichung vom IBM MQ Publish/Subscribe-Broker an den MQTT -Subskribenten gesendet wird. Möglich ist auch, dass eine IBM MQ-Anwendung eine nicht erwartete Nachricht direkt an einen MQTT-Client gesendet hat. In Tabelle 12 auf Seite 143 wird beschrieben, wie die festgelegten Nachrichtenheader in der Nachricht festgelegt werden, die an den MQTT-Client gesendet wird. Alle anderen Daten im IBM MQ-Nachrichtenheader sowie alle anderen Header werden gelöscht. Die Nachrichtendaten in der IBM MQ-Nachricht werden unverändert als Nachrichtennutzdaten in der MQTT-Nachricht gesendet. Die MQTT-Nachricht wird vom Telemetrieservice (MQXR) an den MQTT-Client gesendet.

Tabelle 12. Festlegen von festgelegten Nachrichtenheadern in einer IBM MQ -Nachricht, die an den MQTT -Client gesendet wird		
Feld MQTT	**Typ**	**Wert**
DUP	`Boolesch`	Festgelegt, wenn der Wert `QoS = 1` oder 2 eingestellt wurde und die Nachricht in einer früheren Übertragung an diesen Client gesendet, jedoch nach einer gewissen Zeit noch nicht bestätigt wurde.

QoS	`int`	Wie der Wert von `QoS` in einer ausgehenden Veröffentlichung vom Publish/Subscribe-Broker in IBM MQ festgelegt wird, hängt von der eingehenden Veröffentlichung ab. Hierbei ist relevant, ob die eingehende Veröffentlichung von einem MQTT-Client oder von einer IBM MQ-Anwendung gesendet wurde.
		MQTT
		Legen Sie in der eingehenden Veröffentlichung einen niedrigeren QoS-Wert (Servicequalität) fest. Dasselbe gilt für die vom Subskribenten angeforderte QoS.
		IBM MQ
		Verringern Sie den Wert der von der eingehenden Veröffentlichung abgeleiteten QoS:
		`MQPER_NOT_PERSISTENT→QoS=0`
		`MQPER_PERSISTENT→QoS=2`
		Dasselbe gilt für die vom Subskribenten angeforderte QoS. Wenn die Nachricht ohne Subskription an den Client gesendet wird, wird die Servicequalität (QoS) standardmäßig auf 2 gesetzt. Ein Client kann diesen Wert ändern, indem er `DEFAULT.QoS` mit einer anderen QoS abonniert.

RETAIN	`Boolesch`	Festgelegt, wenn für die eingehende Veröffentlichung die Eigenschaft 'retained' (ständig) festgelegt wurde.

In Tabelle 13 auf Seite 144 wird beschrieben, wie die variablen Nachrichtenheader in der MQTT-Nachricht festgelegt werden, die an den MQTT-Client gesendet wird.

Tabelle 13. Wie variable MQTT-Headereigenschaften in einer an den MQTT-Client gesendeten MQTT-Nachricht festgelegt werde

Feld MQTT	Typ	Wert
Topic name	Zeichenfolge	Die Themenzeichenfolge, mit der die Nachricht veröffentlicht wurde.
Message ID	Zeichenfolge	Die letzten 2 Bytes der Eigenschaft MQMD.MsgId der Veröffentlichung, wenn sie in SYSTEM.MQTT.TRANSMIT.QUEUE gestellt wird.
Payload	byte[]	Direktkopie der Bytes aus der eingehenden Veröffentlichung an den Publish/Subscribe-Broker. Die Länge kann null sein.

Windows Linux AIX Statusunabhängige und statusbehaftete MQTT-Sitzungen

MQTT-Clients können eine statusbehaftete Sitzung mit dem Warteschlangenmanager erstellen. Wenn ein statusbehafteter MQTT-Client getrennt wird, behält der Warteschlangenmanager die vom Client erstellten Subskriptionen und unvollständig verarbeitete Nachrichten. Sobald der Client die Verbindung wiederherstellt, löst er unvollständig verarbeitete Nachrichten auf. Er sendet alle Nachrichten, die für die Zustellung eingereiht wurden und empfängt Nachrichten, die für seine Subskriptionen veröffentlicht wurden, solange er getrennt war.

Wenn ein MQTT-Client eine Verbindung zu einem Telemetriekanal herstellt, startet er entweder eine neue Sitzung oder setzt eine alte Sitzung fort. Eine neue Sitzung verfügt weder über noch nicht bestätigte, ausstehende Nachrichten, noch über Subskriptionen oder Veröffentlichungen, die zur Zustellung anstehen. Wenn ein Client eine Verbindung herstellt, gibt er an, ob der Start mit einer neuen Sitzung erfolgen oder eine bestehende Sitzung fortgesetzt werden soll (siehe Sitzungen bereinigen).

Wenn der Client eine bestehende Sitzung fortsetzt, erfolgt die Verarbeitung, als ob die Verbindung niemals unterbrochen worden wäre. Veröffentlichungen, die zur Zustellung anstehen, werden an den Client gesendet und alle noch nicht festgeschriebenen Nachrichtenübertragungen werden ausgeführt. Wenn ein Client in einer persistenten Sitzung die Verbindung zum Telemetrieservice (MQXR) trennt, bleiben alle Subskriptionen, die vom Client erstellt wurden, erhalten. Veröffentlichungen für die Subskriptionen werden an den Client gesendet, sobald dieser die Verbindung wiederherstellt. Wenn er die Verbindung wiederherstellt, ohne die alte Sitzung fortzusetzen, werden die Veröffentlichungen vom Telemetrieservice (MQXR) gelöscht.

Die Informationen zum Sitzungsstatus werden vom Warteschlangenmanager in der Warteschlange SYSTEM.MQTT.PERSISTENT.STATE gespeichert.

Der IBM MQ-Administrator kann eine Sitzung trennen und löschen.

Windows Linux AIX MQTT-Client ist nicht verbunden

Wenn ein Client nicht verbunden ist, kann der Warteschlangenmanager für ihn weiterhin Veröffentlichungen empfangen. Diese werden an den Client weitergeleitet, sobald er die Verbindung wiederherstellt. Ein Client kann ein "Last Will and Testament" erstellen, das vom Warteschlangenmanager für den Client veröffentlicht wird, wenn die Verbindung des Clients unerwartet getrennt wird.

Wenn Sie bei einer unerwarteten Verbindungstrennung des Clients benachrichtigt werden möchten, können Sie eine Veröffentlichung des Typs 'Last Will and Testament' registrieren (siehe Veröffentlichung 'Last Will and Testament'. Sie wird vom Telemetrieservice (MQXR) gesendet, wenn dieser feststellt, dass die Verbindung zum Client ohne eine entsprechende Trennungsanforderung des Clients unterbrochen wurde.

Ein Client kann jederzeit eine ständige Veröffentlichung veröffentlichen (siehe Ständige Veröffentlichungen und MQTT-Clients). In einer neuen Subskription eines Themas kann angefordert werden, dass alle ständige Veröffentlichungen im Zusammenhang mit diesem Thema gesendet werden. Wenn Sie das Last

Will and Testament als ständige Veröffentlichung erstellen, können Sie mit dessen Hilfe den Status eines Clients überwachen.

Der Client veröffentlicht beispielsweise eine ständige Veröffentlichung, wenn er sich verbindet, in der seine Verfügbarkeit bekannt gemacht wird. Gleichzeitig erstellt er eine ständige Veröffentlichung des Typs 'Last Will and Testament', die seine Nichtverfügbarkeit ankündigt. Außerdem veröffentlicht er seine Nichtverfügbarkeit als ständige Veröffentlichung unmittelbar vor einem geplanten Verbindungsabbau. Um festzustellen, ob der Client verfügbar ist, müssen Sie das Thema der ständigen Veröffentlichung abonnieren. Sie erhalten in diesem Fall immer eine der drei Veröffentlichungen.

Wenn der Client veröffentlichte Nachrichten erhalten soll, wenn er getrennt ist, müssen Sie den Client mit seiner vorherigen Sitzung erneut verbinden; siehe „Statusunabhängige und statusbehaftete MQTT-Sitzungen” auf Seite 144. Seine Subskriptionen sind aktiv, bis sie gelöscht werden oder bis der Client eine neue (bereinigte) Sitzung erstellt.

Windows Linux AIX Lose Kopplung zwischen MQTT-Clients und IBM MQ-Anwendungen

Der Fluss von Veröffentlichungen zwischen MQTT-Clients und IBM MQ-Anwendungen ist lose gekoppelt. Veröffentlichungen können entweder von einem MQTT-Client oder einer IBM MQ-Anwendung stammen und haben keine festgelegte Reihenfolge. Die Veröffentlichungskomponenten und Subskribenten sind lose verbunden. Sie interagieren indirekt über Veröffentlichungen und Subskriptionen. Sie können Nachrichten auch direkt von einer IBM MQ -Anwendung an einen MQTT -Client senden.

MQTT-Clients und IBM MQ-Anwendungen sind auf zwei Arten lose gekoppelt:

1. Veröffentlichungskomponenten und Subskribenten sind durch die Zuordnung einer Veröffentlichung und einer Subskription zu einem Thema lose verbunden. Veröffentlichungskomponenten und Subskribenten haben normalerweise keine Kenntnis von der Adresse oder Identität der anderen Quelle einer Veröffentlichung oder Subskription.
2. MQTT-Clients veröffentlichen, abonnieren, empfangen Veröffentlichungen und verarbeiten Empfangsbestätigungen in separaten Threads.

Eine MQTT-Clientanwendung wartet nicht, bis eine Veröffentlichung zugestellt wurde. Die Anwendung übergibt eine Nachricht an den MQTT-Client und setzt dann den eigenen Thread fort. Die Anwendung wird über ein Zustellungstoken mit der Zustellung einer Veröffentlichung synchronisiert (siehe Zustellungstokens).

Nach der Übergabe einer Nachricht an den MQTT-Client hat die Anwendung die Option, im Rahmen des Zustellungstokens zu warten. Statt zu warten, kann der Client eine Callback-Methode bereitstellen, die aufgerufen wird, wenn die Veröffentlichung an IBM MQ zugestellt wird. Das Zustellungstoken kann auch ignoriert werden.

Abhängig von der Servicequalität, die der Nachricht zugeordnet ist, wird das Zustellungstoken sofort oder möglicherweise auch nach einer langen Zeit an die Callback-Methode zurückgegeben. Das Zustellungstoken kann sogar zurückgegeben werden, nachdem die Verbindung des Clients getrennt und wiederhergestellt wurde. Bei der Servicequalität *fire and forget* wird das Zustellungstoken sofort zurückgegeben. In den anderen beiden Fällen wird das Zustellungstoken nur zurückgegeben, wenn der Client die Bestätigung erhält, dass die Veröffentlichung an Subskribenten gesendet wurde.

Veröffentlichungen, die als Folge einer Clientsubskription an einen MQTT-Client gesendet werden, werden an die Callback-Methode `messageArrived` übermittelt. `messageArrived` wird in einem anderen Thread als die Hauptanwendung ausgeführt.

Nachrichten direkt an einen MQTT-Client senden

Sie können eine Nachricht auf eine von zwei Arten an einen bestimmten MQTT-Client senden.

1. Eine IBM MQ-Anwendung kann eine Nachricht ohne Subskription direkt an einen MQTT-Client senden (siehe Nachricht direkt an einen Client senden).

2. Eine alternative Methode ist die Verwendung Ihrer `ClientIdentifier`-Namenskonvention. Veranlassen Sie, dass alle MQTT-Subskribenten Subskriptionen unter Verwendung ihres eindeutigen `ClientIdentifier`-Eintrags als Thema verwenden. Nehmen Sie Veröffentlichungen für *ClientIdentifier* vor. Die Veröffentlichung wird an den Client gesendet, der das Thema *ClientIdentifier* abonniert hat. Mit diesem Verfahren können Sie eine Veröffentlichung an einen bestimmten MQTT-Subskribenten senden.

Windows Linux AIX MQ Telemetry Sicherheit

Der Schutz von Telemetriegeräten kann wichtig sein, da die Geräte in der Regel tragbar sind und an Orten eingesetzt werden, die nicht genau kontrolliert werden können. Sie können die Verbindung zwischen dem MQTT-Gerät und dem Telemetrieservice (MQXR) mithilfe eines virtuellen privaten Netzes (VPN) schützen. MQ Telemetry bietet die beiden Sicherheitsmechanismen Transport Layer Security (TLS) und Java Authentication and Authorization Service (JAAS).

TLS wird in erster Linie zur Verschlüsselung der Kommunikation zwischen dem Gerät und dem Telemetriekanal und für die Authentifizierung der Geräteverbindung mit dem richtigen Server verwendet (siehe Telemetriekanalauthentifizierung mit TLS. Sie können mithilfe von TLS auch überprüfen, ob sich das Clientgerät mit dem Server verbinden darf (siehe MQTT-Clientauthentifizierung mit TLS).

Mit JAAS wird hauptsächlich überprüft, ob die Serveranwendung von dem Gerätebenutzer verwendet werden darf (siehe MQTT-Clientauthentifizierung mit einem Kennwort). JAAS kann in Verbindung mit LDAP verwendet werden, um ein Kennwort über ein Verzeichnis für einmalige Anmeldungen zu überprüfen.

Wenn TLS und JAAS kombiniert werden, wird die Authentifizierung über zwei Faktoren abgesichert. Sie können die von TLS verwendeten Chiffrierwerte auf Chiffrierwerte beschränken, welche die Federal Information Processing Standards erfüllen.

Mit Zehntausenden von Benutzern oder mehr ist die Bereitstellung einzelner Sicherheitsprofile nicht immer in der Praxis umsetzbar. Außerdem ist es häufig umständlich, die Profile für die Autorisierung einzelner Benutzer für den IBM MQ-Objekte zu verwenden. Gruppieren Sie stattdessen Benutzer in Klassen für die Autorisierung von Veröffentlichungen und Subskriptionen von Themen und für das Senden von Veröffentlichungen an Client.

Konfigurieren Sie jeden Telemetriekanal so, dass Clients zu allgemeinen Client-Benutzer-IDs zugeordnet werden. Verwenden Sie eine allgemeine Benutzer-ID für jeden Client, der sich auf einem bestimmten Kanal verbindet (siehe MQTT-Clientidentität und Autorisierung.

Die Autorisierung von Benutzergruppen umfasst keine Authentifizierung aller Einzelpersonen. Jeder einzelne Benutzer kann beim Client oder Server durch seinen `Benutzernamen` und das zugehörige `Kennwort` authentifiziert und dann mithilfe einer allgemeine Benutzer-ID beim Server autorisiert werden.

Windows Linux AIX MQ Telemetry-Globalisierung

Die Nachrichtennutzdaten im MQTT v3-Protokoll werden als Bytefeldgruppe codiert. Für gewöhnlich erstellen Anwendungen, die den Text verarbeiten, die Nachrichtennutzdaten in UTF-8. Der Telemetriekanal beschreibt die Nachrichtennutzdaten als UTF-8, nimmt jedoch keine Codepagekonvertierungen vor. Die Themenzeichenfolge der Veröffentlichung muss UTF-8 sein.

Die Anwendung ist für die Konvertierung alphabetischer Daten in die richtige Codepage und für die Konvertierung numerischer Daten in die richtige Zahlencodierung verantwortlich.

Der MQTT Java-Client verfügt über eine geeignete `MqttMessage.toString`-Methode. Die Methode behandelt die Nachrichtennutzdaten, als ob sie in dem Standardzeichensatz der lokalen Plattform codiert wären; dies ist für gewöhnlich UTF-8. Sie konvertiert die Nutzdaten in eine Java-Zeichenfolge (String). Java verfügt über die Zeichenfolgemethode `getBytes` zur Konvertierung einer Zeichenfolge in eine Bytefeldgruppe, die unter Verwendung des Standardzeichensatzes der lokalen Plattform codiert wird. Zwei MQTT Java-Programme, die Text in den Nachrichtennutzdaten zwischen Plattformen mit demselben Standardzeichensatz austauschen, können hierbei auf einfache und effiziente Weise UTF-8 nutzen.

Wenn eine der Plattformen nicht den Standardzeichensatz UTF-8 hat, müssen die Anwendungen eine Konvention für den Austausch von Nachrichten erstellen. Die Veröffentlichungskomponente gibt beispielsweise eine Konvertierung aus einer Zeichenfolge in UTF-8 an, wobei die Methode `getBytes("UTF8")` verwendet wird. Zum Empfang des Nachrichtentextes geht der Subskribent davon aus, dass die Nachricht im Zeichensatz UTF-8 codiert ist.

Der Telemetrieservice (MQXR) beschreibt die Codierung aller eingehenden Veröffentlichungen von MQTT-Client als UTF-8. Er legt `MQMD.CodedCharSetId` zu UTF-8und `RFH2.CodedCharSetId` zu `MQCCSI_INHERIT` ; Siehe „Integration von MQ Telemetry mit Warteschlangenmanagern” auf Seite 141. Da das Format der Veröffentlichung auf `MQFMT_NONE` gesetzt ist, kann keine Konvertierung durch Kanäle oder `MQGET` erfolgen.

Windows Linux AIX Leistung und Skalierbarkeit von MQ Telemetry

Beachten Sie die folgenden Faktoren bei der Verwaltung einer großen Anzahl an Clients und der Verbesserung der Skalierbarkeit von MQ Telemetry.

Kapazitätsplanung

Informationen zu Leistungsberichten für MQ Telemetry finden Sie unter MQ Performance documents.

Verbindungen

Verbindungen bringen unter anderem folgenden Aufwand mit sich:

- Den Einrichtungsaufwand für die Verbindung selbst im Hinblick auf die Prozessorbelegung und die Dauer
- Den Netzaufwand
- Die Speicherbelegung, wenn eine Verbindung offen gehalten, jedoch nicht verwendet wird.

Wenn Clients verbunden bleiben, besteht eine zusätzliche Belastung. Wird eine Verbindung offen gehalten, verwenden TCP/IP-Verarbeitungsabläufe und MQTT-Nachrichten das Netz, um zu prüfen, ob die Verbindung immer noch besteht. Außerdem wird für jede Clientverbindung, die offen gehalten wird, Serverspeicher belegt.

Wenn Sie mehr als eine Nachricht pro Minute senden, halten Sie Ihre Verbindung offen, um den Aufwand für den Aufbau einer neuen Verbindung zu vermeiden. Wenn Sie weniger als eine alle 10 bis 15 Minuten senden, sollten Sie erwägen, die Verbindung zu trennen, um den Aufwand zu vermeiden, der entsteht, wenn sie offen gehalten wird. Es kann sinnvoll sein, über längere Zeiträume eine TLS-Verbindung offen, aber inaktiv zu halten, da ihr Aufbau sehr aufwändig ist.

Berücksichtigen Sie außerdem die Funktionalität des Clients. Wenn der Client über die Funktion eines Store-and-forward-Verfahrens verfügt, kann es hilfreich sein, Nachrichten zu stapeln und die Verbindung zwischen dem Versenden der Stapel zu trennen. Wird die Verbindung des Clients jedoch getrennt, kann er keine Nachrichten vom Server empfangen. Der Zweck Ihrer Anwendung beeinflusst daher Ihre Entscheidung.

Wenn Ihr System über einen Client verfügt, der viele Nachrichten sendet (beispielsweise bei Dateiübertragungen), warten Sie nicht bei jeder Nachricht auf eine Serverantwort. Senden Sie stattdessen alle Nachrichten und prüfen Sie am Ende, ob alle empfangen wurden. Sie können auch die Methode der Servicequalität (Quality of Service, QoS) nutzen.

Sie können die Servicequalität je nach Nachricht variieren, indem unwichtige Nachrichten mit der Servicequalität 0 und wichtige Nachrichten mit der Servicequalität 2 übermittelt werden. Der Nachrichtendurchsatz kann bei der Servicequalität 0 etwa doppelt so hoch sein wie bei der Servicequalität 2.

Namenskonventionen

Wenn Sie Ihre Anwendung für viele Clients entwerfen, implementieren Sie eine effektive Namenskonvention. Damit jedem Client die richtige `Client-ID` zugeordnet wird, verwenden Sie eine aussagekräftige

`Client-ID`. Durch eine gute Namenskonvention lässt sich für den Administrator leichter feststellen, welche Clients gerade ausgeführt werden. Eine Namenskonvention ermöglicht dem Administrator das Filtern einer langen Liste mit Clients in IBM MQ Explorer und vereinfacht die Problembestimmung (siehe Client-ID).

Durchsatz

Die Länge der Themennamen wirkt sich auf die Anzahl der Bytes aus, die über das Netz fließen. Beim Veröffentlichen oder Abonnieren kann die Anzahl der Bytes in einer Nachricht eine wichtige Rolle spielen. Begrenzen Sie daher die Anzahl der Zeichen in einem Themennamen. Wenn ein MQTT-Client ein Thema abonniert, gibt ihm IBM MQ einen Namen in folgendem Format:

```
ClientIdentifier: TopicName
```

Mit dem IBM MQ MQSC **DISPLAY** -Befehl können Sie alle Subskriptionen für einen MQTT -Client anzeigen:

```
DISPLAY SUB(' ClientID1:*')
```

Ressourcen in IBM MQ zur Verwendung durch MQTT-Clients definieren

Ein MQTT-Client stellt eine Verbindung zu einem fernen IBM MQ-Warteschlangenmanager her. Es gibt zwei grundlegende Methoden, mit denen eine IBM MQ -Anwendung Nachrichten an einen MQTT -Client senden kann: Setzen Sie die Standardübertragungswarteschlange auf `SYSTEM.MQTT.TRANSMIT.QUEUE` oder verwenden Sie Warteschlangenmanager-Aliasnamen. Definieren Sie bei sehr vielen MQTT-Clients die Standardübertragungswarteschlange eines Warteschlangenmanagers. Durch die Einstellung einer Standardübertragungswarteschlange verringert sich der Verwaltungsaufwand (siehe Verteilte Steuerung von Warteschlangen zum Senden von Nachrichten an MQTT-Clients konfigurieren).

Skalierbarkeit durch das Vermeiden von Subskriptionen verbessern

Wenn ein MQTT V3-Client ein Thema abonniert, wird vom Telemetrieservice (MQXR) in IBM MQ eine Subskription erstellt. Die Subskription leitet Veröffentlichungen für den Client an die Warteschlange `SYSTEM.MQTT.TRANSMIT.QUEUE` weiter. Der Name des fernen Warteschlangenmanagers im Übertragungsheader jeder Veröffentlichung wird auf die `Client-ID` des MQTT-Clients gesetzt, der die Subskription erstellt hat. Wenn viele Clients vorhanden sind, die alle ihre eigenen Subskriptionen erstellen, führt dies dazu, dass viele Proxy-Subskriptionen im gesamten Publish/Subscribe-Cluster oder in der Hierarchie von IBM MQ verwaltet werden müssen. Im Abschnitt Nachricht direkt an einen Client senden finden Sie Informationen darüber, wie Sie anstelle von Publish/Subscribe eine Punkt-zu-Punkt-basierte Lösung verwenden können.

Sehr viele Clients verwalten

Wenn Sie viele gleichzeitig verbundene Clients unterstützen müssen, erhöhen Sie den Speicher, der für den Telemetrieservice (MQXR) zur Verfügung steht. Legen Sie hierfür die JVM-Parameter **-Xms** und **-Xmx** fest. Führen Sie folgende Schritte aus:

1. Suchen Sie die Datei `java.properties` im Konfigurationsverzeichnis des Telemetrieservice (siehe Konfigurationsverzeichnis des Telemetrieservice (MQXR) unter Windows bzw. Konfigurationsverzeichnis des Telemetrieservice unter Linux).
2. Folgen Sie den Anweisungen in der Datei; ein Heapspeicher von 1 GB reicht für 50.000 gleichzeitig verbundene Clients aus.

   ```
   # Heap sizing options - uncomment the following lines to set the heap to 1G
   #-Xmx1024m
   #-Xms1024m
   ```

3. Fügen Sie in der Datei `java.properties` weitere Befehlszeilenargumente für die Übergabe an die Java Virtual Machine, auf der der Telemetrieservice (MQXR) ausgeführt wird, hinzu (siehe JVM-Parameter an den Telemetrieservice (MQXR) übergeben).

Wenn Sie die Anzahl der offenen Dateideskriptoren unter Linux erhöhen möchten, fügen Sie `/etc/security/limits.conf/` folgende Zeilen hinzu und melden Sie sich wieder an.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

Für jeden Socket ist ein Dateideskriptor erforderlich. Da der Telemetrieservice einige zusätzliche Dateideskriptoren benötigt, muss diese Zahl größer als die Anzahl der erforderlichen offenen Sockets sein.

Der Warteschlangenmanager verwendet für jede nicht dauerhafte Subskription eine Objektkennung. Wenn Sie viele aktive, nicht dauerhafte Subskriptionen unterstützen möchten, erhöhen Sie die maximale Anzahl aktiver Kennungen im Warteschlangenmanager. Beispiel:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

Abbildung 48. Maximale Anzahl an Kennungen unter Windows ändern

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Abbildung 49. Maximale Anzahl an Kennungen unter Linux ändern

Weitere Überlegungen

Berücksichtigen Sie bei der Planung Ihrer Systemvoraussetzungen den Zeitaufwand, der für den Neustart des Systems benötigt wird. Die geplante Nichtverfügbarkeit kann sich auf die Anzahl gestauter Nachrichten auswirken, die zur Verarbeitung anstehen. Konfigurieren Sie das System so, dass die Nachrichten in einer angemessenen Zeit erfolgreich verarbeitet werden können. Überprüfen Sie den Plattenspeicher, Hauptspeicher und die Verarbeitungskapazität. Bei einigen Clientanwendungen können bei einer Wiederherstellung der Clientverbindung möglicherweise Nachrichten gelöscht werden. Wenn Nachrichten gelöscht werden sollen, legen Sie `CleanSession` in den Parametern der Clientverbindung fest (siehe Sitzungen bereinigen). Alternativ können Sie die Veröffentlichung und Subskription mit der bestmöglichen Servicequalität (`0`) in einem MQTT -Client durchführen (siehe Servicequalität). Verwenden Sie `nicht persistente` Nachrichten, wenn Sie Nachrichten von IBM MQsenden. Nachrichten mit diesen Servicequalitäten werden bei einem Neustart des Systems oder Verbindung nicht wiederhergestellt.

Windows Linux AIX Von MQ Telemetry unterstützte Geräte

MQTT-Clients können auf einer ganzen Reihe von Geräten ausgeführt werden, von Sensoren über Aktuatoren bis hin zu Handbediengeräten und Fahrzeugsystemen.

MQTT-Clients sind klein und werden auf Geräten ausgeführt, die durch wenig Speicherplatz und niedrige Verarbeitungskapazität eingeschränkt sind. Das MQTT protocol ist zuverlässig und verfügt über kompakte Header, wodurch es sich für Netze eignet, die durch eine geringe Bandbreite, hohen Aufwand und nicht unterbrechungsfreie Verfügbarkeit eingeschränkt sind.

MQ Telemetry kommuniziert über MQTT-Clientanwendungen mit Telemetriegeräten. Diese Anwendungen verwenden die folgenden Ressourcen, die alle das MQTT v3-Protokoll implementieren:

- Die folgenden Clientbibliotheken:
  - *MQTT client for Java* für die Erstellung nativer Anwendungen für beispielsweise Android-, OS X-, Linux- oder Windows-Geräte. Anwendungen, die diese Clientbibliothek verwenden, können alle Varianten von Java von der kleinsten CLDC (Connected Limited Device Configuration)/MIDP (Mobile Information Device Profile) über CDC (Connected Device Configuration)/Foundation, J2SE Java Platform, Standard Edition) und J2EE Java Platform, Enterprise Edition) ausführen. Die angepasste jclRM-

Klassenbibliothek von IBM wird ebenfalls unterstützt. Die Java ME-Plattform wird im Allgemeinen auf kompakten Endgeräten wie Aktuatoren, Sensoren, Mobiltelefonen und sonstigen integrierten Einheiten verwendet. Die Java SE-Plattform ist im Allgemeinen auf integrierten High-End-Einheiten wie Desktop-Computern und Servern installiert.

- *MQTT client for C* für die Erstellung nativer Anwendungen für beispielsweise iOS-, OS X-, Linux- oder Windows-Geräte. Diese Clientbibliothek stellt eine Referenzimplementierung in der Programmiersprache C in Kombination mit dem vordefinierten nativen Client für Windows- und Linux-Systeme bereit. Die Referenzimplementierung in der Programmiersprache C ermöglicht die Portierung von MQTT auf eine breite Palette an Geräten und Plattformen. Einige Windows-Systeme auf Intel-Plattformen, einschließlich Windows 7, RedHat, Ubuntu, und einige Linux-Systeme auf ARM-Plattformen wie Eurotech Viper implementieren Versionen von Linux, die den C-Client ausführen, aber IBM stellt für die Plattformen keine Serviceunterstützung bereit. Sie müssen Probleme im Zusammenhang mit dem Client auf einer unterstützten Plattform reproduzieren, wenn Sie sich an Ihr IBM Support Center wenden möchten.
- *MQTT client for Java* für die Erstellung browserbasierter Webanwendungen.

MQTT-Clientbibliotheken sind bei Eclipse Paho und MQTT.org kostenlos erhältlich. Weitere Informationen finden Sie im Abschnitt IBM MQ Telemetry Transport-Beispielprogramme.

Sicherheit in IBM MQ

In IBM MQ gibt es mehrere Möglichkeiten, für Sicherheit zu sorgen: die Berechtigungsserviceschnittstelle, Kanalexits (benutzerdefiniert oder von einem anderen Hersteller), Kanalsicherheit mit Transport Layer Security (TLS), Kanalauthentifizierungsdatensätze und Nachrichtensicherheit.

Schnittstelle für Berechtigungsservice

Die Berechtigung für die Verwendung von MQI-Aufrufen und Befehlen und den Zugriff auf Objekte wird vom **Objektberechtigungsmanager** erteilt, der standardmäßig aktiviert ist. Der Zugriff auf IBM MQ-Entitäten wird über IBM MQ-Benutzergruppen und den Objektberechtigungsmanager gesteuert. Administratoren können Berechtigungen über eine Befehlszeilenschnittstelle nach Bedarf erteilen oder entziehen.

Weitere Informationen zum Erstellen von Berechtigungsservicekomponenten finden Sie unter Sicherheit auf AIX, Linux, and Windows-Systemen einrichten.

Benutzerdefinierte Kanalexits und Kanalexits anderer Anbieter

Kanäle können benutzerdefinierte Kanalexits oder Kanalexits anderer Anbieter verwenden. Weitere Informationen finden Sie im Abschnitt Kanalexitprogramme für Messaging-Kanäle.

Kanalsicherheit mit TLS

Das TLS-Protokoll (Transport Layer Security) sorgt für standardisierte Kanalsicherheit, indem es Schutz vor Abhören, Manipulation und Identitätsvortäuschung bietet.

Bei TLS werden der Nachrichtenschutz sowie Integrität und gegenseitige Authentifizierung mithilfe von öffentlichen Schlüsseln und symmetrischer Verschlüsselung implementiert.

Umfassende Informationen zur Sicherheit in IBM MQ, darunter auch ausführliche Hinweise zu TLS, finden Sie im Abschnitt Sicherheit. Eine Übersicht über TLS mit Hinweisen zu den in diesem Abschnitt beschriebenen Befehlen finden Sie im Abschnitt Kryptografische Sicherheitsprotokolle: TLS.

Kanalauthentifizierungsdatensätze

Mit Kanalauthentifizierungsdatensätzen können Sie den Zugriff auf Systeme, die eine Verbindung herstellen können, auf Kanalebene steuern. Weitere Informationen finden Sie im Abschnitt Kanalauthentifizierungsdatensätze.

Nachrichtensicherheit

Verwenden Sie Advanced Message Security eine gesondert installierte und lizenzierte Komponente von IBM MQ, um mit IBM MQ gesendete und empfangene Nachrichten mit einem kryptografischen Schutz zu versehen. Weitere Informationen finden Sie in Advanced Message Security.

Zugehörige Tasks

Sicherung

Sicherheitsanforderungen planen

TLS-Unterstützung für verwalteten IBM MQ.NET-Client

Der vollständig verwaltete IBM MQ.NET-Client bietet TLS-Unterstützung (Transport Layer Security) basierend auf dem Kit 'Microsoft.NET SSLStreams'. Dies unterscheidet sich von den anderen IBM MQ -Clients, die auf IBM Global Security Kit (GSKit)basieren.

Sie können IBM MQ.NET-Anwendungen für die Ausführung im verwalteten und im nicht verwalteten Modus entwickeln.

- Im verwalteten Modus agieren .NET-Anwendungen in der .NET-CLR (Common Language Runtime) ohne plattformübergreifenden Aufruf wie beispielsweise den Aufruf des Message Queue Interface in der Programmiersprache C.
- Im nicht verwalteten Modus wird das Message Queue Interface (MQI) in der Programmiersprache C für die zugrunde liegenden MQI-Operationen aufgerufen. Im Wesentlichen umfasst die Schnittstelle im nicht verwalteten Modus die .NET-Wrapperklassen, die auf dem Message Queue Interface in der Programmiersprache C aufsetzen.

Der verwaltete IBM MQ.NET-Client verwendet die Microsoft.NET-Frameworkbibliotheken für die Implementierung von TLS-Secure-Socket-Protokollen. Die Klasse 'System.NET.Security.SSLStream' von Microsoft wird für die Implementierung der Sicherheit (TLS) in IBM MQ.NET verwendet.

Der nicht verwaltete IBM MQ.NET -Clientmodus unterstützt bereits die TLS-Funktion, die auf C MQI (und GSKit) basiert. TLS-Operationen werden also vom Message Queue Interface in der Programmiersprache C verarbeitet. In diesem Fall implementiert GSKit die sicheren TLS-Socketprotokolle.

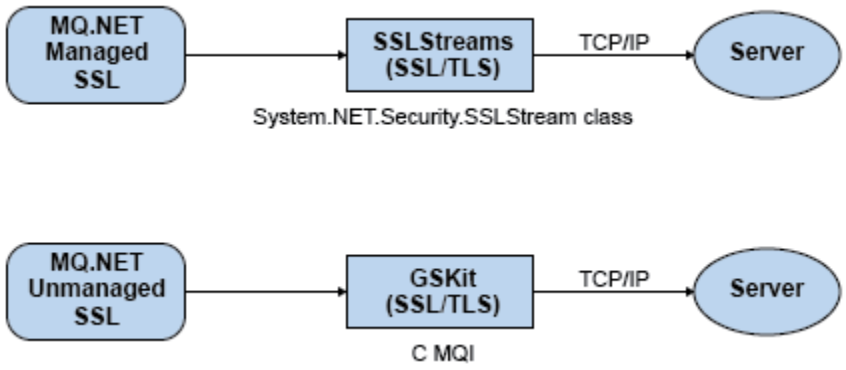


Abbildung 50. Vergleich von IBM MQ.NET-TLS im verwalteten und nicht verwalteten Modus

Die folgende Tabelle enthält eine Zusammenfassung der Unterschiede zwischen den verwalteten und nicht verwalteten Implementierungen:

Tabelle 14. Unterschiede zwischen verwalteten und nicht verwalteten Implementierungen			
Modus	**Protokolle**	**Implementierung**	**Kommentare**
IBM MQ.NET SSL verwaltet	TLS	Klasse System.NET.Security.SSLStream Die SSLStream-Klasse agiert als Datenstrom über ein verbundenes TCP-Socket.	TLS 1.0 TLS 1.2 (nur mit Microsoft.NET Framework v4.5)
IBM MQ.NET SSL nicht verwaltet	TLS	GSKit und C-MQI	Sichere TLS-Socketprotokolle

Zugehörige Konzepte

Unterstützung von Secure Sockets Layer (SSL) und Transport Layer Security (TLS) für .NET

IBM MQ MQI clients

Ein IBM MQ MQI client ist eine Komponente des IBM MQ-Produkts, die auf einem System installiert werden kann, auf dem kein Warteschlangenmanager aktiv ist.

Ein IBM MQ MQI-*Client* ist eine Komponente, die es einer auf einem System aktiven Anwendung ermöglicht, MQI-Aufrufe an Warteschlangenmanager auf einem anderen System abzusetzen. Die Ausgabe eines Aufrufs wird an den Client zurückgesendet, der sie an die Anwendung weitergibt.

Mit einem IBM MQ MQI client kann eine Anwendung, die auf demselben System wie der Client aktiv ist, eine Verbindung zu einem Warteschlangenmanager auf einem anderen System herstellen. Dadurch kann die Anwendung MQI-Aufrufe an diesen Warteschlangenmanager absetzen. Diese Anwendungen werden als IBM MQ MQI clientanwendungen, der Warteschlangenmanager als *Server-Warteschlangenmanager* bezeichnet.

Ein IBM MQ-*Server* ist ein Warteschlangenmanager, der für einen oder mehrere Clients Services zur Steuerung von Warteschlangen bereitstellt. Alle IBM MQ-Objekte (wie beispielsweise Warteschlangen) sind nur auf der Warteschlangenmanager-Maschine (IBM MQ-Servermaschine) vorhanden, nicht auf dem Client. Ein IBM MQ-Server kann auch lokale IBM MQ-Anwendungen unterstützen.

Ein IBM MQ-Server unterscheidet sich von einem normalen Warteschlangenmanager darin, dass zwischen dem Server und den einzelnen Clients dedizierte Kommunikationsverbindungen bestehen. Weitere Informationen zur Erstellung von Kanälen für Clients und Server finden Sie im Abschnitt Verteilte Steuerung von Warteschlangen konfigurieren.

IBM MQ MQI clientanwendungen und Server-Warteschlangenmanager kommunizieren über *MQI-Kanäle* miteinander. Ein MQI-Kanal wird gestartet, wenn die Clientanwendung einen **MQCONN**- oder **MQCONNX**-Aufruf absetzt, um eine Verbindung zum Warteschlangenmanager herzustellen, und endet, wenn die Clientanwendung einen **MQDISC**-Aufruf ausgibt, um die Verbindung zum Warteschlangenmanager zu beenden. Die Eingabeparameter eines MQI-Aufrufs werden im MQI-Kanal in eine Richtung, die Ausgabeparameter in die entgegengesetzte Richtung übertragen.

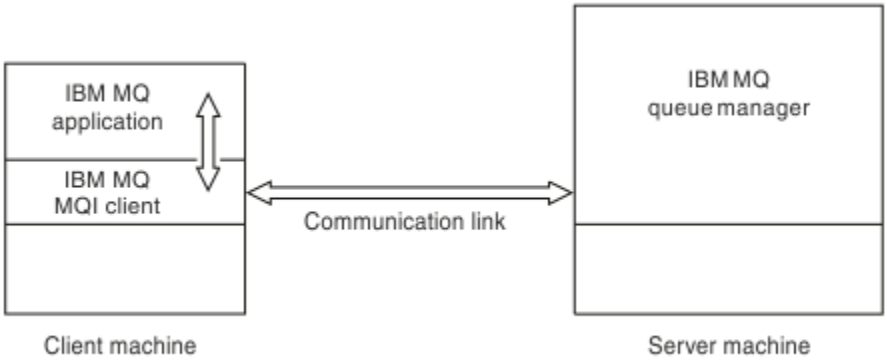


Abbildung 51. Verbindung zwischen Client und Server

Folgende Plattformen können eingesetzt werden. Die zulässigen Kombinationen hängen von dem von Ihnen verwendeten IBM MQ-Produkt ab und sind im Abschnitt „Plattformunterstützung für IBM MQ-Clients" auf Seite 154 beschrieben.

IBM MQ MQI client	**IBM MQ-Server**
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

Die MQI ist für alle auf der Clientplattform aktiven Anwendungen verfügbar; die Warteschlangen und andere IBM MQ-Objekte befinden sich in einem von Ihnen auf einem Server installierten Warteschlangenmanager.

Eine Anwendung, die in der IBM MQ MQI clientumgebung eingesetzt werden soll, muss zunächst mit der entsprechenden Clientbibliothek verbunden werden. Wenn die Anwendung einen MQI-Aufruf ausgibt, leitet der IBM MQ MQI client die Anforderung an einen Warteschlangenmanager weiter, von dem sie verarbeitet und eine Antwort an den IBM MQ MQI client zurückgegeben wird.

Die Verbindung zwischen Anwendung und IBM MQ MQI client wird zur Ausführungszeit dynamisch hergestellt.

Sie können Clientanwendungen auch mithilfe von IBM MQ classes for .NET, IBM MQ classes for Java oder IBM MQ classes for Java Message Service (JMS) entwickeln. Sie können Java- und JMS-Clients auf folgenden Plattformen verwenden:

- IBM i
- AIX
- Linux
- Windows

Auf die Verwendung von Java und JMS wird hier nicht näher eingegangen. Vollständige Informationen zur Installation, Konfiguration und Verwendung von IBM MQ classes for Java und IBM MQ classes for JMS finden Sie in den Abschnitten IBM MQ classes for Java verwenden und IBM MQ classes for JMS verwenden.

IBM MQ-Anwendungen in einer Client/Server-Umgebung

Wenn eine Verbindung zu einem Server besteht, können IBM MQ-Clientanwendungen die meisten MQI-Aufrufe auf dieselbe Weise wie lokale Anwendungen ausgeben. Die Clientanwendung gibt einen MQCONN-

Aufruf aus, um eine Verbindung zu einem angegebenen Warteschlangenmanager herzustellen. Alle weiteren MQI-Aufrufe, in denen die von der Verbindungsanforderung zurückgegebene Verbindungskennung angegeben ist, werden von diesem Warteschlangenmanager verarbeitet.

Sie müssen die Anwendungen mit den entsprechenden Clientbibliotheken verbinden. Weitere Informationen finden Sie unter Anwendungen für IBM MQ MQI clients erstellen.

Zugehörige Konzepte

„Verwendung von IBM MQ-Clients" auf Seite 154
Mit IBM MQ-Clients können IBM MQ-Nachrichtenübertragung und -Warteschlangensteuerung auf effiziente Weise implementiert werden.

„Was ist ein erweiterter Transaktionsclient?" auf Seite 156
Ein erweiterter transaktionsorientierter IBM MQ-Client kann unter der Steuerung eines externen Transaktionsmanagers Ressourcen aktualisieren, die von einem anderen Ressourcenmanager verwaltet werden.

„Verbindung vom Client zum Server herstellen" auf Seite 157
Clients stellen über MQCONN- oder MQCONNX-Aufrufe eine Verbindung zum Server her; die Kommunikation erfolgt über einen Kanal.

„Transaktionsmanagement und -unterstützung" auf Seite 158
Dieser Abschnitt gibt eine Einführung in das Transaktionsmanagement und es wird erläutert, wie Transaktionen in IBM MQ unterstützt werden.

„Funktionen des Warteschlangenmanagers erweitern" auf Seite 160
Sie können die Funktionen eines Warteschlangenmanagers durch Benutzerexits, API-Exits oder installierbare Services erweitern.

Zugehörige Informationen

IBM MQ MQI client einrichten

Verwendung von IBM MQ-Clients

Mit IBM MQ-Clients können IBM MQ-Nachrichtenübertragung und -Warteschlangensteuerung auf effiziente Weise implementiert werden.

Sie können eine Anwendung einsetzen, die eine MQI und einen Warteschlangenmanager verwendet, die sich beide (physisch oder virtuell) auf unterschiedlichen Maschinen befinden. Hier die Vorteile eines solchen Szenarios:

- Es ist keine vollständige IBM MQ-Implementierung auf der Clientmaschine erforderlich.
- Die Hardwarevoraussetzungen für das Clientsystem werden reduziert.
- Die Anforderungen bezüglich der Systemverwaltung werden reduziert.
- Eine auf einem Client aktive IBM MQ-Anwendung kann eine Verbindung zu mehreren Warteschlangenmanagern auf unterschiedlichen Systemen herstellen.
- Es können alternative Kanäle mit unterschiedlichen Übertragungsprotokollen verwendet werden.

Plattformunterstützung für IBM MQ-Clients

IBM MQ auf allen unterstützten Serverplattformen akzeptiert Clientverbindungen von IBM MQ MQI clients auf einer Reihe von Plattformen.

IBM MQ, das als *Basisprodukt und Server* auf allen unterstützten Serverplattformen installiert ist, akzeptiert Verbindungen von IBM MQ MQI clients auf folgenden Plattformen:

- 

IBM i
- 

AIX
- 

Linux
- 

Windows

CCSID (ID des codierten Zeichensatzes) und Kommunikationsprotokoll der Clientverbindungen hängen von der jeweiligen Plattform ab.

Auf einem IBM MQ MQI client ausführbare Anwendungen

In der Clientumgebung wird die MQI vollständig unterstützt. Damit können nahezu alle IBM MQ-Anwendungen für die Ausführung auf IBM MQ MQI clientsystemen konfiguriert werden, indem die Anwendung auf dem IBM MQ MQI client nicht mit der MQI-, sondern mit der MQIC-Bibliothek verbunden wird. Die Ausnahmen sind:

- MQGET mit Signal.
- Anwendungen, bei denen eine Synchronisationspunktkoordination mit anderen Ressourcenmanagern erforderlich ist, benötigen einen erweiterten transaktionsorientierten Client.

Wenn Vorauslesen aktiviert ist, um den Durchsatz beim nicht permanenten Messaging zu optimieren, stehen nicht alle MQGET-Optionen zur Verfügung. In der folgenden Tabelle sind alle zulässigen Optionen aufgeführt; außerdem ist angegeben, ob sie zwischen MQGET-Aufrufen geändert werden können.

Tabelle 15. Zulässige MQGET-Optionen bei aktiviertem Vorauslesen

Werte	**Bei aktiviertem Vorauslesen zulässig und kann zwischen MQGET-Aufrufen ausgetauscht werden**	**Bei aktiviertem Vorauslesen zulässig, kann aber nicht zwischen MQGET-Aufrufen ausgetauscht werden[1]**	**MQGET-Optionen, die bei aktiviertem Vorauslesen nicht zulässig sind[2]**
MQGET MD-Werte	MsgId[3] CorrelId [3]	Encoding CodedCharSetId	
MQGET MQGMO-Optionen	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-Werte		MsgHandle	

1. Wenn diese Optionen zwischen MQGET-Aufrufen ausgetauscht werden, wird der Ursachencode MQRC_OPTIONS_CHANGED zurückgegeben.
2. Wenn diese Optionen im ersten MQGET-Aufruf angegeben werden, wird das Vorauslesen inaktiviert. Werden diese Optionen in einem nachfolgenden MQGET-Aufruf angegeben, wird Ursachencode MQRC_OPTIONS_ERROR zurückgegeben.
3. Clientanwendungen müssen erkennen können, dass bei einer Änderung der Werte für 'MsgId' und 'CorrelId' zwischen MQGET-Aufrufen Nachrichten mit den früheren Werten möglicherweise bereits an den Client gesendet wurden und im Vorauslesepuffer des Clients verbleiben, bis sie verarbeitet (oder automatisch gelöscht) werden.

4. Der erste MQGET-Aufruf bestimmt, ob Nachrichten aus einer Warteschlange angezeigt oder abgerufen werden, wenn Vorauslesen aktiviert ist. Wenn die Anwendung versucht, Anzeige und Abruf zu kombinieren, wird Ursachencode MQRC_OPTIONS_CHANGED zurückgegeben.
5. MQGMO_MSG_UNDER_CURSOR ist bei aktiviertem Vorauslesen nicht möglich. Nachrichten können angezeigt oder abgerufen werden, wenn Vorauslesen aktiviert ist, aber eine Kombination der beiden Funktionen ist nicht möglich.

Eine auf einem IBM MQ MQI client aktive Anwendung kann parallel Verbindungen zu mehreren Warteschlangenmanagern herstellen oder in einem MQCONN- oder MQCONNX-Aufruf einen Warteschlangenmanagernamen mit einem Stern (*) verwenden (siehe Beispiele im Abschnitt IBM MQ MQI client-Anwendungen mit Warteschlangenmanagern verbinden).

Was ist ein erweiterter Transaktionsclient?

Ein erweiterter transaktionsorientierter IBM MQ-Client kann unter der Steuerung eines externen Transaktionsmanagers Ressourcen aktualisieren, die von einem anderen Ressourcenmanager verwaltet werden.

Wenn Sie nicht mit den Grundlagen des Transaktionsmanagements vertraut sind, lesen Sie den Abschnitt „Transaktionsmanagement und -unterstützung” auf Seite 158.

Der transaktionsorientierte XA-Client wird jetzt mit IBM MQ bereitgestellt.

Eine Clientanwendung kann an einer Arbeitseinheit beteiligt sein, die von einem Warteschlangenmanager verwaltet wird, mit dem sie verbunden ist. Innerhalb der Arbeitseinheit kann die Clientanwendung Nachrichten in die Warteschlangen dieses Warteschlangenmanagers einreihen bzw. Nachrichten aus diesen Warteschlangen abrufen. Anschließend kann die Clientanwendung die Arbeitseinheit mit dem **MQCMIT**-Aufruf festschreiben oder mit dem **MQBACK**-Aufruf zurücksetzen. Die Clientanwendung kann allerdings nicht innerhalb einer Arbeitseinheit die Ressourcen (z. B. die Tabellen einer Db2-Datenbank) eines anderen Ressourcenmanagers aktualisieren. Bei Verwendung eines erweiterten transaktionsorientierten IBM MQ-Clients besteht diese Einschränkung nicht.

Ein erweiterter transaktionsorientierter IBM MQ-Client ist ein IBM MQ MQI client mit zusätzlichen Funktionen. Mithilfe dieser Funktionen kann eine Clientanwendung innerhalb derselben Arbeitseinheit Folgendes ausführen:

- Nachrichten in Warteschlangen einreihen, die dem Warteschlangenmanager zugeordnet sind, mit dem die Anwendung verbunden ist, bzw. Nachrichten aus diesem Warteschlangenmanager abrufen.
- Die Ressourcen eines Ressourcenmanagers, bei dem es sich nicht um einen IBM MQ-Warteschlangenmanager handelt, verwalten.

Diese Arbeitseinheit muss von einem externen Transaktionsmanager verwaltet werden, der auf demselben System wie die Clientanwendung aktiv ist. Die Arbeitseinheit darf nicht von dem Warteschlangenmanager verwaltet werden, mit dem die Clientanwendung verbunden ist. Das heißt, dass der Warteschlangenmanager nur als Ressourcenmanager, nicht als Transaktionsmanager dienen kann. Es heißt auch, dass die Clientanwendung die Arbeitseinheit nur über die Anwendungsprogrammierschnittstelle (API) des externen Transaktionsmanagers festschreiben bzw. zurücksetzen kann. Die Clientanwendung kann daher nicht die MQI-Aufrufe (**MQBEGIN**, **MQCMIT** und **MQBACK**) verwenden.

Der externe Transaktionsmanager kommuniziert mit dem Warteschlangenmanager als Ressourcenmanager über denselben MQI-Kanal, der auch von der Clientanwendung verwendet wird, die mit dem Warteschlangenmanager verbunden ist. Bei einer Wiederherstellung im Anschluss an einen Ausfall, wenn keine Anwendungen aktiv sind, kann der Transaktionsmanager jedoch über einen dedizierten MQI-Kanal alle unvollständigen Arbeitseinheiten wiederherstellen, an denen der Warteschlangenmanager zum Zeitpunkt des Ausfalls beteiligt war.

In diesem Abschnitt werden IBM MQ MQI clients ohne die erweiterte transaktionsorientierte Funktion als IBM MQ-Basisclients bezeichnet. Ein erweiterter transaktionsorientierter IBM MQ-Client ist also ein IBM MQ-Basisclient mit erweiterter transaktionsorientierter Funktion.

Anmerkung: IBM i Von IBM MQ MQI client on IBM i wird die erweiterte transaktionsorientierte IBM MQ-Funktion nicht unterstützt.

Plattformunterstützung für erweiterte Transaktionsclients

Multi

Erweiterte transaktionsorientierte Clients sind für alle Multiplatforms verfügbar, die einen Basisclient unterstützen. Die Clients sind nicht für z/OS verfügbar.

Eine Clientanwendung, die einen erweiterten transaktionsorientierten Client verwendet, kann nur eine Verbindung zu einem Warteschlangenmanager der folgenden IBM MQ 9.0oder höher herstellen:

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ fürLinux
- Windows IBM MQ for Windows

z/OS Zwar gibt es keine erweiterten transaktionsorientierten Clients, die unter z/OS ausgeführt werden, jedoch können Clientanwendungen, die einen erweiterten transaktionsorientierten Client verwenden, eine Verbindung zu einem Warteschlangenmanager herstellen, der unter z/OS läuft.

Auf allen Plattformen gelten für erweiterte transaktionsorientierte Clients dieselben Hardware- und Softwarevoraussetzungen wie für den IBM MQ-Basisclient. Programmiersprachen, die vom IBM MQ-Basisclient und dem von Ihnen verwendeten Transaktionsmanager unterstützt werden, werden auch von einem erweiterten transaktionsorientierten Client unterstützt.

Informationen zu den externen Transaktionsmanagern für alle Plattformen finden Sie unter Systemvoraussetzungen für IBM MQ.

Verbindung vom Client zum Server herstellen

Clients stellen über MQCONN- oder MQCONNX-Aufrufe eine Verbindung zum Server her; die Kommunikation erfolgt über einen Kanal.

Eine in der IBM MQ-Clientumgebung aktive Anwendung muss eine aktive Verbindung zwischen Client- und Servermaschinen aufrechterhalten.

Die Verbindung wird über einen MQCONN- oder MQCONNX-Aufruf hergestellt, den die Anwendung absetzt. Clients und Server kommunizieren über *MQI-Kanäle* miteinander oder (bei Verwendung gemeinsam genutzter Verbindungen) über Verbindungen, bei denen eine MQI-Kanalinstanz gemeinsam genutzt wird. Ist der Aufruf erfolgreich, wird die MQI-Kanalinstanz oder Verbindung aufrechterhalten, bis die Anwendung einen MQDISC-Aufruf absetzt. Dies gilt für jeden Warteschlangenmanager, zu dem eine Anwendung eine Verbindung herstellen muss.

Client und Warteschlangenmanager auf einem System

Anwendungen können auch in der IBM MQ MQI client-Umgebung ausgeführt werden, wenn auf demselben System zusätzlich noch ein Warteschlangenmanager installiert ist.

In diesem Fall können Sie eine Verbindung zu den Warteschlangenmanagerbibliotheken oder den Clientbibliotheken herstellen; allerdings müssen auch bei einer Verbindung zu den Clientbibliotheken die Kanalverbindungen definiert werden. Dies kann bei der Entwicklung einer Anwendung hilfreich sein. Sie können das Programm auf der eigenen Maschine ohne Abhängigkeiten von anderen testen und sicher sein, dass es auch dann arbeitet, wenn es in einer unabhängigen IBM MQ MQI clientumgebung eingesetzt wird.

Clients auf verschiedenen Plattformen

In diesem Beispiel kommuniziert die Servermaschine mit drei IBM MQ MQI clients auf verschiedenen Plattformen.

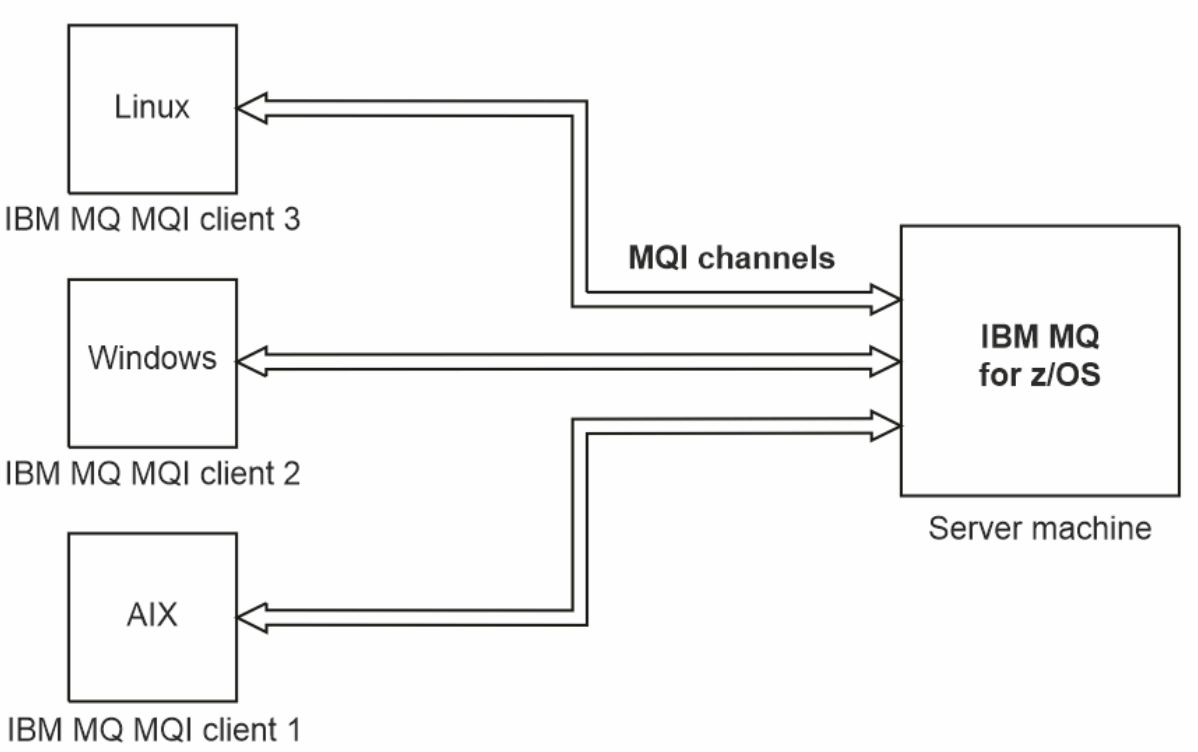


Abbildung 52. Verbindung eines IBM MQ-Servers mit Clients auf unterschiedlichen Plattformen

Es sind weitere komplexere Umgebungen möglich. So kann beispielsweise ein IBM MQ-Client mit mehreren Warteschlangenmanagern verbunden werden oder mit einer beliebigen Anzahl von Warteschlangenmanagern, die als Teil einer Gruppe mit gemeinsamer Warteschlange miteinander verbunden sind.

Verschiedene Versionen von Client- und Server-Software verwenden

Bei Verwendung älterer Versionen von IBM MQ-Produkten müssen Sie sicherstellen, dass der Server die Konvertierung der CCSID (ID des codierten Zeichensatzes) des Clients unterstützt.

Ein IBM MQ -Client kann eine Verbindung zu allen unterstützten Versionen des Warteschlangenmanagers herstellen. Wenn Sie eine Verbindung mit einem Warteschlangenmanager einer früheren Version herstellen, können Sie Funktionen und Strukturen aus einer späteren Version des Produkts in Ihrer IBM MQ-Anwendung auf dem Client nicht verwenden.

Ein IBM MQ -Warteschlangenmanager kann mit Clients unterschiedlicher Versionen mit sich selbst kommunizieren, indem er bis zur höchsten gegenseitig unterstützten Protokollebene aushandelt. Dies bedeutet, dass ältere Clients mit höheren Warteschlangenmanagerversionen verwendet werden können. Es wird empfohlen, dass sowohl der Client als auch der Server IBM MQ -Versionen haben, die derzeit unterstützt werden, um die Problemdiagnose zu vereinfachen und die Unterstützung durch IBMzu aktivieren.

Weitere Informationen finden Sie in den unterstützten Programmiersprachen unter Anwendungen entwickeln.

Transaktionsmanagement und -unterstützung

Dieser Abschnitt gibt eine Einführung in das Transaktionsmanagement und es wird erläutert, wie Transaktionen in IBM MQ unterstützt werden.

Ein *Ressourcenmanager* ist ein Computersubsystem, das Eigner der Ressourcen ist, die von ihm verwaltet werden; Anwendungen können auf diese Ressourcen zugreifen und sie aktualisieren. Hier einige Beispiele für Ressourcenmanager:

- Ein IBM MQ-Warteschlangenmanager; bei den Ressourcen handelt es sich um Warteschlangen
- Eine Db2-Datenbank; bei den Ressourcen handelt es sich um die Datenbanktabellen

Wenn eine Anwendung die Ressourcen auf einem oder auch mehreren Ressourcenmanagern aktualisiert, ist möglicherweise eine Geschäftsanforderung implementiert, mit der sichergestellt wird, dass bestimmte Aktualisierungen entweder zusammen erfolgreich oder gar nicht ausgeführt werden. Dies ist sinnvoll, da die Geschäftsdaten einen inkonsistenten Status hätten, wenn einige Aktualisierungen erfolgreich wären, andere hingegen nicht.

Bei dieser Art von Aktualisierung von Ressourcen, die auf diese Weise verwaltet werden, spricht man von einer *Arbeitseinheit* bzw. *Transaktion*. Ein Anwendungsprogramm kann eine Reihe von Aktualisierungen zu einer solchen Arbeitseinheit zusammenfassen.

Während einer solchen Arbeitseinheit sendet die Anwendung an die Ressourcenmanager Anforderungen zur Aktualisierung der Ressourcen. Die Arbeitseinheit endet, wenn von der Anwendung die Anforderung abgesetzt wird, alle Aktualisierungen festzuschreiben (Commit). Bis zu ihrer Festschreibung bleiben die Aktualisierungen für andere Anwendungen, die auf dieselben Ressourcen zugreifen, unsichtbar. Ebenso kann eine Anwendung die Arbeitseinheit auch abbrechen und eine Anforderung absetzen, alle bis dahin angeforderten Aktualisierungen zurückzusetzen. In diesem Fall werden die Aktualisierung nie sichtbar für andere Anwendungen. Diese Aktualisierungen sind in der Regel logisch miteinander verknüpft und müssen alle erfolgreich ausgeführt werden, damit die Datenintegrität gewährleistet ist. Die Datenintegrität geht verloren, wenn einige Aktualisierungen erfolgreich verlaufen, andere nicht.

Wenn eine Arbeitseinheit erfolgreich abgeschlossen ist, spricht man von einer *Festschreibung*. Nach der Festschreibung sind alle innerhalb einer Arbeitseinheit vorgenommenen Aktualisierungen permanent und können nicht mehr zurückgesetzt werden. Schlägt die Arbeitseinheit fehl, werden alle Aktualisierungen wieder *zurückgesetzt*. Dieser Prozess, bei dem Arbeitseinheiten unter Wahrung der Datenintegrität festgeschrieben oder zurückgesetzt werden, wird als *Synchronisationspunktkoordination* bezeichnet.

Der Zeitpunkt, zu dem alle Aktualisierungen innerhalb einer Arbeitseinheit festgeschrieben oder zurückgesetzt werden, wird als *Synchronisationspunkt* bezeichnet. Bei einer Aktualisierung innerhalb einer Arbeitseinheit spricht man davon, dass sie *synchronisationspunktgesteuert* ausgeführt wird. Fordert eine Anwendung eine *nicht synchronisationspunktgesteuerte* Aktualisierung an, schreibt der Ressourcenmanager die Aktualisierung sofort fest, auch wenn eine Arbeitseinheit noch nicht abgeschlossen wurde und die Aktualisierung nicht mehr zurückgesetzt werden kann.

Das Computersubsystem, das Arbeitseinheiten verwaltet, wird als *Transaktionsmanager* oder *Punktkoordinator* bezeichnet.

Eine *lokale* Arbeitseinheit ist eine Arbeitseinheit, in der nur die Ressourcen des IBM MQ-Warteschlangenmanagers aktualisiert werden. Die Synchronisationspunktkoordination wird vom Warteschlangenmanager selbst mit einer einphasigen Festschreibung vorgenommen.

Bei einer *globalen* Arbeitseinheit werden auch die Ressourcen anderer Ressourcenmanager, beispielsweise einer XA-konformen Datenbank, aktualisiert. Hier wird eine zweiphasige Festschreibung durchgeführt; die Arbeitseinheit kann vom Warteschlangenmanager selbst oder von einem anderen XA-konformen Transaktionsmanager wie IBM TXSeries oder BEA Tuxedo koordiniert werden.

Der Transaktionsmanager muss dafür sorgen, dass entweder alle Aktualisierungen, die an Ressourcen vorgenommen werden, erfolgreich innerhalb einer Arbeitseinheit abgeschlossen werden, oder keine. Anwendungen senden Anforderungen zum Festschreiben oder Zurücksetzen einer Arbeitseinheit an den Transaktionsmanager. Beispiele für Transaktionsmanager sind CICS und WebSphere Application Server, auch wenn beide Produkte noch über andere Funktionen verfügen.

Einige Ressourcenmanager stellen eine eigene Funktion für das Transaktionsmanagement bereit. So kann beispielsweise ein IBM MQ-Warteschlangenmanager Arbeitseinheiten verwalten, bei denen die eigenen Ressourcen und Db2-Tabellen aktualisiert werden. Der Warteschlangenmanager benötigt hierfür keinen separaten Transaktionsmanager; wenn dies vom Benutzer gefordert wird, kann jedoch zusätzlich ein Transaktionsmanager eingesetzt werden. Ein solcher separater Transaktionsmanager wird als *externer Transaktionsmanager* bezeichnet.

Damit eine Arbeitseinheit von einem externen Transaktionsmanager verwaltet werden kann, ist zwischen dem Transaktionsmanager und allen an der Arbeitseinheit beteiligten Ressourcenmanagern eine

Standardschnittstelle erforderlich. Über diese Schnittstellen können der Transaktionsmanager und die Ressourcenmanager miteinander kommunizieren. Eine dieser Schnittstellen ist die *XA-Schnittstelle*, eine Standardschnittstelle, die von einer Reihen von Transaktions- und Ressourcenmanagern unterstützt wird. Die XA-Schnittstelle ist von The Open Group in *Distributed Transaction Processing: The XA Specification* veröffentlicht.

Wenn mehrere Ressourcenmanager an einer Arbeitseinheit beteiligt sind, muss der Transaktionsmanager ein Protokoll für die *zweiphasige Festschreibung* verwenden; damit wird sichergestellt, dass entweder alle Aktualisierungen innerhalb einer Arbeitseinheit erfolgreich ausgeführt werden oder keine, und dies auch im Fall eines Systemausfalls. Wenn der Transaktionsmanager von einer Anwendung eine Anforderung zum Festschreiben einer Arbeitseinheit erhält, wird Folgendes durchgeführt:

Phase 1 (Vorbereitung für die Festschreibung)
Der Transaktionsmanager fragt die einzelnen Ressourcenmanager ab, die an dieser Arbeitseinheit beteiligt sind, um sicherzustellen, dass alle Informationen zu den beabsichtigten Aktualisierungen an den Ressourcen einen wiederherstellbaren Status haben. Dazu schreibt ein Ressourcenmanager die Informationen in der Regel in ein Protokoll und stellt sicher, dass diese Informationen auf der Festplatte gespeichert werden. Phase 1 ist abgeschlossen, wenn der Transaktionsmanager von jedem Ressourcenmanager benachrichtigt wurde, dass die Informationen zu den beabsichtigten Aktualisierungen an den Ressourcen einen wiederherstellbaren Status haben.

Phase 2 (Festschreibung)
Nach Abschluss der Phase 1 bereitet der Transaktionsmanager den unwiderruflichen Schritt vor, mit dem die Arbeitseinheit festgeschrieben wird. Dazu fordert er die einzelnen Ressourcenmanager, die an der Arbeitseinheit beteiligt sind, auf, die Aktualisierungen an ihren Ressourcen festzuschreiben. Wenn ein Ressourcenmanager diese Anforderung empfängt, muss er die Aktualisierungen festschreiben. Zu diesem Zeitpunkt ist es für ihn nicht mehr möglich, die Aktualisierungen zurückzusetzen. Phase 2 ist abgeschlossen, wenn der Transaktionsmanager von jedem Ressourcenmanager benachrichtigt wird, dass die Aktualisierungen an den Ressourcen festgeschrieben wurden.

Die XA-Schnittstelle verwendet ein solches Protokoll für die zweiphasige Festschreibung.

Weitere Informationen finden Sie im Abschnitt Szenarios zur Transaktionsunterstützung.

IBM MQ stellt auch Unterstützung für Microsoft Transaction Server (COM +) bereit. Im Abschnitt Microsoft Transaction Server (COM+) verwenden wird beschrieben, wie IBM MQ für die Nutzung der COM+-Unterstützung konfiguriert wird.

Funktionen des Warteschlangenmanagers erweitern

Sie können die Funktionen eines Warteschlangenmanagers durch Benutzerexits, API-Exits oder installierbare Services erweitern.

Benutzerexits

Benutzerexits ermöglichen es Ihnen, eigenen Code in eine Warteschlangenmanagerfunktion einzufügen. Zu den unterstützten Benutzerexits gehören folgende:

Kanalexits
Über diese Exits kann die Betriebsweise von Kanälen geändert werden. Kanalexits werden im Abschnitt Kanalexitprogramme für Messaging-Kanäle beschrieben.

Datenkonvertierungsexits
Diese Exits erstellen Quellencodefragmente, die in Anwendungsprogramme eingefügt werden, um Daten in ein anderes Format zu konvertieren. Datenkonvertierungsexits werden im Abschnitt Datenkonvertierungsexits schreiben beschrieben.

Exit für Clusterauslastung
Die Funktion dieses Exits wird vom Exit-Provider definiert. Informationen zur Aufrufdefinition finden Sie im Abschnitt MQ_CLUSTER_WORKLOAD_EXIT - Beschreibung des Aufrufs.

API-Exits

API-Exits ermöglichen das Schreiben von Code zur Änderung des Verhaltens von IBM MQ-API-Aufrufen wie MQPUT und MQGET, und fügen diesen Code dann unmittelbar vor oder nach diesen Aufrufen ein. Das Einfügen erfolgt automatisch; der Exit-Code wird dann vom Warteschlangenmanager an den registrierten Punkten ausgeführt. Weitere Informationen zu API-Exits finden Sie im Abschnitt API-Exits schreiben und verwenden.

Installierbare Services

Installierbare Services haben formalisierte Schnittstellen (eine API) mit mehreren Eingangspunkten.

Ein implementierter installierbarer Service wird als *Servicekomponente* bezeichnet. Sie können entweder die zusammen mit IBM MQ bereitgestellten Komponenten verwenden oder eigene Komponenten für die für Ihre Umgebung erforderlichen Funktionen erstellen.

Derzeit werden die folgenden installierbaren Services bereitgestellt:

Berechtigungsservice
Mithilfe des Berechtigungsservice können Sie eine eigene Sicherheitsfunktion erstellen.

Die Standardservicekomponente, mit der der Service implementiert wird, ist der Objektberechtigungsmanager (Object Authority Manager; OAM). Der Objektberechtigungsmanager ist standardmäßig aktiviert; Sie müssen nichts tun, um ihn zu konfigurieren. Mithilfe der Berechtigungsserviceschnittstelle können Sie andere Komponenten erstellen, um den Objektberechtigungsmanager zu ersetzen oder aber zu erweitern. Weitere Informationen zu OAM finden Sie unter Sicherheit auf AIX, Linux, and Windows-Systemen einrichten.

Namensservice
Mit dem Namensservice können Anwendungen Warteschlangen gemeinsam nutzen, indem ferne Warteschlangen wie lokale Warteschlangen angegeben werden.

Sie können eine eigene Namensservicekomponente erstellen. Dies ist beispielsweise sinnvoll, wenn der Namensservice zusammen mit IBM MQ verwendet werden soll. Um den Namensservice zu verwenden, muss entweder eine benutzerdefinierte Komponente oder eine Komponente von einem anderen Softwareanbieter vorhanden sein. Der Namensservice ist standardmäßig inaktiv.

Zugehörige Konzepte
Benutzerexits, API-Exits und installierbare IBM MQ-Services

IBM MQ Java-Sprachenschnittstellen

IBM MQ stellt drei Anwendungsprogrammierschnittstellen (APIs) für die Verwendung in Java -Anwendungen bereit: IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSund IBM MQ classes for Java.

IBM unterstützt und ist ein aktiver Teilnehmer an offenen Standards.

- Ab IBM MQ 8.0implementiert das Produkt den JMS 2.0 -Standard, der eine neue vereinfachte API zusammen mit Features wie gemeinsam genutzte Subskriptionen eingeführt hat.
- Ab IBM MQ 9.3.0wird auch Jakarta Messaging 3.0 unterstützt.
- Außerdem unterstützt WebSphere Liberty JMS 2.0 und Jakarta Messaging 3.0 mit IBM MQ.

In IBM MQ gibt es drei APIs für die Verwendung in Java -Anwendungen:

JM 3.0 **IBM MQ classes for Jakarta Messaging**
IBM MQ classes for Jakarta Messaging ist ein Jakarta Messaging-Provider, der die Jakarta Messaging-Schnittstellen für IBM MQ als Messaging-System implementiert. Jakarta Connectors Architecture bietet eine Standardmethode zum Verbinden von Anwendungen, die in einer Jakarta EE -Umgebung ausgeführt werden, mit einem unternehmensweiten Informationssystem (EIS) wie IBM MQ oder Db2.

JMS 2.0 **IBM MQ classes for JMS**
IBM MQ classes for JMS ist ein JMS-Provider, der die JMS-Schnittstellen für IBM MQ als Messaging-System implementiert. Die Java Platform, Enterprise Edition Connector Architecture (JCA) stellt eine Standardmethode für die Verbindung von Anwendungen, die in einer Java EE-Umgebung aktiv sind, mit einem Enterprise Information System (EIS) wie IBM MQ oder Db2 bereit.

IBM MQ classes for Java
IBM MQ classes for Java ermöglichen Ihnen die Verwendung von IBM MQ in einer Java-Umgebung. IBM MQ classes for Java ermöglichen einer Java-Anwendung eine Verbindung mit IBM MQ als IBM MQ-Client oder eine direkte Verbindung mit einem IBM MQ-Warteschlangenmanager.

Anmerkung:

- JMS 2.0 wurde durch Jakarta Messagingersetzt. IBM MQ classes for JMS unterstützt weiterhin den JMS 2.0 -Standard, aber zukünftige Erweiterungen für das Java -Messaging treten nur in Jakarta Messagingauf, daher in IBM MQ classes for Jakarta Messaging. IBM MQ classes for JMS wird nur für die Verwaltung und Erweiterung vorhandener JMS 2.0 -Anwendungen empfohlen. IBM MQ classes for Jakarta Messaging sollte die bevorzugte Technologie für die Neuentwicklung sein.
- Stabilized IBM MQ classes for Java werden funktional auf der mit IBM MQ 8.0 bereitgestellten Stufe eingefroren. Bestehende Anwendungen, die IBM MQ classes for Java verwenden, werden weiterhin vollständig unterstützt, diese API wird jedoch eingefroren, d. h., es werden keine Änderungen mehr daran vorgenommen (auch nicht auf Kundenanfrage). Vollständig unterstützt bedeutet, dass Fehler behoben und daraus erforderliche Änderungen an den IBM MQ-Systemvoraussetzungen vorgenommen werden.

JM 3.0 Ab IBM MQ 9.3werden die IBM MQ classes for Java, IBM MQ classes for JMSund IBM MQ classes for Jakarta Messaging mit Java 8erstellt. Java -Laufzeitumgebungen mit diesen Ebenen oder höher müssen verwendet werden, um Anwendungen unter Verwendung dieser Schnittstellen auszuführen.

Zugehörige Konzepte
Zugriff auf IBM MQ über Java -Auswahl der API
Warum sollte ich IBM MQ -Klassen für Jakarta Messaging verwenden?
Gründe für die Verwendung von IBM MQ classes for JMS
Gründe für die Verwendung von IBM MQ classes for Java

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Messaging-Provider, die mit IBM MQbereitgestellt werden. Jeder dieser Provider stellt außerdem zwei Gruppen von Erweiterungen für die Messaging API bereit. Diese Messaging-Provider können sowohl von Java Platform, Standard Edition -Anwendungen (Java SE) als auch von Java Platform, Enterprise Edition -Anwendungen (Java EE) verwendet werden.

JM 3.0 IBM MQ 9.3.0 wurde die Unterstützung für Jakarta Messaging 3.0eingeführt. JMS 2.0 wird weiterhin vollständig unterstützt.

Die Spezifikationen JMS und Jakarta Messaging definieren eine Gruppe von Schnittstellen, die Anwendungen für Messaging-Operationen verwenden können. Das Produkt unterstützt die JMS 2.0 -Version des JMS -Standards. Diese Implementierung bietet alle Funktionen der klassischen API, erfordert jedoch weniger Schnittstellen und ist einfacher zu verwenden. Weitere Informationen hierzu finden Sie im Abschnitt „Modell JMS und Jakarta Messaging” auf Seite 166 und in der Spezifikation zu JMS 2.0 auf Java.net.

JM 3.0 Ab IBM MQ 9.3.0wird auch Jakarta Messaging unterstützt.

Das Paket jakarta.jms (Jakarta Messaging 3.0) oder javax.jms (JMS 2.0) gibt die Details der Messaging-Schnittstellen an, und ein Messaging-Provider implementiert diese Schnittstellen für ein bestimmtes Messaging-Produkt. For example:

- IBM MQ classes for JMS ist ein JMS-Provider, der die JMS-Schnittstellen für IBM MQ implementiert und darüber hinaus die beiden folgenden Erweiterungssätze für die JMS-API bereitstellt:

  - IBM MQ JMS-Erweiterungen
  - IBM JMS-Erweiterungen
- Eine Verbindungsfactory, eine Warteschlange oder ein Themenobjekt, die bzw. das mit javax.dimsoder jakarta.jmserstellt wurde, Schnittstellen oder eine Gruppe von JMS -Erweiterungen können mit jeder dieser APIs adressiert werden, d. h., sie können in jede der Schnittstellen umgesetzt werden. Um die Portierbarkeit Ihrer Anwendungen zu erleichtern, sollten Sie stets die allgemeinste API verwenden, die für Ihre Anforderungen gerade noch geeignet ist.

Da JMS und Jakarta Messaging viel gemeinsam genutzt werden, können weitere Verweise auf JMS in diesem Abschnitt als Verweise auf beide Elemente betrachtet werden. Alle Unterschiede werden nach Bedarf hervorgehoben.

IBM MQ JMS-Erweiterungen

IBM MQ classes for JMS stellt Erweiterungen für die JMS-API bereit. IBM MQ classes for JMS enthält Erweiterungen, die in MQConnectionFactory-, MQQueue-und MQTopic-Objekten implementiert sind. Diese Objekte haben IBM MQ-spezifische Eigenschaften und Methoden. Bei den Objekten kann es sich um verwaltete Objekte handeln oder eine Anwendung kann die Objekte dynamisch während der Laufzeit erstellen. Diese Erweiterungen werden als IBM MQ JMS -Erweiterungen bezeichnet. In dieser Dokumentation werden Objekte, die dynamisch von einer Anwendung zur Laufzeit erstellt werden, nicht als verwaltete Objekte betrachtet.

IBM JMS-Erweiterungen

Zusätzlich zu den IBM MQ JMS -Erweiterungen bietet IBM MQ classes for JMS eine allgemeinere Gruppe von Erweiterungen für die JMS -API oder Java als Programmiersprache. Diese Erweiterungen werden als IBM JMS -Erweiterungen bezeichnet und haben die folgenden allgemeinen Ziele:

- Um eine größere Konsistenz zwischen IBM JMS -Providern bereitzustellen.
- Um das Schreiben einer Brückenanwendung zwischen zwei IBM -Messaging-Systemen zu vereinfachen.
- Um das Portieren einer Anwendung von einem IBM JMS -Provider auf einen anderen zu vereinfachen

Der Hauptschwerpunkt dieser Erweiterungen liegt auf der dynamischen Erstellung und Konfiguration von Verbindungsfactorys und Zielen zur Laufzeit, die Erweiterungen bieten jedoch auch Funktionen, die nicht direkt mit Messaging verbunden sind, z. B. Funktionen für die Problembestimmung.

Zugehörige Tasks

IBM MQ -Klassen für JMS/Jakarta Messaging verwenden

JMS-und Jakarta Messaging-Ressourcen konfigurieren

JM 3.0 IBM MQ classes for Jakarta Messaging: Übersicht

IBM MQ 9.3.0 führt Unterstützung für Jakarta Messagingein. Für Jakarta Messaging 3.0die Steuerung der JMS -Spezifikation, die von Oracle in den Java -Community-Prozess verschoben wurde Oracle behält jedoch die Steuerung des Namens "javax", der in anderen Java -Technologien verwendet wird. Obwohl Jakarta Messaging 3.0 funktional äquivalent zu JMS 2.0ist, gibt es einige Unterschiede bei der Benennung. Der offizielle Name für Version 3.0 ist Jakarta Messaging und nicht Java Message Serviceund die Paket-und Konstantennamen haben das Präfix `jakarta` und nicht `javax`.

Hintergrund

Die Java -Plattform wird seit vielen Jahren in zwei Formaten bereitgestellt: Standard Edition und Enterprise Edition.

Java Platform, Standard Edition (manchmal abgekürzt Java SE) ist die Kernsprache und die Klassenbibliotheken, die in einem eigenständigen Kontext ausgeführt werden können. Die meisten Java -Pakete in Java SE haben Namen, die mit "java." beginnen.

Java Platform, Enterprise Edition (Java EE) erweitert dies und fügt Funktionalität wie Messaging, verschiedene Beans, Transaktionalität usw. hinzu. Einige dieser Technologien können auch in einem Java SE -Kontext verwendet werden. Die meisten Java -Pakete in Java EE haben in der Vergangenheit Namen, die mit "javax." beginnen. Es gibt jedoch eine Querverbindung, sodass einige Java SE -Pakete "javax" enthalten. als Präfix für ihren Namen.

Java Message Service (JMS) ist Teil von Java Platform, Enterprise Edition. Java EE 7 enthält JMS 2.0.

Bis Java EE 7standen die Technologien unter der Verantwortung von Oracle.

Die Java EE -Technologien wurden kürzlich von der Verwaltung von Oracle zu einem Community-Prozess verschoben, der von der Eclipse Foundation überwacht wird.

Als "javax." Name konnte nicht in das neue Projekt verschoben werden, neue Benennung wurde übernommen-alle Pakete und Eigenschaftsnamen haben jetzt das Präfix "jakarta". Java Platform, Enterprise Edition wird in Zukunft als "Jakarta EE" bezeichnet. Die Versionsnummerierung wurde fortgesetzt: Version 8 war eine Zwischenversion, die weitgehend ignoriert werden kann, und Jakarta EE 9 ist der Punkt, an dem "jakarta". Präfix wird wirksam.

Die Jakarta EE -Haupttechnologie, die im IBM MQ -Kontext angewendet wird, ist Jakarta Messaging 3.0 -der Nachfolger von Java Message Service (JMS) 2.0. Jakarta EE 9 enthält also Jakarta Messaging 3.0.

IBM MQ unterstützt weiterhin Java EE 7 und JMS 2.0und bietet Unterstützung für Jakarta EE 9 und Jakarta Messaging 3.0.

Was wird bereitgestellt: Java SE

Für Java Platform, Standard Editionwird zusätzlich zu IBM MQ classes for JMS (die JMS 2.0 -Operationen mit IBM MQunterstützen) IBM MQ 9.3.0 und höheren Versionen IBM MQ classes for Jakarta Messagingbereitgestellt. Diese Klassen stellen einen Jakarta Messaging 3.0 -Provider bereit, der in IBM MQintegriert ist und die Verwendung von IBM MQ -Warteschlangenmanagern zur Vereinfachung von Jakarta Messaging -Operationen ermöglicht.

Diese werden als Standard-JAR-Datei `com.ibm.mq.jakarta.client.jar`im Unterverzeichnis `java/lib` der IBM MQ -Installation bereitgestellt.

Für die Verwendung in OSGi-Containern wie Apache Felix oder Eclipse Equinox stellt IBM MQ auch ein Paar OSGi-Bundles bereit:

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

Dabei ist *V.R.M.F* stellt die Version von IBM MQdar, z. B. 9.3.0.0. Diese Bundles befinden sich im Unterverzeichnis `java/lib/OSGi` der IBM MQ -Installation.

Was wird bereitgestellt: Jakarta EE 9

Zur Unterstützung des IBM MQ-basierten Messaging in einem Jakarta EE 9 -kompatiblen Anwendungsserver stellt IBM MQ einen Jakarta EE 9-kompatiblen Ressourcenadapter bereit: `wmq.jakarta.jmsra.rar`. Diese Datei befindet sich im Unterverzeichnis `java/lib/jca` der IBM MQ -Installation.

IBM MQ stellt weiterhin einen Java EE 7 -kompatiblen Ressourcenadapter (`wmq.jmsra.rar`) im Unterverzeichnis `java/lib/jca` der IBM MQ -Installation bereit.

Bereitstellung dieser Artefakte

Diese JAR-Dateien und die RAR-Datei für den Ressourcenadapter werden mit den bereits vorhandenen Artefakten auf den üblichen IBM MQ -Installationsmedien gepackt-sowohl mit den plattformspezifischen Installationsmedien, wie z. B. "`.rpm`" -Dateien, als auch mit den weiterverteilbaren Medien, wie z. B. den selbstextrahierenden weiterverteilbaren Client-JAR-Dateien.

Änderungen zwischen JMS 2.0 und Jakarta Messaging 3.0

Jakarta EE 9 und Jakarta Messaging 3.0 führen keine neue Funktionalität ein. Alles, was sich ändert, sind Namen. Wenn Sie beispielsweise "javax.jms.Connection" in JMS 2.0verwenden, verwenden Sie "jakarta.jms.Connection" in Jakarta Messaging 3.0.

Da die Eclipse Foundation die Jakarta EE -Plattform weiterleitet, wird sie auf dieser Basis aufbauen und diese Namenskonvention wird für neue Funktionen verwendet, die in Zukunft eingeführt werden.

Änderungen zwischen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging

Zusammenfassung

IBM MQ classes for JMS, die Unterstützung für JMS 2.0bereitstellen, bleiben verfügbar und werden hauptsächlich für die Verwaltung und Erweiterung vorhandener Anwendungen empfohlen. Sie werden vollständig unterstützt.

IBM MQ classes for Jakarta Messaging, die Unterstützung für Jakarta Messaging 3.0bereitstellen, werden für die Neuentwicklung empfohlen.

In IBM MQ 9.3.0waren diese beiden Angebote funktional äquivalent. Nur die Benennung unterscheidet sich. Es ist jedoch wahrscheinlicher, dass neue Messaging-Funktionalität in IBM MQ classes for Jakarta Messaging entsteht als in IBM MQ classes for JMS.

Die beiden Angebote sind interoperabel. Nachrichten, die von IBM MQ classes for JMS erzeugt werden, können von IBM MQ classes for Jakarta Messagingverarbeitet werden und umgekehrt. Die beiden Angebote dürfen jedoch nicht in einer einzigen Anwendung koexistieren.

Benennungsänderungen

Tabelle 16. Änderungen an Paketnamen

IBM MQ classes for JMS Paketname	IBM MQ classes for Jakarta Messaging Paketname
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.*

Die Pakete, die sich auf allgemeine Services (Trace, Protokollierung, Unterstützung in der Landessprache usw.) und die JMQI-Implementierungen (lokal und fern) beziehen, sind für IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaginggemeinsam, sodass in diesen Bereichen keine Änderungen erforderlich sind.

Beachten Sie, dass auch Eigenschaftsnamen geändert wurden. Die Eigenschaft zum Aktivieren von IBM MQ -Erweiterungen in IBM MQ classes for Jakarta Messaging ist beispielsweise **`com.ibm.mq.jakarta.jms.SupportMQExtensions`**.

Eigenschaftsnamen, die unabhängig von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingsind, wie z. B. die verschiedenen **`com.ibm.msg.client.commonservices.trace.*`** -Eigenschaften, gelten gleichermaßen für beide Angebote.

Verwaltungsdienstprogramme

Die Dienstprogramme **`crtmqenv`** und **`setmqenv`** akzeptieren jetzt eine Option, mit der angegeben werden kann, ob der Klassenpfad für IBM MQ classes for JMS (`-j 2.0`) oder IBM MQ classes for Jakarta Messaging (`-j 3.0`) konfiguriert werden soll, und es gibt IBM MQ classes for Jakarta Messaging Varianten der **`runjms`** -Dienstprogramme namens **`runjms30`** und ähnliche Namen.

Wenn das Dienstprogramm **dspmqver** aufgefordert wird, Berichte zu Java -Komponenten zu erstellen, wird IBM MQ classes for Jakarta Messaging in die Ausgabe eingeschlossen.

Zum Konfigurieren von IBM MQ classes for Jakarta Messaging -Objekten, die über JNDI abgerufen werden sollen, entspricht das neue Dienstprogramm **JMS30Admin** dem Dienstprogramm **JMSAdmin** für IBM MQ classes for JMS.

Beachten Sie, dass die zugrunde liegenden Objekte aus verschiedenen Paketen stammen. JNDI definitions created by **JMSAdmin** cannot be used by IBM MQ classes for Jakarta Messaging, nor can those created by **JMS30Admin** be used by IBM MQ classes for JMS.

Anmerkung: Es gibt keine Unterstützung für IBM MQ classes for Jakarta Messaging -Objekte, die von IBM MQ Explorerbereitgestellt werden; ihre JNDI-Integration gilt nur für IBM MQ classes for JMS .

Zugehörige Konzepte

Warum sollte ich IBM MQ -Klassen für Jakarta Messaging verwenden?

Modell JMS und Jakarta Messaging

Das Modell JMS und Jakarta Messaging definiert eine Gruppe von Schnittstellen, die Java -Anwendungen für Messaging-Operationen verwenden können. IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind Messaging-Provider, die definieren, wie Java -Messaging-Objekte zu IBM MQ -Konzepten gehören. Die Spezifikationen JMS und Jakarta Messaging erwarten, dass bestimmte Messaging-Objekte verwaltete Objekte sind.

Ab IBM MQ 8.0unterstützt das Produkt die JMS 2.0 -Version des JMS-Standards, die eine vereinfachte API eingeführt hat und gleichzeitig die klassische API aus JMS 1.1beibehält.

JM 3.0 IBM MQ 9.3.0 wurde die Unterstützung für Jakarta Messaging 3.0eingeführt. JMS 2.0 wird weiterhin vollständig unterstützt. Da JMS und Jakarta Messaging viel gemeinsam genutzt werden, können weitere Verweise auf JMS in diesem Abschnitt als Verweise auf beide Elemente betrachtet werden. Alle Unterschiede werden nach Bedarf hervorgehoben.

Vereinfachte API

JMS 2.0 hat die vereinfachte API eingeführt und gleichzeitig die domänenspezifischen und domänenunabhängigen Schnittstellen von JMS 1.1beibehalten. Die vereinfachte API reduziert die Anzahl der zum Senden und Empfangen von Nachrichten erforderlichen Objekte und besteht aus den folgenden Schnittstellen:

ConnectionFactory
Verbindungsfactory - Ein verwaltetes Objekt, das von einem JMS-Client zum Erstellen einer Verbindung verwendet wird. Diese Schnittstelle wird auch in der klassischen API verwendet.

JMSContext
JMS-Kontext - Dieses Objekt vereint die Objekte "Connection" und "Session" der klassischen API. JMSContext-Objekte können aus anderen JMSContext-Objekten erstellt werden, wobei die zugrunde liegende Verbindung dupliziert wird.

JMSProduzent
JMS-Produzent - Ein JMSProducer wird aus einem JMSContext erstellt und zum Senden von Nachrichten an eine Warteschlange oder ein Thema verwendet. Das JMSProducer-Objekt veranlasst die Erstellung der zum Senden einer Nachricht erforderlichen Objekte.

JMSKonsument
JMS-Konsument - Ein JMSConsumer wird aus einem JMSContext erstellt und zum Empfangen der Nachrichten aus einer Warteschlange oder einem Thema verwendet.

Die vereinfachte API hat verschiedene Auswirkungen:

- Das Objekt JMSContext startet die zugrunde liegende Verbindung immer sofort automatisch.
- JMSProducer und JMSConsumer können nun mit der Methode getBody der Nachricht direkt mit dem Nachrichtenhauptteil interagieren, ohne das vollständige Nachrichtenobjekt abrufen zu müssen.

- Nachrichteneigenschaften können nun vor dem Senden des Nachrichten-Bodys (Nachrichtenhauptteils bzw. Inhalt der Nachricht) mittels Methodenverkettung im JMSProducer-Objekt festgelegt werden. Der JMSProducer veranlasst die Erstellung aller Objekte, die zum Senden einer Nachricht erforderlich sind. Mit JMS 2.0 können Eigenschaften wie folgt festgelegt und eine Nachricht gesendet werden:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 hat auch gemeinsam genutzte Subskriptionen eingeführt, bei denen Nachrichten von mehreren Konsumenten gemeinsam genutzt werden können. Alle Subskriptionen aus JMS 1.1 werden wie nicht gemeinsam genutzte Subskriptionen behandelt.

Klassische API

Die folgende Aufstellung geht kurz auf die wichtigsten JMS-Schnittstellen der klassischen API ein:

Destination
: Ziel - Ein Destination-Objekt ist der Ort, an den eine Anwendung Nachrichten sendet, und/oder eine Quelle, aus der eine Anwendung Nachrichten empfängt.

ConnectionFactory
: Verbindungsfactory - Ein ConnectionFactory-Objekt bindet eine Gruppe von Konfigurationseigenschaften für eine Verbindung ein. Eine Anwendung verwendet eine Verbindungsfactory, um eine Verbindung zu erstellen.

Verbindung
: Verbindung - Ein Connection-Objekt bindet die aktive Verbindung einer Anwendung in einen Messaging-Server ein. Eine Anwendung verwendet eine Verbindung, um Sitzungen zu erstellen.

Sitzung
: Sitzung - Ein Session-Objekt ist ein Einzelthreadkontext zum Senden und Empfangen von Nachrichten. Eine Anwendung verwendet eine Sitzung, um Nachrichten, Nachrichtenproduzenten und Nachrichtenkonsumenten zu erstellen. Eine Sitzung ist entweder transaktionsbasiert oder nicht transaktionsbasiert.

Nachricht
: Nachricht - Ein Message-Objekt bindet eine Nachricht ein, die von einer Anwendung gesendet oder empfangen wird.

MessageProducer
: Nachrichtenproduzent - Eine Anwendung verwendet ein MessageProducer-Objekt zum Senden von Nachrichten an ein Ziel.

MessageConsumer
: Nachrichtenkonsument - Eine Anwendung verwendet ein MessageConsumer-Objekt zum Empfangen von Nachrichten, die an ein Ziel gesendet wurden.

In Abbildung 53 auf Seite 168 sind diese Objekte und ihre Beziehungen dargestellt.

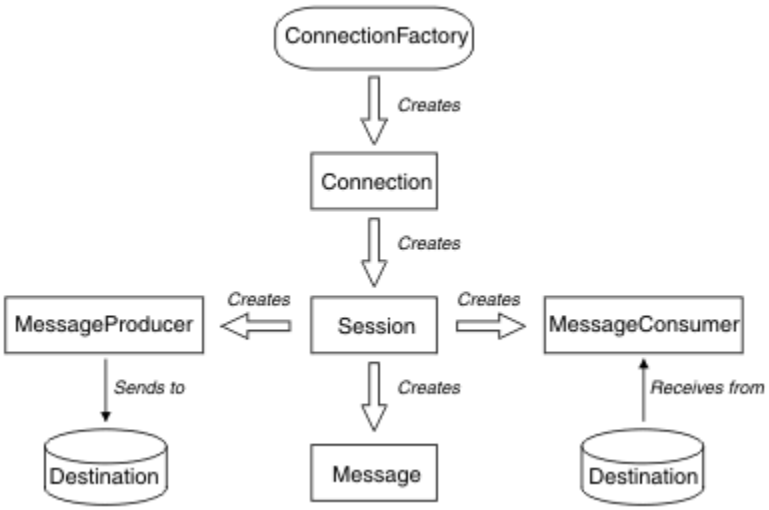


Abbildung 53. JMS-Objekte und ihre Beziehungen

Das Diagramm stellt die wichtigsten Schnittstellen dar: ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message und Destination. Eine Anwendung verwendet eine Verbindungsfactory, um eine Verbindung zu erstellen. Für die Erstellung von Sitzungen verwendet sie eine Verbindung. Die Anwendung kann dann eine Sitzung verwenden, um Nachrichten, Nachrichtenproduzenten und Nachrichtenkonsumenten zu erstellen. Die Anwendung verwendet zum Senden von Nachrichten an ein Ziel einen Nachrichtenproduzenten, während sie zum Empfangen von Nachrichten, die an ein Ziel gesendet wurden, einen Nachrichtenkonsumenten verwendet.

Ein Destination-, ConnectionFactory- oder Connection-Objekt kann gleichzeitig von verschiedenen Threads einer Multithread-Anwendung verwendet werden. Ein Session-, MessageProducer- oder MessageConsumer-Objekt kann hingegen nicht gleichzeitig von verschiedenen Threads verwendet werden. Die einfachste Methode, um sicherzustellen, dass ein Session-, MessageProducer- oder MessageConsumer-Objekt nicht gleichzeitig verwendet wird, ist die Erstellung eines separaten Session-Objekts für jeden Thread.

JMS unterstützt zwei Arten von Messaging:

- Punkt-zu-Punkt-Messaging
- Publish/Subscribe-Messaging

Diese Arten des Messaging werden auch als *Messaging-Domänen* bezeichnet. In einer Anwendung können beide Arten kombiniert werden. In einer Punkt-zu-Punkt-Domäne ist das Ziel eine Warteschlange, in einer Publish/Subscribe-Domäne ist es ein Thema.

Bei JMS-Versionen vor JMS 1.1 wird bei der Programmierung für die Punkt-zu-Punkt-Domäne eine Gruppe von Schnittstellen und Methoden verwendet, während bei der Programmierung für die Publish/Subscribe-Domäne eine andere Gruppe verwendet wird. Auch wenn sich beide ähneln, sind sie völlig getrennt voneinander. Ab JMS 1.1 können Sie eine gemeinsame Gruppe von Schnittstellen und Methoden verwenden, die beide Messaging-Domänen unterstützen. Diese gemeinsamen Schnittstellen bieten eine domänenunabhängige Darstellung der Messaging-Domänen. In Tabelle 17 auf Seite 168 sehen Sie eine Übersicht über die domänenunabhängigen JMS-Schnittstellen und deren domänenspezifischen Entsprechungen.

Tabelle 17. Domänenunabhängige JMS-Schnittstellen und die entsprechenden domänenspezifischen Schnittstellen

Domänenunabhängige Schnittstellen	**Domänenspezifische Schnittstellen für Punkt-zu-Punkt-Domäne**	**Domänenspezifische Schnittstellen für Publish/Subscribe-Domäne**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory

Tabelle 17. Domänenunabhängige JMS-Schnittstellen und die entsprechenden domänenspezifischen Schnittstellen (Forts.)

Domänenunabhängige Schnittstellen	**Domänenspezifische Schnittstellen für Punkt-zu-Punkt-Domäne**	**Domänenspezifische Schnittstellen für Publish/Subscribe-Domäne**
Verbindung	QueueConnection	TopicConnection
Destination	Warteschlange	Thema
Sitzung	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 unterstützt sowohl die früheren domänenspezifischen JMS 1.1 -Schnittstellen als auch die vereinfachte API von JMS 2.0. IBM MQ classes for JMS 2.0 kann daher zur Verwaltung vorhandener Anwendungen verwendet werden, einschließlich der Entwicklung neuer Funktionen in vorhandenen Anwendungen.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 unterstützt die Jakarta Messaging -Versionen derselben Schnittstellen und wird für die Entwicklung neuer Anwendungen empfohlen.

In IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingsind JMS -Objekte wie folgt mit IBM MQ -Konzepten verknüpft:

- Die Eigenschaften eines Connection-Objekts sind von den Eigenschaften der Verbindungsfactory abgeleitet, die zur Erstellung der Verbindung verwendet wurde. Diese Eigenschaften legen fest, wie eine Anwendung eine Verbindung zu einem Warteschlangenmanager herstellt. Beispiele für diese Eigenschaften sind der Name des Warteschlangenmanagers und bei Anwendungen, die sich im Clientmodus mit dem Warteschlangenmanager verbinden, der Hostname oder die IP-Adresse des Systems, auf dem der Warteschlangenmanager ausgeführt wird.
- Ein Session-Objekt umfasst eine IBM MQ-Verbindungskennung, die wiederum den Transaktionsbereich der Sitzung festlegt.
- Sowohl MessageProducer-Objekt als auch MessageConsumer-Objekt umfassen eine IBM MQ-Objektkennung.

Bei Verwendung von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaginggelten alle normalen Regeln von IBM MQ . Beachten Sie insbesondere, dass eine Anwendung zwar Nachrichten an eine ferne Warteschlange senden kann, aber nur Nachrichten aus einer Warteschlange empfangen kann, die zu dem Warteschlangenmanager gehört, mit dem die Anwendung verbunden ist.

Die JMS-Spezifikation erwartet, dass es sich bei den ConnectionFactory- und Destination-Objekten um verwaltete Objekte handelt. Ein Administrator erstellt und pflegt verwaltete Objekte in einem zentralen Repository, aus dem diese Objekte von JMS-Anwendungen über die Java Naming and Directory Interface (JNDI)-Schnittstelle abgerufen werden.

In IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingist die Implementierung der Destination-Schnittstelle eine abstrakte Superklasse von Queue und Topic, sodass eine Instanz von Destination entweder ein Queue-Objekt oder ein Topic-Objekt ist. Die domänenunabhängigen Schnittstellen behandeln eine Warteschlange (Queue) oder ein Thema (Topic) hingegen als Ziel (Destination). Die Messaging-Domäne eines MessageProducer- oder MessageConsumer-Objekts bestimmt sich daraus, ob das Ziel eine Warteschlange oder ein Thema ist.

In IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging können daher Objekte der folgenden Typen verwaltete Objekte sein:

- ConnectionFactory

- QueueConnectionFactory
- TopicConnectionFactory
- Warteschlange
- Thema
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

IBM MQ classes for JMS/Jakarta Messaging -Architektur

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging verfügen über eine Schichtarchitektur. Die oberste Codeebene ist eine allgemeine Ebene, die jeder IBM Java -Messaging-Provider verwenden kann.

JM 3.0 IBM MQ 9.3.0 wurde die Unterstützung für Jakarta Messaging 3.0eingeführt. JMS 2.0 wird weiterhin vollständig unterstützt.

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging verfügen über eine Schichtarchitektur, wie im Diagramm Abbildung 54 auf Seite 170dargestellt. Die oberste Codeebene ist eine allgemeine Ebene, die von jedem IBM JMS -oder Jakarta Messaging-Provider verwendet werden kann. Wenn eine Anwendung eine Methode JMS oder Jakarta Messagingaufruft, wird jede Verarbeitung des Aufrufs, die nicht für ein Messaging-System spezifisch ist, von der allgemeinen Schicht ausgeführt, die auch eine konsistente Antwort auf den Aufruf bereitstellt. Die gesamte Verarbeitung des Aufrufs, die sich speziell auf ein Messaging-System bezieht, wird an eine untergeordnete Schicht delegiert. Im folgenden Diagramm ist der IBM MQ-Messaging-Provider in der unteren Schicht gemeinsam mit zwei weiteren Messaging-Providern (Messaging-Provider A und Messaging-Provider B) dargestellt.

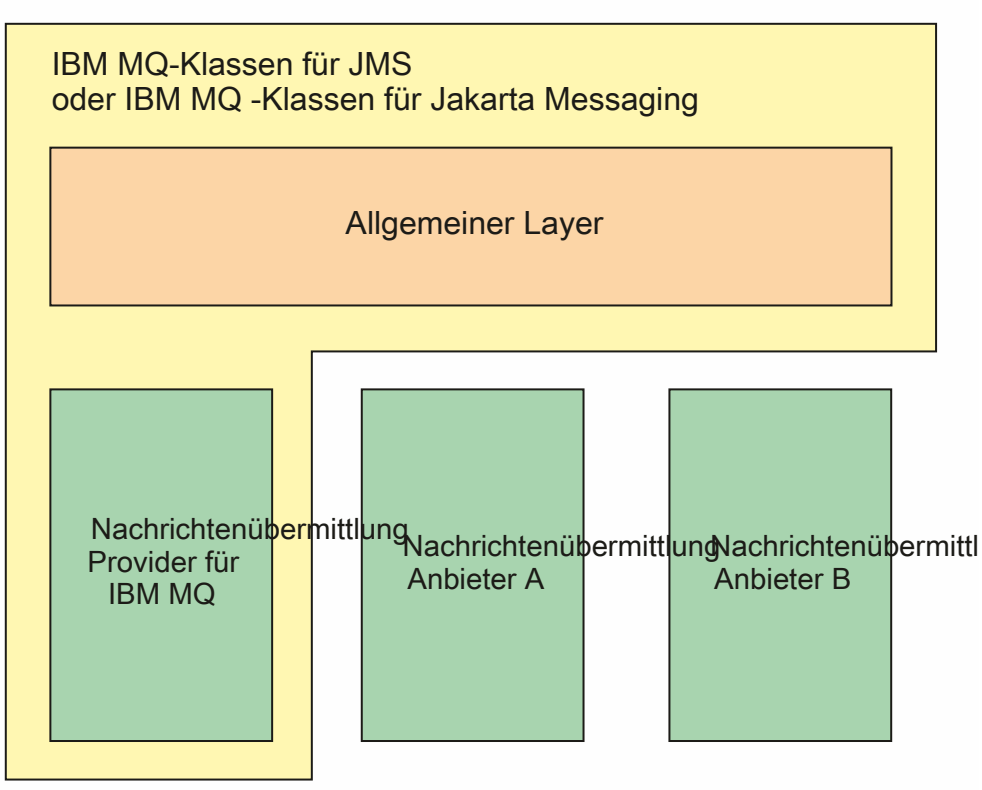


Abbildung 54. Schichtarchitektur für IBM JMS -und Jakarta Messaging -Provider

Eine Schichtarchitektur hat folgende Vorteile:

- Zur Verbesserung der Konsistenz des Verhaltens der verschiedenen IBM JMS -und Jakarta Messaging -Provider
- Vereinfachung des Schreibens einer Bridge-Anwendung zwischen zwei IBM Messaging-Systemen
- Um das Portieren einer Anwendung von einem IBM JMS -oder Jakarta Messaging -Provider auf einen anderen zu vereinfachen

Zugehörige Tasks

IBM MQ -Klassen für JMS/Jakarta Messaging verwenden

Unterstützung für verwaltete Objekte

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging unterstützen die Verwendung verwalteter Objekte.

JM 3.0 Ab IBM MQ 9.3.0wird Jakarta Messaging 3.0 für die Entwicklung neuer Anwendungen unterstützt. IBM MQ 9.3.0 und höher unterstützen weiterhin JMS 2.0 für vorhandene Anwendungen. Die Verwendung der Jakarta Messaging 3.0 -API und der JMS 2.0 -API in derselben Anwendung wird nicht unterstützt. Weitere Informationen finden Sie unter Using IBM MQ classes for JMS/Jakarta Messaging.

Der Logikablauf innerhalb einer JMS -oder IBM MQ classes for Jakarta Messaging -Anwendung beginnt mit ConnectionFactory -und Destination-Objekten. Mit dem Objekt ConnectionFactory erstellt die Anwendung ein Connection-Objekt, das die aktive Verbindung von der Anwendung zum Messaging-Server darstellt. Mit dem Objekt Connection erstellt die Anwendung ein Session-Objekt, das ein einzelner Threaded-Kontext für die Produzierung und Konsumierung von Nachrichten ist. Aus diesem Session-Objekt erstellt die Anwendung dann mit einem Destination-Objekt ein MessageProducer-Objekt, mit dessen Hilfe sie Nachrichten an das angegebene Ziel senden kann. Das Ziel ist entweder eine Warteschlange oder ein Thema im Messaging-System, das im Destination-Objekt gekapselt ist. Aus dem Session- und dem Destination-Objekt kann die Anwendung auch ein MessageConsumer-Objekt erstellen, über das sie die an das angegebene Ziel gesendeten Nachrichten empfängt.

Die Spezifikationen JMS und Jakarta Messaging erwarten, dass ConnectionFactory -und Destination-Objekte verwaltete Objekte sind. Ein Administrator erstellt und verwaltet verwaltete Objekte in einem zentralen Repository und eine JMS -oder Jakarta Messaging -Anwendung ruft diese Objekte mithilfe der Java Naming Directory Interface (JNDI) ab. Das Repository der verwalteten Objekte kann von einer einfachen Datei bis zu einem LDAP-Verzeichnis (Lightweight Directory Access Protocol) reichen.

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging unterstützen die Verwendung verwalteter Objekte. Eine Anwendung kann alle Funktionen von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging verwenden, die über IBM MQ bereitgestellt werden, ohne dass IBM MQspezifische Informationen fest in der Anwendung selbst codiert sind. Dadurch bleibt die Anwendung bis zu einem gewissen Grad von der zugrundeliegenden IBM MQ-Konfiguration unabhängig.

Um diese Unabhängigkeit zu erreichen, kann die Anwendung JNDI verwenden, um Verbindungsfactorys und Ziele abzurufen, die als verwaltete Objekte gespeichert sind, und nur die im Paket `javax.jms` (JMS 2.0) oder `jakarta.jms` (Jakarta Messaging 3.0) definierten Schnittstellen verwenden, um Messaging-Operationen auszuführen.

JMS 2.0 Für JMS 2.0kann ein Administrator das IBM MQ JMS -Verwaltungstool **JMSAdmin**oder IBM MQ Explorerverwenden, um verwaltete Objekte in einem zentralen Repository zu erstellen und zu verwalten.

JM 3.0 Für Jakarta Messaging 3.0können Sie JNDI nicht mit IBM MQ Explorerverwalten. Die JNDI-Verwaltung wird von der Variante Jakarta Messaging 3.0 von **JMSAdmin**(**JMS30Admin**) unterstützt.

Ein Anwendungsserver stellt normalerweise ein eigenes Repository für verwaltete Objekte und eigene

Tools für die Erstellung und Verwaltung der Objekte bereit. Eine Java EE JM 3.0 -oder Jakarta EE -Anwendung kann daher JNDI verwenden, um verwaltete Objekte entweder aus dem Repository des Anwendungsservers oder aus einem zentralen Repository abzurufen.

Zugehörige Tasks

JMS-und Jakarta Messaging-Ressourcen konfigurieren

Unterstützte Kommunikationstypen auf Java EE -und Jakarta EE -Plattformen

Auf den Plattformen Java EE und Jakarta EE unterstützen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging zwei Arten der Kommunikation zwischen einer Komponente einer Anwendung und einem IBM MQ -Warteschlangenmanager.

JM 3.0 IBM MQ 9.3.0 wurde die Unterstützung für Jakarta Messaging 3.0eingeführt. JMS 2.0 wird weiterhin vollständig unterstützt. Da JMS und Jakarta Messaging viel gemeinsam genutzt werden, können weitere Verweise auf JMS in diesem Abschnitt als Verweise auf beide Elemente betrachtet werden. Alle Unterschiede werden nach Bedarf hervorgehoben.

Die folgenden beiden Arten der Kommunikation zwischen einer Anwendungskomponente und einem IBM MQ-Warteschlangenmanager werden unterstützt:

- Abgehende Kommunikation
- Eingehende Kommunikation

Abgehende Kommunikation

Mithilfe der JMS -oder Jakarta Messaging -API erstellt eine Anwendungskomponente eine Verbindung zu einem Warteschlangenmanager und sendet und empfängt anschließend Nachrichten.

Bei der Anwendungskomponente kann es sich zum Beispiel um einen Anwendungsclient, ein Servlet, eine Java Server Page (JSP), ein Enterprise Java Bean (EJB) oder ein Message-driven Bean (MDB) handeln. Bei dieser Art der Kommunikation stellt der Anwendungsserver-Container nur Low-Level-Funktionen für Messaging-Operationen bereit, beispielsweise Verbindungspooling und Thread-Management.

Eingehende Kommunikation

Im Falle der eingehenden Kommunikation wird eine auf einem Ziel eingehende Nachricht einem MDB zugestellt, der diese Nachricht dann verarbeitet.

Java EE JM 3.0 -und Jakarta EE -Anwendungen verwenden MDBs zur asynchronen Verarbeitung von Nachrichten. Ein MDB fungiert als JMS-Nachrichtenlistener und wird durch eine onMessage()-Methode implementiert, die festlegt, wie eine Nachricht verarbeitet wird. Ein MDB wird im EJB-Container des Anwendungsservers implementiert. Die Konfiguration eines MDB ist vom verwendeten Anwendungsserver abhängig. Jedoch muss in der Konfiguration angegeben sein, mit welchem Warteschlangenmanager die Verbindung hergestellt werden soll, wie diese Verbindung erfolgen soll, welches Ziel auf Nachrichten überwacht werden soll und wie sich das MDB hinsichtlich Transaktionen verhalten soll. Diese Informationen werden vom EJB-Container verwendet. Wenn eine Nachricht, die die Auswahlkriterien der MDB erfüllt, am angegebenen Ziel ankommt, verwendet der EJB-Container IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging , um die Nachricht vom Warteschlangenmanager abzurufen, und stellt die Nachricht dann der MDB zu, indem er ihre Methode onMessage() aufruft.

Beziehung zu IBM MQ classes for Java

IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging und IBM MQ classes for JMS sind Peers, die eine gemeinsame Java -Schnittstelle zur MQI verwenden.

Abbildung 55 auf Seite 173 zeigt die Beziehung zwischen IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging und IBM MQ classes for Java.

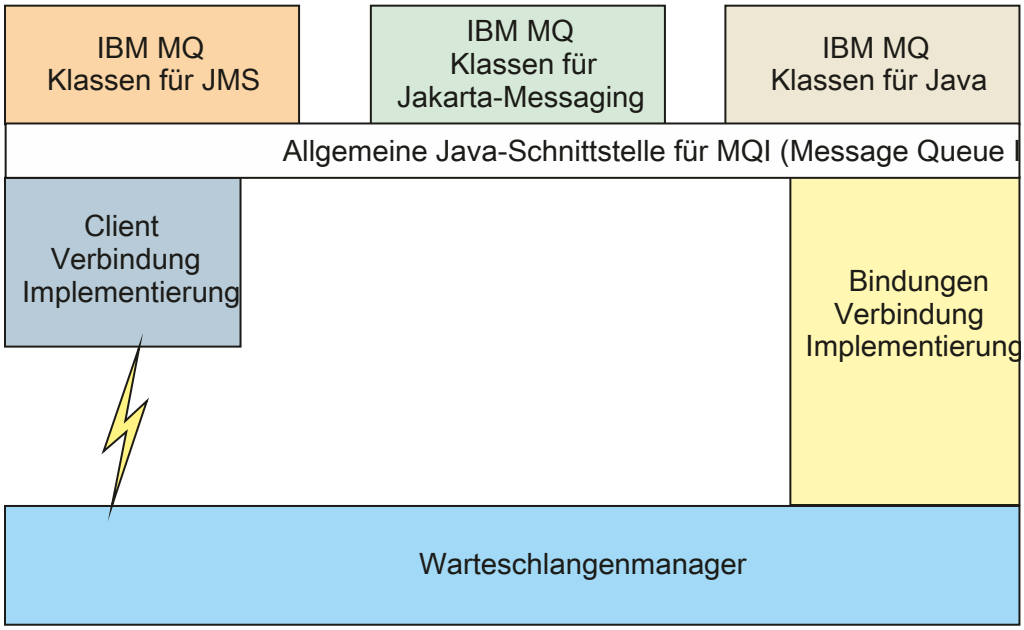


Abbildung 55. Beziehung zwischen IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging und IBM MQ classes for Java

Im Allgemeinen sollten Java -Programme nur eine Schnittstelle für die Schnittstelle mit IBM MQ - IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging oder IBM MQ classes for JMSverwenden. Das Mischen von Schnittstellen wird nicht unterstützt, mit einer Ausnahme. Zur Aufrechterhaltung der Kompatibilität mit Versionen vor IBM WebSphere MQ 7.0 können in Java geschriebene Kanalexit-Klassen nach wie vor die IBM MQ classes for Java-Schnittstellen verwenden, selbst wenn die Kanalexit-Klassen aus IBM MQ classes for JMS aufgerufen werden. Die Verwendung der IBM MQ classes for Java -Schnittstellen bedeutet jedoch, dass Ihre Anwendungen weiterhin von folgenden Komponenten abhängig sind:

- JMS 2.0 Die IBM MQ classes for Java -JAR-Datei `com.ibm.mq.jar` Wenn Sie `com.ibm.mq.jar` nicht in Ihren Klassenpfad aufnehmen möchten, können Sie stattdessen die Schnittstellengruppe im Paket `com.ibm.mq.exits` verwenden.
- JM 3.0 Verwendung von `com.ibm.mq.jakarta.client.jar`bei der Interaktion mit IBM MQ classes for Jakarta Messaging.

Zugehörige Konzepte

Warum sollte ich IBM MQ -Klassen für Jakarta Messaging verwenden?

Vorteile von IBM MQ-Klassen für JMS

Warum sollte ich IBM MQ -Klassen für Javaverwenden?

IBM MQ-Messaging-Provider

Der IBM MQ-Messaging-Provider hat drei Betriebsmodi, den Normalmodus, den Normalmodus mit Einschränkungen und den Migrationsmodus.

Der IBM MQ-Messaging-Provider hat drei Betriebsmodi:

- Normalmodus des IBM MQ-Messaging-Providers
- Normalmodus des IBM MQ-Messaging-Providers mit Einschränkungen
- Migrationsmodus des IBM MQ-Messaging-Providers

Der IBM MQ-Normalmodus für Messaging-Provider nutzt alle Funktionen eines IBM MQ-Warteschlangenmanagers, um JMS zu implementieren. Dieser Modus ist für die Verwendung der API und Funktionalität von JMS 2.0 JM 3.0 oder Jakarta Messaging 3.0 optimiert.

Wenn:

- Der Client gibt die Providerversion 6 auf einem **ConnectionFactory**an. Der Client verhält sich in einer Weise, die mit dem mit IBM WebSphere MQ 6.0bereitgestellten Client kompatibel ist. Es werden nur JMS 1.1 -und JMS 2-Schnittstellen unterstützt, aber einige JMS 2-Funktionen, wie z. B. gemeinsam genutzte Subskriptionen, Zustellungsverzögerung und asynchrones Senden, sind inaktiviert. Es gibt keine gemeinsame Nutzung von Verbindungen.
- Der Client gibt die Providerversion 7 in einem **ConnectionFactory**an. Die JMS-Schnittstellen 1.1 und JMS 2 werden vollständig unterstützt.
- Es wurde keine Providerversion angegeben. Es wird versucht, die Verbindung zu Provider Version 7 herzustellen. Wenn dies fehlschlägt, wird ein weiterer Versuch mit Provider Version 6 unternommen.

Wenn Sie eine Verbindung mit IBM Integration Bus über IBM MQ Enterprise Transport herstellen möchten, verwenden Sie den Migrationsmodus. Bei Verwendung von IBM MQ Real-Time Transport ist der Migrationsmodus automatisch ausgewählt, da Sie im Verbindungsfactory-Objekt explizit Eigenschaften ausgewählt haben. Die Verbindung mit IBM Integration Bus über IBM MQ Enterprise Transport folgt den allgemeinen Regeln der Modusauswahl, die im Abschnitt JMS-Eigenschaft **PROVIDERVERSION** konfigurieren beschrieben ist.

Zugehörige Tasks

JMS-Ressourcen konfigurieren

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 175
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 176
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 178
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 180
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 224

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 237
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 241
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 262
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 278
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 284
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 289
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 252
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 287
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 176 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

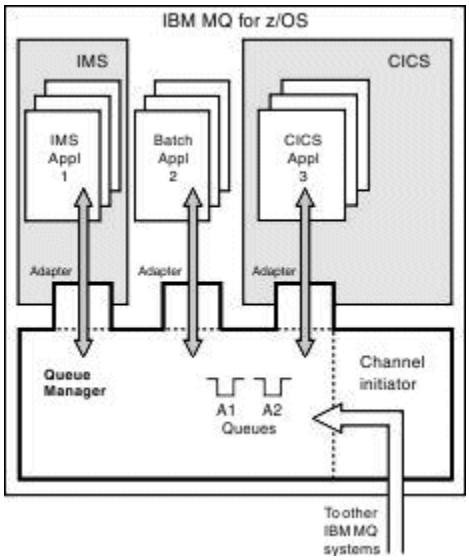


Figure 56. Overview of IBM MQ for z/OS

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 177 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.

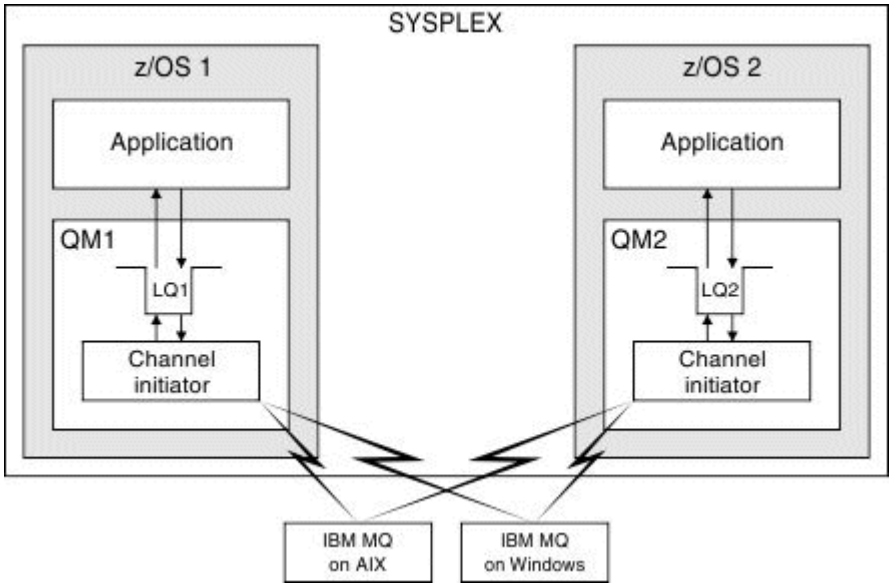


Figure 57. Communication between queue managers

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 241.

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 252.

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 262.

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 278.

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 284.

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 287.

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 291.

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 181 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.

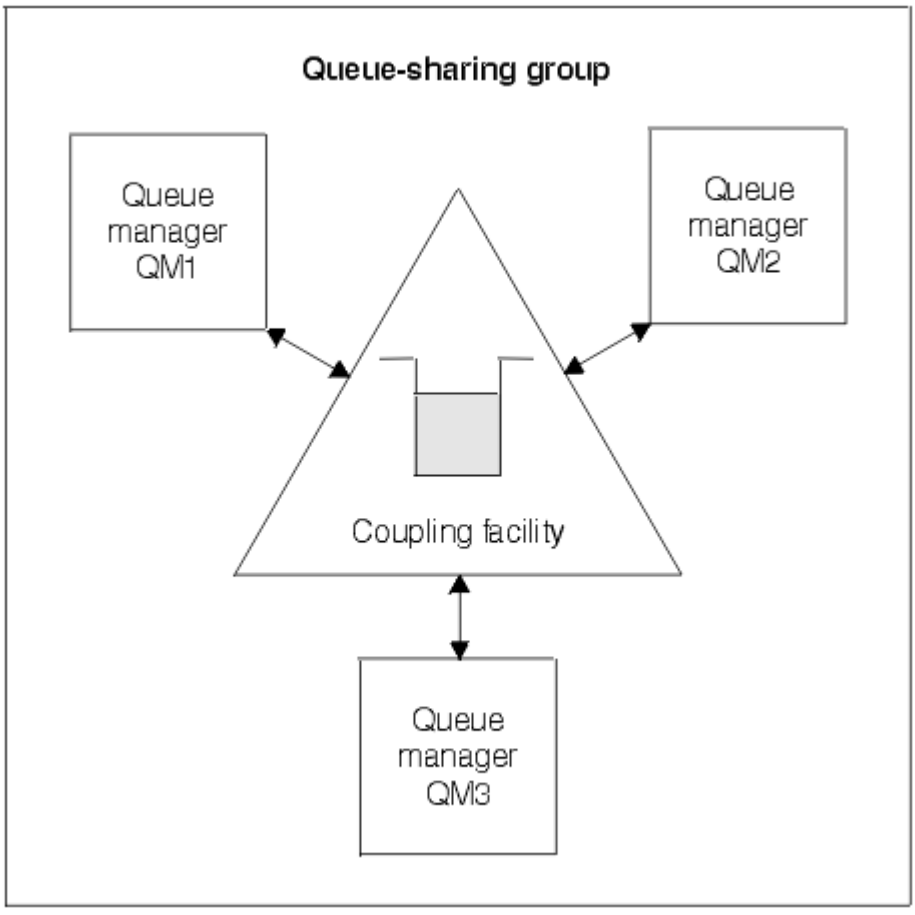


Figure 58. A queue sharing group

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 182 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.

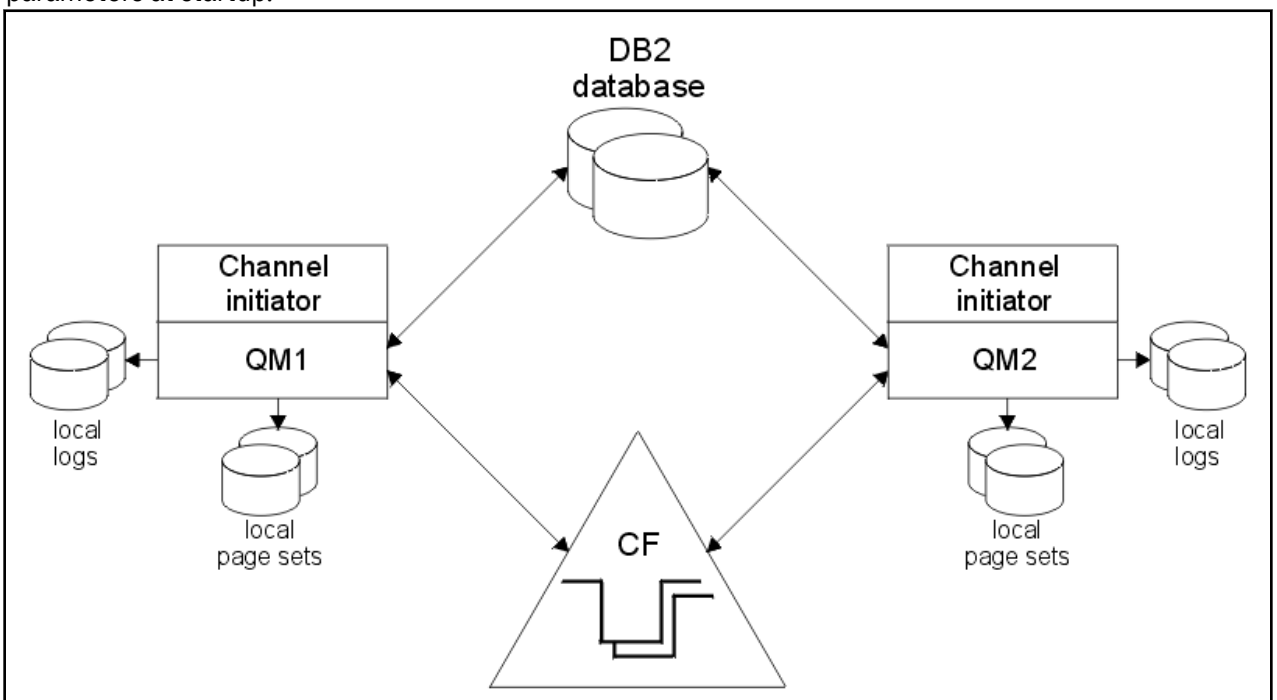


Figure 59. The components of queue managers in a queue sharing group

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 183
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 199
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 218
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 222
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 223
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 185 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 200 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 185
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 187
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 198

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.

[1] `(Hits / (Hits+Misses))* 100`

- 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 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 200.

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 218.

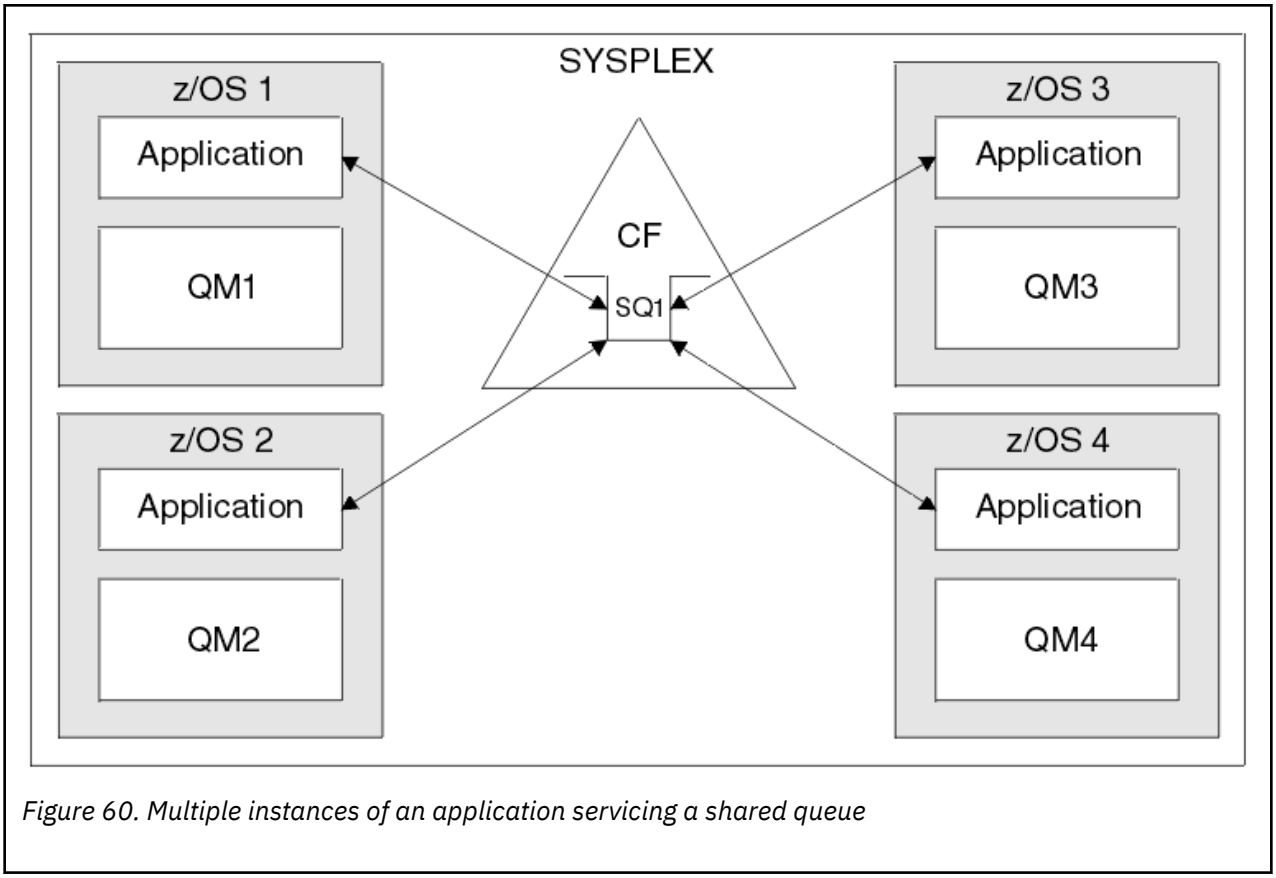


Figure 60. Multiple instances of an application servicing a shared queue

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 207
- “Improved performance - basic configuration” on page 213

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 202 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 208.

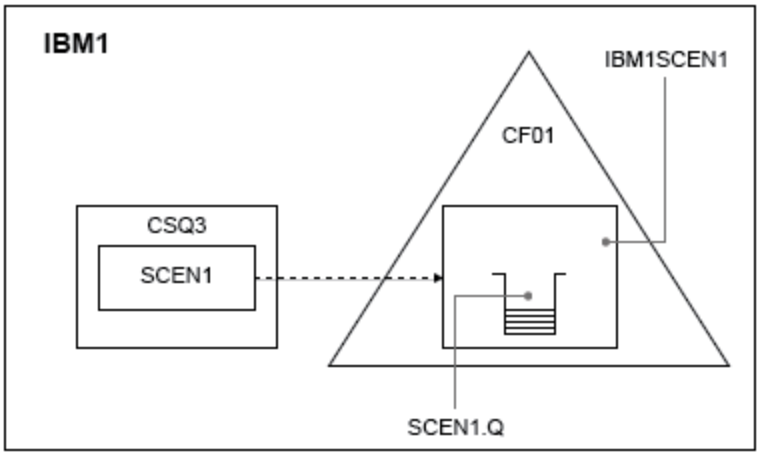


Figure 61. Basic configuration

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 200

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 207. 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 210.

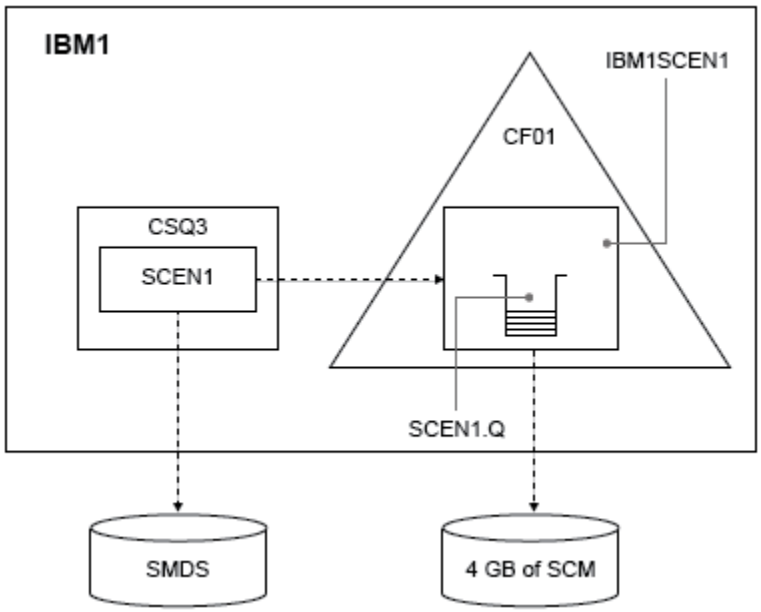


Figure 62. Configuration adding SMDS and SCM for emergency storage

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 212 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 214.

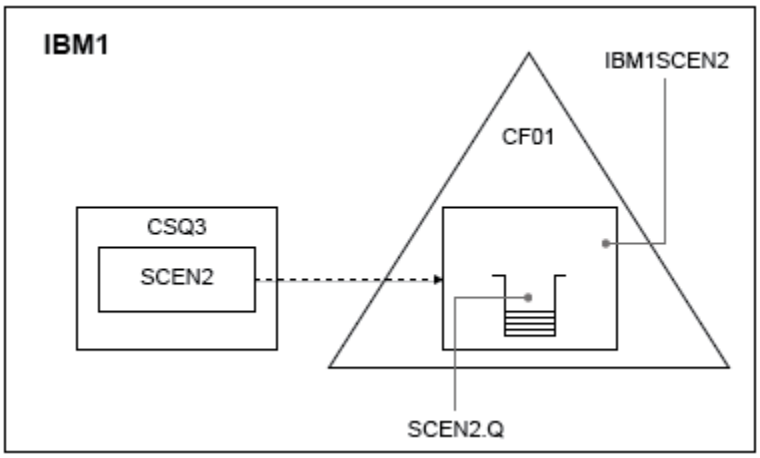


Figure 63. Basic configuration

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 216.

Related concepts

“Use of storage class memory with shared queues” on page 200

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 213. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 217.

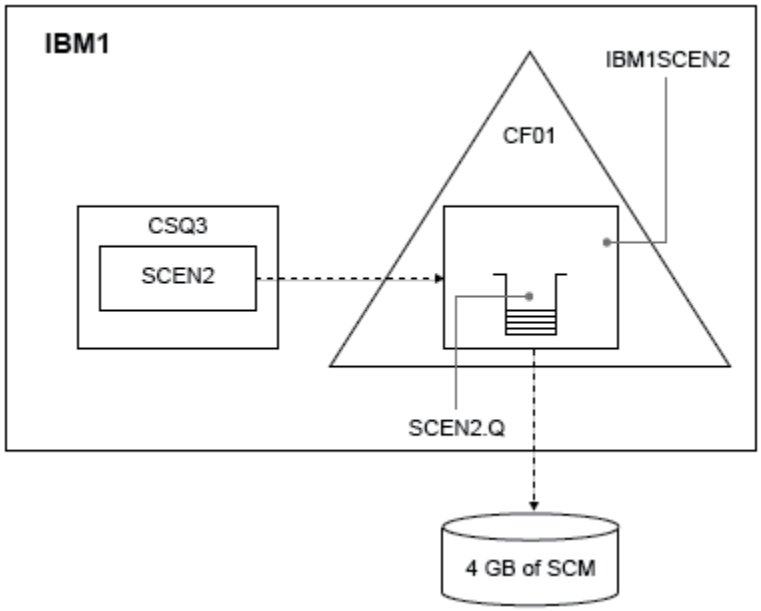


Figure 64. Configuration adding SCM for improved performance

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 213:
  - 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 213.

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 219 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.

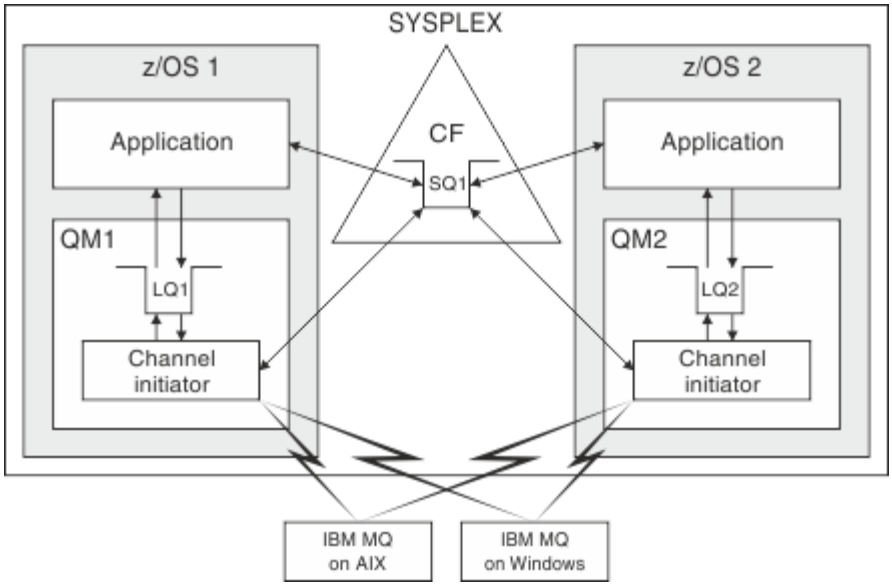


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 219
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 224
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 221
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 222 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

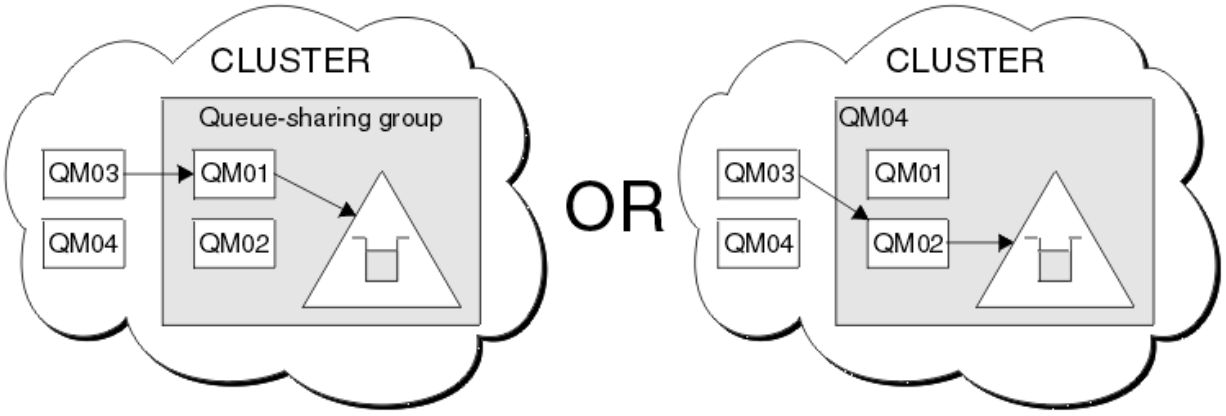


Figure 66. A queue sharing group as part of a cluster

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 262
Queue sharing group security	“Security concepts in IBM MQ for z/OS” on page 278
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 180
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 180.

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.

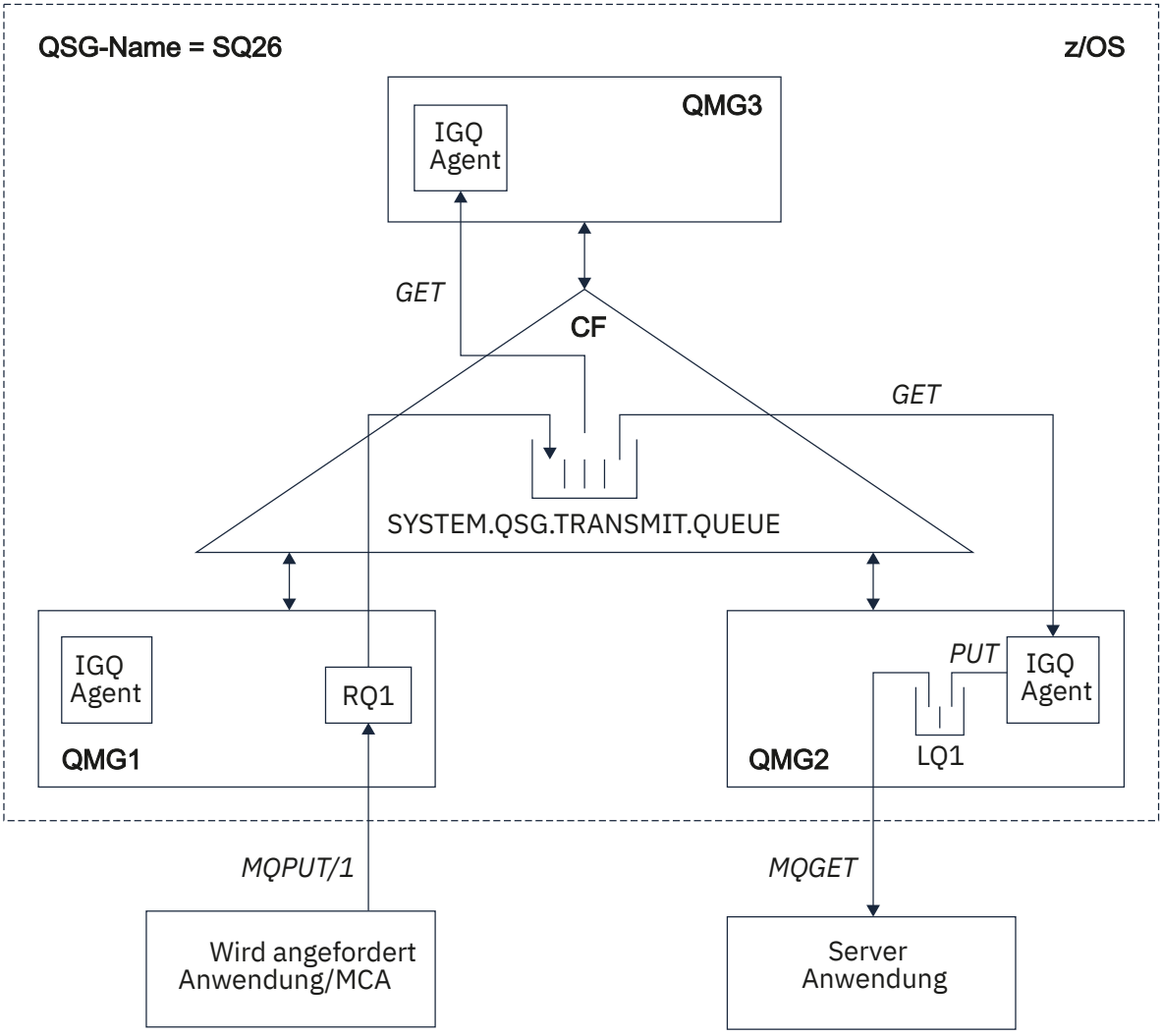


Figure 67. An example of intra-group queuing

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.

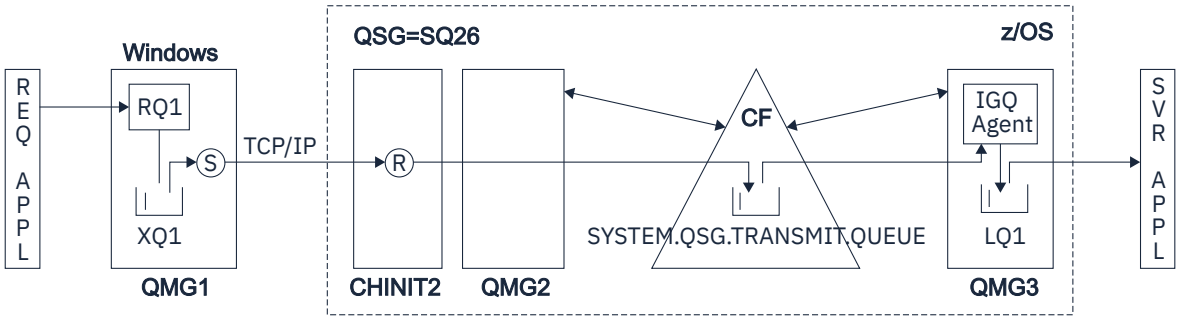


Figure 68. An example of migration support

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 236 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 228 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 236 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 228 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 224 describes the typical configuration.

Related concepts

“Distributed queuing with intra-group queuing (multiple delivery paths)” on page 229
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 231
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 233
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.

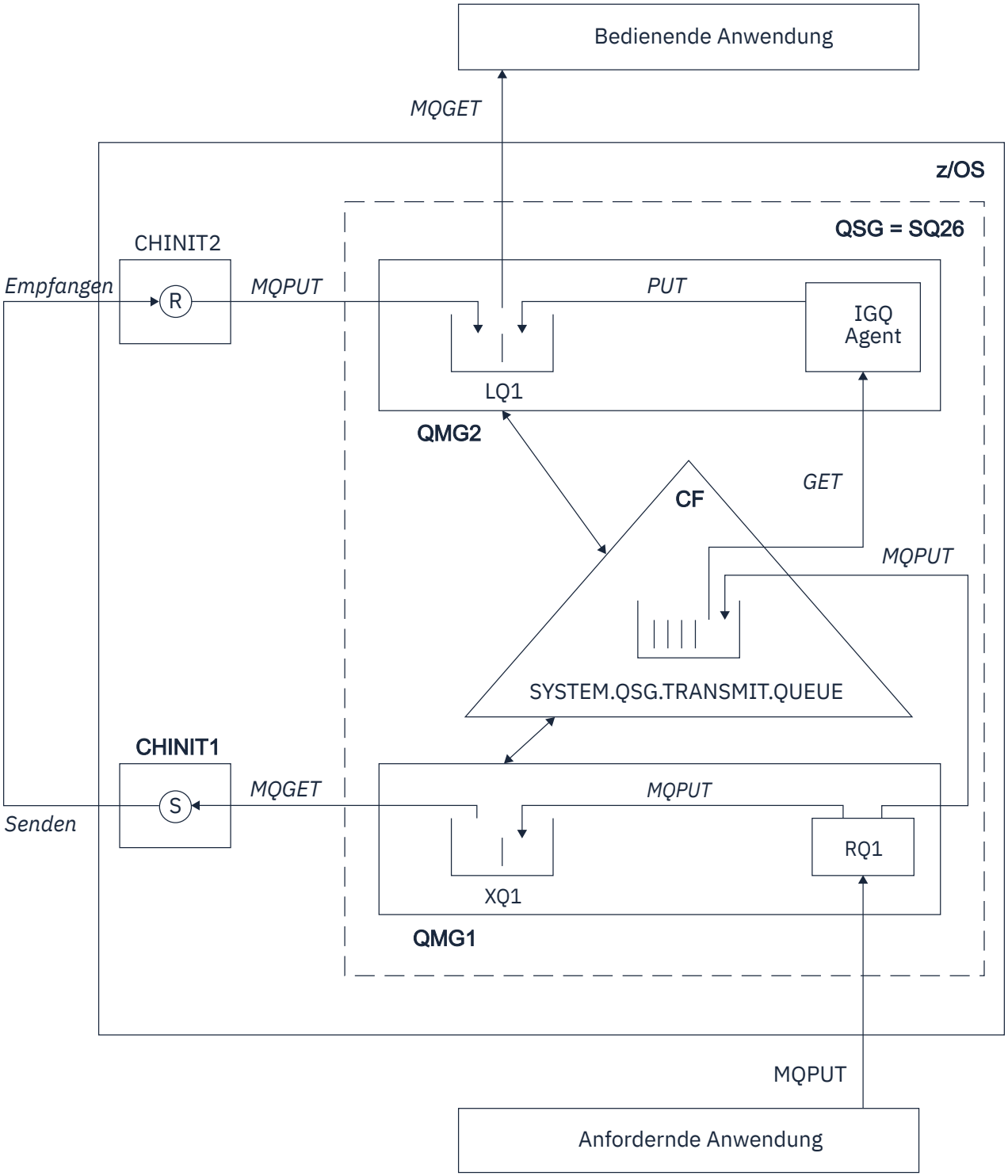


Figure 69. An example configuration

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.TRANS-

MIT.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).

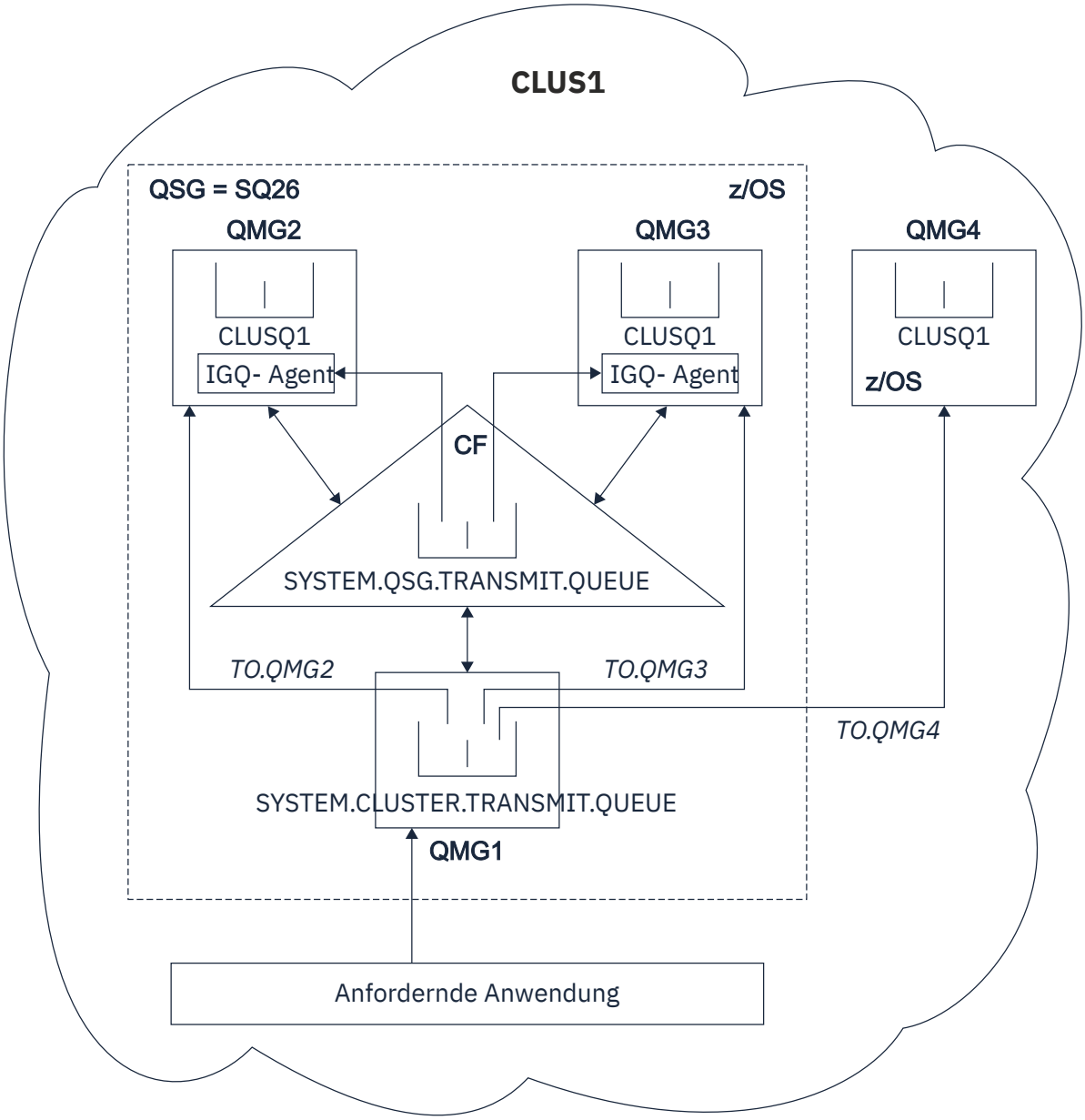


Figure 70. An example of clustering with intra-group queuing

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 229.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 231.

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 234
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 235
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 236.
- 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 236.
- 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 237
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 238
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 240
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 241
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 180, 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 293).

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 239, 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.

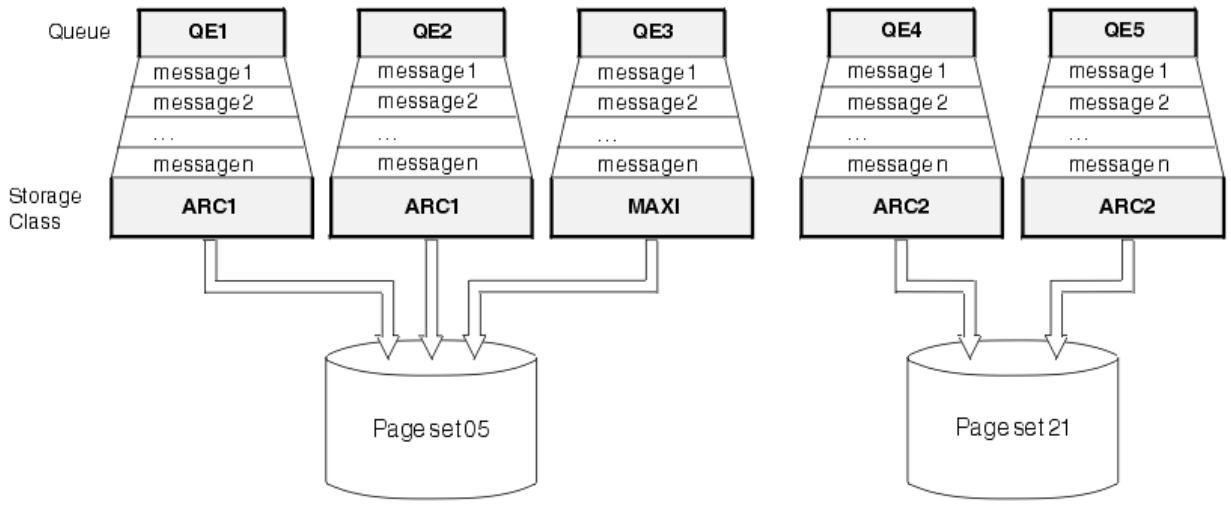


Figure 71. Mapping queues to page sets through storage classes

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 240, 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 240 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.

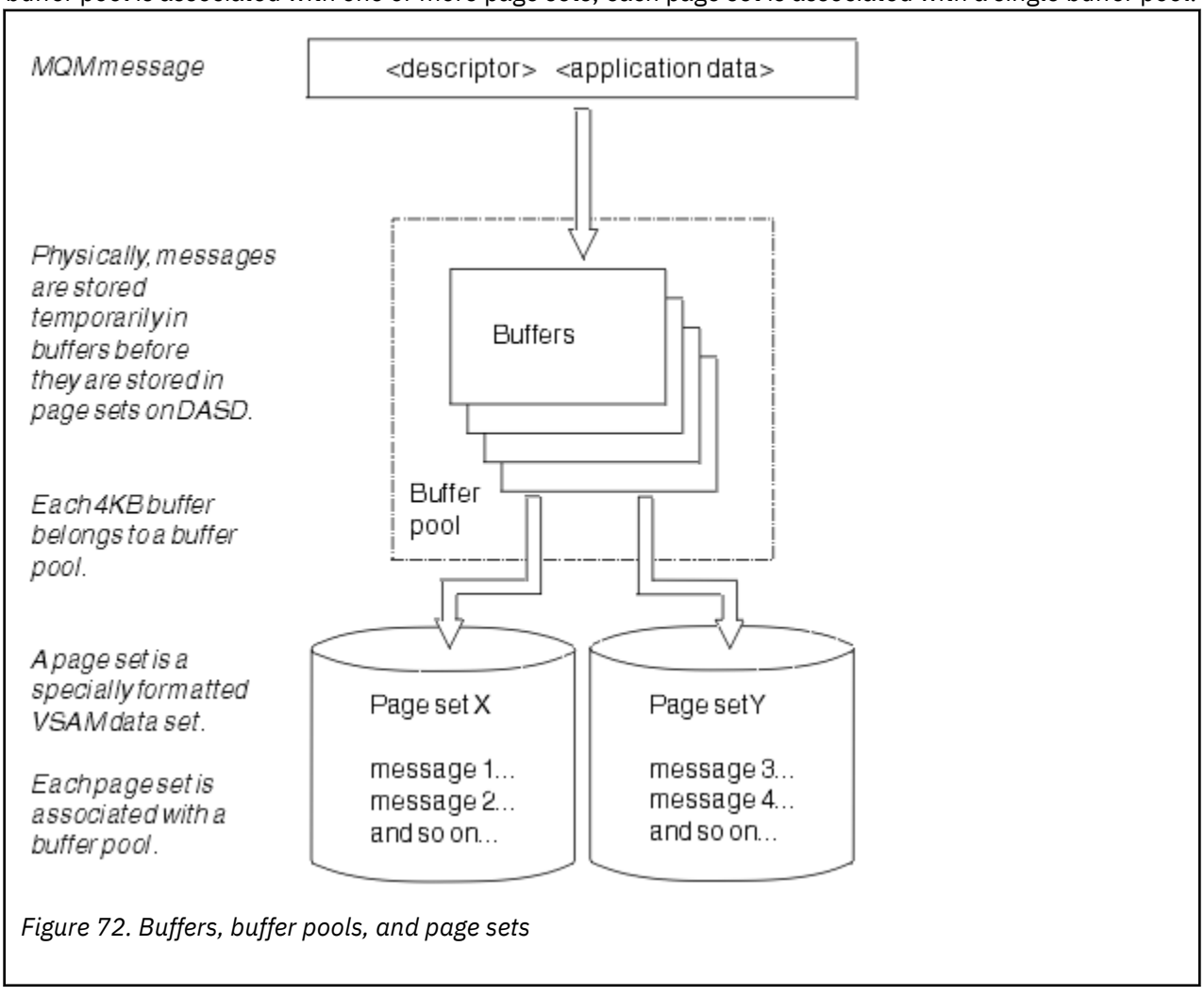


Figure 72. Buffers, buffer pools, and page sets

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 242
- “How the log is structured” on page 245
- “How the IBM MQ for z/OS logs are written” on page 246
- “Larger log Relative Byte Address” on page 249
- “The bootstrap data set” on page 250

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 247 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.

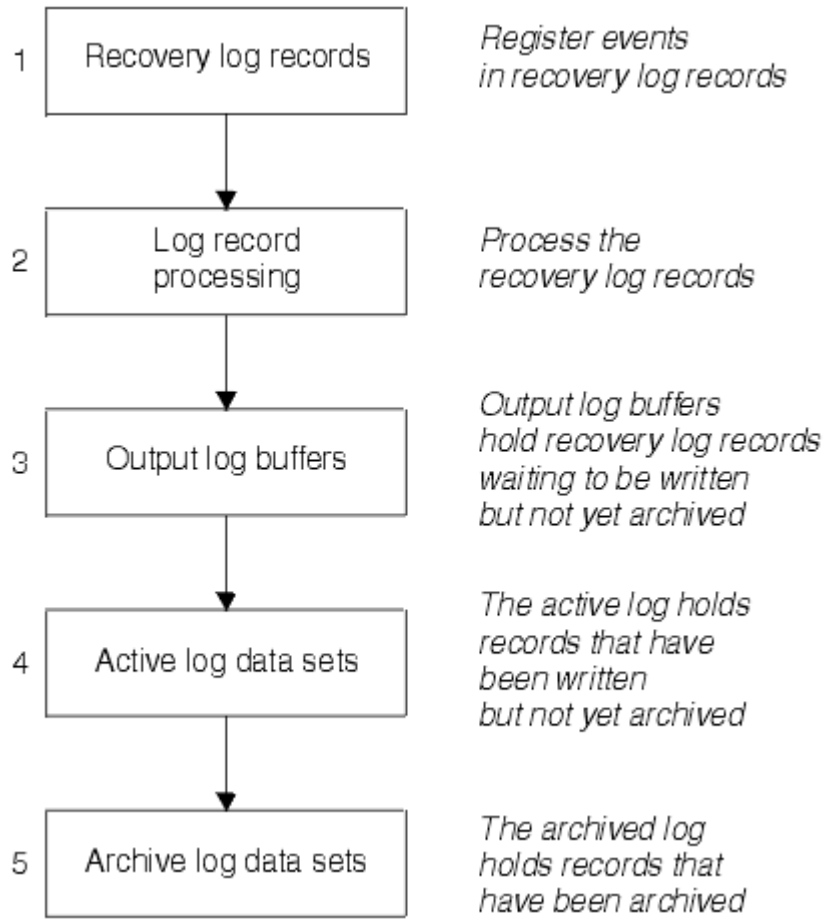


Figure 73. The logging process

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 248
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 248.

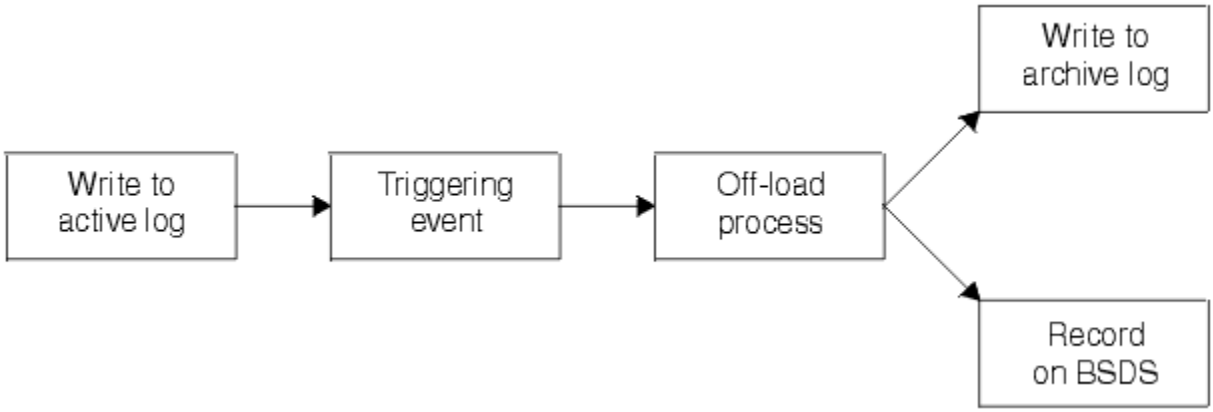


Figure 74. The offloading process

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 verwalten. 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 verwalten.

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 249 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 252
- “Tuning your queue manager on IBM MQ for z/OS” on page 257
- “Sample definitions supplied with IBM MQ for z/OS” on page 258

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 256

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 -Kapazitätsplanung und -optimierung 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 263
- “How consistency is maintained in IBM MQ for z/OS” on page 264
- “What happens during termination in IBM MQ for z/OS” on page 266
- “What happens during restart and recovery in IBM MQ for z/OS” on page 267
- “How in-doubt units of recovery are resolved” on page 269
- “Shared queue recovery” on page 272

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.

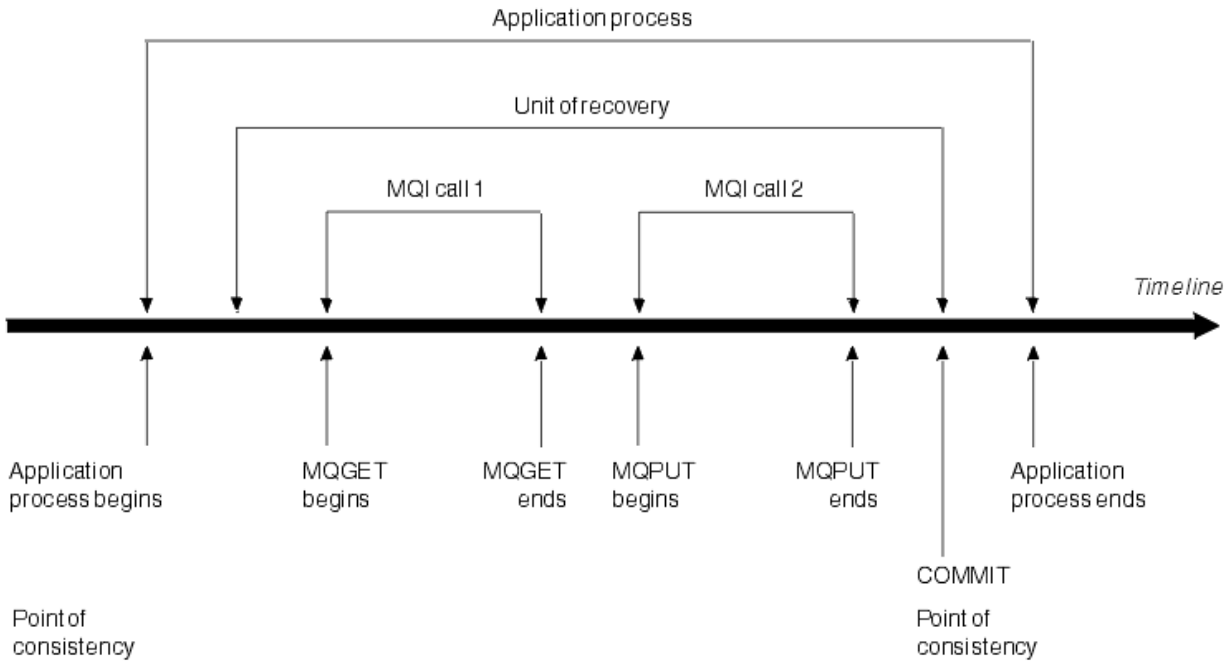


Figure 75. A unit of recovery within an application program

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 263 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 SYNC-POINT.

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 264.

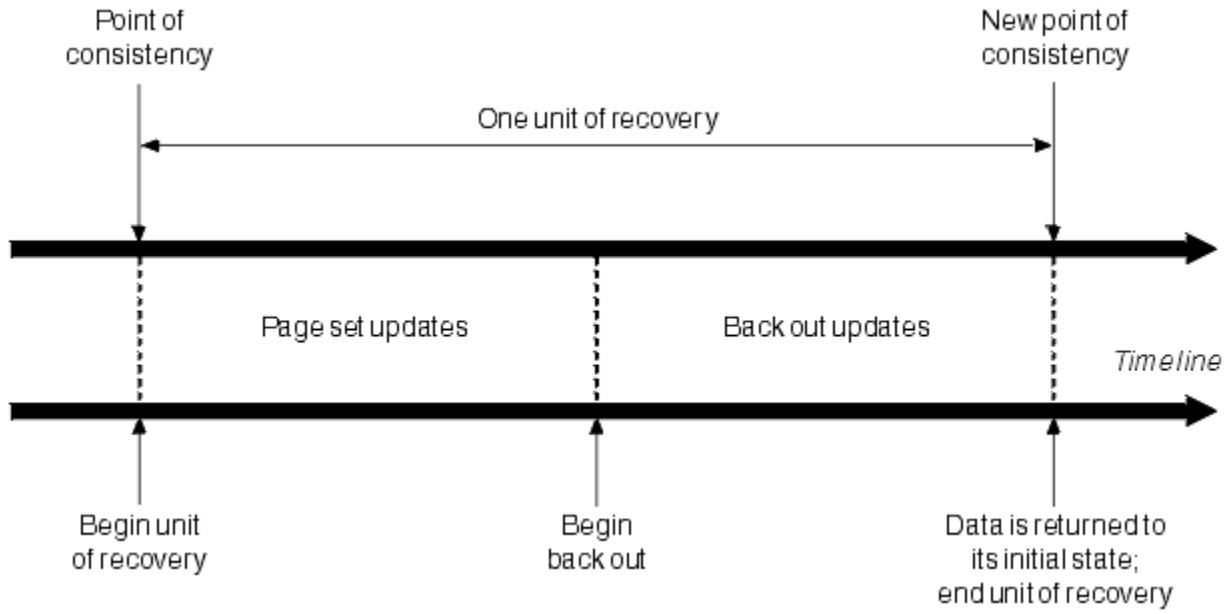


Figure 76. A unit of recovery showing back out

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 265 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.

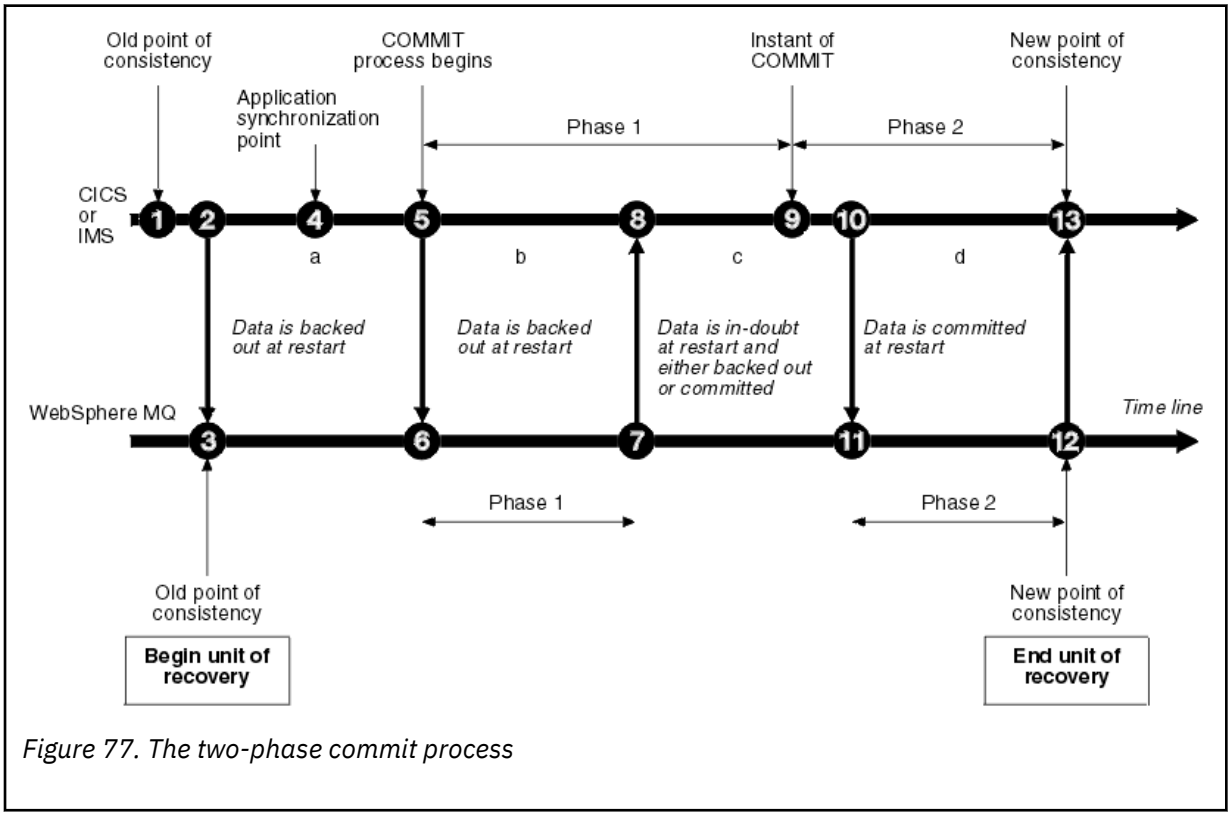


Figure 77. The two-phase commit process

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 265 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 267.

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 269).

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 -Nachrichten, -Beendigungscodes und -Ursachencodes 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 verwalten.

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 verwalten.

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 -Nachrichten, -Beendigungscodes und -Ursachencodes 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 verwalten.

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- System-Befehlsreferenz.

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 282.

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 283
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 199.

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 272.

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 286.

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 219.

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 180
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 288

- “IBM MQ trace” on page 288
- “Events” on page 288

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 289
A queue sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 290
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 289 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 290.

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 290
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 291
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 Basic Skills: Dynamische virtuelle Adressierung 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 291
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 298
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 293
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 296
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 292
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 289

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 294.

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 295.

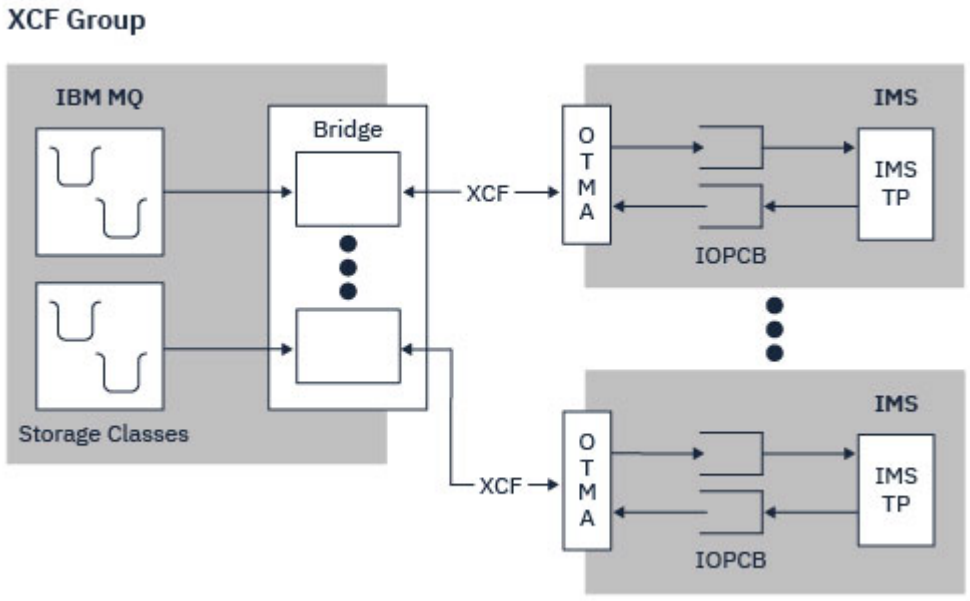


Figure 78. The IBM MQ - IMS bridge

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 293

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

Table 25. Where to find more information about using z/OS Batch with IBM MQ (continued)	
Topic	**Where to look**
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.

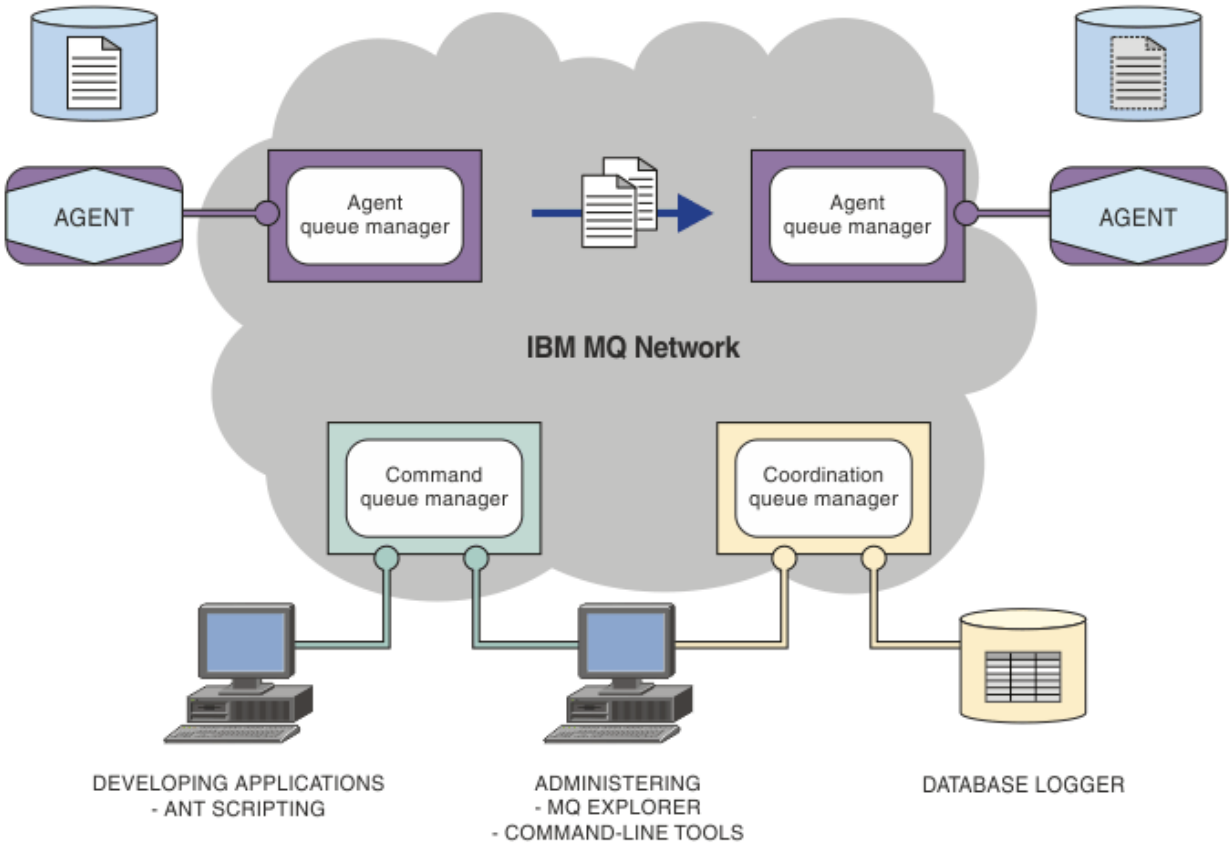
Using IBM MQ functions from JMS applications

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 überträgt Dateien auf kontrollierte und überprüfbare Weise zwischen Systemen - unabhängig von der Dateigröße und den verwendeten Betriebssystemen.

Mit Managed File Transfer können Sie eine angepasste, skalierbare und automatisierte Lösung erstellen, mit der Sie Dateiübertragungen verwalten, sichern und schützen können. Managed File Transfer kommt ganz ohne kostspielige Redundanzen aus. Es verringert die Wartungskosten und maximiert die Ergebnisse aus Ihren IT-Investitionen.



Das Diagramm zeigt eine einfache Managed File Transfer-Topologie. Die Topologie enthält zwei Agenten, von denen jeder innerhalb des IBM MQ-Netzes mit seinem eigenen Agentenwarteschlangenmanager verbunden ist. Die Dateiübertragung findet vom Agenten auf der einen Seite des Diagramms über das IBM MQ-Netz zum Agenten auf der anderen Seite des Diagramms statt. Das IBM MQ-Netz enthält außerdem einen Koordinationswarteschlangenmanager und einen Befehlswarteschlangenmanager. Anwendungen und Tools stellen Verbindungen zu diesen Warteschlangenmanagern her, um Managed File Transfer-Aktivitäten im IBM MQ-Netz zu konfigurieren, zu verwalten, auszuführen und zu protokollieren.

Für Managed File Transfer stehen vier Installationsoptionen zur Auswahl; für welche Option Sie sich entscheiden, hängt von Ihrem Betriebssystem und von Ihrer Konfiguration ab. Diese Optionen sind Managed File Transfer Agent, Managed File Transfer Logger, Managed File Transfer Service oder Managed File Transfer Tools. Weitere Informationen finden Sie im Abschnitt Produktoptionen für Managed File Transfer.

Mit Managed File Transfer können Sie folgende Tasks ausführen:

- Gesteuerte Dateiübertragungen erstellen
  - Windows Linux Neue Dateiübertragungen aus IBM MQ Explorer auf Linux -oder Windows -Plattformen erstellen.
  - Auf allen Plattformen neue Dateiübertragungen über die Befehlszeile erstellen.
  - Dateiübertragungsfunktion in das Tool Apache Ant integrieren.
  - Anwendungen schreiben, die Managed File Transfer durch Übergabe von Nachrichten in die Befehlswarteschlangen der Agenten steuern.
  - Dateiübertragungen planen, sodass sie zu einem späteren Zeitpunkt ausgeführt werden. Außerdem können geplante Dateiübertragungen basierend auf einer Reihe von Dateisystemereignissen ausgelöst werden, z. B. beim Erstellen einer neuen Datei.

  - Eine Ressource (beispielsweise ein Verzeichnis) ständig überwachen; erfüllt der Inhalt dieser Ressource eine vordefinierte Bedingung, wird eine bestimmte Task gestartet. Bei dieser Task kann es sich um eine Dateiübertragung, ein Ant-Script oder einen JCL-Job handeln.
  - Dateien zu und von IBM MQ-Warteschlangen übertragen.
  - Dateien zu und von FTP-, FTPS- oder SFTP-Servern übertragen.
  - Dateien zu und von Connect:Direct-Knoten übertragen.
  - Sowohl Textdateien als auch binäre Dateien übertragen. Bei Textdateien werden Codepages und Zeilenendekonventionen automatisch zwischen den Quellen- und Zielsystemen konvertiert.
  - Übertragungen können gesichert werden. Dazu werden die Industriestandards für auf Secure Socket Layer (SSL) basierende Verbindungen angewendet.
- Laufende Übertragungen anzeigen und Informationen zu allen Übertragungen in Ihrem Netz protokollieren
  - Windows Linux Zeigen Sie den Status von Übertragungen in Bearbeitung von IBM MQ Explorer auf Linux -oder Windows -Plattformen an.
  - Windows Linux Überprüfen Sie den Status abgeschlossener Übertragungen mithilfe der IBM MQ Explorer auf Linux -oder Windows -Plattformen.
  - Mit der Datenbankprotokollfunktion von Managed File Transfer Protokollnachrichten in einer Db2- oder Oracle-Datenbank speichern.

Managed File Transfer setzt aufIBM MQ auf und stellt somit die gleiche sichere und zuverlässige Zustellung von Nachrichten zwischen Anwendungen bereit. Auch in diesem Tool können Sie die Vorteile der verschiedenen Funktionen von IBM MQ nutzen. Beispielsweise können Sie mit einem Kanalausdruck die zwischen Agenten über IBM MQ-Kanäle übertragenen Daten komprimieren und mittels SSL-Kanälen die zwischen Agenten ausgetauschten Daten schützen. Dateien können zuverlässig übertragen werden, mit Fehlertoleranz gegenüber Ausfällen der Infrastruktur, über die die Dateiübertragung ausgeführt wird. Bei einem Netzausfall wird die Dateiübertragung am Abbruchpunkt neu gestartet, sobald die Verbindung wiederhergestellt ist.

Durch die Konsolidierung der Dateiübertragungsfunktion mit Ihrem bestehenden IBM MQ-Netz sind keine weiteren Ressourcen mehr zur Verwaltung einer zweiten Infrastruktur erforderlich. Wenn Sie noch kein IBM MQ-Kunde sind, errichten Sie durch die Einrichtung eines IBM MQ-Netzes zur Unterstützung von Managed File Transfer den Backbone für eine zukünftige SOA-Implementierung. Wenn Sie IBM MQ bereits verwenden, können Sie mit Managed File Transfer auch die Vorteile Ihrer bestehenden IBM MQ-Infrastruktur einschließlich IBM MQ Internet Pass-Thru und IBM Integration Bus nutzen.

Sie können IBM MQ -Hochverfügbarkeitslösungen nutzen, um die Ausfallsicherheit Ihrer Managed File Transfer -Konfiguration zu verbessern. Wenn Ihre Agenten replizierte Datenwarteschlangenmanager (RDQMs) verwenden, müssen Sie sie für die Verwendung der Funktion für variable IP-Adressen konfigurieren. Dies bedeutet, dass Agenten dieselbe IP-Adresse für die Kommunikation mit jeder der drei RDQM-Instanzen verwenden, die derzeit ausgeführt wird, und die Verbindung bei einem Failover automatisch wiederherstellen (siehe RDQM-Hochverfügbarkeit und Variable IP-Adresse erstellen und löschen). Wenn Sie die Multi-Instanz-Warteschlangenmanager-Lösung verwenden, verwenden Anwendungen eine andere IP-Adresse für die Kommunikation mit jeder Instanz, die von der Wiederherstellung der Clientverbindung bei der Übernahme verarbeitet wird (siehe Multi-Instanz-Warteschlangenmanager und Kanal und Clientverbindung).

Managed File Transfer lässt sich mit verschiedenen IBM-Produkten integrieren:

IBM Integration Bus

Mit diesem Produkt können Sie Dateien verarbeiten, die von Managed File Transfer im Zuge eines IBM Integration Bus Nachrichtenflusses übertragen wurden. Weitere Informationen finden Sie im Abschnitt Mit MFT aus IBM Integration Bus arbeiten.

IBM Sterling Connect:Direct

Übertragen Sie Dateien zu und von einem vorhandenen Connect:Direct-Netz unter Verwendung der Managed File Transfer Connect:Direct-Bridge. Weitere Informationen finden Sie im Abschnitt Die Connect:Direct-Bridge.

IBM Tivoli Composite Application Manager

IBM Tivoli Composite Application Manager stellt einen Agenten bereit, mit dem Sie die auf dem Koordinationswarteschlangenmanager veröffentlichten Informationen überwachen können.

Zugehörige Konzepte

Produktoptionen für Managed File Transfer

„Übersicht über die MFT-Topologie" auf Seite 302
Dieser Abschnitt gibt eine Übersicht darüber, wie Managed File Transfer-Agenten mit dem Koordinationswarteschlangenmanager in einem IBM MQ-Netz verbunden sind.

„Funktionsweise von MFT mit IBM MQ" auf Seite 301
Managed File Transfer interagiert auf verschiedene Weise mit IBM MQ.

Funktionsweise von MFT mit IBM MQ

Managed File Transfer interagiert auf verschiedene Weise mit IBM MQ.

- Managed File Transfer überträgt Dateien zwischen Agentenprozessen, indem die einzelnen Dateien in eine oder mehrere Nachrichten aufgeteilt und diese Nachrichten über das IBM MQ-Netz übertragen werden.
- Die Agentenprozesse verschieben Dateidaten unter Verwendung nicht persistenter Nachrichten, um die Auswirkungen auf Ihre IBM MQ-Protokolle zu minimieren. Durch Kommunikation untereinander steuern die Agentenprozesse den Fluss von Nachrichten mit Dateidaten. Dadurch wird die Ansammlung von Nachrichten mit Dateidaten in IBM MQ-Übertragungswarteschlangen verhindert und sichergestellt, dass im Fall nicht zugestellter nicht persistenter Nachrichten die Dateidaten erneut gesendet werden.
- Managed File Transfer-Agenten verwenden verschiedene IBM MQ-Warteschlangen. Weitere Informationen finden Sie im Abschnitt MFT-Systemwarteschlangen und das System.
- Zwar sind einige dieser Warteschlangen ausschließlich zur internen Verwendung vorgesehen, ein Agent kann jedoch Anforderungen in Form speziell formatierter Befehlsnachrichten akzeptieren, die an eine bestimmte vom Agenten überwachte Warteschlange gesendet werden. Sowohl die Befehlszeilenbefehle als auch das IBM MQ Explorer-Plug-in senden IBM MQ-Nachrichten an den Agenten, um diesen zur Ausführung der gewünschten Aktion anzuleiten. Auch Sie können IBM MQ-Anwendungen schreiben, die in dieser Weise mit dem Agenten interagieren. Weitere Informationen finden Sie im Abschnitt MFT durch Einreihen von Nachrichten in die Befehlswarteschlange des Agenten steuern.
- Managed File Transfer-Agenten senden Informationen zum eigenen Status sowie zum Fortschritt und Ergebnis der Übertragungen an einen MQ-Warteschlangenmanager, der als Koordinationswarteschlangenmanager fungiert. Diese Informationen werden vom Koordinationswarteschlangenmanager publiziert und können von Anwendungen subskribiert werden, um den Übertragungsfortschritt zu überwachen oder die ausgeführten Übertragungen zu protokollieren. Die publizierten Informationen können sowohl von den Befehlszeilenbefehlen als auch vom IBM MQ Explorer-Plug-in verwendet werden. Sie können IBM MQ-Anwendungen schreiben, die diese Informationen nutzen. Weitere Informationen zu dem Thema, in dem die Informationen veröffentlicht werden, finden Sie unter SYSTEM.FTE Thema.
- Die Schlüsselkomponenten von Managed File Transfer nutzen die Fähigkeit der IBM MQ-Warteschlangenmanager, Nachrichten speichern und weiterleiten zu können. Wenn es also zu einem Ausfall kommt, können unbeeinflusste Teile Ihrer Infrastruktur weiterhin Dateien übertragen. Dies erstreckt sich auch auf den Koordinationswarteschlangenmanager, dem eine Kombination aus Speichern, Weiterleiten und permanenter Subskription eine Ausfalltoleranz ermöglicht, ohne dass wichtige Informationen zu ausgeführten Dateiübertragungen verloren gehen.

Übersicht über die MFT-Topologie

Dieser Abschnitt gibt eine Übersicht darüber, wie Managed File Transfer-Agenten mit dem Koordinationswarteschlangenmanager in einem IBM MQ-Netz verbunden sind.

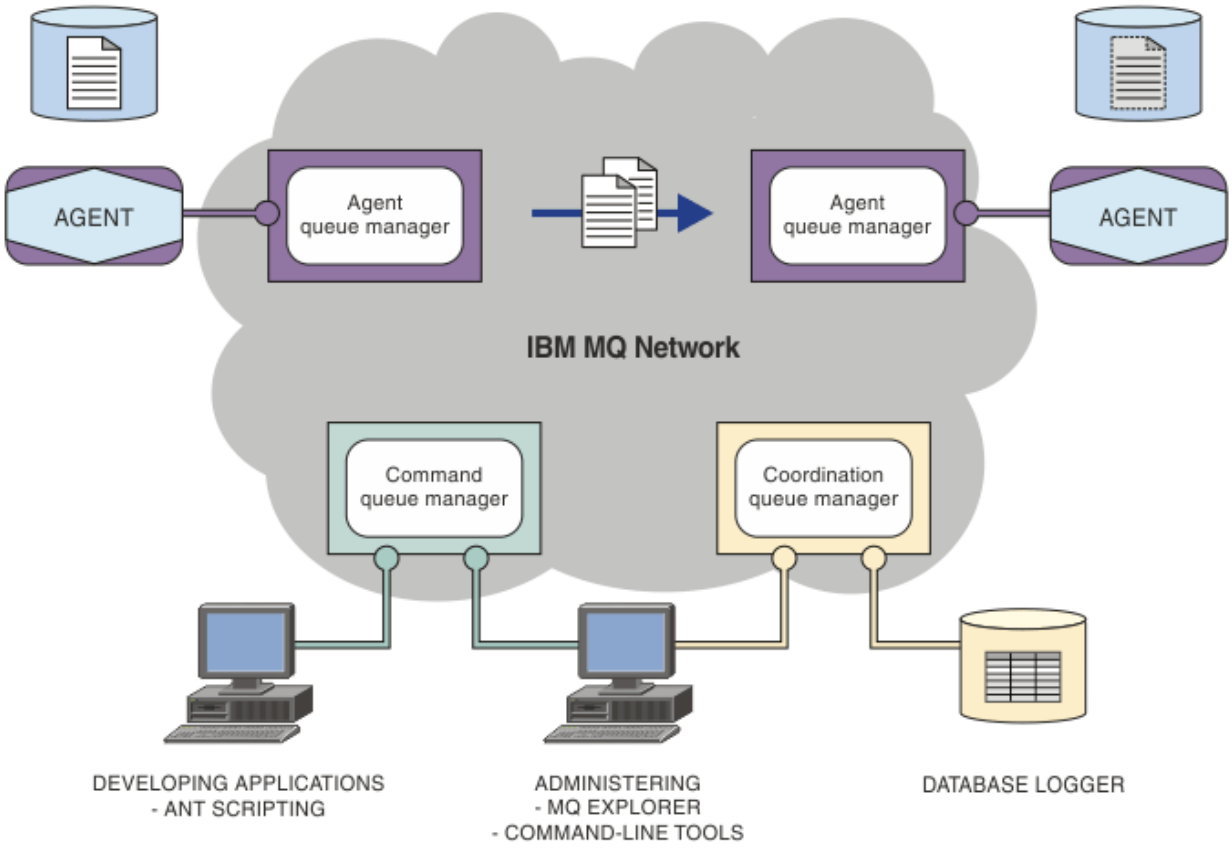
Managed File Transfer-Agenten senden und empfangen die Dateien, die übertragen werden. Jeder Agent verfügt über eine eigene Warteschlangengruppe im zugeordneten Warteschlangenmanager. Der Agent ist mit seinem Warteschlangenmanager entweder in Bindungsmodus oder Clientmodus verbunden. Ein Agent kann auch den Koordinationswarteschlangenmanager als seinen Warteschlangenmanager verwenden.

Der Koordinationswarteschlangenmanager sendet Informationen betreffs Prüfung und Dateiübertragung. Der Koordinationswarteschlangenmanager ist die zentrale Stelle für die Erfassung von Prüfinformationen zu Agenten, zum Übertragungsstatus und zu den Übertragungen. Der Koordinationswarteschlangenmanager muss nicht verfügbar sein, damit eine Übertragung ausgeführt werden kann. Die Übertragungen werden normal fortgesetzt, auch wenn der Koordinationswarteschlangenmanager vorübergehend nicht verfügbar sein sollte. Prüf- und Statusnachrichten werden in diesem Fall in den Agentenwarteschlangenmanagern gespeichert, bis der Koordinationswarteschlangenmanager wieder verfügbar ist, und anschließend wie gewohnt verarbeitet.

Agenten werden beim Koordinationswarteschlangenmanager registriert und publizieren ihre Details auf dem Warteschlangenmanager. Diese Agenteninformationen werden vom Managed File Transfer-Plug-in dazu verwendet, Übertragungen aus IBM MQ Explorer zu starten. Die im Koordinationswarteschlangenmanager erfassten Informationen zum Agenten werden auch von den Befehlen verwendet, um Agenteninformationen und den Agentenstatus anzuzeigen.

Prüfinformationen zum Übertragungsstatus und zur Übertragung werden im Koordinationswarteschlangenmanager veröffentlicht. Mithilfe des Übertragungsstatus und der Prüfinformationen zur Übertragung kann das Managed File Transfer-Plug-in den Fortschritt der Übertragung aus IBM MQ Explorer überwachen. Die im Koordinationswarteschlangenmanager gespeicherten Prüfinformationen zur Übertragung können später zur Überprüfbarkeit herangezogen werden.

Mit dem Befehlswarteschlangenmanager wird eine Verbindung zum IBM MQ-Netz hergestellt. Außerdem wird zu diesem Warteschlangenmanager eine Verbindung hergestellt, wenn Managed File Transfer-Befehle ausgegeben werden.



Zugehörige Konzepte

„Managed File Transfer" auf Seite 298
Managed File Transfer überträgt Dateien auf kontrollierte und überprüfbare Weise zwischen Systemen - unabhängig von der Dateigröße und den verwendeten Betriebssystemen.

„Funktionsweise von MFT mit IBM MQ" auf Seite 301
Managed File Transfer interagiert auf verschiedene Weise mit IBM MQ.

Managed File Transfer-Szenario

MFT REST API - Übersicht

Die REST API unterstützt bestimmte Managed File Transfer-Befehle, einschließlich Auflistung von Übertragungen und Details zu Dateiübertragungsagenten.

Die REST API enthält Optionen zum Auflisten aller aktuellen Managed File Transfer-Übertragungen und zum Abfragen des Status von Managed File Transfer-Agenten. Weitere Informationen finden Sie im Abschnitt Erste Schritte mit der REST API MFT.

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru (MQIPT) ist eine optionale Komponente von IBM MQ, mit der Messaging-Lösungen zwischen fernen Sites über das Internet implementiert werden können.

Wenn Sie die Installationsdateien von MQIPT für IBM MQ 9.4.x abrufen möchten, wechseln Sie zu IBM Fix Central für IBM MQ.

Sie können MQIPT verwenden, um eine Verbindung zu einer beliebigen unterstützten Version von IBM MQherzustellen. Sie müssen keine anderen IBM MQ -Komponenten mit derselben Version wie MQIPTinstallieren.

Wenn Sie IBM MQ-Berechtigung erworben haben, können Sie so viele Kopien von MQIPT installieren wie erforderlich. MQIPT-Installationen werden nicht auf Ihre erworbene IBM MQ-Berechtigung angerechnet. Weitere Informationen zur IBM MQ -Lizenzierung finden Sie unter IBM MQ -Lizenzinformationen.

Anmerkung: Diese Dokumentation bezieht sich auf MQIPT in IBM MQ 9.4. Informationen zur Dokumentation zum MQIPT-Support-Pack (Version 2.1) im IBM Documentation finden Sie unter MQIPT (SupportPac MS81) in der Dokumentation zu IBM MQ 9.0.

Anmerkung: Wenn Sie MQIPT 2.1 oder eine frühere Version verwenden, wird Ihnen empfohlen, ein Upgrade auf MQIPT for IBM MQ 9.4durchzuführen, da das Ende des Unterstützungszeitraums für das MQIPT Support Pack 30th September 2020 war.

IBM MQ Internet Pass-Thru wird als eigenständiger Service ausgeführt, der IBM MQ-Nachrichtenflüsse zwischen zwei IBM MQ-Warteschlangenmanagern oder zwischen einem IBM MQ-Client und einem IBM MQ-Warteschlangenmanager empfangen und weiterleiten kann.

Mit MQIPT wird diese Verbindung aktiviert, wenn der Client und der Server sich nicht im gleichen physischen Netz befinden.

Eine oder mehrere Instanzen von MQIPT können im Kommunikationspfad zwischen zwei IBM MQ-Warteschlangenmanagern oder zwischen einem IBM MQ-Client und einem IBM MQ-Warteschlangenmanager angeordnet werden. Mit den Instanzen von MQIPT können die beiden IBM MQ-Systeme Nachrichten austauschen, ohne dass eine direkte TCP/IP-Verbindung zwischen den beiden Systemen erforderlich ist. Diese Architektur ist nützlich, wenn die Firewallkonfiguration eine direkte TCP/IP-Verbindung zwischen den beiden Systemen verhindert.

MQIPT ist an mindestens einem TCP/IP-Port für eingehende Verbindungen empfangsbereit. Diese Verbindungen können entweder normale IBM MQ -Nachrichten, IBM MQ -Nachrichten, die in HTTP getunnelt werden, oder Nachrichten, die mit TLS (Transport Layer Security) oder SSL (Secure Sockets Layer) verschlüsselt werden, enthalten. MQIPT kann mehrere gleichzeitig bestehende Verbindungen verarbeiten.

Der IBM MQ-Kanal, der die ursprüngliche TCP/IP-Verbindung herstellt, wird als *aufrufender Kanal* bezeichnet, der Kanal, zu dem eine Verbindung herstellt werden soll, als *antwortender Kanal* und der Warteschlangenmanager, der schließlich kontaktiert werden soll, als *Zielwarteschlangenmanager*.

MQIPT speichert Daten beim Weiterleiten von der Quelle an das entsprechende Ziel im Hauptspeicher. Es werden keine Daten auf Platte gespeichert (mit Ausnahme des Speichers, den das Betriebssystem auf Platte auslagert). MQIPT greift nur explizit auf die Platte zu, um ihre Konfigurationsdatei zu lesen und Verbindungsprotokoll-und Tracedatensätze zu schreiben.

Der vollständige Bereich der IBM MQ -Kanaltypen kann über mindestens eine Instanz von MQIPTverbunden werden. Das Vorhandensein von MQIPT in einem Kommunikationspfad hat keine Auswirkung auf die Funktionsmerkmale der verbundenen IBM MQ -Komponenten. Die Leistung der Nachrichtenübertragung kann jedoch beeinträchtigt werden.

MQIPT kann mit IBM MQ verwendet werden, wie in „Mögliche Konfigurationen von MQIPT” auf Seite 308beschrieben.

Informationen zur Installation von MQIPT finden Sie im Abschnitt MQIPT installieren.

Zugehörige Tasks

IBM MQ Internet Pass-Thru konfigurieren

IBM MQ Internet Pass-Thru verwalten und konfigurieren

Zugehörige Verweise

IBM MQ Internet Pass-Thru -Konfigurationsreferenz

Verwendung von MQIPT

Es gibt verschiedene Möglichkeiten der Verwendung von IBM MQ Internet Pass-Thru (MQIPT).

MQIPT kann als Kanalkonzentrator verwendet werden

Durch diese Verwendung von MQIPT können Kanäle zu bzw. von mehreren separaten Hosts von einer Firewall als einziger Kanal zum bzw. vom MQIPT-Host wahrgenommen werden. Dies erleichtert die Definition und Verwaltung der Filterregeln für die Firewall.

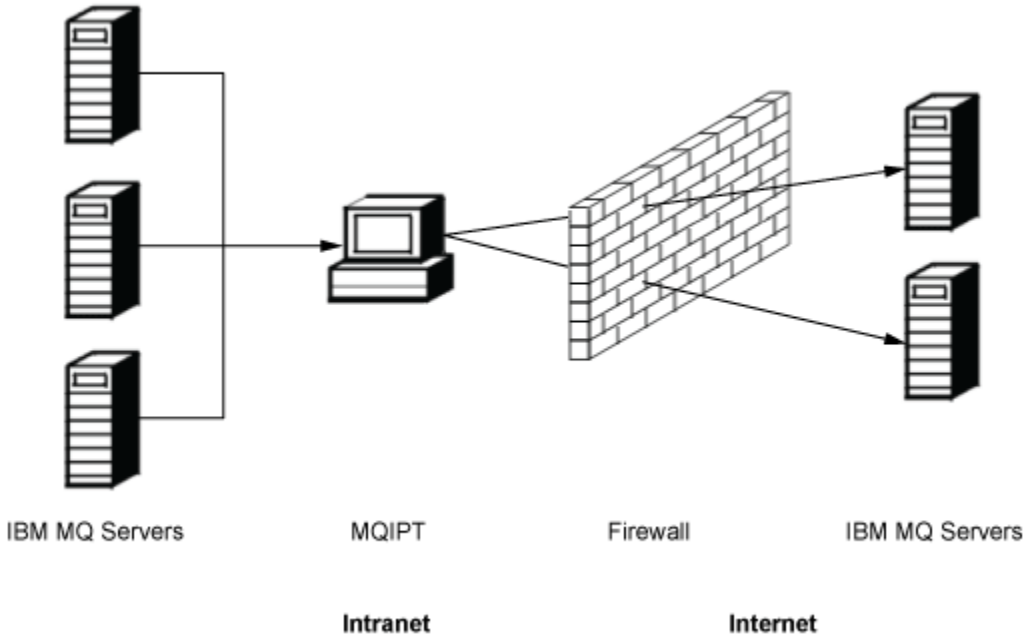


Abbildung 79. Beispiel für die Verwendung von MQIPT als Kanalkonzentrator

MQIPT kann in eine "Demilitarized Zone" (DMZ) gestellt werden, um einen zentralen Zugriffspunkt bereitzustellen

Bei Einsatz von MQIPT auf einem Computer mit einer bekannten und gesicherten IP-Adresse innerhalb einer DMZ-Firewall (eine Firewallkonfiguration für die Sicherung von lokalen Netzen) kann MQIPT zum Überwachen eingehender IBM MQ-Kanalverbindungen verwendet werden, die es dann an das gesicherte Intranet weiterleitet (DMZ = Demilitarized Zone). Die innere Firewall muss diesem vertrauenswürdigen Computer die Herstellung eingehender Verbindungen gestatten. In dieser Konfiguration verhindert MQIPT, dass externe Zugriffsanforderungen die tatsächliche IP-Adresse der Maschinen im gesicherten Intranet erkennen. Auf diese Weise stellt MQIPT einen einzelnen zentraler Zugriffspunkt dar. Bei Bedarf kann MQIPT so konfiguriert werden, dass TLS-Verbindungen akzeptiert werden und Daten über eine separate TLS-Verbindung an das Ziel weitergeleitet werden, wodurch die TLS-Sitzung in der DMZ beendet wird.

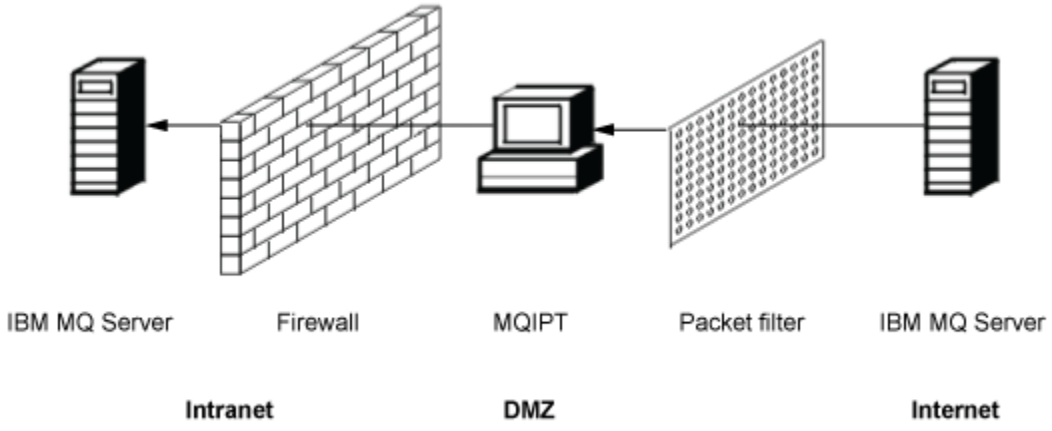


Abbildung 80. Beispiel für die Verwendung von MQIPT in einer DMZ-Firewall

MQIPT kann über die HTTP-Tunnelung kommunizieren

Wenn zwei Instanzen von MQIPT nebeneinander eingesetzt werden, können sie über HTTP miteinander kommunizieren. Durch die HTTP-Tunnelung können Anforderungen unter Verwendung vorhandener HTTP-Proxys durch Firewalls hindurch übergeben werden. Der erste MQIPT fügt das IBM MQ-Protokoll in das HTTP-Protokoll ein und der zweite extrahiert das IBM MQ-Protokoll aus seinem HTTP-Wrapper und leitet es an den Zielwarteschlangenmanager weiter.

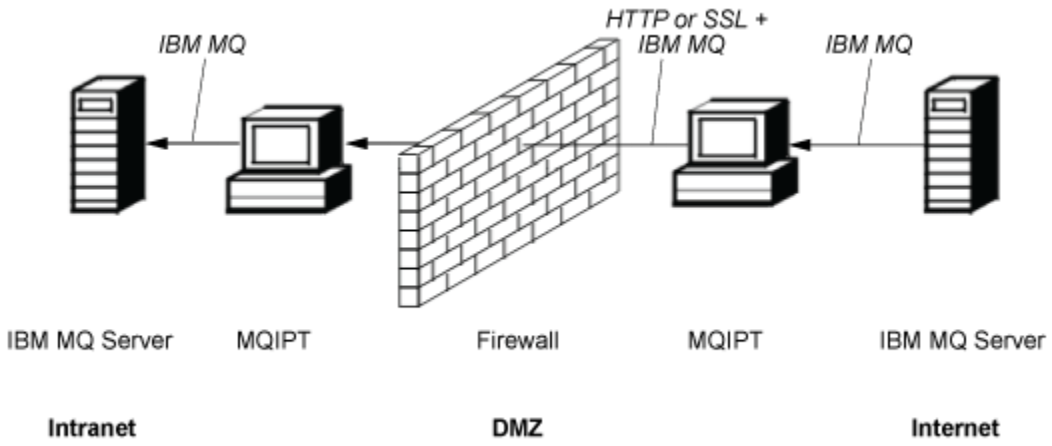


Abbildung 81. Beispiel für die MQIPT- und HTTP-Tunnelung

MQIPT kann Nachrichten verschlüsseln

Wenn MQIPT wie im vorherigen Beispiel konfiguriert ist, können Anforderungen vor der Übertragung durch Firewalls verschlüsselt werden. Die erste Instanz von MQIPT verschlüsselt die Daten, die zweite Instanz entschlüsselt sie mithilfe von SSL/TLS und sendet sie an den Zielwarteschlangenmanager.

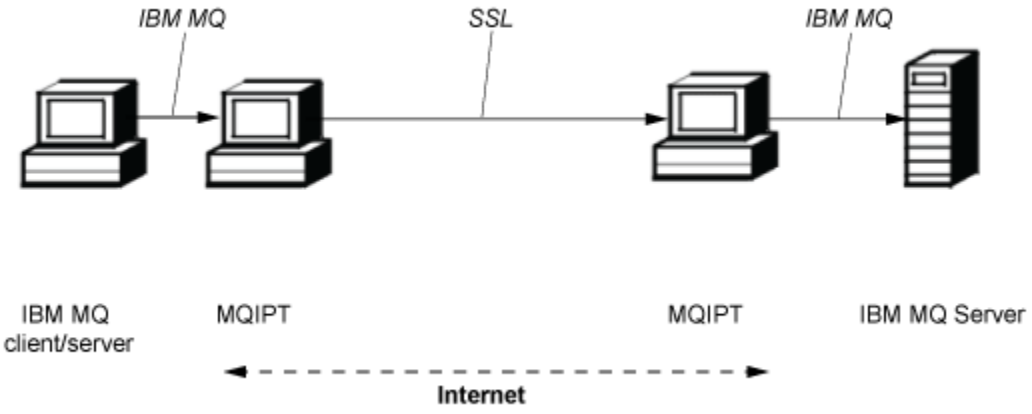


Abbildung 82. Beispiel für MQIPT und SSL/TLS

Funktionsweise von MQIPT

In seiner einfachsten Konfiguration fungiert MQIPT als IBM MQ-Protokollweiterleitungsfunktion. Es ist an einem TCP/IP-Port empfangsbereit und akzeptiert Verbindungsanforderungen von IBM MQ-Kanälen.

Wenn eine korrekt formatierte Anforderung eingeht, stellt MQIPT eine weitere TCP/IP-Verbindung zwischen sich selbst und dem IBM MQ-Zielwarteschlangenmanager her. Anschließend leitet es alle Protokollpakete, die es von seiner eingehenden Verbindung empfängt, an den Zielwarteschlangenmanager weiter und gibt Protokollpakete vom Zielwarteschlangenmanager an die ursprüngliche eingehende Verbindung zurück.

Es liegt keine Änderung am IBM MQ-Protokoll (Client/Server oder Warteschlangenmanager zu Warteschlangenmanager) vor, da keines der beiden Enden direkt über die Anwesenheit des Vermittlers informiert ist. Neue Versionen des IBM MQ-Client- oder -Servercodes sind nicht erforderlich.

Um MQIPT zu verwenden, muss der aufrufende Kanal so konfiguriert sein, dass er den Hostnamen und den Port von MQIPT verwendet und nicht den Hostnamen und Port des Zielwarteschlangenmanagers. Dies wird mit der Eigenschaft **CONNAME** des IBM MQ-Kanals definiert. MQIPT liest die eingehenden Daten und übergibt sie einfach an den Zielwarteschlangenmanager. Andere Konfigurationsfelder, wie z. B. die Benutzer-ID und das Kennwort in einem Client/Server-Kanal, werden ebenfalls an den Zielwarteschlangenmanager übergeben.

Mehrere Warteschlangenmanager

MQIPT kann verwendet werden, um den Zugriff auf mehr als einen Zielwarteschlangenmanager zu ermöglichen. Damit dies funktioniert, muss ein Mechanismus vorhanden sein, um MQIPT zu informieren, zu welchem Warteschlangenmanager eine Verbindung hergestellt werden soll, sodass MQIPT die eingehende TCP/IP-Portnummer verwendet, um zu ermitteln, zu welchem Warteschlangenmanager eine Verbindung hergestellt werden soll.

Daher können Sie MQIPT so konfigurieren, dass es auf mehreren TCP/IP-Ports empfangsbereit ist. Jeder Empfangsport ist über eine MQIPT-*Route* einem Zielwarteschlangenmanager zugeordnet. Sie können bis zu 100 solche Routen definieren, die einen empfangsbereiten TCP/IP-Port mit dem Hostnamen und dem Port des Zielwarteschlangenmanagers verknüpfen. Somit ist der Hostname (IP-Adresse) des Zielwarteschlangenmanagers für den Ursprungskanal niemals sichtbar. Jede Route kann mehrere Verbindungen zwischen ihrem Empfangsport und dem Ziel verarbeiten, wobei jede Verbindung unabhängig voneinander agiert.

Konfigurationsdatei MQIPT

MQIPT verwendet eine Konfigurationsdatei mit dem Namen `mqipt.conf`. Diese Datei enthält Definitionen aller Routen und ihrer zugehörigen Eigenschaften. Weitere Informationen zu `mqipt.conf` finden Sie im Abschnitt IBM MQ Internet Pass-Thru verwalten und konfigurieren.

Beim Start von MQIPT wird jede in der Konfigurationsdatei aufgelistete Route gestartet. In die Systemkonsole werden Nachrichten geschrieben, die den Status jeder einzelnen Route anzeigen. Wenn die Nachricht MQCPI078 für eine Route angezeigt wird, ist diese Route bereit, Verbindungsanforderungen zu akzeptieren.

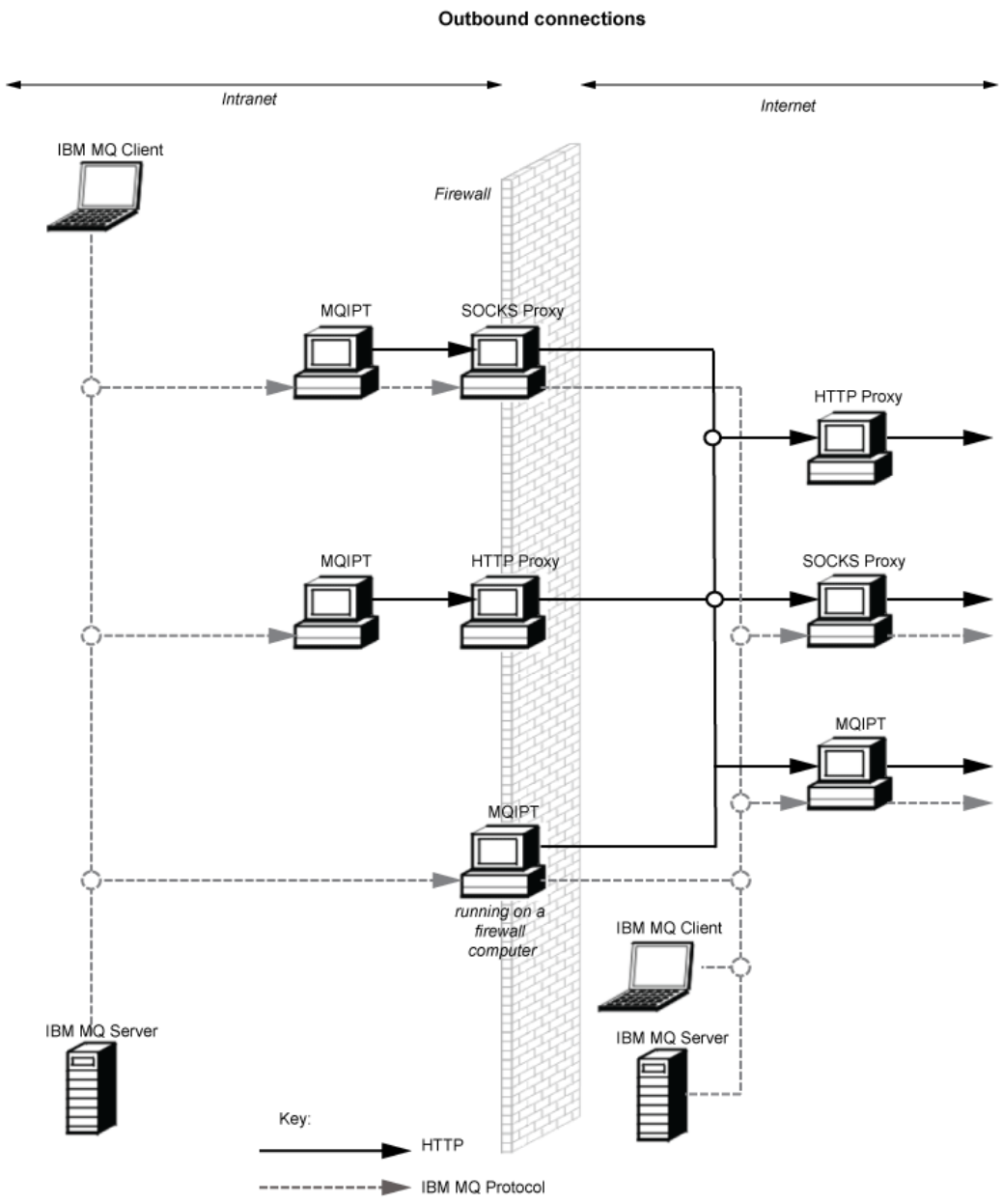
Mögliche Konfigurationen von MQIPT

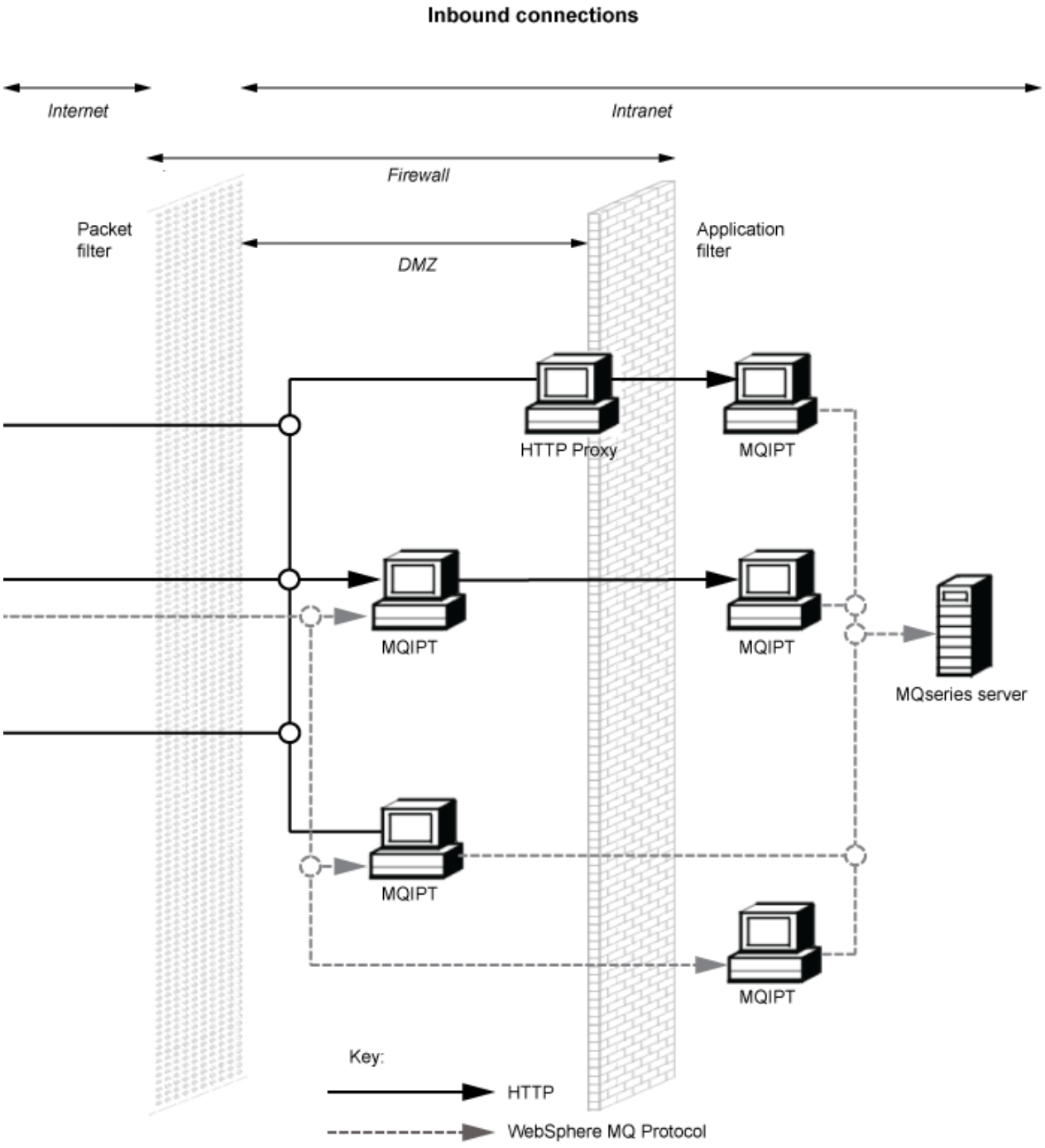
MQIPT kann in Verbindung mit IBM MQ und IBM Integration Busverwendet werden.

Die folgende aus mehreren Teilen bestehende Abbildung zeigt viele mögliche Konfigurationen für MQIPT in einer IBM MQ-Topologie. Sie veranschaulicht verschiedene Möglichkeiten, wie MQIPT Nachrichten senden kann. In der Abbildung sind Clients und Server in einem Intranet innerhalb einer Firewall und im Internet außerhalb der Firewall zu sehen, die Nachrichten an MQIPT, einen HTTP-Proxy oder SOCKS-Proxy übergeben, der diese weiterleitet.

Die Nachrichten werden von einem MQIPT-Proxy oder einem HTTP-Proxy in einer DMZ empfangen, bevor sie über die eingehende Firewall an einen Server übergeben werden.

Beachten Sie, dass der HTTP-Proxy, der SOCKS-Proxy und die MQIPT-Computer auf der Intranetseite der Firewall die Möglichkeit einer Verkettung mehrerer Computer im Internet darstellen. So könnte beispielsweise ein MQIPT-Computer über einen oder mehrere SOCKS- oder HTTP-Proxy-Computer oder weitere MQIPT-Computer kommunizieren, bevor er sein Ziel erreicht.


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



Kompatible Konfigurationen

Kompatible Verbindungsszenarios, bei denen ein IBM MQ-Client oder -Warteschlangenmanager mit MQIPT kommuniziert. Für die Kommunikation mit einem Zielwarteschlangenmanager wird dieselbe oder eine zweite MQIPT-Route verwendet.

Kompatible Konfigurationen einer einzigen MQIPT-Route

Sie können eine einzelne MQIPT -Route für die Kommunikation mit IBM MQverwenden.

Die Spalten in Tabelle 26 auf Seite 311 enthalten folgende Informationen:

1. Das zwischen IBM MQ und der MQIPT-Route verwendete Protokoll. Die Verbindung kann entweder über einen IBM MQ-Client oder einen Warteschlangenmanager hergestellt werden und kann entweder IBM MQ-Formate und Protokolle (FAP) oder ein SSL/TLS-Protokoll verwenden.
2. Der Modus, in dem die MQIPT-Route ausgeführt wird. Das Format der Kommunikation über das Internet zwischen MQIPT und IBM MQ wird durch die Konfiguration der MQIPT -Route bestimmt. Beachten Sie, dass in den Fällen, in denen die Tabelle SSL erwähnt, auch TLS verwendet werden kann.
3. Das zwischen der MQIPT-Route und dem Zielwarteschlangenmanager verwendete Protokoll.

Tabelle 26. Gültige Konfigurationen mit einer einzelnen Instanz von MQIPT

1. IBM MQ Quellenprotokoll	**2. Modus der MQIPT-Route**	**3. IBM MQ Zielprotokoll**
FAP	FAP-Proxy (Standardwert)	FAP
	FAP-Server und SSL-Client	SSL/TLS
SSL/TLS	SSL-Proxy	SSL/TLS
	SSL-Server und FAP-Client	FAP
	SSL-Server und SSL-Client	SSL/TLS

Kompatible Konfigurationen mit mehr als einer MQIPT-Route

Möglicherweise wollen Sie auf einer oder mehreren Instanzen von MQIPT mehr als eine Route verwenden, um mit IBM MQ zu kommunizieren.

Die Spalten in Tabelle 27 auf Seite 312 enthalten folgende Informationen:

1. Das zwischen IBM MQ und der ersten MQIPT-Route verwendete Protokoll. Die Verbindung kann entweder über einen IBM MQ-Client oder einen Warteschlangenmanager hergestellt werden und kann entweder IBM MQ-Formate und Protokolle (FAP) oder ein SSL/TLS-Protokoll verwenden.
2. Der Modus, in dem die erste MQIPT-Route ausgeführt wird. Das Format der Kommunikation über das Internet zwischen MQIPT und IBM MQ wird durch die Konfiguration der MQIPT -Route bestimmt. Beachten Sie, dass in den Fällen, in denen die Tabelle SSL erwähnt, auch TLS verwendet werden kann.
3. Der Modus, in dem die zweite MQIPT-Route ausgeführt wird.
4. Das zwischen der zweiten MQIPT-Route und dem Zielwarteschlangenmanager verwendete Protokoll.

Tabelle 27. Gültige Konfigurationen mit mehreren Instanzen von MQIPT			
1. IBM MQ Quellenprotokoll	**2. Modus der ersten MQIPT-Route**	**3. Modus der zweiten MQIPT-Route**	**4. IBM MQ Zielprotokoll**
FAP (Standardwert)	FAP-Proxy (Standardwert)	FAP-Proxy (Standardwert)	FAP
	FAP-Server und SSL-Client	SSL-Proxy	SSL/TLS
		SSL-Server und FAP-Client	FAP
		SSL-Server und SSL-Client	SSL/TLS
	HTTP-Client	HTTP-Server und SSL-Client	SSL/TLS
	HTTPS-Client	HTTPS-Server und SSL-Client	SSL/TLS
	HTTP-Client	HTTP-Server	FAP
	HTTPS-Client	HTTPS-Server	FAP
SSL/TLS	SSL-Proxy	SSL-Proxy	SSL/TLS
		SSL-Server und FAP-Client	FAP
		SSL-Server und SSL-Client	SSL/TLS
	HTTP-Client	HTTP-Server	FAP
	HTTPS-Client	HTTPS-Server	SSL/TLS
	HTTP-Client	HTTP-Server und SSL-Client	FAP
	HTTPS-Client	HTTPS-Server und SSL-Client	SSL/TLS

Unterstützte Kanalkonfigurationen

Es werden alle IBM MQ-Kanaltypen unterstützt, die Konfiguration ist jedoch auf TCP/IP-Verbindungen beschränkt. Einem IBM MQ-Client oder -Warteschlangenmanager wird MQIPT so angezeigt, als ob es sich um den Zielwarteschlangenmanager handeln würde. Wenn die Kanalkonfiguration einen Zielhost und eine Portnummer erfordert, werden der Hostname und die Listener-Portnummer von MQIPT angegeben.

Client-/Server-Kanäle

MQIPT ist für eingehende Clientverbindungsanforderungen empfangsbereit und leitet diese anschließend mithilfe von HTTP-Tunnelung, SSL/TLS oder als IBM MQ-Standardprotokollpakete weiter. Wenn MQIPT HTTP-Tunnelung oder SSL/TLS verwendet, erfolgt die Weiterleitung über eine Verbindung zu einem zweiten MQIPT. Wenn HTTP-Tunnelung nicht verwendet wird, erfolgt die Weiterleitung über eine Verbindung zu dem, was als Zielwarteschlangenmanager angesehen wird (obwohl es sich hierbei wiederum un einen weiteren MQIPT handeln könnte). Wenn der Zielwarteschlangenmanager die Clientverbindung akzeptiert hat, werden Pakete zwischen Client und Server übermittelt.

Clustersender-/-empfängerkanäle

Wenn MQIPT eine eingehende Anforderung von einem Clustersenderkanal erhält, wird davon ausgegangen, dass der Warteschlangenmanager SOCKS-fähig ist und die tatsächliche Zieladresse wird während des SOCKS-Handshakeverfahrens ermittelt. Die Anforderung wird genau wie bei Clientver-

bindungskanälen an den nächsten MQIPT oder Zielwarteschlangenmanager weitergeleitet. Hierzu gehören auch automatisch definierte Clustersenderkanäle.

Sender/Empfänger
Wenn MQIPT eine eingehende Anforderung von einem Senderkanal erhält, wird diese in genau derselben Weise wie bei Clientverbindungskanälen an den nächsten MQIPT oder Zielwarteschlangenmanager weitergeleitet. Der Zielwarteschlangenmanager überprüft die eingehende Anforderung und startet gegebenenfalls den Empfängerkanal. Die gesamte Kommunikation zwischen Sender-und Empfängerkanal (einschließlich Sicherheitsdatenflüssen) wird übermittelt.

Requester/Server
Diese Kombination wird wie die vorherigen Konfigurationen gehandhabt. Die Validierung der Verbindungsanforderung wird vom Serverkanal auf dem Zielwarteschlangenmanager durchgeführt.

Requester/Sender
Die "Callback"-Konfiguration könnte hilfreich sein, wenn die beiden Warteschlangenmanager keine direkten Verbindungen zueinander herstellen können, jedoch beide berechtigt sind, eine Verbindung zu MQIPT herzustellen und von dort Verbindungen zu akzeptieren.

Server/Requester und Server/Empfänger
Diese werden von MQIPT auf dieselbe Weise behandelt wie die `Sender/Receiver` -Konfiguration.

Kanalbeendigung und Fehlerbedingungen

Wenn MQIPT das Schließen (entweder normales oder abnormales) eines IBM MQ-Kanals erkennt, leitet es die Kanalschließung weiter. Wenn Sie eine Route mit Hilfe von MQIPT schließen, werden alle über diese Route verlaufenden Kanäle geschlossen.

MQIPT stellt eine optionale Funktion für Inaktivitätszeitlimits bereit. Wenn MQIPT feststellt, dass ein Kanal während eines Zeitraums, der das Zeitlimit überschreitet, inaktiv war, führt er eine sofortige Beendigung der betreffenden beiden Verbindungen durch.

Die IBM MQ-Systeme an jedem Ende des Kanals betrachten diese abnormalen Beendigungsbedingungen entweder als Netzausfälle oder als Beendigung des Kanals durch ihren Partner. Der Kanal kann dann erneut gestartet und wiederhergestellt werden (wenn der Fehler während eines Protokollzeitraums auftritt), als ob MQIPT nicht verwendet würde.

Sicherheit von Nachrichten

Durch das verteilte Warteschlangenmanagement von IBM MQ wird sichergestellt, dass Nachrichten ordnungsgemäß zugestellt werden. Dies ist auch dann der Fall, wenn MQIPT zwischen den beiden Enden des Kanals vorhanden ist. MQIPT speichert keine Nachrichtendaten und beteiligt sich auch nicht an der Synchronisationspunktprozedur, die eine korrekte Nachrichtenübermittlung sicherstellt.

Bei der Verwendung von schnellen, nicht persistenten IBM MQ-Nachrichten kann eine Nachricht verloren gehen, wenn die MQIPT-Route ausfällt oder erneut gestartet wird, während die IBM MQ-Nachricht unterwegs ist. Bevor Sie die Route erneut starten, müssen Sie sicherstellen, dass alle IBM MQ-Kanäle, die die MQIPT-Route verwenden, inaktiv sind.

Weitere Informationen zur Sicherheit von Nachrichten in IBM MQ finden Sie im Abschnitt Sicherheit von Nachrichten.

Multi-Instanz-Warteschlangenmanager und Hochverfügbarkeit

MQIPT kann mit Multi-Instanz-Warteschlangenmanagern in Hochverfügbarkeitsumgebungen verwendet werden.

MQIPT hat keinen persistenten Status, daher bietet ein Failover von MQIPT zu einem anderen System keinen Vorteil. Stattdessen sollten mehrere Instanzen von MQIPT mit identischen Konfigurationsdateien `mqipt.conf` auf verschiedenen Systemen ausgeführt werden. Überwachen Sie jede Instanz von MQIPT auf Verfügbarkeit und starten Sie sie gegebenenfalls (auf demselben System) erneut. So wird eine Gruppe identischer MQIPT-Instanzen bereitgestellt, die für die Weiterleitung von Verbindungen verwendet wer-

den können. Sie müssen dann sicherstellen, dass IBM MQ Verbindungen zu MQIPT weiterleiten kann und MQIPT in der Lage ist, diese Verbindungen an den Zielwarteschlangenmanager weiterzuleiten.

IBM MQ-Kanäle für abgehende Nachrichten können auf verschiedene Arten an eine verfügbare MQIPT-Instanz weitergeleitet werden. Beispiele:

- Verwendung einer Lastausgleichsfunktion oder eines Hochverfügbarkeitsrouters wie z. B. des IBM Network Dispatchers aus dem WebSphere Edge Components-Produkt.
- Angabe mehrerer Verbindungsnamen in der IBM MQ-Kanaldefinition in Form einer durch Kommas getrennten Liste. IBM MQ versucht dann, eine Verbindung zu den einzelnen MQIPT-Adressen herzustellen, bis eine verfügbare MQIPT-Instanz gefunden wird.

Sie müssen auch Verbindungen von MQIPT zum Zielwarteschlangenmanager leiten. Wenn die Hochverfügbarkeitskonfiguration für ein Failover der IP-Adresse mit dem Zielwarteschlangenmanager sorgt, ist keine besondere MQIPT-Konfiguration erforderlich: Geben Sie die IP-Zieladresse in der Routeneigenschaft **`Destination`** an und lassen Sie zu, dass die Failover-Operation die IP-Adresse mit dem Warteschlangenmanager verschiebt.

Wenn sich die IP-Adresse des Warteschlangenmanagers jedoch nach einem Failover ändert, müssen Sie dafür sorgen, dass MQIPT die Verbindung an das richtige Ziel weiterleitet. Dies kann auf eine der folgenden Arten erfolgen:

- Schreiben eines Routing-Exits, der prüft, welche IP-Adresse und Portnummer zugänglich ist, und anschließendes Überschreiben des Routenziels für jede Verbindung. Mit MQIPT werden einige Beispielrouting-Exits bereitgestellt; diese können zu diesem Zweck angepasst werden.
- Verwendung einer Hochverfügbarkeits-Lastausgleichsfunktion, um die Verbindung umzuleiten.
- Definition mehrerer MQIPT-Routen (eine für jede IP-Adresse und jeden Port, an der/dem der Warteschlangenmanager ausgeführt werden könnte). Anschließend Leitung der IBM MQ-Verbindungen zu den verschiedenen MQIPT-Routen, beispielsweise durch Auflistung aller Routen-IP-Adressen und Portnummern in Form einer durch Kommas getrennten Liste im Verbindungsnamen des Kanals für abgehende Nachrichten.

Es ist auch wichtig, alle End-to-End-Komponenten im Netzpfad zu optimieren:

1. Verbindungsversuche zu nicht verfügbaren Systemen müssen schnell fehlschlagen, sodass bei Verbindungswiederholungsversuchen zum ersten verfügbaren Ziel übergegangen werden kann.

   Optimieren Sie für MQIPT-SSL-Routen die Routeneigenschaft **`SSLClientConnectTimeout`**, damit bei nicht verfügbaren Zielen umgehend ein Verbindungsfehler ausgegeben wird. Weitere Informationen zu den Optimierungsparametern von IBM MQ finden Sie in der IBM MQ-Dokumentation. Darüber hinaus finden Sie in der Betriebssystemdokumentation Informationen zur TCP/IP-Optimierung für das Betriebssystem. In allen Fällen sollte bei fehlgeschlagenen Verbindungsversuchen schnell ein Netzfehler (z. B. ein TCP-Rücksetzungspaket) zurückgegeben werden oder es sollte ohne unnötige Verzögerung eine Zeitlimitüberschreitung erfolgen.
2. Aktive Verbindungen zu einem fehlgeschlagenen System müssen umgehend getrennt werden, damit neue Verbindungen hergestellt werden können.

   Sie sollten auch bedenken, welche Auswirkungen ein Failover zu einem Zeitpunkt hat, zu dem MQIPT aktiv von Verbindungen genutzt wird. Wahrscheinlich werden Netzverbindungen während eines Failovers getrennt. Für Clientanwendungen können Sie die IBM MQ-Funktion für die automatische Clientverbindungswiederholung verwenden, um getrennte Verbindungen wiederherzustellen. Für Nachrichtenkanäle können Sie ein kurzes Wiederholungsintervall angeben, sodass der Kanal sofort wieder eine Verbindung herstellt. Weitere Informationen zur Konfiguration der automatischen Clientverbindungswiederholung und der Nachrichtenkanalwiederholung finden Sie in der IBM MQ-Dokumentation.

IBM MQ Console und REST API

Sie können IBM MQ Console und REST API verwenden, um IBM MQzu verwalten und Messaging-Operationen über HTTP auszuführen.

- Sie können die IBM MQ Console verwenden, um grundlegende Verwaltungstasks über einen Web-Browser auszuführen. Weitere Informationen hierzu finden Sie im Abschnitt Verwaltung mit dem IBM MQ Console .
- Sie können die administrative REST API verwenden, um IBM MQ-Objekte zu verwalten, wie z. B. Warteschlangenmanager und Warteschlangen sowie Managed File Transfer-Agenten und -Übertragungen. Weitere Informationen finden Sie im Abschnitt Verwaltung mit der REST API.
- Sie können messaging REST API verwenden, um einfaches Punkt-zu-Punkt-und Publish-Messaging durchzuführen. Weitere Informationen finden Sie unter Messaging mit REST API.

Installationsoptionen

Die IBM MQ Console und REST API werden auf einem WebSphere Liberty -Server mit dem Namen mqweb ausgeführt. Ab IBM MQ 9.3.5können Sie den mqweb-Server als optionale Komponente in einer IBM MQ -Installation oder als eigenständige IBM MQ Web Server -Installation installieren.

Linux V 9.4.0 **Eigenständige Installation von IBM MQ Web Server**

Ab IBM MQ 9.4.0kann der mqweb-Server in einer eigenständigen Installation von IBM MQ Web Serverausgeführt werden. Mit einer eigenständigen IBM MQ Web Server -Installation können Sie den mqweb-Server auf Systemen installieren und ausführen, die von Ihren IBM MQ -Installationen getrennt sind. Die Installation eines eigenständigen IBM MQ Web Server s bietet größere Flexibilität hinsichtlich der Systeme und der Anzahl der Systeme, auf denen Sie Ihre mqweb-Server ausführen möchten. Bei Bedarf können mehrere Instanzen des mqweb-Server auf verschiedenen Maschinen ausgeführt werden, um die erforderliche Skalierbarkeit und Verfügbarkeit bereitzustellen.

Wenn Sie eine IBM MQ -Berechtigung erworben haben, können Sie so viele Kopien installieren, wie für den eigenständigen IBM MQ Web Servererforderlich sind. IBM MQ Web Server-Installationen werden nicht auf Ihre erworbene IBM MQ-Berechtigung angerechnet. Weitere Informationen zur Lizenzierung von IBM MQ finden Sie im Abschnitt IBM MQ - Lizenzinformationen.

In einer eigenständigen IBM MQ Web Server -Installation gelten die folgenden Einschränkungen:

- Mit IBM MQ Console können nur ferne Warteschlangenmanager verwaltet werden.
- messaging REST API kann nur mit fernen Warteschlangenmanagern verwendet werden.
- Die administrative REST API ist nicht verfügbar.

Der eigenständige IBM MQ Web Server wird nur auf Linux -Plattformen unterstützt.

Weitere Informationen zur Installation des eigenständigen IBM MQ Web Servers finden Sie unter Eigenständigen IBM MQ Web Serverinstallieren.

Optionale Komponente einer IBM MQ -Installation

Sie können die Komponenten IBM MQ Console und REST API im Rahmen einer IBM MQ -Installation installieren.

Alle IBM MQ Console -und REST API -Funktionen sind verfügbar, wenn der mqweb-Server in einer IBM MQ -Installation ausgeführt wird.

- Mit IBM MQ Console können lokale und ferne Warteschlangenmanager verwaltet werden.
- messaging REST API kann mit lokalen und fernen Warteschlangenmanagern verwendet werden.
- Mit administrative REST API können lokale und ferne Warteschlangenmanager verwaltet werden.

Wenn Sie die Komponenten IBM MQ Console und REST API verwenden möchten, installieren Sie die folgende Komponente als Teil Ihrer IBM MQ -Installation:

- AIX Installieren Sie unter AIX die Dateigruppe `mqm.web.rte`.
- IBM i Installieren Sie unter IBM idie WEB-Komponente.
- Linux Installieren Sie unter Linux die Komponente `MQSeriesWeb`.

- Windows Installieren Sie unter Windows das Feature `Web Administration`.
- z/OS Installieren Sie unter z/OS das Feature `IBM MQ for z/OS UNIX System Services Web Components`.

Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden.

Möglicherweise bietet IBM die in diesem Dokument beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf Produkte, Programme oder Services von IBM bedeuten nicht, dass nur Produkte, Programme oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder andere Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb von Fremdprodukten, Fremdprogrammen und Fremdservices liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Défense
U.S.A.

Bei Lizenzanforderungen zu Double-Byte-Information (DBCS) wenden Sie sich bitte an die IBM Abteilung für geistiges Eigentum in Ihrem Land oder senden Sie Anfragen schriftlich an folgende Adresse:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Europe, Middle East & Africa
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in diesen Informationen beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren und können Namen von Personen, Firmen, Marken oder Produkten enthalten. Sämtliche dieser Namen sind fiktiv. Ähnlichkeiten mit Namen und Adressen tatsächlicher Unternehmen oder Personen sind zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Beispielanwendungsprogramme, die in Quellensprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Beispielprogramme kostenlos ohne Zahlung an IBM in jeder Form kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Beispielprogramme geschrieben sind. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten.

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farbabbildungen.

Informationen zu Programmierschnittstellen

Die bereitgestellten Informationen zur Programmierschnittstelle sollen Sie bei der Erstellung von Anwendungssoftware für dieses Programm unterstützen.

Dieses Handbuch enthält Informationen zu geplanten Programmierschnittstellen, die es dem Kunden ermöglichen, Programme zum Abrufen der Services von IBM MQzu schreiben.

Diese Informationen können jedoch auch Angaben über Diagnose, Bearbeitung und Optimierung enthalten. Die Informationen zu Diagnose, Bearbeitung und Optimierung sollten Ihnen bei der Fehlerbehebung für die Anwendungssoftware helfen.

Wichtig: Verwenden Sie diese Diagnose-, Änderungs-und Optimierungsinformationen nicht als Programmierschnittstelle, da sie Änderungen unterliegen.

Marken

IBM, das IBM Logo, ibm.com, sind Marken der IBM Corporation in den USA und/oder anderen Ländern. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein.

Microsoft und Windows sind eingetragene Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Dieses Produkt enthält Software, die von Eclipse Project (https://www.eclipse.org/) entwickelt wurde.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Teilenummer:

(1P) P/N: