9.4

Anwendungen für IBM MQ entwickeln

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen” auf Seite 1359 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

Anwendungen für IBM MQ entwickeln

Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

Entwicklung von Anwendungen für IBM MQ - Informationen für Einsteiger

Wenn Sie mehr über die Entwicklung von Anwendungen für IBM MQ wissen möchten, besuchen Sie IBM Developer:

- IBM MQ Developer Essentials *(Grundlagen lernen, Demo ausführen, App codieren, an Lernprogrammen für Fortgeschrittene teilnehmen)*
- IBM MQ Downloads für Entwickler *(einschließlich kostenloser Entwicklereditionen und Testversionen)*

Die Entwicklung solcher Anwendungen sollte Ihnen auch leichter fallen, wenn Sie sich mit den Konzepten vertraut machen, die in folgenden Abschnitten beschrieben werden:

- „Konzepte für die Anwendungsentwicklung” auf Seite 7
- „Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 52

Unterstützung für objektorientierte Sprachen und Frameworks

IBM MQ stellt grundlegende Unterstützung für Anwendungen bereit, die in folgenden Sprachen und Frameworks entwickelt werden:

- JMS
- Java
- C++
- .NET

Weitere Informationen hierzu finden Sie im Abschnitt „Objektorientierte Anwendungen” auf Seite 16.

.NET unterstützt Anwendungen, die in vielen Sprachen entwickelt werden. Um die Verwendung der IBM MQ-Klassen für .NET für den Zugriff auf IBM MQ-Warteschlangen zu veranschaulichen, enthält die MQ-Produktdokumentation Informationen für folgende Sprachen:

- C#-Beispielcode und -Beispielanwendungen
- Beispielanwendungen in C++
- Visual Basic-Beispielanwendungen

Weitere Informationen finden Sie unter „IBM MQ .NET-Programme schreiben und implementieren” auf Seite 608.

IBM MQ unterstützt .NET Core für Anwendungen in Windows -Umgebungen von IBM MQ 9.1.1 und für Anwendungen in Linux® -Umgebungen von IBM MQ 9.1.2. Weitere Informationen finden Sie unter „Installieren von IBM MQ classes for .NET” auf Seite 584.

ALW IBM MQ unterstützt auch die AMQP-Clients, die das OASIS AMQP 1.0-Protokoll implementieren.

MQ Light, Apache Qpid-Clients wie Apache Qpid Proton-und Apache Qpid JMS-APIs basieren auf diesem Protokoll.

Die MQ Light-APIs sind unter IBM MQ Lightverfügbar.

Die Apache Qpid-Clients sind unter QPid-Protonverfügbar.

Die folgenden programmiersprachenbezogenen Bindungen werden wie folgt bereitgestellt:

- eine Go-Bindung
- eine JavaScript-API-Implementierung, die mit Node.js-Anwendungen arbeitet

Unterstützung für programmgesteuerte REST-APIs

IBM MQ bietet zum Senden und Empfangen von Nachrichten Unterstützung für die folgenden programmgestützten REST-APIs:

- IBM MQ messaging REST API
- z/OS IBM z/OS Connect EE
- IBM Integration Bus
- IBM DataPower Gateway

Siehe „REST-Anwendungen mit IBM MQ entwickeln" auf Seite 738 sowie das Lernprogramm Get started with the IBM MQ messaging REST API im IBM MQ-Bereich von IBM Developer. Dieses Lernprogramm enthält Beispiele in folgenden Sprachen, die unverändert zur Verwendung mit der IBM MQ messaging REST API bereitgestellt werden:

- Beispiel mit Verwendung der MQ-Messaging-REST-API
- Node.js-Beispiel mit HTTPS-Modul
- Node.js-Beispiel mit Promise-Modul

Unterstützung für prozedurale Programmiersprachen

IBM MQ bietet Unterstützung für Anwendungen, die in folgenden prozeduralen Programmiersprachen entwickelt werden:

- C
- Windows Visual Basic (nur Windows-Systeme)
- COBOL
- z/OS Assembler (nur IBM MQ for z/OS)
- z/OS PL/I (nur IBM MQ for z/OS)
- IBM i RPG (nur IBM MQ for IBM i)

Diese Sprachen rufen über die Schnittstelle für Nachrichtenwarteschlangen (MQI) Services zur Steuerung von Nachrichtenwarteschlangen auf. Weitere Informationen finden Sie unter „MQI-Anwendungen mit IBM MQ entwickeln" auf Seite 754. Beachten Sie, dass das von den objektorientierten Sprachen und Frameworks verwendete IBM MQ-Objektmodell zusätzliche Funktionen bereitstellt, die für die prozeduralen Programmiersprachen, die die MQI verwenden, nicht verfügbar sind.

Anwendungsnamen angeben

ALW

Bereits vor IBM MQ 9.1.2 konnten Sie einen Anwendungsnamen für Java- oder JMS-Clientanwendungen angeben. Ab IBM MQ 9.1.2 können Sie zusätzlich den Anwendungsnamen in weiteren Programmiersprachen angeben. Weitere Informationen finden Sie unter „Anwendungsname in unterstützten Programmiersprachen angeben" auf Seite 55.

Zugehörige Tasks

„Anwendungen für MQ Telemetry entwickeln" auf Seite 1300

„Microsoft Windows Communication Foundation -Anwendungen mit IBM MQ entwickeln" auf Seite 1327
Der angepasste WCF-Kanal (Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

Zugehörige Verweise

„Anwendungen für Managed File Transfer entwickeln” auf Seite 1278
Geben Sie Programme an, die mit Managed File Transfer ausgeführt werden sollen, verwenden Sie Apache Ant mit Managed File Transfer, passen Sie Managed File Transfer mit Benutzerexits an und steuern Sie Managed File Transfer, indem Sie Nachrichten in die Befehlswarteschlange des Agenten einreihen.

Konzepte für die Anwendungsentwicklung

Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

Informationen zu den Typen der Anwendungen, die Sie für IBM MQ schreiben können, finden Sie unter „Anwendungen für IBM MQ entwickeln” auf Seite 5 und „Von Anwendungen ausführbare Aktionen” auf Seite 7.

Zugehörige Konzepte

„Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 52
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

Von Anwendungen ausführbare Aktionen

Sie können Anwendungen zum Senden und Empfangen von Nachrichten entwickeln, die zur Unterstützung von Geschäftsprozessen erforderlich sind. Darüber hinaus können Sie Anwendungen für das Management Ihrer Warteschlangenmanager und zugehörigen Ressourcen entwickeln.

Von Anwendungen in IBM MQ for Multiplatforms ausführbare Aktionen

Sie können Anwendungen schreiben, die unter Multiplatforms folgende Aktionen ausführen:

- Nachrichten an andere Anwendungen senden, die unter denselben Betriebssystemen ausgeführt werden. Die Anwendungen können entweder auf demselben oder auf einem anderen System installiert sein.
- Nachrichten an Anwendungen senden, die auf IBM MQ-Plattformen ausgeführt werden.
- Message-Queuing in CICS für die folgenden Systeme verwenden:
  - 

TXSeries fürAIX
  - 

IBM i
  - 

Windows
- Message-Queuing in Encina für die folgenden Systeme verwenden:
  - 

AIX
  - 

Windows
- Message-Queuing in Tuxedo für die folgenden Systeme verwenden:
  - 

AIX
  - AT&T
  - 

Windows
- Mit IBM MQ als Transaktionsmanager Aktualisierungen koordinieren, die externe Ressourcenmanager inIBM MQ-Arbeitseinheiten vorgenommen haben. Die folgenden externen Ressourcenmanager werden unterstützt und entsprechen den Anforderungen der X/OPEN XA-Schnittstelle:

  - Db2
  - Informix
  - Oracle
  - Sybase
- Mehrere Nachrichten gemeinsam als eine Arbeitseinheit verarbeiten, die festgeschrieben oder zurückgesetzt werden kann.
- Ausführung in einer vollständigen IBM MQ-Umgebung oder in einer IBM MQ-Clientumgebung.

Von Anwendungen in IBM MQ for z/OS ausführbare Aktionen

Sie können Anwendungen schreiben, die unter z/OS folgende Aktionen ausführen:

- Message-Queuing in CICS oder IMS verwenden.
- Nachrichten zwischen Stapel-, CICS- und IMS-Anwendungen unter Auswahl der jeweils geeignetsten Umgebung für die betreffende Funktion senden.
- Nachrichten an Anwendungen senden, die auf IBM MQ-Plattformen ausgeführt werden.
- Mehrere Nachrichten gemeinsam als eine Arbeitseinheit verarbeiten, die festgeschrieben oder zurückgesetzt werden kann.
- Per IMS-Brücken-Nachrichten an IMS-Anwendungen senden bzw. mit diesen interagieren.
- An Arbeitseinheiten mitwirken, die von RRS koordiniert werden.

Jede Umgebung in z/OS zeichnet sich durch eigene Merkmale und Kenndaten sowie Vorteile und Nachteil aus. Der Vorteil von IBM MQ for z/OS liegt darin, dass die Anwendungen nicht an eine bestimmte Umgebung gebunden sind, sondern verteilt werden können, um die Vorteile der jeweiligen Umgebung zu nutzen. Sie können beispielsweise Endbenutzerschnittstellen mittels TSO oder CICS entwickeln, rechenintensive Module im z/OS-Stapel verarbeiten und Datenbankanwendungen in IMS oder CICS ausführen. In sämtlichen Fällen können die verschiedenen Anwendungskomponenten unter Verwendung von Nachrichten und Warteschlangen kommunizieren.

Entwickler von IBM MQ-Anwendungen müssen die Unterschiede und Einschränkungen dieser Umgebungen kennen und berücksichtigen. For example:

- IBM MQ stellt Funktionen für die übergreifende Kommunikation zwischen Warteschlangenmanagern bereit (auch als *verteilte Steuerung von Warteschlangen* bezeichnet).
- Der Stapel und die CICS-Umgebungen verwenden unterschiedliche Methoden zur Festschreibung und Zurücksetzung von Änderungen.
- IBM MQ for z/OS bietet Unterstützung in der IMS -Umgebung für Onlinenachrichtenverarbeitungsprogramme (MPPs), interaktive Fast Path-Programme (IFPs) und Stapelnachrichtenverarbeitungsprogramme (BMPs). Beachten Sie beim Schreiben von DL/I-Programmen die Anweisungen in den Abschnitten „Building z/OS batch applications” auf Seite 1077 und „z/OS batch considerations” auf Seite 769 für z/OS-Stapelprogramme.
- Obwohl mehrere Instanzen von IBM MQ for z/OS auf einem z/OS-System vorhanden sein können, kann eine CICS-Region jeweils nur eine Verbindung zu einem Warteschlangenmanager herstellen. Es können jedoch mehrere CICS-Regionen mit demselben Warteschlangenmanager verbunden werden. In den IMS- und z/OS-Stapelumgebungen können Programme Verbindungen zu mehreren Warteschlangenmanagern herstellen.
- Da IBM MQ for z/OS die gemeinsame Nutzung lokaler Warteschlangen durch mehrere Warteschlangenmanager unterstützt, werden Durchsatz und Verfügbarkeit verbessert. Solche Warteschlangen werden als *gemeinsam genutzte Warteschlangen* bezeichnet und die Warteschlangenmanager bilden eine *Gruppe mit gemeinsamer Warteschlange*, die Nachrichten in denselben gemeinsam genutzten Warteschlangen verarbeiten kann. Stapelanwendungen können eine Verbindung zu einem von mehreren Warteschlangenmanagern in einer Gruppe mit gemeinsamer Warteschlange herstellen, indem nicht der Name eines bestimmten Warteschlangenmanagers, sondern der Name der Gruppe mit gemeinsamer

Warteschlange angegeben wird. Dies wird als *Gruppen-Batch-Anschluss* oder kurz *Gruppenanschluss* bezeichnet. Weitere Informationen bietet der Abschnitt Gemeinsam genutzte Warteschlangen und Gruppen mit gemeinsamer Warteschlange.

z/OS Ausführliche Erläuterungen der Unterschiede zwischen den unterstützten Umgebungen und ihren jeweiligen Einschränkungen finden Sie in Abschnitt „Using and writing applications on IBM MQ for z/OS" auf Seite 938.

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung" auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

„Konzepte für den Entwurf von IBM MQ-Anwendungen" auf Seite 52
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

„Prozedurale Anwendungen für die Warteschlangensteuerung erstellen" auf Seite 758
Dieser Abschnitt enthält Informationen zum Erstellen von Anwendungen für die Warteschlangensteuerung, zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager und zum Öffnen und Schließen von Objekten.

„Prozedurale Clientanwendungen schreiben" auf Seite 961
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„IBM MQ classes for JMS/Jakarta Messaging verwenden" auf Seite 85
IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Java -Messaging-Provider, die mit IBM MQbereitgestellt werden. Neben der Implementierung der in den Spezifikationen JMS und Jakarta Messaging definierten Schnittstellen fügen diese Messaging-Provider zwei Gruppen von Erweiterungen zur Java -Messaging-API hinzu.

„IBM MQ classes for Java verwenden" auf Seite 364
Verwenden Sie 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.

„C++-Anwendungen entwickeln" auf Seite 554
In IBM MQ werden C++-Klassen bereitgestellt, die IBM MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen.

„Prozedurale Anwendung erstellen" auf Seite 1052
Sie können eine IBM MQ-Anwendung in einer von mehreren prozeduralen Sprachen schreiben und die Anwendung auf mehreren unterschiedlichen Plattformen ausführen.

Zugehörige Tasks

„Prozedurale IBM MQ-Beispielprogramme verwenden" auf Seite 1111
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

„.NET-Anwendungen entwickeln" auf Seite 582
IBM MQ classes for .NET ermöglichen .NET -Anwendungen, eine Verbindung zu IBM MQ als IBM MQ MQI client oder direkt zu einem IBM MQ -Server herzustellen.

„Microsoft Windows Communication Foundation -Anwendungen mit IBM MQ entwickeln" auf Seite 1327
Der angepasste WCF-Kanal (Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

Sicherung

Multi Anwendungen, Anwendungsnamen und Anwendungsinstanzen

Bevor Sie beginnen, Ihre Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden Konzepten zu Anwendungen, Anwendungsnamen und Anwendungsinstanzen vertraut machen.

Anwendungen

Multi

Verbindungen zu einem Warteschlangenmanager werden als die gleiche *Anwendung* angesehen, wenn der gleiche *Anwendungsname* angegeben wird. Der Anwendungsname wird als Attribut APPLTAG des Befehls DISPLAY CONN(*) TYPE CONN angezeigt.

Anmerkungen:

1. Bei Anwendungen, die einen IBM MQ client mit einer früheren Version als IBM MQ 9.1.2 verwenden, wird der Anwendungsname automatisch durch den IBM MQ client festgelegt. Der zugehörige Wert ist von der Anwendungsprogrammiersprache und der Plattform abhängig, auf der die Anwendung ausgeführt wird. Weitere Informationen finden Sie unter PutApplName.
2. Bei IBM MQ client-Anwendungen, die einen IBM MQ client der Version IBM MQ 9.1.2 oder höher verwenden, kann der Anwendungsname auf einen bestimmten Wert festgelegt werden. In den meisten Fällen sind dafür keine Änderungen am Anwendungscode oder eine erneute Kompilierung der Anwendung erforderlich. Weitere Informationen finden Sie im Abschnitt „Anwendungsnamen in unterstützten Programmiersprachen verwenden” auf Seite 57.

Anwendungsinstanzen

Multi

Verbindungen werden weiter in *Anwendungsinstanzen* unterteilt. Eine Instanz einer Anwendung ist eine Gruppe eng verwandter Verbindungen, die eine 'Ausführungseinheit' für diese Anwendung bereitstellen. In der Regel handelt es sich um einen einzelnen Betriebssystemprozess, der eine Reihe von Threads und zugehörigen IBM MQ-Verbindungen aufweisen kann.

In IBM MQ for Multiplatforms ist eine Anwendungsinstanz einem bestimmten Verbindungstag zugeordnet. Der Warteschlangenmanager ordnet neue Verbindungen automatisch einer vorhandenen Anwendungsinstanz zu, wenn er erkennen kann, dass sie zusammen gehören.

Anmerkungen:

- Bei Verwendung von Clientverbindungen können diese Prozesse über einen oder mehrere aktive Kanäle eine Verbindung zum Warteschlangenmanager herstellen.
- Bei JMS-Anwendungen wird eine Anwendungsinstanz einer bestimmten JMS-Verbindung und allen ihr zugehörigen JMS-Sitzungen zugeordnet.

Anwendungsinstanzen sind besonders wichtig in IBM MQ for Multiplatforms, wenn der automatische Lastausgleich für Anwendungen in einem einheitlichen Cluster verwendet wird. Auf Plattformen mit IBM MQ for Multiplatforms können Sie mit dem Befehl DISPLAY APSTATUS die aktuell verbundenen Anwendungsinstanzen anzeigen.

In einigen Fällen kann der Warteschlangenmanager die Verbindung zu einer Anwendungsinstanzzuordnung nicht ordnungsgemäß herstellen, insbesondere in folgenden Fällen:

- Wenn ausgehend von demselben Prozess mehrere Verbindungen für eine gemeinsame Dialognutzung unter Verwendung unterschiedlicher Anwendungsnamen hergestellt werden.
- Wenn Clientbibliotheken einer früheren Version verwendet werden. Dies gilt beispielsweise für IBM MQ JMS-Clientinstallationen der IBM MQ 9.1.2 und früher.

Wenn sich die Anwendungen in diesen Fällen nicht als wiederverbindungsfähig definieren, ist dies zwar zulässig, aber einige der Anwendungsinstanzgruppierungen sind dann möglicherweise falsch. Wenn für einige der Verbindungen der Status MQCNO_RECONNECT deklariert ist, wirkt sich dies in erheblichem

Maße negativ auf den Lastausgleich für Anwendungen aus, und als Folge wird der MQCONN-Aufruf mit dem Fehler MQCNO_RECONNECT_INCOMPATIBLE abgelehnt.

Zugehörige Konzepte

„Anwendungsname in unterstützten Programmiersprachen angeben" auf Seite 55
Bereits vor IBM MQ 9.2.0 konnten Sie einen Anwendungsnamen für Java- oder JMS-Clientanwendungen angeben. Ab IBM MQ 9.2.0 wird diese Funktion in IBM MQ for Multiplatforms auf andere Programmiersprachen erweitert.

Anwendungsprogramme, die die MQI verwenden

Zur erfolgreichen Ausführung benötigen IBM MQ-Anwendungsprogramme bestimmte Objekte.

Abbildung 1 auf Seite 11 zeigt eine Anwendung, die Nachrichten aus einer Warteschlange entfernt, sie verarbeitet und anschließend einige Ergebnisse an eine andere Warteschlange im selben Warteschlangenmanager sendet.

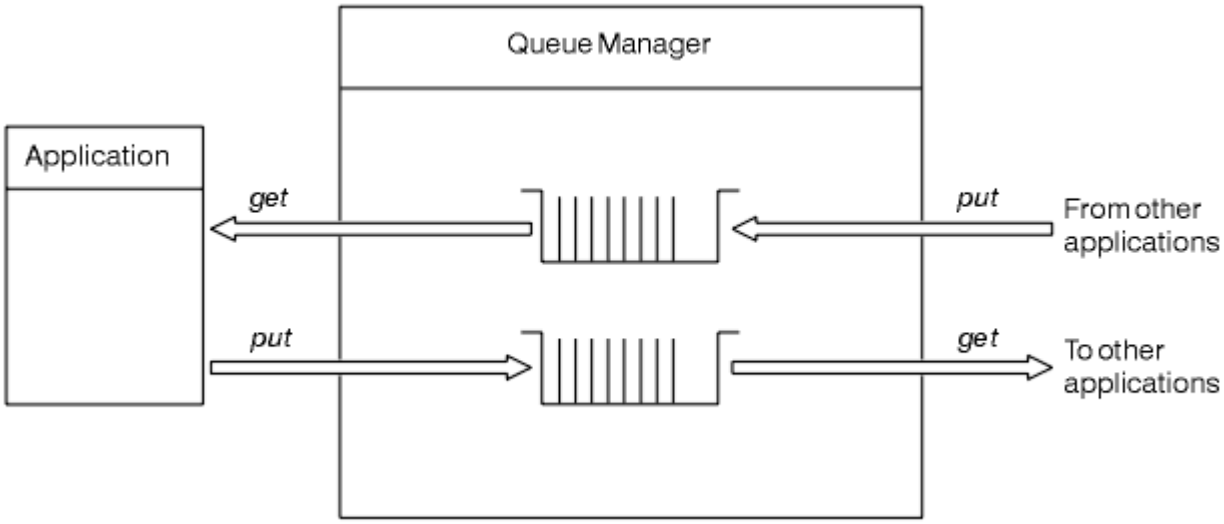


Abbildung 1. Warteschlangen, Nachrichten und Anwendungen

Anwendungen können Nachrichten mittels MQPUT zwar in lokale oder ferne Warteschlangen einreihen, das direkte Abrufen mittels MQGETist jedoch nur von lokalen Warteschlangen möglich.

Folgende Bedingungen müssen erfüllt sein, um diese Anwendung ausführen zu können:

- Der Warteschlangenmanager muss vorhanden und aktiv sein.
- Die erste Anwendungswarteschlange, aus der die Nachrichten entfernt werden, muss definiert sein.
- Die zweite Warteschlange, in die die Anwendung Nachrichten stellt, muss ebenfalls definiert sein.
- Die Anwendung muss eine Verbindung mit dem Warteschlangenmanager herstellen können. Dazu ist einer Verknüpfung mit IBM MQ erforderlich. Siehe „Prozedurale Anwendung erstellen" auf Seite 1052.
- Die Anwendungen, die die Nachrichten in die erste Warteschlange stellen, müssen ebenfalls mit einem Warteschlangenmanager verbunden sein. Handelt es sich um ferne Anwendungen, müssen für sie außerdem Übertragungswarteschlangen und Kanäle konfiguriert sein. Diese Systemkomponente ist in Abbildung 1 auf Seite 11 nicht dargestellt.

Clientverbindungen für die Verbindung zu mehreren IBM MQ-Warteschlangenmanagern verwenden

Es ist möglich, Clientanwendungen so zu konfigurieren, dass sie eine Verbindung zu mehreren Warteschlangenmanagern herstellen (aus Gründen des Lastausgleichs oder der Serviceverfügbarkeit).

Die primären Mechanismen, um dies im IBM MQ -Client zu erreichen, sind die Verwendung von Definitionstabellen für Clientkanäle (siehe Definitionstabellen für Clientkanäle konfigurieren) oder Verbindungslisten.

Es ist auch möglich, ein ähnliches Verhalten mit externen Lastausgleichsprodukten zu erzielen oder IBM MQ -Verbindungscode in einen 'Stub' einzuschließen, der Hostnamen oder IP-Adressen umleiten kann.

Jedes dieser Verfahren hat einige Einschränkungen und kann für bestimmte Anwendungsanforderungen mehr oder weniger geeignet sein. In den folgenden Abschnitten werden, obwohl sie nicht vollständig sind, bestimmte Aspekte beschrieben, die Sie berücksichtigen sollten, sowie die Auswirkungen dieser verschiedenen Ansätze auf diese Aspekte.

IBM MQ -Uniform-Cluster (siehe Informationen zu Uniform-Clustern) stellen einen leistungsfähigen Mechanismus für die horizontale Skalierung von Anwendungen über mehrere Warteschlangenmanager hinweg bereit, der auf dem Basismechanismus der CCDT aufbaut, um mehrere Ziele bereitzustellen. Uniform-Cluster können Funktionen bereitstellen, die über das hinausgehen, was mit einer externen Lastausgleichsfunktion möglich ist, die die zugrunde liegenden IBM MQ -Protokolle nicht kennt, und einige der unten beschriebenen Probleme vermeiden. Daher sollten Sie die Verwendung eines Uniform-Clusters gegenüber anderen Verfahren, soweit zutreffend, bevorzugen.

Achtung: Sie sollten Anwendungen mit Vorsicht verwenden, die IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingverwenden, einschließlich Anwendungen, die einen der IBM MQ -Ressourcenadapter verwenden, die über Lastausgleichstechnologien eine Verbindung zu Warteschlangenmanagern herstellen. Wenn Probleme auftreten, reproduzieren Sie diese Probleme, ohne den Lastausgleich zu verwenden.

Es gibt mehrere Probleme, die alle bedeuten, dass solche Verbindungen bestenfalls problematisch und schlimmstenfalls völlig unzuverlässig sind:

- Besondere Vorsicht ist erforderlich, wenn eine Anwendung, die mehrere Verbindungen zum Warteschlangenmanager herstellt, eine beliebige Form des Lastausgleichs verwendet. Dies schließt alle Anwendungen ein, die IBM MQ Classes for JMS/Jakarta Messaging verwenden, da diese mehrere IBM MQ -Verbindungen in allgemeiner Verwendung erstellen. Wenn Sie eine externe Lastausgleichsfunktion oder einen angepassten Code-Stub verwenden, muss dies jederzeit Verbindungen von derselben Anwendungsinstanz zu demselben Warteschlangenmanager weiterleiten.
- Die Verwendung von XA-Transaktionsmanagement oder JTA (Java Transaction API) beruht auf der Fähigkeit, konsistent eine Verbindung zu demselben Warteschlangenmanager herzustellen. In der Praxis ist dies bei jeder Form des Lastausgleichs wahrscheinlich nicht praktikabel.
- -Die einheitliche Clusterverwaltung beruht auf der Möglichkeit, Clients anzuweisen, die Verbindung zu bestimmten Warteschlangenmanagern ohne Beeinträchtigung wiederherzustellen. Es ist nicht ratsam, den externen Lastausgleich mit der Verwendung von Uniform-Clustern zu kombinieren

Sie sollten die IBM MQ -Funktionalität für einheitliche Cluster verwenden, um eine horizontale Skalierung von Anwendungen über mehrere Warteschlangenmanager hinweg zu erreichen, anstatt externe Lastausgleichstechnologien zu verwenden. Informationen zu Uniform-Clustern, einschließlich der Erstellung und Verwendung von Uniform-Clustern, finden Sie unter Uniform-Cluster konfigurierenund in den folgenden Abschnitten.

In diesen Informationen verwendete Begriffe

CCDT-multi-QMGR (CCDT für mehrere Warteschlangenmanager)

Eine CCDT-Datei, die mehrere Clientverbindungskanäle (CLNTCONN) mit derselben Gruppe enthält, d. h. das Attribut QMNAME CLNTCONN (Warteschlangenmanagername-Clientverbindung), wobei verschiedene CLNTCONN-Einträge in verschiedene Warteschlangenmanager aufgelöst werden.

Dies ist anders als bei einer CCDT-Datei, die mehrere CLNTCONN-Einträge enthält, bei denen es sich einfach um unterschiedliche IP-Adressen oder Hostnamen für denselben Multi-Instanz-Warteschlangenmanager handelt - ein Konzept, für das Sie sich entscheiden könnten, um es mit einem Code-Stub zu kombinieren.

Wenn Sie sich für ein CCDT-Konzept mit mehreren Warteschlangenmanagern entscheiden, müssen Sie festlegen, ob die Einträge priorisiert werden sollen oder ein randomisiertes Workload-Management (WLM) erfolgen soll:

Priorisiert
Verwendung mehrerer alphabetisch geordneter Einträge mit den Attributen CLNTWGHT(1) und AFFINITY(PREFERRED), um die letzte gute Verbindung zu speichern.

Randomisiert
Verwendung der Attribute CLNTWGHT(1) und AFFINITY(NONE). Sie können die WLM-Gewichtung auf unterschiedlich skalierten IBM MQ-Servern durch Anpassung von CLNTWGHT anpassen.

Anmerkung: Große Unterschiede zwischen den Kanälen bei CLNTWGHT sollten vermieden werden.

Lastausgleichsfunktion
Eine Netzappliance mit einer virtuellen IP-Adresse (VIP), die mit Anschlussüberwachung der TCP/IP-Listener von mehreren IBM MQ-Warteschlangenmanagern konfiguriert ist. Die Konfiguration der VIP in der Netzappliance hängt von der verwendeten Netzappliance ab.

Die folgenden Auswahlmöglichkeiten beziehen sich nur auf Anwendungen, die Nachrichten senden oder synchrone Anforderungs- und Antwortnachrichten initialisieren. Für Anwendungen, die diese Nachrichten und Anforderungen verarbeiten, beispielsweise Listener, gelten völlig andere Überlegungen, die im Abschnitt "Connecting a message listener to a queue" ausführlich erörtert werden.

Umfang der erforderlichen Codeänderung für vorhandene Anwendungen, die eine Verbindung zu einem einzelnen Warteschlangenmanager herstellen

CONNAME-Liste, CCDT multi-QMGR und Lastausgleichsfunktion
MQCONN("QMNAME") in MQCONN("*QMNAME")

Der Name des Warteschlangenmanagers könnte in der JNDI-Konfiguration (Java Naming and Directory Interface) für Java Platform, Enterprise Edition-Anwendungen (Java EE) enthalten sein. Andernfalls ist eine ein Zeichen betreffende Codeänderung erforderlich.

Code-Stub
Ersetzen Sie vorhandene JMS-oder MQI-Verbindungslogik durch einen Code-Stub.

Unterstützung für verschiedene WLM-Strategien

CONNAME-Liste
Nur priorisiert.

Dies wird sich wahrscheinlich negativ auf den Code auswirken.

CCDT multi-QMGR (CCDT, mehrere Warteschlangenmanager)
Priorisiert oder randomisiert.

Dies wird wahrscheinlich keine Auswirkungen auf den Code haben.

Lastausgleichsfunktion
Alle, einschließlich jeder einzelnen Verbindung für alle Nachrichten.

Dies wird sich wahrscheinlich positiv auf den Code auswirken.

Code-Stub
Alle, einschließlich jeder einzelnen Nachricht für alle Nachrichten.

Dies wird sich wahrscheinlich positiv auf den Code auswirken.

Leistungseinbußen, wenn der primäre Warteschlangenmanager nicht verfügbar ist

CONNAME-Liste
Es wird immer der erste Eintrag in der Liste ausprobiert.

Dies wird sich wahrscheinlich negativ auf den Code auswirken.

CCDT multi-QMGR (CCDT, mehrere Warteschlangenmanager)
Speichert die letzte gute Verbindung.

Dies wird sich wahrscheinlich positiv auf den Code auswirken.

Lastausgleichsfunktion
Durch Anschlussüberwachung werden fehlerhafte Warteschlangenmanager vermieden.

Dies wird sich wahrscheinlich positiv auf den Code auswirken.

Code-Stub
Kann die letzte gute Verbindung speichern und einen intelligenten Wiederholungsversuch unternehmen.

Dies wird sich wahrscheinlich positiv auf den Code auswirken.

Unterstützung für XA-Transaktionen

CONNAME-Liste, CCDT multi-QMGR und Lastausgleichsfunktion
Der Transaktionsmanager muss Wiederherstellungsinformationen speichern, die erneut eine Verbindung zu derselben Warteschlangenmanagerressource herstellen.

Ein MQCONN-Aufruf, der in verschiedene Warteschlangenmanager aufgelöst wird, macht dies in der Regel ungültig. In Java EE beispielsweise sollte eine einzelne Verbindungsfactory bei Verwendung von XA in einen einzelnen Warteschlangenmanager aufgelöst werden.

Dies wird sich wahrscheinlich negativ auf den Code auswirken.

Code-Stub
Der Code-Stub kann die XA-Anforderungen für einen Transaktionsmanager, beispielsweise mehrere Verbindungsfactorys, erfüllen.

Dies wird sich wahrscheinlich positiv auf den Code auswirken.

Verwaltungsflexibilität, um Infrastrukturänderungen vor Apps zu verbergen

CONNAME-Liste
Nur DNS.

Dies wird sich wahrscheinlich negativ auf den Code auswirken.

CCDT multi-QMGR (CCDT, mehrere Warteschlangenmanager)
DNS und gemeinsam genutztes Dateisystem oder gemeinsam genutztes Dateisystem oder CCDT-Datei-Push-Operation.

Dies wird wahrscheinlich keine Auswirkungen auf den Code haben.

Lastausgleichsfunktion
Dynamische virtuelle IP-Adresse (VIP).

Dies wird sich wahrscheinlich positiv auf den Code auswirken.

Code-Stub
DNS oder CCDT-Einträge eines einzelnen Warteschlangenmanagers.

Dies wird wahrscheinlich keine Auswirkungen auf den Code haben.

Vermeidung von Unterbrechungen bei geplanter Wartung

Sie müssen noch eine weitere Situation berücksichtigen und einplanen: die Vermeidung von Anwendungsunterbrechungen, also beispielsweise Fehlern und Timeouts, die für den Endbenutzer erkennbar sind, während der geplanten Wartung eines Warteschlangenmanagers. Der beste Ansatz zur Vermeidung von Unterbrechungen besteht darin, alle Arbeiten von einem Warteschlangenmanager zu entfernen, bevor dieser gestoppt wird.

Stellen Sie sich ein Szenario mit Anforderungen und Antworten vor. Sie möchten, dass alle momentan ausgeführten Anforderungen abgeschlossen und die Antworten von der Anwendung verarbeitet werden, es soll jedoch keine weitere Arbeit an das System übergeben werden. Den Warteschlangenmanager einfach stillzulegen, wird diesem Anspruch nicht gerecht, da gut codierte Anwendungen eine Ausnahme mit

dem Rückkehrcode RC2161 MQRC_Q_MGR_QUIESCING empfangen, bevor sie ihre Antwortnachrichten für momentan ausgeführte Anforderungen erhalten.

Sie können für die Anforderungswarteschlangen, über die Arbeit übergeben wird, PUT(DISABLED) festlegen und gleichzeitig für die Antwortwarteschlangen PUT(ENABLED) und GET(ENABLED) belassen. Auf diese Weise können Sie die Verschachtelungstiefe der Anforderungs-, Übertragungs- und Antwortwarteschlangen überwachen. Wenn sich alle stabilisiert haben, also momentan ausgeführte Anforderungen abgeschlossen sind oder das Zeitlimit erreicht haben, können Sie den Warteschlangenmanager stoppen.

Allerdings müssen die anfordernden Anwendungen gut codiert sein, um eine Anforderungswarteschlange, für die PUT(DISABLED) eingestellt ist, handhaben zu können, bei der der Rückkehrcode RC2051 MQRC_PUT_INHIBITED ausgegeben wird, wenn versucht wird, eine Nachricht zu senden.

Beachten Sie, dass es nicht zu der Ausnahmebedingung kommt, wenn die Verbindung zu IBM MQ hergestellt oder die Anforderungswarteschlange geöffnet wird. Die Ausnahmebedingung tritt nur auf, wenn versucht wird, mit dem MQPUT-Aufruf eine Nachricht zu senden.

Wenn Sie einen Code-Stub erstellen, der diese Logik zur Fehlerbehandlung für Szenarios mit Anforderungen und Antworten enthält, und Ihre Anwendungsteams bitten, in Zukunft einen solchen Code-Stub zu verwenden, kann Ihnen dies helfen, Anwendungen mit konsistentem Verhalten zu entwickeln.

Flexible und skalierbare Clientanwendungen entwickeln

Zur Unterstützung von Fehlertoleranz und Skalierbarkeit können Clientanwendungen, die Verbindungsoptionen unterstützen, in Uniform-Clustern implementiert werden, damit die Instanzen der Anwendung zwischen den Warteschlangenmanagern neu verteilt werden können.

Einen Überblick über Uniform-Cluster bietet der Abschnitt Informationen zu Uniform-Clustern.

Im Idealfall ist diese Neuverteilung für die Anwendung nicht sichtbar; geeignet für diesen Implementierungstyp sind jedoch nur bestimmte Anwendungstypen und unter Umständen müssen beim Anwendungsdesign bestimmte Aspekte berücksichtigt werden.

Diese Aspekte beziehen sich auf zwei Hauptkategorien:

- Seltene *Fehlerpfade*, die es möglicherweise bereits für wiederverbindbare Anwendungen gibt, die aber wahrscheinlicher werden, wenn die Implementierung in einem Uniform-Cluster erfolgt. Beispielsweise könnten nach einer Verbindungswiederholung alle gerade ausgeführten Arbeitseinheiten und Anzeigecursor zurückgesetzt werden. In der aktuellen Umgebung Ihrer wiederverbindbaren Anwendung könnte es sich hierbei um ein seltenes Ereignis handeln, das daher im Anwendungscode nicht so exakt wie möglich verarbeitet wird. Eine Überprüfung der Anwendungslogik, mit der eine geeignete Behandlung solcher Situationen sichergestellt wird, trägt dazu bei, dass das Auftreten unerwarteter Probleme vermieden wird.
- *Affinitäten* für einen bestimmten Warteschlangenmanager. Wenn Sie wissen, dass eine Anwendung immer eine Verbindung zu demselben oder zu einem bestimmten Warteschlangenmanager herstellen muss, sollte die Anwendung so konfiguriert sein, dass die Verbindung zu diesem Warteschlangenmanager wiederholt wird oder dass die Verbindung zu diesem Warteschlangenmanager nicht aktiviert ist. Solche Affinitäten können jedoch temporär sein, beispielsweise beim Warten auf eine Antwortnachricht. Die Beeinflussung des Ausgleichsalgorithmus zur Berücksichtigung dieser Affinitäten über den Anwendungscode ist im folgenden Abschnitt erläutert. Weitere Informationen zu diesen Optionen und zur Umsetzung einer ähnlichen Strategie über die Konfiguration und nicht über den Anwendungscode finden Sie unter Neuverteilung von Anwendungen in Uniform-Clustern beeinflussen.

Optionen für Verbindungswiederholung in der MQI beeinflussen

Weitere Informationen zu MQCNO_RECONNECT finden Sie unter Optionen für Verbindungswiederholung.

Wenn Sie wissen, dass eine Anwendung immer eine Verbindung zu demselben oder zu einem bestimmten Warteschlangenmanager herstellen muss, sollte für sie die Einstellung MQCNO_RECONNECT_Q_MGR oder MQCNO_RECONNECT_DISABLED konfiguriert sein.

Ausgleichsalgorithmus in der MQI beeinflussen

Es kann jedoch sinnvoll sein, das Neuverteilungsverhalten zu steuern oder zu beeinflussen, um den Anforderungen bestimmter Anwendungstypen gerecht zu werden, also beispielsweise Unterbrechungen von gerade ausgeführten Transaktionen zu minimieren oder auch sicherzustellen, dass anfordernde Anwendungen vor dem Versetzen ihre Antworten empfangen.

Im Abschnitt Neuverteilung von Anwendungen in Uniform-Clustern beeinflussen werden bestimmte erwünschte Verhaltensmuster angenommen und erläutert. Das Verhalten bestimmter Anwendungen kann über die Datei 'client.ini' aber auch zum Zeitpunkt der Konfiguration oder der Implementierung beeinflusst werden; dies ist im genannten Abschnitt beschrieben.

In anderen Situationen kann es sinnvoller sein, das Ausgleichsverhalten und die Anforderungen in die Anwendungslogik zu integrieren. In diesen Fällen können dieselben relevanten Merkmale der Anwendung bei der Verbindung mit dem Warteschlangenmanager im MQCONNX-Aufruf in einer Struktur namens MQBNO (Ausgleichsoptionen) an IBM MQ übergeben werden.

Wenn Sie eine MQBNO-Struktur angeben, muss sie alle von IBM MQ benötigten Informationen bereitstellen, damit entschieden werden kann, wie und wann die Anwendung aufgefordert werden soll, die Verbindung zu einem anderen Warteschlangenmanager wiederherzustellen.

Sie müssen Folgendes angeben:

- **Type** der Anwendung
- Zeitlimit für die Neuverteilung der Instanz ungeachtet des Status (**Timeout**)
- Gegebenenfalls spezielle Ausgleichsoptionen (**BalanceOptions**)

Eine Ausnahme bildet in diesem Fall, dass Sie für das Zeitlimit bei Bedarf die Einstellung MQBNO_TIMEOUT_DEFAULT beibehalten können. Dann wird das Zeitlimit (sofern angegeben) in einen beliebigen Wert aus der Datei 'client.ini', der Anwendung oder der globalen Zeilengruppen aufgelöst bzw. bei einer fehlgeschlagenen Auflösung in den Basisstandardwert von 10 Sekunden.

Weitere Informationen zum Format dieser Struktur bietet der Abschnitt MQBNO.

Informationen zu den .NET-Anwendungen finden Sie unter Neuverteilung von Anwendungen in .NET beeinflussen.

Objektorientierte Anwendungen

IBM MQ unterstützt JMS, Java, C++ und .NET. Diese Sprachen und Frameworks verwenden das IBM MQ-Objektmodell. Es stellt Klassen bereit, die über dieselbe Funktionalität verfügen wie Aufrufe und Strukturen von IBM MQ.

Einige der Sprachen und Frameworks, die das IBM MQ-Objektmodell verwenden, unterstützen zusätzliche Funktionen, die den prozeduralen Programmiersprachen, die die Schnittstelle für Nachrichtenwarteschlangen (MQI) verwenden, nicht zur Verfügung stehen.

Ausführliche Informationen zu den Klassen, Methoden und Eigenschaften dieses Modells finden Sie unter „IBM MQ-Objektmodel” auf Seite 17.

JMS

IBM MQ stellt Klassen bereit, die die Spezifikationen Jakarta Messaging 3.0 und Java Message Service 2.0 implementieren. Details zu IBM MQ classes for JMSfinden Sie unter IBM MQ classes for JMS verwenden. Informationen zu den Unterschieden zwischen IBM MQ classes for Java und IBM MQ classes for JMS, die Ihnen bei der Entscheidung helfen, welche verwendet werden sollen, finden Sie in „JMS/Jakarta Messaging -und Java -Anwendungen entwickeln” auf Seite 85.

IBM MQ Message Service Client (XMS) for C/C++ und IBM MQ Message Service Client (XMS) for .NET stellen eine Anwendungsprogrammierschnittstelle (API) namens XMS XMS bereit. Diese besitzt dieselben Schnittstellen wie die Java Message Service (JMS)-API. Weitere Informationen finden Sie unter „XMS .NET-Anwendungen entwickeln” auf Seite 646.

Java
Informationen zum Codieren von Programmen mit dem IBM MQ -Objektmodell in Javafinden Sie unter IBM MQ classes for Java verwenden .

Stabilized IBM wird keine weiteren Erweiterungen für die IBM MQ classes for Java durchführen; sie werden auf der in IBM MQ 8.0 ausgelieferten Stufe stabilisiert. Informationen zu den Unterschieden zwischen den IBM MQ classes for Java und den IBM MQ classes for JMS, die Ihnen bei der Entscheidung für eine dieser Technologien helfen, finden Sie im Abschnitt „JMS/Jakarta Messaging -und Java -Anwendungen entwickeln” auf Seite 85.

C++
In IBM MQ werden C++-Klassen bereitgestellt, die IBM MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen. Informationen zum Codieren von Programmen mit dem IBM MQ-Objektmodell in C++ finden Sie unter C++ verwenden. Message Service Clients für C/C++ und .NET stellen eine Anwendungsprogrammierschnittstelle (API) namens XMS bereit. Diese besitzt dieselben Schnittstellen wie die API von Java Message Service (JMS).

.NET
Informationen zum Codieren von .NET -Programmen mithilfe der IBM MQ .NET -Klassen finden Sie in .NET-Anwendungen entwickeln . Message Service Clients für C/C++ und .NET stellen eine Anwendungsprogrammierschnittstelle (API) namens XMS bereit. Diese besitzt dieselben Schnittstellen wie die Java Message Service (JMS)-API.

Zugehörige Konzepte

„MQI-Anwendungen mit IBM MQ entwickeln” auf Seite 754
IBM MQ unterstützt die Programmiersprachen C, Visual Basic, COBOL, Assembler, RPG, pTAL und PL/I. Diese prozeduralen Programmiersprachen rufen Message-Queuing-Services über die Schnittstelle für Nachrichtenwarteschlangen (MQI) auf.

Technische Übersicht

„Konzepte für die Anwendungsentwicklung” auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

Zugehörige Verweise

Referenzinformationen zum Entwickeln von Anwendungen

IBM MQ-Objektmodell

Das IBM MQ-Objektmodell besteht aus Klassen, Methoden und Eigenschaften.

Das IBM MQ-Objektmodell besteht aus Folgendem:

- *Klassen* stellen bekannte IBM MQ-Konzepte wie Warteschlangenmanager, Warteschlangen und Nachrichten dar.
- *Methoden* der einzelnen Klassen entsprechend MQI-Aufrufen.
- *Eigenschaften* der einzelnen Klassen entsprechend den Attributen von IBM MQ-Objekten.

Wenn Sie eine IBM MQ-Anwendung mithilfe des IBM MQ-Objektmodells erstellen, werden Instanzen dieser Klassen in der Anwendung erstellt. In der objektorientierten Programmierung wird die Instanz einer Klasse als *Objekt* bezeichnet. Wenn ein Objekt erstellt wurde, interagieren Sie mit dem Objekt, indem Sie die Werte seiner Eigenschaften untersuchen oder festlegen (gleichbedeutend mit der Ausgabe eines MQINQ- oder MQSET-Aufrufs) sowie über Methodenaufrufe des Objekts (gleichbedeutend mit der Ausgabe der anderen MQI-Aufrufe).

Klassen

Das IBM MQ-Objektmodell enthält folgenden Basissatz an Klassen.

Die eigentliche Implementierung des Modells variiert je nach unterstützter objektorientierter Umgebung.

MQQueueManager
Ein Objekt der Klasse 'MQQueueManager' stellt eine Verbindung zu einem Warteschlangenmanager dar. Es verfügt über die Methoden Connect(), Disconnect(), Commit() und Backout() (Verbinden, Verbindung trennen, Festschreiben und Zurücksetzen (entsprechen MQCONN bzw. MQCONNX, MQDISC, MQCMIT und MQBACK). Seine Eigenschaften entsprechen den Attributen eines Warteschlangenmanagers. Beim Zugriff auf die Eigenschaft eines Warteschlangenmanagerattributs wird implizit eine Verbindung zum Warteschlangenmanager hergestellt, falls diese nicht bereits besteht. Beim Löschen eines MQQueueManager-Objekts wird die Verbindung zum Warteschlangenmanager implizit getrennt.

MQQueue
Ein Objekt der Klasse 'MQQueue' stellt eine Warteschlange dar. Es verfügt über Put()- und Get()-Methoden zum Einreihen von Nachrichten in die Warteschlange bzw. zum Abrufen von Nachrichten aus der Warteschlange (entsprechen MQPUT und MQGET). Seine Eigenschaften entsprechen den Attributen einer Warteschlange. Beim Zugriff auf die Eigenschaft eines Warteschlangenattributs oder beim Aufruf einer Put()- oder Get()-Methode wird die Warteschlange implizit geöffnet (entspricht MQOPEN). Beim Löschen eines MQQueue-Objekts wird die Warteschlange implizit geschlossen (entspricht MQCLOSE).

MQTopic
Ein Objekt der Klasse 'MQTopic' stellt ein Thema dar dar. Es verfügt über Put()- und Get()-Methoden zum Veröffentlichen von Nachrichten im Thema bzw. zum Empfangen oder Subskribieren von Nachrichten aus dem Thema (entsprechen MQPUT und MQGET). Seine Eigenschaften entsprechen den Attributen eines Themas. Ein MQTopic-Objekt kann immer nur entweder zur Veröffentlichung oder zur Subskription aufgerufen werden. Ein gleichzeitiger Zugriff ist nicht möglich. Wenn das MQTopic-Objekt für empfangende Nachrichten verwendet wird, kann es mit einer nicht verwalteten oder verwalteten Subskription und als ein dauerhafter oder nicht dauerhafter Subskribent erstellt werden. Für diese unterschiedlichen Szenarios stehen mehrere überladene Konstruktoren zur Verfügung.

MQMessage
Ein Objekt der Klasse 'MQMessage' stellt eine Nachricht dar, die in eine Warteschlange eingereiht oder aus einer Warteschlange abgerufen wird. Es enthält einen Puffer und bindet sowohl Anwendungsdaten als auch MQMD ein. Seine Eigenschaften entsprechen MQMD-Feldern und es verfügt über Methoden, mit denen Sie unterschiedliche Benutzerdaten (z. B. Zeichenfolgen, lange und kurze Ganzzahlen, einzelne Bytes) in den Puffer schreiben bzw. aus dem Puffer lesen können.

MQPutMessageOptions
Ein Objekt der Klasse 'MQPutMessageOptions' stellt die MQPMO-Struktur dar. Seine Eigenschaften entsprechen MQPMO-Feldern.

MQGetMessageOptions
Ein Objekt der Klasse 'MQGetMessageOptions' stellt die MQGMO-Struktur dar. Seine Eigenschaften entsprechen MQGMO-Feldern.

MQProcess
Ein Objekt der Klasse 'MQProcess' stellt eine Prozessdefinition dar (verwendet bei Triggering). Seine Eigenschaften stellen die Attribute einer Prozessdefinition dar.

MQDistributionList
Ein Objekt der Klasse 'MQDistributionList' stellt eine Verteilerliste dar (zum Senden mehrerer Nachrichten mit einem einzelnen MQPUT-Befehl). Es enthält eine Liste mit MQDistributionListItem-Objekten.

MQDistributionListItem
Ein Objekt der Klasse 'MQDistributionListItem' stellt ein einzelnes Verteilerlistenziel dar. Es bindet die MQOR-, MQRR- und MQPMR-Strukturen ein und seine Eigenschaften entsprechen den Feldern dieser Strukturen.

Objektverweise

In einem IBM MQ-Programm, das die MQI verwendet, gibt IBM MQ Verbindungs- und Objektkennungen an das Programm zurück.

Diese Kennungen müssen bei nachfolgenden IBM MQ-Aufrufen als Parameter übergeben werden. Das IBM MQ-Objektmodell blendet die Kennungen für das Anwendungsprogramm aus. Stattdessen führt die Erstellung eines Objekts aus einer Klasse zur Rückgabe eines Objektverweises an das Anwendungsprogramm. Dieser Objektverweis wird bei Methodenaufrufen und beim Zugriff auf Eigenschaften des Objekts verwendet.

Rückkehrcodes

Die Ausgabe eines Methodenaufrufs oder das Einrichten eines Eigenschaftswerts führt zum Festlegen von Rückgabecodes.

Diese Rückgabecodes bestehen aus einem Beendigungscode und einem Ursachencode, beide sind Eigenschaften des Objekts. Die Werte des Beendigungscodes und des Ursachencodes sind mit den für die MQI definierten Werten identisch, zusätzlich gibt es noch einige spezifische Werte für die objektorientierte Umgebung.

IBM MQ-Nachrichten

Eine IBM MQ-Nachricht besteht aus Nachrichteneigenschaften und Anwendungsdaten. Der Message-Queuing-Nachrichtendeskriptor (MQMD) enthält die Steuerinformationen, die bei der Übertragung einer Nachricht zwischen der sendenden und empfangenden Anwendung mit den Anwendungsdaten mitgesendet werden.

Bestandteile einer Nachricht

IBM MQ-Nachrichten bestehen aus zwei Teilen:

- Nachrichteneigenschaften
- Anwendungsdaten

Abbildung 2 auf Seite 19 stellt eine Nachricht dar und zeigt deren logische Aufteilung in Nachrichteneigenschaften und Anwendungsdaten.

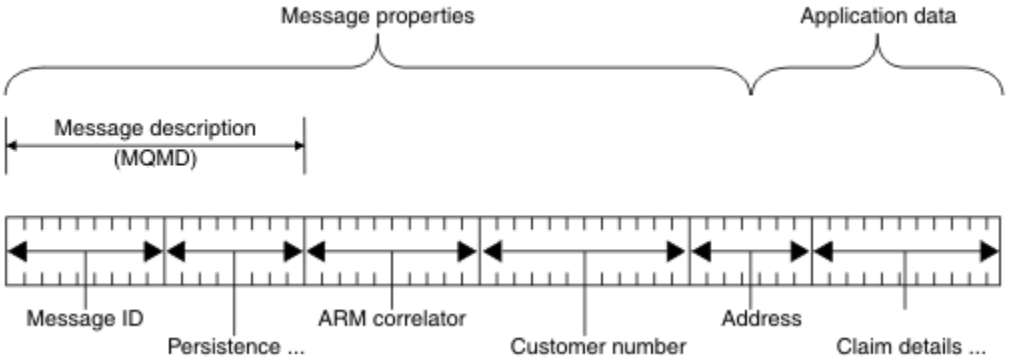


Abbildung 2. Darstellung einer Nachricht

Die Anwendungsdaten, die in einer IBM MQ-Nachricht übertragen werden, werden nur im Falle einer Konvertierung von einem Warteschlangenmanager geändert. Zudem schränkt IBM MQ den Inhalt dieser Daten nicht ein. Die Datenlänge der einzelnen Nachrichten darf die Werte des Attributs **MaxMsgLength** der Warteschlange und des Warteschlangenmanagers nicht überschreiten.

ALW Unter AIX, Linux, and Windows ist das Attribut *MaxMsgLength* des Warteschlangenmanagers und der Warteschlange standardmäßig auf 4 MB (4 194 304 Bytes) gesetzt. Falls erforderlich, können Sie den Wert in maximal 100 MB (104 857 600 Byte) ändern.

IBM i Unter IBM i ist das Attribut *MaxMsgLength* des Warteschlangenmanagers und der Warteschlange standardmäßig auf 4 MB (4 194 304 Bytes) gesetzt. Falls erforderlich, können Sie den Wert in maximal 100 MB (104 857 600 Byte) ändern. Informationen für den Fall, dass Sie IBM MQ-Nachrichten

mit mehr als 15 MB in IBM i verwenden möchten, finden Sie im Abschnitt „Prozedurale Anwendung auf IBM i erstellen” auf Seite 1059.

z/OS Unter z/OS ist der Wert des Attributs **MaxMsgLength** des Warteschlangenmanagers auf 100 MB festgelegt. Der Standardwert des Attributs **MaxMsgLength** der Warteschlange beträgt 4 MB (4.194.304 Byte). Bei Bedarf kann er auf maximal 100 MB erhöht werden.

Unter gewissen Umständen sollten Ihre Nachrichten etwas kürzer als **MaxMsgLength** sein. Weitere Informationen finden Sie unter „Die Daten der Nachricht” auf Seite 797.

Sie erstellen eine Nachricht, wenn Sie die MQI-Aufrufe MQPUT bzw. MQPUT1 verwenden. Als Eingabe für diese Aufrufe geben Sie die Steuerinformationen (also beispielsweise die Priorität der Nachricht und den Namen einer Antwortwarteschlange) und Ihre Daten an. Der Aufruf stellt die Nachricht dann in eine Warteschlange. Weitere Informationen zu diesen Aufrufen finden Sie unter MQPUT und MQPUT1.

Nachrichtendeskriptor

Mithilfe der MQMD-Struktur, die den *Nachrichtendeskriptor* definiert, können Sie auf Nachrichtensteuerungsinformationen zugreifen.

Eine umfassende Beschreibung der MQMD-Struktur finden Sie im Abschnitt MQMD - Nachrichtendeskriptor.

Eine Erläuterung zur Verwendung der MQMD-Felder mit Informationen zum Ursprung der Nachricht finden Sie im Abschnitt „Nachrichtenkontext” auf Seite 50.

Es gibt verschiedene Versionen des Nachrichtendeskriptors. Weitere Informationen zur Gruppierung und Segmentierung von Nachrichten (siehe „Nachrichtengruppen” auf Seite 46) erhalten Sie in Version 2 des Nachrichtendeskriptors (oder in MQMDE). Dieser entspricht dem Nachrichtendeskriptor Version 1, verfügt aber noch über zusätzliche Felder. Diese Felder werden unter MQMDE - Erweiterung des Nachrichtendeskriptors erläutert.

Nachrichtentypen

In IBM MQ sind vier Nachrichtentypen definiert.

Diese vier Nachrichten sind:

- Datagramme
- Anforderungsnachrichten
- Antwortnachrichten
- Berichtsnachrichten
  - Arten von Berichtsnachrichten
  - Berichtsnachrichtenoptionen

Mit den ersten drei Nachrichtentypen können sich Anwendungen gegenseitig Informationen übermitteln. Mithilfe des vierten Typs, der Berichtsnachricht, können Anwendungen und Warteschlangenmanager Informationen zu Ereignissen wie z. B. dem Auftreten eines Fehlers zurückmelden.

Jeder Nachrichtentyp ist mit einem MQMT_*-Wert gekennzeichnet. Sie können auch eigene Nachrichtentypen definieren. Informationen zum in Frage kommenden Wertebereich finden Sie im Abschnitt Nachrichtentypen.

Datagramme

Verwenden Sie ein *Datagramm*, wenn Sie von der Anwendung, welche die Nachricht empfängt (die Nachricht also aus der Warteschlange abruft), keine Antwort benötigen.

Beispielsweise könnte eine Anwendung, die Fluginformationen in einer Flughafen-Lounge anzeigt, Datagramme verwenden. In einer Nachricht könnten die Daten für eine gesamte Bildschirmanzeige mit Fluginformationen enthalten sein. Eine solche Anwendung wird höchstwahrscheinlich keine Nachrichten-

bestätigung anfordern, da es vermutlich keine Rolle spielt, wenn eine Nachricht nicht zugestellt wird. Nach kurzer Zeit sendet die Anwendung eine Aktualisierungsnachricht.

Anforderungsnachrichten

Verwenden Sie eine *Anforderungsnachricht*, wenn Sie von der Anwendung, welche die Nachricht empfängt, eine Antwort erhalten möchten.

Anforderungsnachrichten könnten beispielsweise von einer Anwendung verwendet werden, welche die Bilanz eines Girokontos anzeigt. In der Anforderungsnachricht könnte die Kontonummer angegeben sein, in der Antwortnachricht wäre der Kontostand enthalten.

Falls Sie die Antwortnachricht mit der Anforderungsnachricht verbinden möchten, stehen Ihnen hierfür zwei Möglichkeiten zur Verfügung:

- Sie können es so einrichten, dass die Anwendung, welche die Anforderungsnachricht bearbeitet, Informationen in die Antwortnachricht stellen muss, die sich auf die Anforderungsnachricht beziehen.
- Sie können im Nachrichtendeskriptor der Anforderungsnachricht im Berichtsfeld den Inhalt der Felder *MsgId* und *CorrelId* der Antwortnachricht angeben:
  - Sie können anfordern, dass entweder das Feld *MsgId* oder das Feld *CorrelId* der Originalnachricht in das Feld *CorrelId* der Antwortnachricht kopiert wird (standardmäßig wird das Feld *MsgId* kopiert).
  - Sie können anfordern, dass für die Antwortnachricht entweder ein neues Feld *MsgId* generiert wird oder dass das Feld *MsgId* der Originalnachricht in das Feld *MsgId* der Antwortnachricht kopiert wird (standardmäßig wird eine neue Nachrichten-ID generiert).

Antwortnachrichten

Verwenden Sie eine *Antwortnachricht* als Antwort auf eine andere Nachricht.

Beachten Sie beim Erstellen einer Antwortnachricht alle Optionen, die im Nachrichtendeskriptor der zu beantwortenden Nachricht definiert sind. Berichtsoptionen geben den Inhalt der Felder für die Nachrichten-ID (*MsgId*) und die Korrelations-ID (*CorrelId*) an. Anhand dieser Felder kann die Anwendung, welche die Antwort empfängt, die Antwort mit der ursprünglichen Anforderung korrelieren.

Berichtsnachrichten

Berichtsnachrichten informieren Anwendungen über Ereignisse wie z. B. das Auftreten eines Fehlers bei der Verarbeitung einer Nachricht.

Sie können von folgenden Komponenten generiert werden:

- Warteschlangenmanager
- Nachrichtenkanalagent (wenn die Nachricht beispielsweise nicht zugestellt werden kann) oder
- Anwendung (wenn diese beispielsweise die Daten in der Nachricht nicht verwenden kann).

Berichtsnachrichten können jederzeit geniert werden und in einer Warteschlange eintreffen, wenn die Anwendung dies gar nicht erwartet.

Arten von Berichtsnachrichten

Wenn Sie eine Nachricht in eine Warteschlange stellen, können Sie angeben, dass Sie folgende Nachrichten empfangen möchten:

- Eine *Ausnahmeberichtsnachricht*. Diese wird als Antwort auf eine Nachricht gesendet, für die das Ausnahme-Flag gesetzt ist. Sie wird vom Nachrichtenkanalagenten oder der Anwendung generiert.
- Eine *Ablaufberichtsnachricht*. Diese gibt an, dass eine Anwendung versucht hat, eine Nachricht abzurufen, die bereits den Ablaufschwellenwert erreicht hatte. Die Nachricht ist zum Löschen markiert. Diese Art von Bericht wird vom Warteschlangenmanager generiert.
- Eine *Berichtsnachricht mit einer Bestätigung bei Eingang*. Diese gibt an, dass die Nachricht ihre Zielwarteschlange erreicht hat. Sie wird vom Warteschlangenmanager generiert.

- Eine *Berichtsnachricht mit einer Empfangsbestätigung*. Diese gibt an, dass die Nachricht von einer empfangenden Anwendung abgerufen wurde. Sie wird vom Warteschlangenmanager generiert.
- Eine *Berichtsnachricht mit einer Benachrichtigung über eine positive Aktion*. Diese gibt an, dass eine Anforderung erfüllt (und somit die in der Nachricht angeforderte Aktion erfolgreich ausgeführt) wurde. Diese Art von Bericht wird von der Anwendung generiert.
- Eine *Berichtsnachricht mit einer Benachrichtigung über eine negative Aktion*. Diese gibt an, dass eine Anforderung nicht erfüllt (und somit die in der Nachricht angeforderte Aktion nicht erfolgreich ausgeführt) wurde. Diese Art von Bericht wird von der Anwendung generiert.

Anmerkung: In den einzelnen Arten von Berichtsnachrichten kann jeweils Folgendes enthalten sein:

- Die gesamte ursprüngliche Nachricht
- Die ersten 100 Datenbytes der ursprünglichen Nachricht
- Keine Daten aus der ursprünglichen Nachricht

Wenn Sie eine Nachricht in eine Warteschlange stellen, können Sie mehrere Arten von Berichtsnachrichten anfordern. Wenn Sie sich für die Berichtsnachricht mit dem Zustellnachweis und die Ausnahmeberichtsnachricht entscheiden, erhalten Sie eine Ausnahmeberichtsnachricht, wenn die Nachricht nicht zugestellt werden kann. Wenn Sie dagegen nur die Berichtsnachricht mit dem Zustellnachweis auswählen und die Nachricht nicht zugestellt werden kann, erhalten Sie keine Ausnahmeberichtsnachricht.

Sie erhalten nur die angeforderten Berichtsnachrichten, sofern die entsprechenden Kriterien für die Nachrichten erfüllt sind.

Berichtsnachrichtenoptionen

Nachdem eine Ausnahme aufgetreten ist, können Sie eine Nachricht *löschen*. Wenn Sie die Löschoption auswählen und eine Ausnahmeberichtsnachricht angefordert haben, wird die Berichtsnachricht an *ReplyToQ* und *ReplyToQMgr* übermittelt, während die ursprüngliche Nachricht gelöscht wird.

Anmerkung: Ein Vorteil dieser Option besteht darin, dass die Anzahl der Nachrichten, die in die Warteschlange für nicht zustellbare Nachrichten gelangen, reduziert werden kann. Allerdings muss Ihre Anwendung, sofern sie nicht nur Datagrammnachrichten sendet, zurückgegebene Nachrichten bearbeiten. Wenn eine Ausnahmeberichtsnachricht generiert wird, übernimmt sie die Persistenz der ursprünglichen Nachricht.

Kann eine Berichtsnachricht nicht zugestellt werden (weil beispielsweise die Warteschlange voll ist), wird sie in die Warteschlange für nicht zustellbare Nachrichten gestellt.

Wenn Sie eine Berichtsnachricht empfangen möchten, geben Sie den Namen der Empfangswarteschlange für Antworten im Feld *ReplyToQ* an. Andernfalls schlägt MQPUT oder MQPUT1 der Originalnachricht mit dem Fehler MQRC_MISSING_REPLY_TO_Q fehl.

Im Nachrichtendeskriptor (MQMD) einer Nachricht können Sie mithilfe anderer Berichtsoptionen angeben, welchen Inhalt die Felder *MsgId* und *CorrelId* in jeglichen Berichtsnachrichten haben sollen, die für diese Nachricht erstellt werden:

- Sie können anfordern, dass entweder das Feld *MsgId* oder das Feld *CorrelId* der Originalnachricht in das Feld *CorrelId* der Berichtsnachricht kopiert werden soll. Standardmäßig wird die Nachrichten-ID kopiert. Verwenden Sie MQRO_COPY_MSG_ID_TO_CORRELID, da auf diese Weise der Absender einer Nachricht die Antwort oder Berichtsnachricht mit der ursprünglichen Nachricht korrelieren kann. Die Korrelations-ID der Antwort oder Berichtsnachricht ist mit der Nachrichten-ID der ursprünglichen Nachricht identisch.
- Sie können anfordern, dass für die Berichtsnachricht entweder ein neues Feld *MsgId* generiert wird oder dass das Feld *MsgId* der Originalnachricht in das Feld *MsgId* der Berichtsnachricht kopiert wird. Standardmäßig wird eine neue Nachrichten-ID generiert. Verwenden Sie MQRO_NEW_MSG_ID, da auf diese Weise sichergestellt ist, dass jede Nachricht im System über eine andere Nachrichten-ID verfügt und eindeutig von allen anderen Nachrichten im System zu unterscheiden ist.

- Fachanwendungen müssen unter Umständen MQRO_PASS_MSG_ID oder MQRO_PASS_CORREL_ID verwenden. Die Anwendung, welche die Nachrichten aus der Warteschlange liest, müssen Sie jedoch selbst entwickeln, um sicherzugehen, dass sie korrekt funktioniert, wenn die Warteschlange beispielsweise mehrere Nachrichten mit derselben Nachrichten-ID enthält.

  Serveranwendungen müssen die Einstellungen dieser Flags in der Anforderungsnachricht prüfen und die Felder *MsgId* und *CorrelId* in der Antwort oder der Berichtsnachricht entsprechend festlegen.

  Als Intermediäre zwischen einer anfordernden Anwendung und einer Serveranwendung fungierende Anwendungen müssen die Einstellungen dieser Flags nicht prüfen. Dies liegt daran, dass diese Anwendungen die Nachricht in der Regel mit unveränderten Feldern *MsgId*, *CorrelId* und *Report* an die Serveranwendung weiterleiten müssen. Somit kann die Serveranwendung das Feld *MsgId* aus der ursprünglichen Nachricht in das Feld *CorrelId* der Antwortnachricht kopieren.

Beim Generieren eines Berichts zu einer Nachricht müssen Serveranwendungen prüfen, ob diese Optionen gesetzt wurden.

Weitere Informationen zur Verwendung von Berichtsnachrichten finden Sie im Abschnitt Bericht.

Mit einer Reihe von Rückkopplungscodes geben Warteschlangenmanager die Berichtsgattung an. Sie fügen diese Codes in das Feld *Feedback* des Nachrichtendeskriptors einer Berichtsnachricht ein. Zudem können Warteschlangenmanager auch MQI-Ursachencodes im Feld *Feedback* zurückgeben. In IBM MQ ist ein Bereich möglicher Rückkopplungscodes für Anwendungen definiert.

Weitere Informationen zu Rückkopplungs- und Ursachencodes finden Sie im Abschnitt Feedback.

Rückkopplungscodes könnten beispielsweise von einem Programm verwendet werden, welches die Auslastung anderer Programme überwacht, die für eine Warteschlange zuständig sind. Ist mehr als eine Instanz eines Programms für eine Warteschlange zuständig und ist dies durch die Anzahl in der Warteschlange eintreffender Nachrichten nicht mehr gerechtfertigt, kann ein solches Programm eine Berichtsnachricht (mit dem Rückkopplungscode MQFB_QUIT) an eines der zuständigen Programme senden, um diesem mitzuteilen, dass es seine Aktivität beenden soll. (Ein Überwachungsprogramm könnte mit dem Aufruf MQINQ herausfinden, wie viele Programme für eine Warteschlange zuständig sind.)

Multi *Berichte und segmentierte Nachrichten*

Wenn Sie für eine segmentierte Nachricht die Generierung von Berichten anfordern, erhalten Sie unter Umständen mehr Berichte als bei einer nicht segmentierten Nachricht. Berichte zu segmentierten Nachrichten sind nur auf Multiplatforms verfügbar.

Segmentierte Nachrichten werden im Abschnitt „Nachrichtensegmentierung” auf Seite 833 beschrieben.

Für Berichte, die von IBM MQ generiert werden

Wenn Sie Ihre Nachrichten segmentieren oder dem Warteschlangenmanager die Nachrichtensegmentierung gestatten, können Sie nur dann den Empfang eines Einzelberichts für die gesamte Nachricht erwarten, wenn Sie ausschließlich Berichte mit Bestätigung bei Zustellung (COD-Berichte) angefordert und MQGMO_COMPLETE_MSG in der abrufenden Anwendung angegeben haben.

Andernfalls muss Ihre Anwendung für die Handhabung verschiedener Berichte vorbereitet sein, i.d.R. wird pro Segment ein Bericht generiert.

Anmerkung: Wenn Sie Ihre Nachrichten segmentieren und nur die ersten 100 Bytes der ursprünglichen Nachrichtendaten zurückgegeben werden sollen, ändern Sie die Einstellung der Berichtsoptionen, um Berichte ohne Daten für Segmente mit einem Offset von mindestens 100 anzufordern. Wenn Sie die Einstellung jedoch unverändert lassen, sodass jedes Segment 100 Datenbytes anfordert, und die Berichtsnachrichten mit einem einzelnen MQGET abrufen, der MQGMO_COMPLETE_MSG angibt, werden die Berichte zu einer komplexen Nachricht mit 100 Bytes an gelesenen Daten an jedem entsprechenden Offset zusammengestellt. In diesem Fall benötigen Sie entweder einen großen Puffer oder müssen MQGMO_ACCEPT_TRUNCATED_MSG angeben.

Für von Anwendungen generierte Berichte

Wenn Ihre Anwendung Berichte generiert, kopieren Sie stets die IBM MQ-Header am Anfang der Daten der Originalnachricht in die Daten der Berichtsnachricht.

Fügen Sie anschließend keine, 100 Bytes oder alle Daten der Originalnachricht (bzw. die Anzahl, die Sie normalerweise einschließen würden) den Daten der Berichtsnachricht hinzu.

Die zu kopierenden IBM MQ-Header erkennen sie, indem Sie sich die aufeinanderfolgenden Formatnamen ansehen. Beginnen Sie bei MQMD und arbeiten Sie sich durch alle vorhandenen Header. Die folgenden `Format`-Namen zeigen diese IBM MQ-Header an:

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH* steht für einen beliebigen Namen, der mit den Zeichen MQH beginnt.

Der `Format`-Name tritt an bestimmten Stellen für MQDLH und MQXQH auf. Für die anderen IBM MQ-Header tritt er hingegen an derselben Stelle auf. Die Header-Länge ist in einem Feld enthalten, das für MQMDE-, MQIMS- und alle MQH*-Header ebenfalls an derselben Stelle auftritt.

Wenn Sie einen MQMD der Version 1 verwenden und einen Bericht zu einem Segment, einer Nachricht in einer Gruppe oder einer Nachricht mit zulässiger Segmentierung erstellen, müssen die Berichtsdaten mit einem MQMDE beginnen. Setzen Sie das Feld *`OriginalLength`* auf die Länge der Daten der Originalnachricht und schließen Sie die Längen aller bestehenden IBM MQ-Header aus.

Berichte abrufen

Wenn Sie COA- oder COD-Berichte anfordern, können Sie diese mit MQGMO_COMPLETE_MSG erneut assemblieren lassen.

Ein MQGET-Aufruf mit MQGMO_COMPLETE_MSG ist erfüllt, wenn genügend Berichtsnachrichten (eines einzelnen Typs, z. B. COA, und mit derselben *`GroupId`*) in der Warteschlange vorhanden sind und diese eine vollständige ursprüngliche Nachricht darstellen. Dies trifft auch dann zu, wenn die Berichtsnachrichten nicht die vollständigen Originaldaten enthalten. In jeder Berichtsnachricht gibt das Feld *`Original-Length`* die Länge der Originaldaten an, die von der Berichtsnachricht dargestellt werden, selbst wenn die Daten nicht vorhanden sind.

Auf diese Weise können Sie auch vorgehen, wenn die Warteschlange verschiedene Berichtstypen enthält (z. B. COA und COD), da ein MQGET mit MQGMO_COMPLETE_MSG Berichtsnachrichten nur dann neu erstellt, wenn diese denselben *`Feedback`*-Code aufweisen. Bei Ausnahmeberichten ist dieses Vorgehen allerdings nicht möglich, da diese für gewöhnlich unterschiedliche *`Feedback`*-Codes haben.

Verwenden Sie dieses Verfahren, wenn Sie eine positive Meldung abrufen möchten, dass die gesamte Nachricht angekommen ist. In den meisten Fällen müssen Sie jedoch die Möglichkeit berücksichtigen, dass manche Segmente ankommen, während andere unter Umständen eine Ausnahme generieren (oder einen Ablauf, falls Sie dies zugelassen haben). In diesem Fall können Sie MQGMO_COMPLETE_MSG nicht verwenden, da Sie vermutlich verschiedene *`Feedback`*-Codes für unterschiedliche Segmente und auch mehr als einen Bericht für ein Segment erhalten. Stattdessen kann jedoch MQGMO_ALL_SEGMENTS_AVAILABLE verwendet werden.

Dazu müssen Sie unter Umständen Berichte bei deren Ankunft abrufen und in Ihrer Anwendung ein Bild davon erstellen, was mit der Originalnachricht geschehen ist. Mit dem Feld *`GroupId`* in der Berichtsnachricht können Sie Berichte mit dem Feld *`GroupId`* der Originalnachricht korrelieren und mit dem Feld *`Feedback`* geben Sie den Typ der jeweiligen Berichtsnachrichten an. Die Art und Weise, in der Sie dies ausführen, hängt von Ihren Anwendungsanforderungen ab.

Ein Ansatz sieht wie folgt aus:

- Fordern Sie COD-Berichte und Ausnahmeberichte an.

- Überprüfen Sie nach einer bestimmten Zeit mittels MQGMO_COMPLETE_MSG, ob eine vollständige Reihe von COD-Berichten empfangen wurde. Ist dies der Fall, weiß Ihre Anwendung, dass die gesamte Nachricht verarbeitet wurde.
- Wurde keine vollständige Reihe empfangen und sind Ausnahmeberichte zu dieser Nachricht vorhanden, beheben Sie das Problem wie im Falle nicht segmentierter Nachrichten. Stellen Sie allerdings sicher, dass Sie verwaiste Segmente dabei löschen.
- Wenn für manche Segmente keine Berichte vorliegen, warten die ursprünglichen Segmente (oder die Berichte) unter Umständen auf die Wiederherstellung einer Kanalverbindung oder das Netz ist eventuell momentan überlastet. Falls überhaupt keine Ausnahmeberichte empfangen wurden (oder Sie der Meinung sind, dass es sich bei den Ihnen vorliegenden Berichten unter Umständen nur um temporäre Berichte handelt), können Sie die Wartezeit Ihrer Anwendung ein wenig verlängern.

  Wie zuvor ist auch dies ähnlich wie bei nicht segmentierten Nachrichten. Allerdings müssen Sie hier die Möglichkeit berücksichtigen, verwaiste Segmente zu löschen.

Handelt es sich bei der Originalnachricht nicht um eine kritische Nachricht (z. B. eine Abfrage oder Nachricht, die später wiederholt werden kann), legen Sie die Ablaufzeit fest, um sicherzustellen, dass verwaiste Segmente entfernt werden.

Warteschlangenmanager einer früheren Version

Wenn ein Warteschlangenmanager, der die Segmentierung unterstützt, einen Bericht generiert, der jedoch auf einem Warteschlangenmanager empfangen wird, der keine Segmentierung unterstützt, wird die MQMDE-Struktur (gibt die durch den Bericht dargestellten Felder *Offset* und *OriginalLength* an) zusätzlich zu keinen, 100 Byte oder allen ursprünglichen Nachrichtendaten immer in die Berichtsdaten einbezogen.

Wenn allerdings ein Segment einer Nachricht einen Warteschlangenmanager durchläuft, der keine Segmentierung unterstützt, und dort ein Bericht generiert wird, wird die MQMDE-Struktur in der Originalnachricht wie reine Daten behandelt. Sie wird also nicht in die Berichtsdaten einbezogen, wenn null Bytes der ursprünglichen Daten angefordert wurden. Ohne den MQMDE ist die Berichtsnachricht unter Umständen nicht hilfreich.

Fordern Sie mindestens 100 Bytes an Daten in Berichten an, wenn die Nachricht möglicherweise über einen Warteschlangenmanager einer früheren Version übergeben wird.

Format von Nachrichtensteuerungsinformationen und Nachrichtendaten

Der Warteschlangenmanager ist nur am Format der Steuerungsinformationen innerhalb einer Nachricht interessiert. Die Anwendungen, die die Nachricht bearbeiten, sind hingegen sowohl am Format der Steuerungsinformationen als auch der Nachrichtendaten interessiert sind.

Format von Nachrichtensteuerungsinformationen

Steuerungsinformationen in den Zeichenfolgefeldern des Nachrichtendeskriptors müssen in dem vom Warteschlangenmanager verwendeten Zeichensatz formatiert sein.

Dieser Zeichensatz wird vom Attribut **CodedCharSetId** des Warteschlangenmanager-Objekts definiert. Wenn Anwendungen Nachrichten von einem Warteschlangenmanager an einen anderen übergeben, legen die Nachrichtenkanalagenten, die die Nachrichten übertragen, anhand dieses Attributs fest, welche Datenkonvertierung ausgeführt wird. Daher müssen Steuerungsinformationen diesen Zeichensatz verwenden.

Format von Nachrichtendaten

Sie können Folgendes angeben:

- Das Format der Anwendungsdaten
- Den Zeichensatz der Zeichendaten
- Das Format der numerischen Daten

Verwenden Sie dazu diese Felder:

Format
Zeigt dem Empfänger einer Nachricht das Format der Anwendungsdaten in der Nachricht an.

Wenn der Warteschlangenmanager eine Nachricht erstellt, gibt er in manchen Fällen mit dem Feld *Format* das Format dieser Nachricht an. Kann ein Warteschlangenmanager beispielsweise eine Nachricht nicht zustellen, reiht er die Nachricht in eine Warteschlange für nicht zustellbare Nachrichten ein. Er fügt der Nachricht einen Header (mit weiteren Steuerinformationen) hinzu und ändert das Feld *Format* entsprechend, damit diese Informationen angezeigt werden.

Der Warteschlangenmanager verfügt über mehrere *integrierte Formate*, deren Namen mit MQ beginnen, beispielsweise MQFMT_STRING. Sollten diese für Ihre Anforderungen nicht ausreichen, können Sie eigene Formate (*benutzerdefinierte Formate*) definieren, dürfen für diese jedoch keine Namen verwenden, die mit MQ beginnen.

Wenn Sie eigene Formate erstellen und verwenden, müssen Sie ein Datenkonvertierungsexit schreiben, um ein Programm zu unterstützen, das die Nachrichten mithilfe von MQGMO_CONVERT abruft.

CodedCharSetId
Definiert den Zeichensatz der Zeichendaten in der Nachricht. Wenn Sie diesen Zeichensatz auf den des Warteschlangenmanagers setzen möchten, können Sie dieses Feld auf die Konstante MQCCSI_Q_MGR oder MQCCSI_INHERIT setzen.

Wenn Sie eine Nachricht von einer Warteschlange abrufen, vergleichen Sie den Wert des Felds *CodedCharSetId* mit dem Wert, den Ihre Anwendung erwartet. Sind die beiden Werte nicht identisch, müssen Sie unter Umständen alle Zeichendaten in der Nachricht konvertieren oder ein Datenkonvertierungs-Nachrichtenexit verwenden, falls verfügbar.

Encoding
Beschreibt das Format von numerischen Nachrichtendaten, die binäre Ganzzahlen, gepackt-dezimale Ganzzahlen und Gleitkommazahlen enthalten. Es ist i.d.R. entsprechend der Maschine codiert, auf der der Warteschlangenmanager ausgeführt wird.

Wenn Sie eine Nachricht in eine Warteschlange einreihen, geben Sie im Feld *Encoding* für gewöhnlich die Konstante MQENC_NATIVE an. Das bedeutet, dass die Codierung Ihrer Nachrichtendaten der Codierung der Maschine entspricht, auf der Ihre Anwendung ausgeführt wird.

Wenn Sie eine Nachricht von einer Warteschlange abrufen, vergleichen Sie den Wert des Felds *Encoding* im Nachrichtendeskriptor mit dem Wert der Konstanten MQENC_NATIVE auf Ihrer Maschine. Sind die beiden Werte nicht identisch, müssen Sie unter Umständen alle numerischen Daten in der Nachricht konvertieren oder ein Datenkonvertierungs-Nachrichtenexit verwenden, falls verfügbar.

Anwendungsdatenkonvertierung
Wenn mit mehreren Plattformen gearbeitet wird, müssen Anwendungsdaten unter Umständen in den Zeichensatz und die Codierung konvertiert werden, die für eine andere Anwendung erforderlich sind.

Die Konvertierung kann am sendenden oder am empfangenden Warteschlangenmanager stattfinden. Wenn die Bibliothek integrierter Formate nicht Ihren Anforderungen entspricht, können Sie eine eigene Bibliothek definieren. Der Konvertierungstyp hängt von dem Nachrichtenformat ab, das im Formatfeld des Nachrichtendeskriptors MQMD angegeben wird.

Anmerkung: Nachrichten mit angegebenem MQFMT_NONE werden nicht konvertiert.

Konvertierung am sendenden Warteschlangenmanager

Setzen Sie das Kanalattribut CONVERT auf YES, wenn der sendende Nachrichtenkanalagent (MCA) die Anwendungsdaten konvertieren soll.

Die Konvertierung wird am sendenden Warteschlangenmanager für bestimmte integrierte Formate und benutzerdefinierte Formate ausgeführt, wenn ein geeigneter Benutzerexit bereit steht.

Integrierte Formate

Hierzu gehören folgende Aufrufe:

- Reine Zeichennachrichten (Formatname MQFMT_STRING)
- IBM MQ-definierte Nachrichten, z. B. Programmable Command Formats

  IBM MQ verwendet Programmable Command Format-Nachrichten für Verwaltungsnachrichten und -ereignisse (in diesem Fall wird der Formatname MQFMT_ADMIN verwendet). Sie können das gleiche Format (mit dem Formatnamen MQFMT_PCF) für Ihre eigenen Nachrichten verwenden und die Vorteile der integrierten Datenkonvertierung nutzen.

Die Namen der integrierten Formate des Warteschlangenmanager beginnen immer mit MQFMT. Eine Liste und Beschreibung finden Sie unter Format.

Anwendungsdefinierte Formate

Bei benutzerdefinierten Formate müssen die Anwendungsdaten von einem Datenkonvertierungsexitprogramm konvertiert werden. Weitere Informationen hierzu finden Sie unter „Datenkonvertierungsexits schreiben" auf Seite 1035). In einer Client/Server-Umgebung wird der Exit auf den Server geladen. Dort wird auch die Konvertierung ausgeführt.

Konvertierung am empfangenden Warteschlangenmanager

Anwendungsnachrichtendaten können vom empfangenden Warteschlangenmanager sowohl für integrierte als auch für benutzerdefinierte Formate konvertiert werden.

Die Konvertierung wird während der Verarbeitung eines MQGET-Aufrufs ausgeführt, wenn Sie die Option MQGMO_CONVERT angeben. Ausführliche Informationen finden Sie im Abschnitt Optionen

Codierte Zeichensätze

IBM MQ-Produkte unterstützen die codierten Zeichensätze, die vom jeweiligen Betriebssystem bereitgestellt werden.

Wenn Sie einen Warteschlangenmanager erstellen, basiert die verwendete ID des codierten Zeichensatzes des Warteschlangenmanagers (CCSID) auf dieser zu Grunde liegenden Umgebung. Handelt es sich dabei um eine Seite mit gemischtem Code, verwendet IBM MQ den SBCS-Teil der Seite als Warteschlangenmanager-CCSID.

Wenn das zu Grunde liegende Betriebssystem Seiten mit DBCS-Code unterstützt, kann IBM MQ sie bei der allgemeinen Datenkonvertierung verwenden.

Ausführliche Informationen zu den unterstützten codierten Zeichensätzen finden Sie in der Dokumentation Ihres Betriebssystems.

Wenn Sie Anwendungen für mehrere Plattformen schreiben, müssen Sie die Konvertierung der Anwendungsdaten, die Formatnamen sowie Benutzerexits berücksichtigen. Informationen zum Aufrufen und Schreiben von Datenkonvertierungsexits finden Sie unter „Datenkonvertierungsexits schreiben" auf Seite 1035.

Nachrichtenprioritäten

Sie können die Priorität der Nachricht auf einen numerischen Wert setzen oder die Standardpriorität der Warteschlange übernehmen.

Die Priorität einer Nachricht legen Sie beim Einreihen der Nachricht in eine Warteschlange im Feld *Priority* der MQMD-Struktur fest. Sie können entweder einen numerischen Wert für die Priorität festlegen oder die Nachricht die Standardpriorität der Warteschlange verwenden lassen.

Das Attribut **MsgDeliverySequence** der Warteschlange legt fest, ob Nachrichten in der Warteschlange in FIFO-Reihenfolge (First In/First Out) oder nach ihrer Priorität in FIFO-Reihenfolge gespeichert werden. Wenn dieses Attribut auf MQMDS_PRIORITY gesetzt ist, werden Nachrichten mit der im Nachrichtendeskriptor im Feld *Priority* angegebenen Priorität eingereiht. Ist das Attribut dagegen auf MQMDS_FIFO gesetzt, werden Nachrichten mit der Standardpriorität der Warteschlange eingereiht. Nachrichten von gleicher Priorität werden in der Warteschlange in der Reihenfolge ihrer Ankunft gespeichert.

Das Attribut **DefPriority** einer Warteschlange legt die Standardpriorität der in diese Warteschlange eingereihten Nachrichten fest. Dieser Wert wird bei Erstellung der Warteschlange festgelegt, kann jedoch anschließend geändert werden. Aliaswarteschlangen und lokale Definitionen von fernen Warteschlangen können unterschiedliche Standardprioritäten der Basiswarteschlangen haben, auf die sie zurückgreifen. Wenn der Auflösungspfad mehrere Warteschlangendefinitionen enthält (siehe „Namensauflösung” auf Seite 783), wird zum Zeitpunkt der PUT-Operation die Standardpriorität des Attributs **DefPriority** der im Befehl 'Open' angegebenen Warteschlange übernommen.

Der Wert des Attributs **MaxPriority** des Warteschlangenmanagers gibt die maximale Priorität an, die Sie einer von diesem Warteschlangenmanager verarbeiteten Nachricht zuweisen können. Der Wert dieses Attributs kann nicht verändert werden. In IBM MQ hat das Attribut den Wert 9. Sie können Nachrichten mit Prioritären zwischen 0 (niedrigster Wert) und 9 (höchster Wert) erstellen.

Nachrichteneigenschaften

Mittels Nachrichteneigenschaften ermöglichen Sie es einer Anwendung, die zu verarbeitenden Nachrichten auszuwählen oder Informationen zu einer Nachricht abzurufen, ohne auf den MQMD- oder den MQRFH2-Header zugreifen zu müssen. Sie erleichtern auch die Kommunikation zwischen IBM MQ- und JMS-Anwendungen.

Eine Nachrichteneigenschaft besteht aus Daten, die einer Nachricht zugeordnet sind (alphanumerischer Name und Wert eines bestimmten Typs). Nachrichteneigenschaften werden von Nachrichtenselektoren zum Filtern von Veröffentlichungen nach Themen oder zur Auswahl bestimmter Nachrichten zum Abrufen aus Warteschlangen verwendet. Über Nachrichteneigenschaften können Geschäftsdaten oder Statusinformationen eingeschlossen werden, ohne sie in den Anwendungsdaten speichern zu müssen. Anwendungen müssen nicht auf Daten im MQ-Nachrichtendeskriptor (MQMD) oder auf MQRFH2-Header zugreifen, da Felder in diesen Datenstrukturen als Nachrichteneigenschaften mittels Message-Queue Interface-(MQI)-Funktionsaufrufen aufgerufen werden können.

Die Verwendung von Nachrichteneigenschaften in IBM MQ imitiert die Verwendung von Eigenschaften in JMS. Sie können also Eigenschaften in einer JMS-Anwendung festlegen und sie in einer prozeduralen IBM MQ-Anwendung abrufen und umgekehrt. Sie stellen einer JMS-Anwendung eine Eigenschaft zur Verfügung, indem Sie der Eigenschaft das Präfix "usr" zuweisen. Diese ist dann als JMS-Nachrichtenbenutzereigenschaft (ohne das Präfix) verfügbar. Beispiel: Die IBM MQ -Eigenschaft *usr.myproperty* (eine Zeichenfolge) ist für eine JMS -Anwendung über den JMS -Aufruf `message.getStringProperty('myproperty')` zugänglich. Beachten Sie, dass JMS-Anwendungen keine Eigenschaften mit dem Präfix "usr" aufrufen können, wenn sie mindestens zwei U+002E-Zeichen (".") enthalten. Eine Eigenschaft ohne Präfix und ohne U+002E-Zeichen (".") wird so behandelt, als enthielte sie das Präfix "usr". Umgekehrt kann auf eine Benutzereigenschaftengruppe in einer JMS -Anwendung in einer IBM MQ -Anwendung zugegriffen werden, indem "usr" hinzugefügt wird. dem in einem MQINQMP-Aufruf abgefragten Eigenschaftsnamen das Präfix "usr." hinzufügen.

Nachrichteneigenschaften und Nachrichtenlänge

Mit dem Attribut *MaxPropertiesLength* des Warteschlangenmanagers steuern Sie die Größe der Eigenschaften, die mit jeder Nachricht in einem IBM MQ-Warteschlangenmanager übertragen werden.

Wenn Sie Eigenschaften mithilfe von MQSETMP festlegen, entspricht die Größe der Eigenschaft der Länge des Eigenschaftsnamens in Bytes zuzüglich der Länge des Eigenschaftswerts in Bytes, die an den MQSETMP-Aufruf übermittelt werden. Der Zeichensatz des Eigenschaftsnamens und der Eigenschaftswert können sich während der Übertragung der Nachricht an ihren Empfängerändern, da sie in Unicode konvertiert werden können. In diesem Fall ändert sich unter Umständen die Größe der Eigenschaft.

Bei einem MQPUT- bzw. MQPUT1-Aufruf zählen die Eigenschaften der Nachricht nicht zur Länge der Nachricht für die Warteschlange und den Warteschlangenmanager hinzu. Sie sind jedoch zur Länge der vom Warteschlangenmanager empfangenen Eigenschaften anzurechnen (unabhängig davon, ob sie mit den MQI-Aufrufen der Nachrichteneigenschaft festgelegt wurden oder nicht).

Wenn die Größe der Eigenschaften die maximale Eigenschaftenlänge überschreitet, wird die Nachricht mit MQRC_PROPERTIES_TOO_BIG zurückgewiesen. Da sich die Größe der Eigenschaften nach ihrer Darstellung richtet, sollten Sie die maximale Eigenschaftenlänge mit einem möglichst hohen Wert angeben.

Eine Anwendung kann eine Nachricht erfolgreich mit einem Puffer einreihen, der größer ist als der Wert von *MaxMsgLength*, wenn die Eigenschaften im Puffer eingeschlossen sind. Nachrichteneigenschaften werden nämlich auch dann nicht zur Länge der Nachricht angerechnet, wenn sie als MQRFH2-Elemente dargestellt werden. Die Felder des MQRFH2-Headers werden nur dann der Eigenschaftenlänge angerechnet, wenn mindestens ein Ordner enthalten ist und jeder Ordner im Header wiederum Eigenschaften enthält. Wenn keiner der im MQRFH2-Header enthaltenen Ordner Eigenschaften enthält, werden die Felder des MQRFH2-Headers stattdessen der Nachrichtenlänge angerechnet.

Bei einem MQGET-Aufruf zählen die Eigenschaften der Nachricht nicht zur Länge der Nachricht für die Warteschlange und den Warteschlangenmanager hinzu. Da die Eigenschaften jedoch separat gezählt werden, kann der Puffer möglicherweise von einem MQGET-Aufruf zurückgegeben werden, der größer ist als der Wert des Attributs *MaxMsgLength*.

Vor dem MQGET-Aufruf sollten Ihre Anwendungen nicht den Wert von *MaxMsgLength* abfragen und Sie sollten daraufhin keinen Puffer dieser Größe zuweisen, sondern einen Puffer mit einer Größe, die Sie für ausreichend groß halten. Schlägt der MQGET-Aufruf fehl, weisen Sie einen Puffer zu, der sich an der Größe des Parameters *DataLength* orientiert.

Der Parameter *DataLength* des MQGET-Aufrufs gibt die Länge der Anwendungsdaten in Bytes sowie alle Eigenschaften zurück, die in dem von Ihnen bereitgestellten Puffer zurückgegeben wurden, wenn in der MQGMO-Struktur kein Nachrichtenhandle angegeben wurde.

Der Parameter *Buffer* des MQPUT-Aufrufs enthält die zu sendenden Anwendungsnachrichtendaten sowie alle Eigenschaften aus den Nachrichtendaten.

Die Länge der Nachrichteneigenschaften ist auf 100 MB begrenzt, ausgenommen der Nachrichtendeskriptor bzw. die Erweiterung für jede Nachricht.

Die Größe der Eigenschaft in ihrer internen Darstellung entspricht der Länge des Namens plus der Größe ihres Werts plus einiger Steuerdaten für die Eigenschaft. Zudem sind einige Steuerdaten für die Eigenschaften vorhanden, nachdem der Nachricht eine Eigenschaft hinzugefügt wurde.

Eigenschaftsnamen

Ein Eigenschaftsname ist eine Zeichenfolge. Hinsichtlich ihrer Länge und der zulässigen Zeichen gelten gewisse Einschränkungen.

Bei einem Eigenschaftsnamen handelt es sich um eine Zeichenfolge, bei der die Groß-/Kleinschreibung beachtet werden muss. Sofern keine sonstigen Einschränkungen durch den Kontext bestehen, ist sie auf +4095 Zeichen beschränkt. Dieser Grenzwert ist in der Konstante MQ_MAX_PROPERTY_NAME_LENGTH enthalten.

Wenn Sie diese maximale Länge bei der Verwendung eines MQI-Nachrichteneigenschaftenaufrufs überschreiten, schlägt der Aufruf mit dem Ursachencode MQRC_PROPERTY_NAME_LENGTH_ERR fehl.

Da es in JMS keine maximale Eigenschaftsnamenlänge gibt, kann eine JMS-Anwendung einen gültigen JMS-Eigenschaftsnamen festlegen, der beim Speichern in einer MQRFH2-Struktur kein gültiger IBM MQ-Eigenschaftsname ist.

In diesem Fall werden bei einer Syntaxanalyse nur die ersten 4095 Zeichen des Eigenschaftsnamens verwendet. Alle nachfolgenden Zeichen werden abgeschnitten. Dies kann dazu führen, dass in einer Anwendung, die Selektoren verwendet, eine Auswahlzeichenfolge nicht als Übereinstimmung gefunden wird. Möglicherweise stimmen aufgrund der Tatsache, dass mehrere Eigenschaften auf denselben Namen abgeschnitten werden, auch umgekehrt Zeichenfolgen überein, bei denen dies nicht der Fall sein sollte. Wird ein Eigenschaftsname abgeschnitten, gibt WebSphereMQ eine Fehlerprotokollnachricht aus.

Für alle Eigenschaftsnamen gelten die durch die Java-Sprachspezifikation für Java-IDs festgelegten Regeln. Ausnahme: Das Unicode-Zeichen U+002E (.) ist als Bestandteil des Namens erlaubt, darf jedoch nicht am Anfang stehen. Die Regeln für Java-IDs entsprechen denen, die in der JMS-Spezifikation für Eigenschaftsnamen enthalten sind.

Leerzeichen und Vergleichsoperatoren sind nicht erlaubt. Eingebettete Nullen sind in einem Eigenschaftsnamen zwar zulässig, werden jedoch nicht empfohlen. Wenn Sie eingebettete Nullen verwenden, kann die

Konstante MQVS_NULL_TERMINATED in Verbindung mit der MQCHARV-Struktur nicht für die Angabe von Zeichenfolgen variabler Länge verwendet werden.

Verwenden Sie möglichst einfache Eigenschaftsnamen, da Anwendungen Nachrichten auf Basis der Eigenschaftsnamen auswählen können und die Konvertierung zwischen dem Zeichensatz des Namens und demjenigen des Selektors zu einem unerwarteten Fehlschlagen der Auswahl führen könnte.

IBM MQ-Eigenschaftsnamen verwenden das Zeichen U+002E (.) zur logischen Gruppierung von Eigenschaften. Dadurch wird der Namensbereich für Eigenschaften unterteilt. Eigenschaften mit den folgenden Präfixwerten sind in jeder beliebigen Kombination aus Groß- und Kleinschreibung für die Verwendung durch das Produkt reserviert:

- `mcd`
- `jms`
- `usr`
- `mq`
- `sib`
- `wmq`
- `Root`
- `Body`
- `Properties`

Wenn Sie darauf achten, dass die Namen der Nachrichteneigenschaften aller Anwendungen den zugehörigen Internetdomänennamen als Präfix enthalten, lassen sich Namenskollisionen am einfachsten vermeiden. Wenn Sie zum Beispiel eine Anwendung mit dem Domänennamen `ourcompany.com` entwickeln, könnten Sie alle Eigenschaften mit dem Präfix `com.ourcompany` benennen. Diese Namenskonvention erleichtert auch die Auswahl von Eigenschaften, so kann zum Beispiel eine Anwendung alle Nachrichteneigenschaften abfragen, die mit `com.ourcompany.%` beginnen.

Im Abschnitt Einschränkungen bei Eigenschaftsnamen finden Sie weitere Informationen zur Verwendung von Eigenschaftsnamen.

Einschränkungen bei Eigenschaftsnamen
Bei der Benennung einer Eigenschaft müssen Sie bestimmte Regeln beachten.

Für Eigenschaftsnamen gelten die folgenden Einschränkungen:

1. Eine Eigenschaft darf nicht mit den folgenden Zeichenfolgen beginnen:
   - "JMS" - reserviert zur Verwendung durch IBM MQ classes for JMS.
   - "usr.JMS" - nicht gültig.

   Ausgenommen sind folgende Eigenschaften, die Synonyme für JMS-Eigenschaften sind:

Eigenschaft	**Synonym für**
JMSCorrelationID	Root .MQMD.CorrelId oder jms.Cid
JMSDeliveryMode	Root .MQMD.Persistence oder jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root .MQMD.Expiry oder jms.Exp
JMSMessageID	Root .MQMD.MsgId
JMSPriority	Root .MQMD.Priority oder jms.Pri
JMSRedelivered	Root .MQMD.BackoutCount
JMSReplyTo (eine als URI codierte Zeichenfolge)	Root .MQMD.ReplyToQ oder Root .MQMD.ReplyToQMgr oder jms.Rto

Eigenschaft	Synonym für
JMSTimestamp	Root .MQMD.PutDate oder Root .MQMD.PutTime oder jms.Tms
JMSType	mcd.Type oder mcd.Set oder mcd.Fmt
JMSXAppID	Root .MQMD.PutApplName
JMSXDeliveryCount	Root .MQMD.BackoutCount
JMSXGroupID	Root .MQMD.GroupId oder jms.Gid
JMSXGroupSeq	Root .MQMD.MsgSeqNumber oder jms.Seq
JMSXUserID	Root .MQMD.UserIdentifier

Mit diesen Synonymen kann eine MQI-Anwendung auf ähnliche Weise wie eine IBM MQ classes for JMS -Clientanwendung auf JMS -Eigenschaften zugreifen. Von diesen Eigenschaften können nur JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupID und JMSXGroupSeq mithilfe von MQI festgelegt werden.

Beachten Sie, dass die JMS_IBM_*-Eigenschaften, die aus IBM MQ classes for JMS heraus verfügbar sind, bei Verwendung von MQI nicht verfügbar sind. MQI-Anwendungen können anderweitig auf die von den JMS_IBM_*-Eigenschaften referenzierten Felder zugreifen.

2. Unabhängig von ihrer Schreibweise (Groß-/Kleinschreibung) darf eine Eigenschaft nie wie folgt heißen: "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" und "ESCAPE". Diese Namen sind SQL-Schlüsselwörter, die in Auswahlzeichenfolgen verwendet werden.
3. Ein Eigenschaftsname, der mit " beginnt mq " In jeder Mischung aus Klein-oder Großbuchstaben und nicht beginnend mit "mq_usr" kann nur ein "." Zeichen (U+002E). Mehrere "." Zeichen sind in Eigenschaften mit diesen Präfixen nicht zulässig.
4. Zwei "." Zeichen müssen andere Zeichen dazwischen enthalten; ein leerer Punkt in der Hierarchie ist nicht zulässig. Ebenso darf ein Eigenschaftsname nicht mit einem "." enden. gesetzt.
5. Wenn eine Anwendung die Eigenschaft "a.b" und dann die Eigenschaft "a.b.c" festlegt, ist nicht erkennbar, ob "b" in der Hierarchie einen Wert oder eine andere logische Gruppierung enthält. Eine solche Hierarchie wird als "gemischter Inhalt" bezeichnet und nicht unterstützt. Die Festlegung einer Eigenschaft, die zu gemischtem Inhalt führt, ist nicht zulässig.

Die Umsetzung dieser Einschränkungen wird vom Mechanismus der Gültigkeitsprüfung wie folgt erzwungen:

- Die Gültigkeit von Eigenschaftsnamen wird überprüft, wenn eine Eigenschaft mit dem Aufruf MQSETMP-Nachrichteneigenschaft festlegen festgelegt wird, falls eine Gültigkeitsprüfung beim Erstellen des Nachrichtenhandle angefordert wurde. Wenn eine versuchte Gültigkeitsprüfung für eine Eigenschaft aufgrund eines Fehlers in der Angabe des Eigenschaftsnamens fehlschlägt, wird der Beendigungscode MQCC_FAILED mit der entsprechenden Ursache zurückgegeben:
  - MQRC_PROPERTY_NAME_ERROR für die Ursachen 1-4.
  - MQRC_MIXED_CONTENT_NOT_ALLOWED für die Ursache 5.
- Eigenschaftsnamen, die direkt als MQRFH2-Elemente angegeben werden, werden nicht immer vom Aufruf MQPUT auf ihre Gültigkeit hin überprüft.

Nachrichtendeskriptorfelder als Eigenschaften
Die meisten Nachrichtendeskriptorfelder können als Eigenschaften behandelt werden. Der Eigenschaftsname wird erstellt, indem dem Namen des Nachrichtendeskriptorfelds ein Präfix hinzugefügt wird.

Wenn eine MQI-Anwendung eine in einem Nachrichtendeskriptorfeld (z. B. in einer Selektorzeichenfolge oder mittels der APIs der Nachrichteneigenschaft) enthaltene Nachrichteneigenschaft identifizieren möchte, verwenden Sie die folgende Syntax:

Eigenschaftsname	Nachrichtendeskriptorfeld
Root.MQMD.*Field*	*Feld*

Geben Sie *Field* mit dem gleichen Fall wie für die MQMD-Strukturfelder in der Deklaration der Programmiersprache C an. Beispiel: Der Eigenschaftsname `Root.MQMD.AccountingToken` greift auf das Feld `AccountingToken` des Nachrichtendeskriptors zu.

Die Felder `StrucId` und `Version` des Nachrichtendeskriptors können über die gezeigte Syntax nicht aufgerufen werden.

Nachrichtendeskriptorfelder werden in einem MQRFH2-Header niemals für andere Eigenschaften dargestellt.

Wenn die Nachrichtendaten mit einem MQMDE beginnen, der vom Warteschlangenmanager erkannt wird, ist der Zugriff auf die MQMDE-Felder über die beschriebene Notation `Root.MQMD.`*`Field`* möglich. In diesem Fall werden die MQMDE-Felder als logische Komponente des MQMD aus der Eigenschaftsperspektive behandelt. Siehe Übersicht über MQMDE.

Merkmalsdatentypen und -werte

Eine Eigenschaft kann boolesch, eine Bytefolge, eine Zeichenfolge, eine Gleitkomma- oder Ganzzahlzahl sein. Die Eigenschaft kann jeden gültigen Wert im Bereich des Datentyps speichern, sofern nicht anderweitig durch den Kontext beschränkt.

Der Datentyp eines Eigenschaftswerts muss einer der folgenden Werte sein:

- MQBOOL
- MQBYTE[ ]
- MQCHAR[ ]
- MQFLOAT32
- MQFLOAT64
- MQINT8
- MQINT16
- MQINT32
- MQINT64

Eine Eigenschaft kann existieren, darf aber keinen definierten Wert haben; es ist eine Nulleigenschaft. Eine Nulleigenschaft unterscheidet sich von einer Byteeigenschaft (MQBYTE[ ]) oder der Eigenschaft einer Zeichenfolge (MQCHAR[ ]) dadurch, dass sie einen definierten, aber leeren Wert (d. h. ein Wert mit Nulllänge) aufweist.

Bytefolge ist in JMS oder XMS kein gültiger Eigenschaftsdatentyp. Verwenden Sie im Ordner *usr* keine Bytefolgeneigenschaften.

Nachrichten aus Warteschlangen auswählen

Die Auswahl von Nachrichten aus Warteschlangen ist mithilfe der Felder 'MsgId' und 'CorrelId', über einen MQGET-Aufruf oder unter Verwendung einer Auswahlzeichenfolge (SelectionString) in einem MQOPEN- bzw. MQSUB-Aufruf möglich.

Selectors

Ein Nachrichtenselektor ist eine Zeichenfolge variabler Länge, mit der sich eine Anwendung für den Empfang nur solcher Nachrichten registriert, deren Eigenschaften die als Auswahlzeichenfolge dargestellte SQL-Abfrage (Structured Query Language) erfüllen.

Auswahl mithilfe der Funktionsaufrufe MQSUB und MQOPEN

Für eine Auswahl mithilfe der Aufrufe MQSUB und MQOPEN verwenden Sie die Struktur *SelectionString*. Dabei handelt es sich um eine Struktur des Typs MQCHARV.

Mit der Struktur *SelectionString* wird eine Auswahlzeichenfolge variabler Länge an den Warteschlangenmanager übergeben.

Die ID des codierten Zeichensatzes, die der Selektorzeichenfolge zugeordnet ist, wird über das Feld VSCCSID der MQCHARV-Struktur festgelegt. Für die ID des codierten Zeichensatzes muss ein Wert verwendet werden, der für Selektorzeichenfolgen unterstützt wird. Eine Liste der unterstützten Codepages finden Sie im Abschnitt Codepagekonvertierung.

Wenn Sie eine CCSID angeben, für die IBM MQ keine Unicode-Konvertierung unterstützt, führt dies zum Fehler MQRC_SOURCE_CCSID_ERROR. Dieser Fehler wird gemeldet, wenn der Selektor an den Warteschlangenmanager übergeben wird, also beim Aufruf MQSUB, MQOPEN oder MQPUT1.

Der Standardwert für das Feld *VSCCSID* lautet MQCCSI_APPL. Dies bedeutet, dass die ID des codierten Zeichensatzes der Auswahlzeichenfolge mit der ID des codierten Zeichensatzes des Warteschlangenmanagers bzw. des Clients (falls die Verbindung über einen Client erfolgt) identisch ist. Die Konstante MQCCSI_APPL kann jedoch vor der Kompilierung durch eine Anwendung in einer erneuten Definition überschrieben werden.

Falls der MQCHARV-Selektor eine NULL-Zeichenfolge darstellt, wird für diesen Nachrichtenkonsumenten keine Auswahl getroffen. Stattdessen werden die Nachrichten zugestellt, als ob kein Selektor verwendet worden wäre.

Die maximale Länge einer Auswahlzeichenfolge wird lediglich durch die Festlegung des MQCHARV-Felds *VSLength* begrenzt.

Die Auswahlzeichenfolge ('SelectionString') wird in der Ausgabe eines MQSUB-Aufrufs mit der Subskriptionsoption MQSO_RESUME zurückgegeben, wenn ein Puffer bereitgestellt wird und in 'VSBufSize' eine positive Puffergröße angegeben ist. Wird kein Puffer bereitgestellt, wird nur die Länge der Auswahlzeichenfolge im Feld 'VSLength' der MQCHARV-Struktur zurückgegeben. Ist der bereitgestellte Puffer kleiner als der für die Rückgabe des Feldes erforderliche Speicherplatz, werden nur VSBufSize-Bytes im bereitgestellten Puffer zurückgegeben.

Eine Anwendung kann eine Auswahlzeichenfolge erst ändern, wenn sie zuvor entweder die Kennung für die Warteschlange (bei MQOPEN) oder für die Subskription (bei MQSUB) geschlossen hat. In einem nachfolgenden MQOPEN- oder MQSUB-Aufruf kann dann eine neue Auswahlzeichenfolge angegeben werden.

MQOPEN
Schließen Sie die geöffnete Kennung mit MQCLOSE und geben Sie dann in einem nachfolgenden MQOPEN-Aufruf eine neue Auswahlzeichenfolge an.

MQSUB
Schließen Sie die zurückgegebene Subskriptionskennung (hSub) mit MQCLOSE und geben Sie dann in einem nachfolgenden MQSUB-Aufruf eine neue Auswahlzeichenfolge an.

Abbildung 3 auf Seite 34 zeigt den Auswahlprozess unter Verwendung des Aufrufs MQSUB.

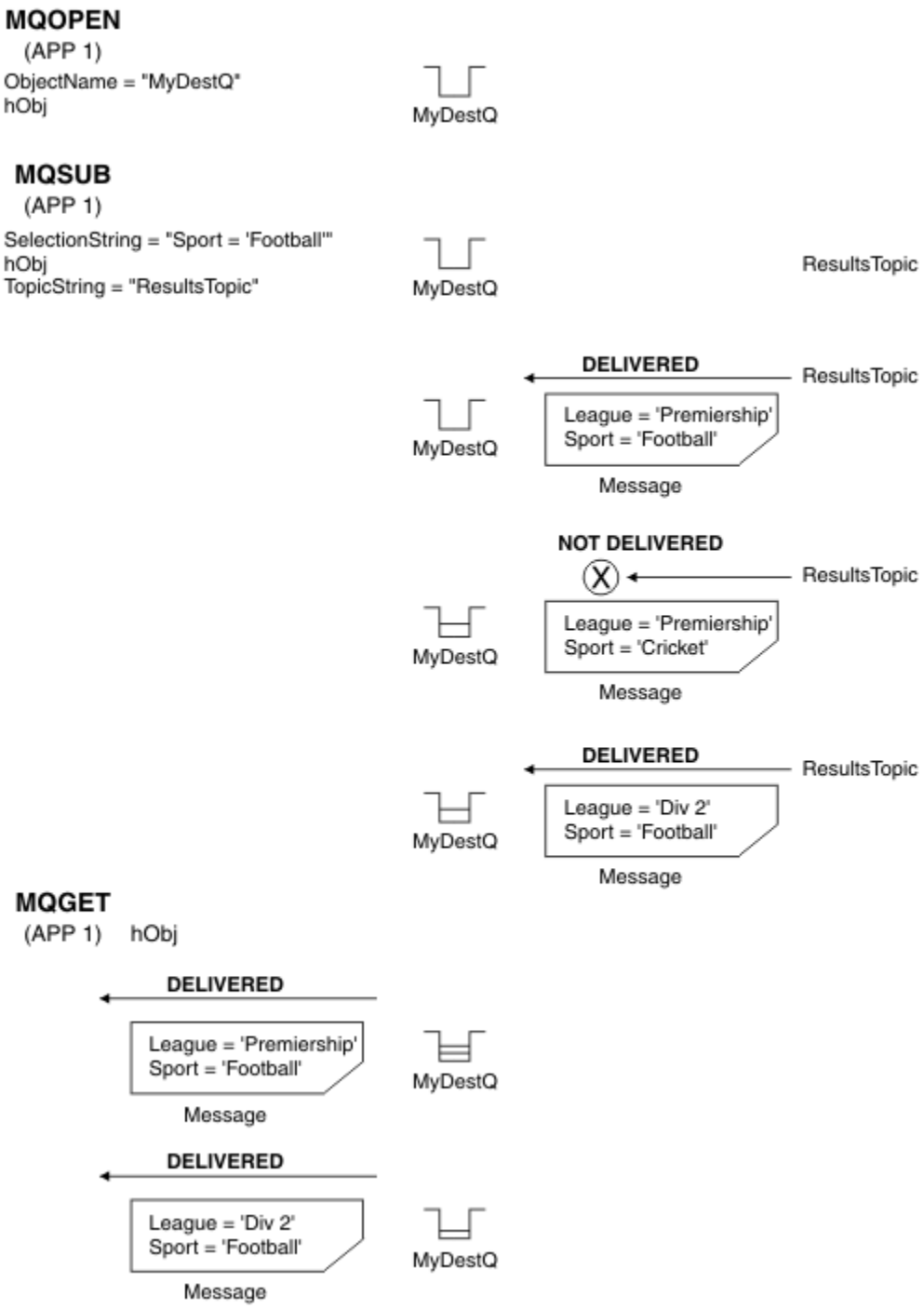


Abbildung 3. Auswahl mithilfe des Aufrufs MQSUB

Ein Selektor kann einem MQSUB-Aufruf über das Feld *SelectionString* der MQSD-Struktur übergeben werden. Die Übergabe eines Selektors im Aufruf MQSUB bewirkt, dass in der Zielwarteschlange nur Nachrichten zur Verfügung gestellt werden, die zu dem subskribierten Thema veröffentlicht werden und einer angegebenen Auswahlzeichenfolge entsprechen.

Abbildung 4 auf Seite 35 zeigt den Auswahlprozess unter Verwendung des Aufrufs MQOPEN.

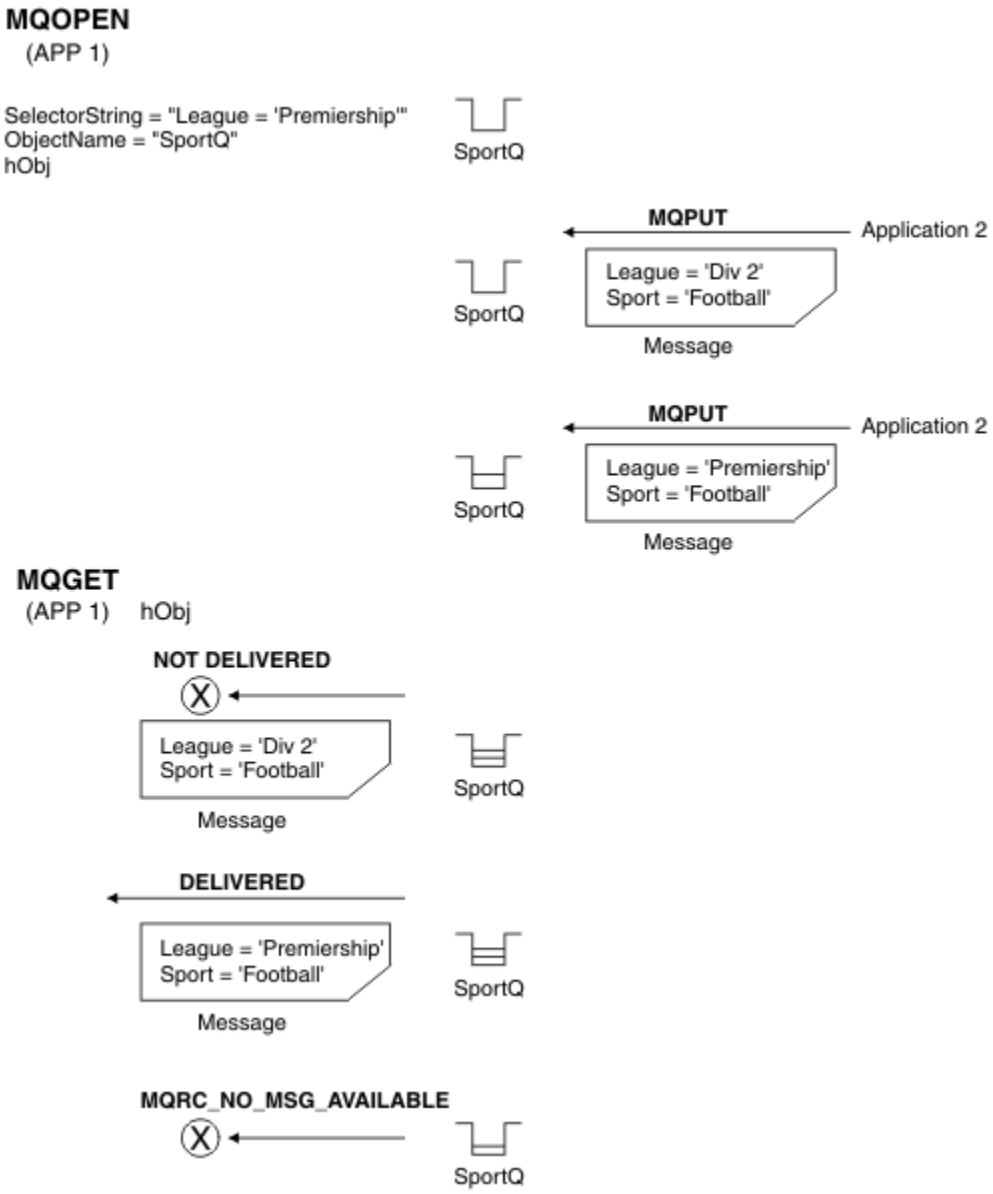


Abbildung 4. Auswahl mithilfe des Aufrufs MQOPEN

Ein Selektor kann einem MQOPEN-Aufruf über das Feld *SelectionString* der MQOD-Struktur übergeben werden. Die Übergabe eines Selektors im Aufruf MQOPEN bewirkt, dass nur die Nachrichten in der geöffneten Warteschlange an den Nachrichtenkonsumenten zugestellt werden, die einem Selektor entsprechen.

Der Selektor im Aufruf MQOPEN wird hauptsächlich bei einer Punkt-zu-Punkt-Übermittlung verwendet, bei der eine Anwendung festlegen kann, dass nur die Nachrichten in einer Warteschlange empfangen werden sollen, die einem Selektor entsprechen. Das vorherige Beispiel veranschaulicht ein einfaches Szenario, bei dem zwei Nachrichten in eine mit MQOPEN geöffnete Warteschlange eingereiht werden. Es wird jedoch nur eine Nachricht von der Zielanwendung empfangen, da lediglich diese Nachricht einem Selektor entspricht.

Beachten Sie, dass bei nachfolgenden MQGET-Aufrufen der Code MQRC_NO_MSG_AVAILABLE ausgegeben wird, da die Warteschlange keine weiteren Nachrichten enthält, die dem angegebenen Selektor entsprechen.

Zugehörige Konzepte

„Regeln und Einschränkungen für Auswahlzeichenfolgen” auf Seite 43

Machen Sie sich mit diesen Regeln zur Interpretation von Auswahlzeichenfolgen vertraut und informieren Sie sich über Zeichenbeschränkungen, um potenzielle Probleme bei der Verwendung von Selektoren zu vermeiden.

Auswahlverhalten

Übersicht über das IBM MQ-Auswahlverhalten.

Die Felder in einer MQMDE-Struktur gelten als Nachrichteneigenschaften für die entsprechenden Nachrichtendeskriptoreigenschaften, wenn der MQMD folgende Kriterien erfüllt:

- Er hat das Format MQFMT_MD_EXTENSION.
- Im Anschluss an den MQMD folgt unmittelbar eine gültige MQMDE-Struktur.
- Er weist die Version 1 auf oder enthält nur Felder der Standardversion 2.

Es kann vorkommen, dass eine Auswahlzeichenfolge bereits vor dem Abgleich mit Nachrichteneigenschaften entweder in TRUE oder FALSE aufgelöst wird. Dies kann beispielsweise der Fall sein, wenn die Auswahlzeichenfolge auf `"TRUE <> FALSE"` gesetzt ist. Eine derartige frühzeitige Auswertung kann nur garantiert werden, wenn die Auswahlzeichenfolge keine Nachrichteneigenschaftenverweise enthält.

Wenn eine Auswahlzeichenfolge vor der Berücksichtigung von Nachrichteneigenschaften in TRUE aufgelöst wird, werden alle Nachrichten zugestellt, die zu dem vom Konsumenten subskribierten Thema veröffentlicht werden. Wird eine Auswahlzeichenfolge vor der Berücksichtigung von Nachrichteneigenschaften in FALSE aufgelöst, werden für den Funktionsaufruf, von dem der Selektor übergeben wurde, der Ursachencode MQRC_SELECTOR_ALWAYS_FALSE und der Beendigungscode MQCC_FAILED zurückgegeben.

Selbst wenn eine Nachricht neben den Headereigenschaften keine weiteren Nachrichteneigenschaften enthält, kann sie dennoch für die Auswahl infrage kommen. Wenn eine Auswahlzeichenfolge auf eine nicht existente Nachrichteneigenschaft verweist, wird angenommen, dass diese Eigenschaft den Wert NULL oder 'Unknown' hat.

Beispiel: Eine Nachricht erfüllt unter Umständen weiterhin eine Auswahlzeichenfolge wie z. B. `'Color IS NULL'`, wobei `'Color'` nicht als Nachrichteneigenschaft in der Nachricht existiert.

Die Auswahl kann nur für die Eigenschaften erfolgen, die einer Nachricht zugeordnet sind. Eine Auswahl für die Nachricht selbst ist nur möglich, wenn ein erweiterter Nachrichtenauswahlbereitsteller verfügbar ist. Eine Auswahl hinsichtlich der Nachrichtennutzdaten ist ebenfalls nur möglich, wenn ein erweiterter Nachrichtenauswahlbereitsteller verfügbar ist.

Jeder Nachrichteneigenschaft ist ein Typ zugeordnet. Bei einer Auswahl müssen Sie sicherstellen, dass die Werte, die in den Ausdrücken zum Testen der Nachrichteneigenschaften verwendet werden, den richtigen Typ aufweisen. Wenn keine Typübereinstimmung vorhanden ist, wird der betreffende Ausdruck in FALSE aufgelöst.

Sie sind dafür verantwortlich, dass die Auswahlzeichenfolge und Nachrichteneigenschaften kompatible Typen verwenden.

Die Auswahlkriterien finden auch für inaktive permanente Subskribenten weiter Anwendung, damit nur Nachrichten behalten werden, die der ursprünglich angegebenen Auswahlzeichenfolge entsprechen.

Auswahlzeichenfolgen können nicht geändert werden, wenn eine permanente Subskription mit einer Änderungsaktion (MQSO_ALTER) wiederaufgenommen wird. Wird eine andere Auswahlzeichenfolge dargestellt, wenn ein permanenter Subskribent seine Aktivität wiederaufnimmt, wird MQRC_SELECTOR_NOT_ALTERABLE an die Anwendung zurückgegeben.

Anwendungen erhalten den Rückkehrcode MQRC_NO_MSG_AVAILABLE, wenn eine Warteschlange keine Nachricht enthält, die den Auswahlkriterien entspricht.

Wenn eine Anwendung eine Auswahlzeichenfolge angegeben hat, die Eigenschaftswerte enthält, kommen nur die Nachrichten für die Auswahl infrage, die übereinstimmende Eigenschaften aufweisen. Wenn ein Subskribent beispielsweise die Auswahlzeichenfolge 'a = 3' angibt und eine Nachricht ohne Eigenschaften veröffentlicht wird bzw. mit Eigenschaften, in denen 'a' nicht vorhanden ist oder nicht 3 ergibt, empfängt der Subskribent diese Nachricht nicht in ihrer Zielwarteschlange.

Messaging-Leistung

Bei der Auswahl von Nachrichten aus einer Warteschlange ist es erforderlich, dass IBM MQ jede Nachricht in der Warteschlange nacheinander überprüft. Diese Überprüfung erfolgt so lange, bis eine Nachricht gefunden wird, die die Auswahlkriterien erfüllt, oder bis keine zu prüfenden Nachrichten mehr vorhanden sind. Eine Nachrichtenauswahl in sehr umfangreichen Warteschlangen wirkt sich daher negativ auf die Messaging-Leistung aus.

Um die Nachrichtenauswahl aus sehr umfangreichen Warteschlangen zu optimieren, wenn die Auswahl auf JMSCorrelationID oder JMSMessageID basiert, verwenden Sie eine Auswahlzeichenfolge in folgendem Format:

- JMSCorrelationID ='ID:*Korrelations-ID*'
- JMSMessageID='ID:*Nachrichten-ID*'

Dabei gilt:

- *Korrelations-ID* ist eine Zeichenfolge, die eine IBM MQ-Standardkorrelations-ID enthält.
- *Nachrichten-ID* ist eine Zeichenfolge, die eine IBM MQ-Standardnachrichten-ID enthält.

Anmerkung: Der Selektor sollte nur eine der Eigenschaften referenzieren. Die Verwendung eines Selektors, der eines dieser Formate aufweist, bietet eine erhebliche Leistungsverbesserung, wenn die Auswahl auf JMSCorrelationID basiert, und eine marginale Leistungsverbesserung für JMSMessageID. Weitere Informationen finden Sie unter „Nachrichtenselektoren in JMS" auf Seite 152.

Komplexe Selektoren verwenden

Selektoren können viele Komponenten enthalten, beispielsweise:

a und b oder c und d oder e und `f` oder g und h oder `i` und `j`... oder y und z

Die Verwendung solcher komplexer Selektoren kann sich auf die Leistung auswirken und einen übermäßigen Ressourcenbedarf zur Folge haben. Daher schützt IBM MQ das System, indem die Verarbeitung zu komplexer Selektoren fehlschlägt, da andernfalls eine Knappheit von Systemressourcen entstünde. Der Schutz kann für Auswahlzeichenfolgen erfolgen, die über 100 Tests enthalten, oder wenn IBM MQ feststellt, dass der Betriebssystemstack die maximale Größe erreicht. Probieren Sie die Verwendung von Auswahlzeichenfolgen sorgfältig mit mehreren Komponenten auf den entsprechenden Plattformen aus und testen Sie sie gründlich, um sicherzustellen, dass die Schutzgrenzwerte nicht überschritten werden.

Sie können die Leistung und Komplexität von Selektoren verbessern, indem Sie zusätzliche Klammern verwenden, um Komponenten zu kombinieren. For example:

(a und b oder c und d) oder (e und `f` oder g und h) oder (`i` und `j`) ...

Zugehörige Konzepte

„Regeln und Einschränkungen für Auswahlzeichenfolgen" auf Seite 43
Machen Sie sich mit diesen Regeln zur Interpretation von Auswahlzeichenfolgen vertraut und informieren Sie sich über Zeichenbeschränkungen, um potenzielle Probleme bei der Verwendung von Selektoren zu vermeiden.

Syntax der Nachrichtenselektoren

Ein IBM MQ-Nachrichtenselektor ist eine Zeichenfolge, deren Syntax auf einer Untergruppe der SQL92-Syntax für bedingte Ausdrücke basiert.

Die Reihenfolge der Auswertung eines Nachrichtenselektors erfolgt innerhalb einer Rangfolgestufe von links nach rechts. Sie können diese Reihenfolge durch die Verwendung von Klammern ändern. Vordefinierte Selektorliterale und Operatorbezeichnungen werden hier in Großbuchstaben geschrieben, die Groß-/Kleinschreibung muss jedoch nicht beachtet werden.

Wenn der Selektor über die API bereitgestellt wird, prüft IBM MQ die Richtigkeit der Syntax eines Nachrichtenselektors zum Zeitpunkt der Darstellung. Wenn die Syntax der Auswahlzeichenfolge falsch ist oder ein Eigenschaftsname nicht gültig ist und ein erweiterter Nachrichtenauswahlanbieter nicht verfügbar ist,

wird MQRC_SELECTION_NOT_AVAILABLE an die Anwendung zurückgegeben. Ist die Syntax der Auswahlzeichenfolge falsch oder ein Eigenschaftsname nicht gültig, wenn eine Subskription wiederaufgenommen wird, wird ein MQRC_SELECTOR_SYNTAX_ERROR an die Anwendung zurückgegeben. Wenn die Prüfung des Eigenschaftsnamens beim Festlegen der Eigenschaft inaktiviert war (indem MQCMHO_NONE anstatt MQCMHO_VALIDATE angegeben wurde) und eine Anwendung nachfolgend eine Nachricht mit einem ungültigen Eigenschaftsnamen einreiht, wird diese Nachricht nie ausgewählt.

Zum Zeitpunkt der Darstellung des Selektors wird kein Fehler zurückgegeben, wenn IBM MQ feststellt, dass ein administrativ definierter Subskriptionsselektor die erweiterte Nachrichtensyntax verwendet, wie durch den **DISPLAY SUB** -Parameter **SELTYPE** mit dem Wert ERWEITERTangegeben. In diesem Fall wird die Überprüfung der Syntax der Auswahlzeichenfolge bis zum Zeitpunkt der Veröffentlichung verzögert (siehe MQRC_SELECTION_NOT_AVAILABLE).

Ein Selektor kann Folgendes enthalten:

- Literale:
  - Zeichenfolgeliterale stehen zwischen einfachen Anführungszeichen. Zwei aufeinanderfolgende einfache Anführungszeichen (Hochkommas) stellen ein einfaches Anführungszeichen dar, Beispiele: 'literal' und 'literal''s'. Genau wie Java-Zeichenfolgeliterale verwenden sie die Unicode-Zeichencodierung. Ein Zeichenfolgeliteral darf nicht in Anführungszeichen stehen. Zwischen einfachen Anführungszeichen darf jede Abfolge von Bytes verwendet werden.
  - Eine Bytefolge besteht aus mindestens einem Paar Hexadezimalzeichen in Anführungszeichen und hat das Präfix 0x. Beispiele sind "0x2F1C" oder "0XD43A". Die Länge einer Bytefolge muss mindestens ein Byte betragen. Wenn eine Selektorbytefolge an eine Nachrichteneigenschaft vom Typ MQTYPE_BYTE_STRING angepasst wird, wird keine besondere Aktion an einer führenden oder abschließenden Null vorgenommen. Die Bytes werden als weiteres Zeichen behandelt. Endianness wird ebenfalls nicht berücksichtigt. Die Länge von Selektor- und Eigenschaftsbytefolgen muss identisch sein, ebenso wie die Bytefolge.

    Beispiele übereinstimmender Bytefolgeauswahlen (wenn *myBytes* = 0AFC23) sind:

    - "myBytes = "0x0AFC23"" = TRUE

    Folgende Zeichenfolgenauswahlen stimmen nicht überein:

    - "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (die Anzahl der Bytes ist kein Vielfaches von zwei)
    - "myBytes = "0x0AFC2300"" = FALSE (die abschließende Null ist im Vergleich signifikant)
    - "myBytes = "0x000AFC23"" = FALSE (die führende Null ist im Vergleich signifikant)
    - "myBytes = "0x23FC0A"" = FALSE (Endianness wird nicht berücksichtigt)
  - Hexadezimalzahlen beginnen mit einer Null, gefolgt von x in Groß- oder Kleinschreibung. Der Rest des Literals enthält ein oder mehrere gültige Hexadezimalzeichen. Beispiele: 0xA, 0xAF, 0X2020.
  - Eine führende Null, gefolgt von mindestens einer Ziffer zwischen 0-7, wird immer als Beginn einer Oktalzahl interpretiert. Sie können keine Dezimalzahl mit dem Präfix Null darstellen. 09 gibt beispielsweise einen Syntaxfehler zurück, da 9 keine gültige oktale Ziffer ist. Oktalzahlen sind beispielsweise 0177, 0713.
  - Ein exaktes numerisches Literal ist ein numerischer Wert ohne Dezimalzeichen, wie beispielsweise 57, -957 und +62. Ein exaktes numerisches Literal kann mit einem L in Groß- oder Kleinschreibung abschließen. Dies hat keine Auswirkung auf das Speichern oder Interpretieren der Zahl. IBM MQ unterstützt exakte Numerale von -9,223,372,036,854,775,808 bis 9,223,372,036,854,775,807.
  - Ein ungefähres (approximatives) numerisches Literal ist ein numerischer Wert in Exponentialschreibweise, wie beispielsweise 7E3 oder -57.9E2, oder ein numerischer Wert mit einer Dezimalstelle, wie beispielsweise 7., -95.7 oder +6.2. IBM MQ unterstützt Zahlen von -1.797693134862315E+308 bis 1.797693134862315E+308.

Das Festkommateil muss auf ein optionales Vorzeichen folgen (+ oder -). Der Festkommateil muss entweder eine ganze Zahl oder eine Bruchzahl sein. Der Bruchteil des Festkommateils muss eine führende Ziffer aufweisen.

Ein E in Groß- oder Kleinschreibung zeigt den Beginn eines optionalen Exponenten an. Der Exponent verfügt über eine Dezimalbasis und der Zahlenteil des Exponenten kann ein optionales Vorzeichen als Präfix haben.

Ungefähre numerische Literale können durch das Zeichen F oder D beendet werden (Groß-/Kleinschreibung muss nicht beachtet werden). Diese Syntax unterstützt das sprachenübergreifende Tagging von Zahlen mit einfacher oder doppelter Genauigkeit. Diese Zeichen sind optional und haben keine Auswirkung auf das Speichern oder Verarbeiten ungefährer numerischer Literale. Diese Zahlen werden immer mit doppelter Genauigkeit gespeichert und verarbeitet.

– Boolesche Literale TRUE und FALSE.

Anmerkung: Nicht finite IEEE-754-Darstellungen wie `NaN`, `+Infinity` oder `-Infinity` werden in Auswahlzeichenfolgen nicht unterstützt. Daher können diese Werte nicht als Operanden in einem Ausdruck verwendet werden. Für mathematische Operationen wird eine negative Null wie eine positive Null behandelt.

- Kennungen:

  Eine Kennung ist eine Zeichenfolge mit variabler Länge, die mit einem gültigen Kennungsanfangszeichen beginnen muss, gefolgt von einer Null oder mehreren gültigen Kennungsteilzeichen. Kennungsnamen und Nachrichteneigenschaftsnamen unterliegen denselben Regeln. Weitere Informationen hierzu finden Sie unter „Eigenschaftsnamen” auf Seite 29 und „Einschränkungen bei Eigenschaftsnamen” auf Seite 30.

  Anmerkung: Eine Auswahl hinsichtlich der Nachrichtennutzdaten ist ebenfalls nur möglich, wenn ein erweiterter Nachrichtenauswahlbereitsteller verfügbar ist.

  Bei Kennungen handelt es sich entweder Verweise auf Headerfelder oder auf Eigenschaften. Dieser Eigenschaftswerttyp in einem Nachrichtenselektor muss dem Typ entsprechen, mit dem die Eigenschaft festgelegt wird, obwohl so weit wie möglich eine sogenannte 'Numeric Promotion' (Konvertierung) durchgeführt wird. Stimmen die Typen nicht überein, lautet das Ergebnis des Ausdrucks FALSE. Wenn eine Eigenschaft referenziert wird, die nicht in einer Nachricht vorhanden ist, lautet ihr Wert `NULL`.

  Typenkonvertierungen, die für die GET-Methoden für Eigenschaften gelten, sind nicht anwendbar, wenn eine Eigenschaft in einem Nachrichtenselektorausdruck verwendet wird. Wenn Sie beispielsweise eine Eigenschaft als Zeichenfolgewert festlegen und diesen dann mit einem Selektor als numerischen Wert abfragen, gibt der Ausdruck FALSE zurück.

  Namen von JMS-Feldern und Eigenschaften, die mit den Eigenschaftsnamen oder den Namen der `MQMD`-Felder übereinstimmen, sind ebenfalls gültige Kennungen in einer Auswahlzeichenfolge. IBM MQ stimmt die erkannten Namen von JMS-Feldern und -eigenschaften auf die Werte der Nachrichteneigenschaft ab. Weitere Informationen hierzu finden Sie unter „Nachrichtenselektoren in JMS” auf Seite 152. Beispiel: Die Auswahlzeichenfolge `"JMSPriority >="` trifft ihre Auswahl anhand der Eigenschaft `Pri` im Ordner `jms` der aktuellen Nachricht.

- Überlauf/Unterlauf:

  Folgende Bedingungen sind für dezimale und ungefähre numerische Zahlen nicht definiert:

  – Angabe einer Zahl außerhalb des definierten Bereichs

  – Angabe eines arithmetischen Ausdrucks, der einen Über- bzw. Unterlauf verursachen könnte

  Diese Bedingungen werden nicht geprüft.

- Leerzeichen:

  Definiert als Leerzeichen, Seitenvorschub, Zeilenumbruch, Rücklauf, Horizontal- oder Vertikaltabulator. Folgende Unicode-Zeichen werden als Leerzeichen erkannt:

  – `\u0009 to \u000D`

  – `\u0020`

  - \u001C
  - \u001D
  - \u001E
  - \u001F
  - \u1680
  - \u180E
  - \u2000 bis \u200A
  - \u2028
  - \u2029
  - \u202F
  - \u205F
  - \u3000
- Ausdrücke:
  - Ein Selektor ist ein Bedingungsausdruck. Ein Selektor, der als 'wahr' (true) ausgewertet wird, ist eine Übereinstimmung; ein Selektor, der als 'falsch' (false) oder 'unbekannt' (unknown) ausgewertet wird, ist keine Übereinstimmung.
  - Arithmetische Ausdrücke bestehen aus sich selbst, aus arithmetischen Operationen, Kennungen (Kennungswert wird als numerisches Literal behandelt) und aus numerischen Literalen.
  - Bedingungsausdrücke bestehen aus sich selbst, aus Vergleichsoperationen und aus logischen Operationen.
- Die Standardverwendung von Klammern () zur Festlegung der Auswertungsreihenfolge von Ausdrücken wird unterstützt.
- logische Operatoren in der Reihenfolge ihrer Priorität: NOT, AND, OR.
- Vergleichsoperatoren: =, >, >=, <, <=, <> (ungleich).
  - Zwei Bytefolgen sind nur dann gleich, wenn die Zeichenfolgen dieselbe Länge und Bytefolge haben.
  - Nur Werte des gleichen Typs können verglichen werden. Als einzige Ausnahme ist jedoch der Vergleich von exakten und ungefähren numerischen Werte möglich (die erforderliche Typenkonvertierung wird durch die Regeln der Java Numeric Promotion definiert). Falls versucht wird, verschiedene Typen zu vergleichen, ist der Selektor immer 'false'.
  - Der Vergleich von Zeichenfolgen und booleschen Werten beschränkt sich auf = und <>. Zwei Zeichenfolgen sind nur gleich, wenn sie dieselbe Zeichenfolge enthalten.
- Arithmetische Operatoren mit Rangordnung:
  - +, - unär.
  - * Multiplikation und / Division.
  - + Addition und - Subtraktion.
  - Rechenoperationen für einen NULL-Wert werden nicht unterstützt. Bei einem entsprechenden Versuch wird der komplette Selektor immer als 'False' zurückgegeben.
  - Arithmetische Operationen müssen Java Numeric Promotion anwenden.
- Vergleichsoperator arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 und arithmetic-expr3:
  - `Age BETWEEN 15 and 19` entspricht `age>= 15 AND age <= 19`.
  - `Age NOT BETWEEN 15 and 19` entspricht `age < 15 OR age > 19`.
  - Ist einer der Ausdrücke einer BETWEEN-Operation NULL, hat die Operation den Wert 'False'. Ist einer der Ausdrücke einer NOT BETWEEN-Operation NULL, hat die Operation den Wert 'True'.
- Kennung `[NOT] IN (string-literal1, string-literal2,...)` Vergleichsoperator, wobei Kennung eine Zeichenfolge oder einen NULL -Wert hat.

  - `Country IN ('UK', 'US', 'France')` gibt Wert 'True' für `'UK'` und 'False' für `'Peru'` zurück. Er ist äquivalent zum Ausdruck `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - `Country NOT IN ('UK', 'US', 'France')` gibt 'False' für `'UK'` und 'True' für `'Peru'` zurück. Er ist äquivalent zum Ausdruck `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`.
  - Hat die Kennung einer `IN`- oder `NOT IN`-Operation den Wert `NULL`, ist der Wert der Operation 'Unknown' (unbekannt).
- `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` -Vergleichsoperator, wobei `identifier` einen Zeichenfolgewert hat. *Musterwert* ist ein Zeichenfolgeliteral, wobei _ für ein beliebiges einzelnes Zeichen steht und % für eine beliebige Zeichenfolge (einschließlich der leeren Sequenz). Alle anderen Zeichen stehen für sich. Das optionale *Escapezeichen* ist ein Zeichenfolgeliteral mit einem einzelnen Zeichen, mit dem die besondere Bedeutung von _ und % in *Musterwert* geschützt wird. Der Operator `LIKE` darf nur zum Vergleich von zwei Zeichenfolgewerten verwendet werden.
  - `phone LIKE '12%3'` gibt 'True' für 123 und 12993 und 'False' für 1234 zurück.
  - `word LIKE 'l_se'` gibt 'True' für 'lose' und 'False' für 'loose' zurück.
  - `underscored LIKE '\_%' ESCAPE '\'` gibt Wert 'True' für `_foo` und 'False' für `bar` zurück.
  - `phone NOT LIKE '12%3'` gibt 'False' für 123 und 12993 und 'True' für 1234 zurück.
  - Hat die Kennung einer `LIKE`- oder `NOT LIKE`-Operation den Wert `NULL`, ist der Wert der Operation 'Unknown' (unbekannt).

  Anmerkung: Der Operator `LIKE` muss zum Vergleich von zwei Zeichenfolgewerten verwendet werden. Der Wert von `Root.MQMD.CorrelId` ist ein 24-Byte-Array, keine Zeichenfolge. Die Selektorzeichenfolge `Root.MQMD.CorrelId LIKE 'ABC%'` wird vom Parser als syntaktisch gültig akzeptiert, ergibt jedoch 'False'. Wenn Sie ein Byte-Array mit einer Zeichenfolge vergleichen, kann `LIKE` daher nicht verwendet werden.
- Vergleichsoperator `identifier IS NULL` testet den Headerfeldwert `NULL` oder einen fehlenden Eigenschaftswert.
- Vergleichsoperator `identifier IS NOT NULL` testet das Vorhandensein eines Headerfeldwerts ungleich null oder eines Eigenschaftswerts.
- Nullwerte

  Die Evaluierung von Selektorausdrücken, die `NULL`-Werte enthalten, wird von der `NULL`-Semantik SQL 92 definiert. Zusammenfassung:
  - SQL behandelt einen `NULL`-Wert als unbekannten Wert ('unknown').
  - Ein Vergleich oder eine Arithmetik mit einem unbekannten Wert ergibt immer einen unbekannten Wert.
  - Die Operatoren `IS NULL` und `IS NOT NULL` konvertieren einen unbekannten Wert in die Werte `TRUE` und `FALSE`.

  Die booleschen Operatoren verwenden eine dreiwertige Logik (`T=TRUE`, `F=FALSE`, `U=UNKNOWN`)

Tabelle 1. Ergebniswert der booleschen Operatoren bei der Logik `A AND B`

Operator A	**Operator B**	**Ergebnis (A AND B)**
T	F	F
T	U	U
T	T	T
F	T	F
F	U	F
F	F	F

Tabelle 1. Ergebniswert der booleschen Operatoren bei der Logik A AND B (Forts.)		
Operator A	**Operator B**	**Ergebnis (A AND B)**
U	T	U
U	U	U
U	F	F

Tabelle 2. Ergebniswert der booleschen Operatoren bei der Logik A OR B		
Operator A	**Operator B**	**Ergebnis (A OR B)**
T	F	T
T	U	T
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

Tabelle 3. Ergebniswert der booleschen Operatoren bei der Logik NOT A	
Operator A	**Ergebnis (NOT A)**
T	F
F	T
U	U

Der folgende Nachrichtenselektor wählt Nachrichten mit dem Nachrichtentyp 'car' (Auto), der Farbe 'blue' (Blau) und einer höheren Gewichtsangabe als 2.500 Pfund aus:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

SQL unterstützt Vergleich und Arithmetik fester Dezimalzahlen, Nachrichtenselektoren hingegen bieten keine Unterstützung. Daher sind exakte numerische Literale auf diejenigen ohne Dezimalstellen beschränkt. Dies ist auch der Grund, weshalb numerische Werte mit einer Dezimalstelle als alternative Darstellung für einen näherungsweise berechneten numerischen Wert verwendet werden können.

SQL-Kommentare werden nicht unterstützt.

Zugehörige Konzepte

„Nachrichteneigenschaften” auf Seite 28
Mittels Nachrichteneigenschaften ermöglichen Sie es einer Anwendung, die zu verarbeitenden Nachrichten auszuwählen oder Informationen zu einer Nachricht abzurufen, ohne auf den MQMD- oder den MQRFH2-Header zugreifen zu müssen. Sie erleichtern auch die Kommunikation zwischen IBM MQ- und JMS-Anwendungen.

Zugehörige Verweise

MsgHandle

MQBUFMH - Konvertieren von Puffern in Nachrichtenkennungen

Regeln und Einschränkungen für Auswahlzeichenfolgen
Machen Sie sich mit diesen Regeln zur Interpretation von Auswahlzeichenfolgen vertraut und informieren Sie sich über Zeichenbeschränkungen, um potenzielle Probleme bei der Verwendung von Selektoren zu vermeiden.

- Beim Publish/Subscribe-Messaging werden die Nachrichten so ausgewählt, wie sie vom Publisher gesendet wurden. Weitere Informationen bietet der Abschnitt Auswahlzeichenfolgen.
- Äquivalenz wird mittels eines einzelnen Gleichheitszeichens getestet; zum Beispiel `a = b` ist richtig, `a == b` hingegen ist falsch.
- Ein Operator, mit dem viele Programmiersprachen 'ungleich' darstellen, ist `!=`. Diese Darstellung ist kein gültiges Synonym für <>; so ist `a <> b` beispielsweise gültig, `a != b` hingegen ist ungültig.
- Einfache Anführungszeichen (Hochkommas) werden nur erkannt, wenn sie mit dem Zeichen ' (U+0027) dargestellt werden. Entsprechend müssen Anführungszeichen, die nur zum Einschließen von Bytefolgen gültig sind, mit dem Zeichen " (U+0022) dargestellt werden.
- Die Symbole &, &&, | und || sind keine Synonyme für eine logische Konjunktion/Disjunktion; beispielsweise muss `a && b` als `a AND b` angegeben werden.
- Die Platzhalterzeichen * und ? sind keine Synonyme für % und _.
- Selektoren, die zusammengesetzte Ausdrücke wie `20 < b < 30` enthalten, sind nicht gültig. Der Parser evaluiert Operatoren mit identischer Rangfolge von links nach rechts. Das Beispiel ändert sich also entsprechend zu `(20 < b) < 30`, was keinen Sinn ergibt. Stattdessen muss der Ausdruck als `(b > 20) AND (b < 30)` geschrieben werden.
- Bytefolgen müssen in Anführungszeichen eingeschlossen werden; wenn einzelne Anführungszeichen verwendet werden, wird die Bytefolge als Zeichenfolgeliteral interpretiert. Die Anzahl der Zeichen (nicht die Anzahl, die die Zeichen darstellen), die auf `0x` folgen, muss ein Vielfaches von zwei sein.
- Das Schlüsselwort `IS` ist kein Synonym für das Gleichheitszeichen. Die Auswahlzeichenfolgen `a IS 3` und `b IS 'red'` sind daher ungültig. Das Schlüsselwort `IS` existiert nur zur Unterstützung von `IS NULL` und `IS NOT NULL`.

Zugehörige Konzepte
„Auswahlverhalten” auf Seite 36
Übersicht über das IBM MQ-Auswahlverhalten.

Zugehörige Verweise
Auswahlzeichenfolgen

Überlegungen zu UTF-8 und Unicode bei der Verwendung von Nachrichtenselektoren

Zeichen, die nicht zwischen einfachen Anführungszeichen stehen und die reservierten Schlüsselwörter einer Auswahlzeichenfolge bilden, müssen in Basic Latin Unicode eingegeben werden (Zeichenbereich U+0000 bis U+0007F). Andere Codepunktdarstellungen alphanumerischer Zeichen sind nicht erlaubt. Die Zahl 1 muss beispielsweise mit dem Unicode-Zeichen U+0031 dargestellt werden. Die Verwendung des äquivalenten vollbreiten Ziffernformats U+FF11 oder des arabischen Unicode-Äquivalents U+0661 ist nicht gestattet.

Die Namen von Nachrichteneigenschaften können mit jeder gültigen Folge von Unicode-Zeichen angegeben werden. Nachrichteneigenschaftsnamen, die in Auswahlzeichenfolgen enthalten sind, welche in UTF-8 codiert sind, werden auch dann validiert, wenn sie Mehrbytezeichen enthalten. Die Validierung von Mehrbyte-UTF-8 ist streng. Stellen Sie daher sicher, dass Sie gültige UTF-8-Folgen für die Namen von Nachrichteneigenschaften verwenden. Zeichen, die außerhalb des Unicode-Bereichs 'Basic Multilingual Plane' (BMP) von U+0000 bis U+FFFF liegen und in UTF-16 durch Ersatzcodepunkte (X'D800' bis X'DFFF') bzw. in UTF-8 durch vier Byte dargestellt werden, werden für die Verwendung in Nachrichteneigenschaftsnamen nicht unterstützt.

Beim Vergleich auf Gleichheit werden Eigenschaftsnamen oder -werte nicht gesondert verarbeitet. Es findet also beispielsweise keine Prä-/Dekomposition statt und Ligaturen erhalten keine besondere Bedeutung. Das Unterkompositions-Umlautzeichen U+00FC wird beispielsweise nicht als Äquivalent von U+0075 + U+0308 behandelt und die Zeichenfolge 'ff' nicht als Äquivalent des Unicode-Zeichens U+FB00 (LATIN SMALL LIGATURE FF).

Eigenschaftsdaten in einfachen Anführungszeichen können durch jede beliebige Folge von Bytes dargestellt werden und werden nicht validiert.

Inhalt einer Nachricht auswählen

Eine Subskription, die auf der Auswahl von Nachrichtennutzdateninhalt basiert (auch als Inhaltsfilterung bezeichnet), ist möglich. Die Entscheidung, welche Nachrichten einer solchen Subskription bereitgestellt werden, kann jedoch nicht direkt von IBM MQ vorgenommen werden; stattdessen ist ein erweiterter Nachrichtenauswahlbereitsteller zur Verarbeitung der Nachrichten erforderlich, beispielsweise IBM Integration Bus.

Wenn eine Anwendung eine Topic-Zeichenfolge veröffentlicht und mindestens ein Subskribent den Inhalt der Nachricht über eine Auswahlzeichenfolge auswählen lässt, erfordert IBM MQ, dass der erweiterte Nachrichtenauswahlbereitsteller die Syntax der Veröffentlichung analysiert und IBM MQ informiert, ob die Veröffentlichung den Auswahlkriterien aller Subskribenten mit einem Inhaltsfilter entspricht.

Wenn der erweiterte Nachrichtenauswahlbereitsteller feststellt, dass die Veröffentlichung mit der Auswahlzeichenfolge des Subskribenten übereinstimmt, wird die Nachricht an den Subskribenten übermittelt.

Stellt der Provider für die erweiterte Nachrichtenauswahl fest, dass die Veröffentlichung nicht mit den Auswahlkriterien übereinstimmt, wird sie nicht an den Subskribenten weitergegeben. Dies kann dazu führen, dass der Aufruf MQPUT oder MQPUT1 mit Ursachencode MQRC_PUBLICATION_FAILURE fehlschlägt. Kann der Provider für erweiterte Nachrichtenauswahl die Veröffentlichung nicht parsen, wird der Ursachencode MQRC_CONTENT_ERROR zurückgegeben und der MQPUT- oder MQPUT1-Aufruf schlägt fehl.

Wenn der erweiterte Nachrichtenauswahlbereitsteller nicht verfügbar ist oder nicht entscheiden kann, ob der Subskribent die Veröffentlichung erhalten soll, wird Ursachencode MQRC_SELECTION_NOT_AVAILABLE zurückgegeben und der Aufruf MQPUT oder MQPUT1 schlägt fehl.

Wenn eine Subskription mit einem Inhaltsfilter erstellt wird und der erweiterte Nachrichtenauswahlbereitsteller nicht verfügbar ist, schlägt der Aufruf MQSUB mit Ursachencode MQRC_SELECTION_NOT_AVAILABLE fehl. Wenn eine Subskription mit einem Inhaltsfilter wiederaufgenommen wird und der erweiterte Nachrichtenauswahlbereitsteller nicht verfügbar ist, gibt der Aufruf MQSUB die Warnung MQRC_SELECTION_NOT_AVAILABLE zurück; die Subskription kann jedoch wiederaufgenommen werden.

Zugehörige Verweise

Auswahlzeichenfolgen

Asynchrone Verarbeitung von IBM MQ-Nachrichten

Bei der asynchronen Verarbeitung werden mehrere Message Queue Interface-Erweiterungen (MQI-Erweiterungen) verwendet. Es handelt sich dabei um die MQI-Aufrufe MQCB und MQCTL, mit denen eine MQI-Anwendung für die Verarbeitung von Nachrichten aus mehreren Warteschlangen geschrieben werden kann. Nachrichten werden der Anwendung mithilfe einer von der Anwendung angegebenen 'Codeeinheit' zugestellt, wobei die Nachricht oder ein Token, das die Nachricht repräsentiert, übergeben wird.

In den einfachsten Anwendungsumgebungen wird die Codeeinheit durch einen Funktionszeiger definiert. In anderen Umgebungen jedoch kann die Codeeinheit durch einen Programm- oder Modulnamen definiert sein.

In der asynchronen Nachrichtenverarbeitung werden die folgenden Begriffe verwendet:

Nachrichtenkonsument
: Ein Programmierkonstrukt, mit dem Sie ein Programm oder eine Funktion definieren können, das bzw. die mit einer Nachricht aufgerufen wird, sobald eine Nachricht verfügbar ist, die den Anwendungsanforderungen entspricht.

Event handler (Ereigniskennung)
: Ein Programmierkonstrukt, mit dem Sie ein Programm oder eine Funktion definieren können, das bzw. die aufgerufen wird, sobald ein asynchrones Ereignis auftritt, wie z. B. die Stilllegung eines Warteschlangenmanagers.

Callback
Ein generischer Begriff, der sich entweder auf die Routine eines Nachrichtenkonsumenten oder Ereignishandlers bezieht.

Mithilfe der asynchronen Verarbeitung können die Gestaltung und Implementierung neuer Anwendungen vereinfacht werden. Dies gilt insbesondere für Anwendungen, die mehrere Eingabewarteschlangen oder Subskriptionen verarbeiten. Wenn Sie jedoch mehrere Eingabewarteschlangen verwenden und Nachrichten prioritätsgesteuert verarbeiten, gilt in jeder einzelnen Warteschlange die jeweilige Prioritätsfolge unabhängig von den anderen Warteschlangen: Es kann vorkommen, dass Sie Nachrichten mit niedriger Priorität aus einer bestimmten Warteschlange vor Nachrichten mit hoher Priorität aus einer anderen Warteschlange erhalten. Die ordnungsgemäße Reihenfolge der Nachrichten kann nicht über mehrere Warteschlangen hinweg garantiert werden. Bei der Verwendung von API-Exits müssen Sie außerdem beachten, dass diese möglicherweise geändert werden müssen, damit sie die Aufrufe MQCB und MQCTL enthalten.

Die folgenden Abbildungen veranschaulichen eine beispielhafte Verwendung dieser Funktion.

In Abbildung 5 auf Seite 45 ist eine Multithread-Anwendung dargestellt, die Nachrichten aus zwei Warteschlangen verarbeitet. In diesem Beispiel werden alle Nachrichten dargestellt, die an eine einzelne Funktion zugestellt werden.

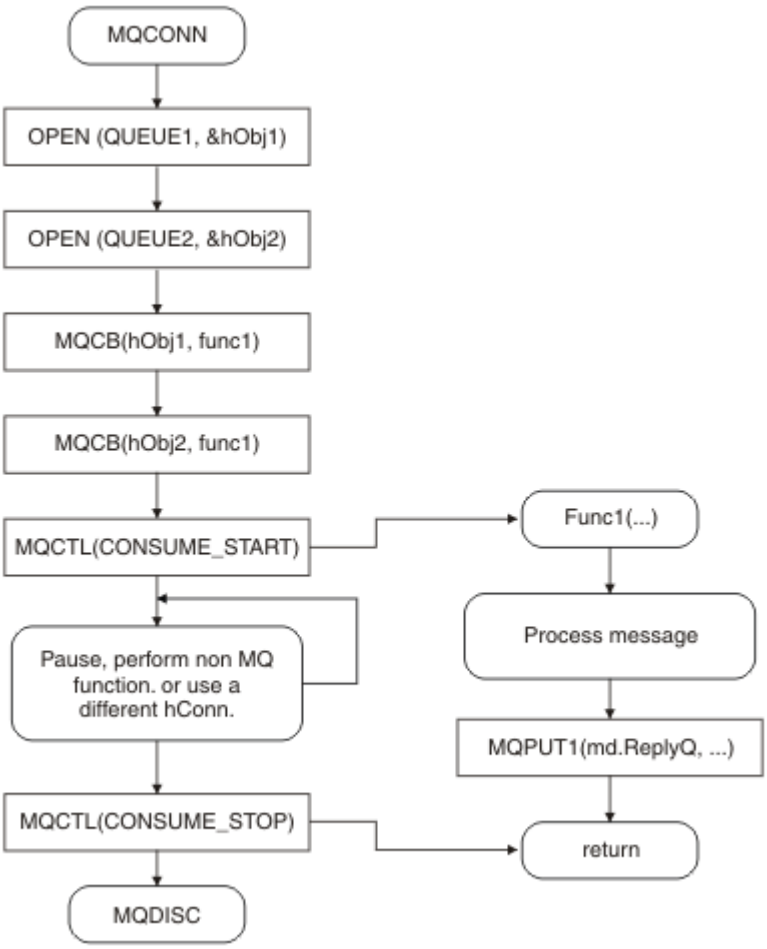


Abbildung 5. Nachrichtengesteuerte Standardanwendung, die Nachrichten aus zwei Warteschlangen verarbeitet

z/OS Unter z/OS muss der Hauptsteuer-Thread vor Beendigung einen MQDISC-Aufruf ausgeben. Dadurch können Callback-Threads beendet und Systemressourcen freigegeben werden.

Abbildung 6 auf Seite 46Dieser Beispielablauf zeigt eine Einzelthread-Anwendung, die Nachrichten aus zwei Warteschlangen verarbeitet. In diesem Beispiel werden alle Nachrichten dargestellt, die an eine einzelne Funktion zugestellt werden.

Der Unterschied zur asynchronen Verarbeitung besteht darin, dass die Steuerung erst an den Ausgeber des MQCTL-Aufrufs zurückgegeben wird, wenn sich alle Konsumenten selbst deaktiviert haben. Hierfür muss ein Konsument die Anforderung MQCTL STOP ausgeben oder der Warteschlangenmanager muss stillgelegt werden.

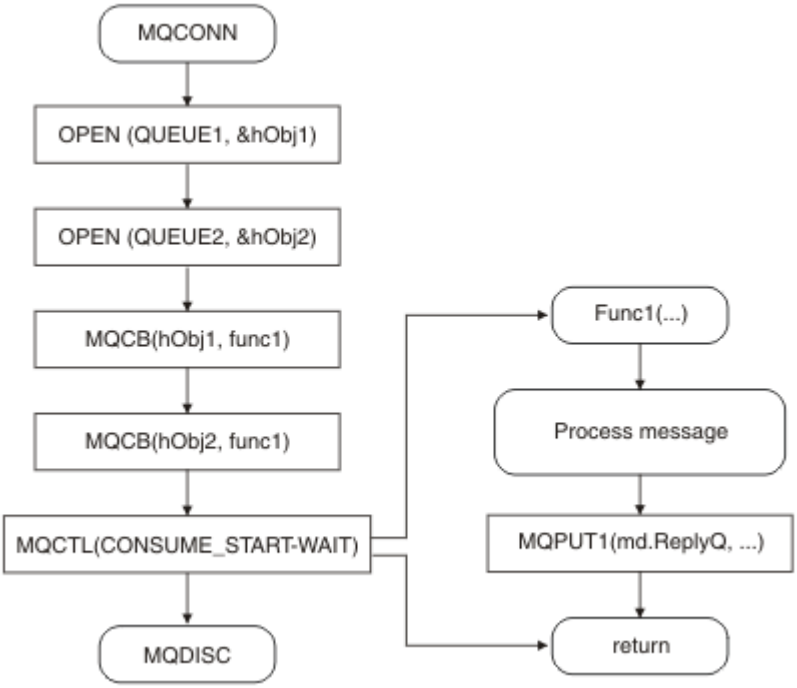


Abbildung 6. Nachrichtengesteuerte Einzelthread-Anwendung, die Nachrichten aus zwei Warteschlangen verarbeitet

Nachrichtengruppen

Nachrichten können innerhalb von Gruppen auftreten, um eine Anordnung der Nachrichten zu ermöglichen.

Mithilfe von Nachrichtengruppen können mehrere Nachrichten als untereinander zugehörig gekennzeichnet werden und Sie können eine logische Reihenfolge auf die Gruppe anwenden (siehe „Logische und physische Reihenfolge" auf Seite 814). Unter Multiplatforms ermöglicht die Nachrichtensegmentierung die Aufteilung großer Nachrichten in kleinere Segmente. Gruppierte oder segmentierte Nachrichten können nicht in ein Thema eingereiht werden.

Die Hierarchie innerhalb einer Gruppe gestaltet sich wie folgt:

Gruppe
Das ist die höchste Stufe der Hierarchie und wird durch eine *GroupId* angegeben. Sie besteht aus mindestens einer Nachricht mit derselben *GroupId*. Diese Nachrichten können standortunabhängig in der Warteschlange gespeichert werden.

Anmerkung: Der Begriff *Nachricht* wird hier als Bezeichnung für ein Element in einer Warteschlange verwendet, wie er durch ein einzelnes MQGET zurückgegeben werden kann, das kein MQGMO_COMPLETE_MSG angibt.

Abbildung 7 auf Seite 47 zeigt eine Gruppe logischer Nachrichten:

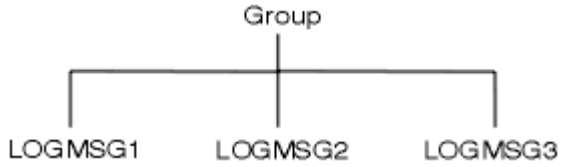


Abbildung 7. Gruppe logischer Nachrichten

Durch Öffnen einer Warteschlange und Angeben von MQOO_BIND_ON_GROUP werden alle an diese Warteschlange gesendeten Nachrichten in eine Gruppe gezwungen, die zur gleichen Instanz der Warteschlange übermittelt wird. Weitere Informationen zur Option BIND_ON_GROUP finden Sie unter Handhabung von Nachrichtenaffinitäten.

Logische Nachricht

Logische Nachrichten innerhalb einer Gruppe werden durch die Felder *GroupId* und *MsgSeqNumber* gekennzeichnet. Die *MsgSeqNumber* beginnt bei 1 für die erste Nachricht einer Gruppe. Auch wenn eine Nachricht keiner Gruppe angehört, ist dieser Wert 1.

Verwenden Sie logische Nachrichten in einer Gruppe für Folgendes:

- Anordnung sicherstellen (wenn dies unter den Umständen, unter denen die Nachricht übermittelt wurde, nicht garantiert ist).
- Anwendungen erlauben, ähnliche Nachrichten zu gruppieren (z. B. die Nachrichten, die von derselben Serverinstanz verarbeitet werden müssen).

Jede Nachricht in einer Gruppe besteht aus einer physischen Nachricht, außer wenn sie in mehrere Segmente aufgeteilt wird. Jede Nachricht ist aus logischer Sicht eine separate Nachricht, und nur die Felder *GroupId* und *MsgSeqNumber* im MQMD geben einen Hinweis darauf, dass die Nachricht in Beziehung zu anderen Nachrichten der Gruppe steht. Andere Felder im MQMD sind unabhängig; einige können für alle Nachrichten in der Gruppe identisch sein, wogegen andere unterschiedlich sein können. Nachrichten können z. B. in einer Gruppe verschiedene Formatnamen, CCSIDs und Codierungen tragen.

Segment

Segmente werden für die Bearbeitung von Nachrichten verwendet, die entweder zum Einreihen oder Abrufen der Anwendung oder für den Warteschlangenmanager zu groß sind (einschließlich eingreifender Warteschlangenmanager, über die die Nachricht übermittelt wird). Weitere Informationen finden Sie unter „Nachrichtensegmentierung” auf Seite 833.

Eine einzelne Nachricht wird in kleinere Nachrichten aufgeteilt, die als *Segmente* bezeichnet werden. Ein Segment einer Nachricht wird durch die Felder *GroupId*, *MsgSeqNumber* und *Offset* gekennzeichnet. Das Feld *Offset* beginnt für das erste Segment einer Nachricht bei Null.

Jedes Segment besteht aus einer physischen Nachricht, die zu einer Gruppe gehören kann (Abbildung 8 auf Seite 48 zeigt ein Beispiel von Nachrichten in einer Gruppe). Ein Segment ist aus logischer Sicht Teil einer einzelnen Nachricht. Daher sollten sich zwischen den einzelnen Segmenten derselben Nachricht nur die MQMD-Felder *MsgId*, *Offset* und *MsgFlags* unterscheiden. Wenn ein Segment nicht eintrifft, wird der Ursachencode MQRC_INCOMPLETE_GROUP oder MQRC_INCOMPLETE_MSG soweit erforderlich ausgegeben.

Abbildung 8 auf Seite 48 zeigt eine Gruppe logischer Nachrichten, von denen einige segmentiert sind:

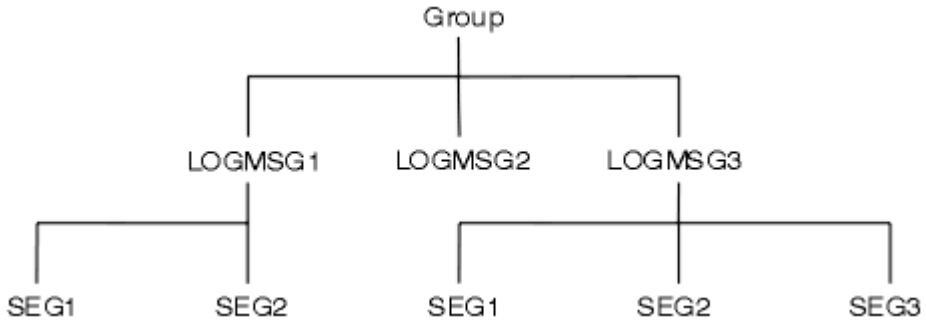


Abbildung 8. Segmentierte Nachrichten

z/OS Segmentierung wird unter IBM MQ for z/OS nicht unterstützt.

Sie können segmentierte oder gruppierte Nachricht mit Publish/Subscribe nicht verwenden.

Zugehörige Konzepte

„Nachrichtensegmentierung” auf Seite 833
In diesem Abschnitt erfahren Sie, wie Nachrichten segmentiert werden. Dieses Feature wird unter IBM MQ for z/OS oder nicht von Anwendungen unterstützt, die IBM MQ classes for JMSverwenden.

Zugehörige Verweise

„Logische und physische Reihenfolge” auf Seite 814
Innerhalb jeder Prioritätsstufe können Nachrichten in Warteschlangen in *physischer* oder *logischer* Reihenfolge auftreten.

MQMD - Nachrichtendeskriptor

Nachrichtenpersistenz

Persistente Nachrichten werden in Protokolle und Warteschlangendatendateien geschrieben. Wird ein Warteschlangenmanager nach einem Fehler erneut gestartet, werden die persistenten Nachrichten nach Bedarf aus den protokollierten Daten wiederhergestellt. Nicht persistente Nachrichten werden beim Stopp eines Warteschlangenmanagers gelöscht, wobei es keine Rolle spielt, ob der Stopp auf einen Bedienerbefehl oder einen Fehler einer Systemkomponente zurückzuführen ist.

z/OS Eine Ausnahme bilden hier nicht persistente Nachrichten in einer Coupling-Facility (CF) unter z/OS. Sie bleiben bestehen, solange die Coupling-Facility verfügbar bleibt.

Wenn Sie den Nachrichtendeskriptor (MQMD) bei der Erstellung einer Nachricht mit den Standardwerten initialisieren, wird die Nachrichtenpersistenz dem Attribut **DefPersistence** der im Befehl MQOPEN angegebenen Warteschlange entnommen. Alternativ können Sie die Nachricht über das Feld *Persistence* der MQMD-Struktur als persistent oder nicht persistent definieren.

Die Verwendung persistenter Nachrichten wirkt sich auf die Leistung der Anwendung aus. Wie stark diese Auswirkungen sind, hängt von den Leistungsmerkmalen des E/A-Subsystems der Maschine sowie von der Verwendung der Synchronisationspunktoptionen auf den einzelnen Plattformen ab:

- Eine persistente Nachricht außerhalb der aktuellen Arbeitseinheit wird bei jedem Einreihen und Abrufen auf Platte geschrieben. Siehe „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901.
- z/OS ALW Auf allen Plattformen mit Ausnahme von IBM i wird eine persistente Nachricht in der aktuellen Arbeitseinheit nur beim Festschreiben der Arbeitseinheit protokolliert, und die Arbeitseinheit kann mehrere Warteschlangenoperationen enthalten.

Nicht persistente Nachrichten können für einen schnellen Nachrichtenaustausch eingesetzt werden. Weitere Informationen zum schnellen Nachrichtenaustausch finden Sie unter Sicherheit von Nachrichten.

Anmerkung: Werden persistente Nachrichten sowohl innerhalb als auch außerhalb einer Arbeitseinheit geschrieben, so kann dies innerhalb Ihrer Anwendungen zu schwerwiegenden Leistungsproblemen füh-

ren. Besonders trifft dies zu, wenn für beide Operationsarten die gleiche Zielwarteschlange verwendet wird.

Nachrichten mit fehlgeschlagener Übermittlung

Wenn ein Warteschlangenmanager eine Nachricht nicht in eine Warteschlange einreihen kann, haben Sie verschiedene Möglichkeiten.

Sie haben folgende Möglichkeiten:

- Versuchen Sie, die Nachricht erneut in die Warteschlange einzureihen.
- Fordern Sie an, dass die Nachricht an den Absender zurückgegeben wird.
- Stellen Sie eine Nachricht in die Warteschlange für nicht zustellbare Nachrichten.

Weitere Informationen finden Sie in „Prozedurale Programmfehler handhaben” auf Seite 1090.

Zurückgesetzte Nachrichten

Beim Verarbeiten von Nachrichten aus einer Warteschlange, die über eine Arbeitseinheit gesteuert werden, kann die Arbeitseinheit aus mindestens einer Nachricht bestehen. Bei einer Zurücksetzung werden die aus der Warteschlange abgerufenen Nachrichten in der Warteschlange wiederhergestellt und können dann in anderen Arbeitseinheiten erneut verarbeitet werden. Tritt bei der Verarbeitung einer bestimmten Nachricht ein Problem auf, wird die Arbeitseinheit erneut zurückgesetzt. Hierbei kann es zu einer Verarbeitungsschleife kommen. Nachrichten, die in eine Warteschlange eingereiht wurden, werden aus der Warteschlange entfernt.

Eine Anwendung kann Nachrichten in einer solchen Schleife durch Überprüfung des MQMD-Felds *BackoutCount* finden. Die Anwendung kann die Situation entweder korrigieren oder eine Warnung an den Bediener ausgeben.

Multi Der Rücksetzungszähler wird auch bei einem Neustart des Warteschlangenmanagers fortgeführt. Änderungen des Attributs **HardenGetBackout** werden ignoriert.

z/OS Bei gemeinsam genutzten Warteschlangen bleibt der Rücksetzungszähler auch bei einem Neustart des Warteschlangenmanagers erhalten. Um bei allen anderen Konfigurationen unter z/OS sicherzustellen, dass der Rücksetzungszähler für private Warteschlangen beim Neustart des Warteschlangenmanagers erhalten bleibt, setzen Sie das Attribut *HardenGetBackout* auf MQQA_BACKOUT_HARDENED; andernfalls wird der Rücksetzungszähler für die einzelnen Nachrichten bei einem Neustart des Warteschlangenmanagers nicht fortgeführt. Bei dieser Attributfestlegung ist jedoch ein zusätzlicher Verarbeitungsaufwand erforderlich.

Weitere Informationen zum Festschreiben und Rücksetzen von Nachrichten finden Sie unter „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901.

Empfangswarteschlange für Antworten und Warteschlangenmanager

Es kann passieren, dass Sie Nachrichten empfangen, die als Antworten auf von Ihnen übermittelte Nachrichten gesendet werden:

- Eine Antwortnachricht als Antwort auf eine Anforderungsnachricht
- Eine Berichtsnachricht über ein nicht erwartetes Ereignis oder einen Fälligkeitstermin
- Eine Berichtsnachricht über eine Bestätigung bei Eingang oder eine Empfangsbestätigung
- Eine Berichtsnachricht über ein PAN- (Benachrichtigung zu positiver Aktion) oder ein NAN-Ereignis (Benachrichtigung zu negativer Aktion)

Geben Sie im Feld *ReplyToQ* der MQMD-Struktur den Namen der Warteschlange an, der sie Antworten und Berichtsnachrichten übermitteln möchten. Geben Sie im Feld *ReplyToQMgr* den Namen des Warteschlangenmanagers an, der Eigner der Empfangswarteschlange für Antworten ist.

Wenn Sie das Feld *ReplyToQMgr* leer lassen, legt der Warteschlangenmanager den Inhalt folgender Nachrichtendeskriptorfelder in der Warteschlange fest:

ReplyToQ
Wenn *ReplyToQ* eine lokale Definition einer fernen Warteschlange ist, wird im Feld *ReplyToQ* der Name der fernen Warteschlange eingetragen; andernfalls wird das Feld nicht geändert.

ReplyToQMgr
Wenn *ReplyToQ* eine lokale Definition einer fernen Warteschlange ist, wird im Feld *ReplyToQMgr* der Name des Warteschlangenmanagers eingetragen, der Eigner der fernen Warteschlange ist; andernfalls wird im Feld *ReplyToQMgr* der Name des Warteschlangenmanagers eingetragen, mit dem Ihre Anwendung verbunden ist.

Anmerkung: Sie können anfordern, dass ein Warteschlangenmanager mehr als einmal versucht, eine Nachricht zuzustellen, und Sie können anfordern, dass die Nachricht bei einem Fehlschlag gelöscht wird. Wird die Nachricht nach fehlgeschlagener Übermittlung nicht gelöscht, reiht sie der ferne Warteschlangenmanager in ihre Warteschlange für nicht zustellbare Nachrichten ein (siehe „Warteschlange für nicht zustellbare Nachrichten verwenden” auf Seite 1094).

Nachrichtenkontext

Nachrichtenkontext-Informationen ermöglichen der Anwendung, die die Nachricht abruft, Informationen über den Absender der Nachricht zu erhalten.

Die abrufende Anwendung kann Folgendes wollen:

- Prüfen Sie, ob die sendende Anwendung über die korrekte Berechtigungsstufe verfügt
- Führen Sie einige Abrechnungsfunktionen aus, damit die sendende Anwendung für alle auszuführenden Arbeiten geladen werden kann
- Führen Sie ein Auditprotokoll aller Nachrichten, mit denen sie gearbeitet hat

Wenn Sie den MQPUT- oder MQPUT1-Aufruf verwenden, um eine Nachricht in eine Warteschlange einzureihen, können Sie angeben, dass der Warteschlangenmanager dem Nachrichtendeskriptor einige Standard-Kontextinformationen hinzufügen soll. Anwendungen, die über die entsprechende Berechtigungsstufe verfügen, können zusätzliche Kontextinformationen enthalten. Weitere Informationen zur Angabe von Kontextinformationen finden Sie unter „Nachrichtenkontextinformationen steuern” auf Seite 799.

Der Benutzerkontext wird vom Warteschlangenmanager beim Erstellen folgender Berichtsnachrichtenarten verwendet:

- Bestätigung bei Zustellung
- Verfall

Bei der Erstellung dieser Berichtsnachrichten wird der Benutzerkontext auf +put- und +passid-Berechtigungen für das Berichtsziel überprüft. Dort, wo der Benutzerkontext eine unzureichende Berechtigung aufweist, wird die Berichtsnachricht in der Warteschlange für nicht zustellbare Nachrichten eingegliedert, sofern eine definiert wurde. Dort, wo keine Warteschlange für nicht zustellbare Nachrichten vorhanden ist, wird die Berichtsnachricht gelöscht.

Alle Kontextinformationen werden in den Kontextfeldern des Nachrichtendeskriptors gespeichert. Der Informationstyp gehört zu Identität-, Ursprungs- und Benutzerkontextinformaionen

Identitätskontext

Identitätskontext-Informationen geben den Benutzer der Anwendung an, der die Nachricht zuerst in einer Warteschlange einreiht. Entsprechend berechtigte Anwendungen können folgende Felder festlegen:

- Der Warteschlangenmanager gibt im Feld *UserIdentifier* einen Namen an, der den Benutzer identifiziert; wie diese Identifizierung erfolgt, ist von der Umgebung abhängig, in der die Anwendung ausgeführt wird.
- Der Warteschlangenmanager trägt im Feld *AccountingToken* ein Token oder eine Zahl ein, das bzw. die aus der Anwendung ermittelt wurde, von der die Nachricht eingereiht wurde.

- Im Feld *ApplIdentityData* können Anwendungen zusätzliche Informationen zum Benutzer eintragen (beispielsweise ein verschlüsseltes Kennwort).

Eine Windows -Systemsicherheitskennung (SID) wird im Feld *AccountingToken* gespeichert, wenn eine Nachricht unter IBM MQ for Windowserstellt wird. Mit der SID können Sie das Feld *UserIdentifier* ergänzen und die Berechtigungsnachweise eines Benutzers einrichten.

Weitere Informationen zu den Angaben des Warteschlangenmanagers in den Feldern *UserIdentifier* und *AccountingToken* finden Sie in den Beschreibungen dieser Felder in den Themen UserIdentifier und AccountingToken.

Anwendungen, die Nachrichten von einem Warteschlangenmanager zum anderen übergeben, sollten außerdem die Identitätskontextinformationen weitergeben, damit andere Anwendungen die Identität des Absenders der Nachricht kennen.

Ursprungskontext

Ursprungskontext-Informationen beschreiben die Anwendung, die die Nachricht in die Warteschlange eingereiht hat, in der sich die Nachricht derzeit befindet. Der Nachrichtendeskriptor enthält folgende Felder für die ursprünglichen Kontextinformationen:

- *PutApplType* definiert den Typ der Anwendung, die die Nachricht eingereiht hat (z. B. eine CICS-Transaktion).
- *PutApplName* definiert den Namen der Anwendung, die die Nachricht eingereiht hat (z. B. den Namen eines Jobs oder einer Transaktion).
- *PutDate* definiert das Datum, an dem die Nachricht in die Warteschlange eingereiht wurde.
- *PutTime* definiert die Uhrzeit, zu der die Nachricht in die Warteschlange eingereiht wurde.
- *ApplOriginData* definiert zusätzliche Informationen, die eine Anwendung über den Ursprung der Nachricht hinzufügt. Beispielsweise könnte bei entsprechend berechtigten Anwendungen einstellt sein, ob der Identität der Daten vertraut wird.

Ursprungskontextinformationen werden in der Regel vom Warteschlangenmanager ausgegeben. Die Werte in den Feldern *PutDate* und *PutTime* werden in Greenwich Mean Time (GMT) angegeben. Informationen zu diesen Feldern finden Sie in den Abschnitten PutDate und PutTime.

Eine Anwendung mit entsprechender Berechtigung kann ihren eigenen Kontext bereitstellen. Dadurch können Abrechnungsdaten erhalten werden, wenn ein einzelner Benutzer in jedem System, das eine Nachricht bearbeitet, die es ausgegeben hat, über eine unterschiedliche Benutzer-ID verfügt.

IBM MQ-Objekte

Hier finden Sie ausführliche Informationen zu IBM MQ-Objekten, darunter Warteschlangenmanager, Gruppen mit gemeinsamer Warteschlange, Warteschlangen, Topic-Verwaltungsobjekte, Namenslisten, Prozessdefinitionen, Authentifizierungsdatenobjekte, Kanäle, Speicherklassen, Listener und Services.

Warteschlangenmanager definieren die Eigenschaften (die als 'Attribute' bezeichnet werden) dieser Objekte. Die Werte dieser Attribute wirken sich auf die Verarbeitung der Objekte durch IBM MQ aus. In den Anwendungen werden diese Objekte über die MQI (Message Queue Interface) gesteuert. Programme verweisen auf Objekte anhand eines *Objektdeskriptors* (MQOD).

Wenn Sie mithilfe von IBM MQ-Befehlen Objekte definieren, ändern oder löschen, prüft der Warteschlangenmanager, ob Sie über die erforderliche Berechtigungsstufe zum Ausführen dieser Operationen 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.

Zugehörige Konzepte

„Nachrichtenkontextinformationen steuern” auf Seite 799
Wenn Sie den MQPUT- oder MQPUT1-Aufruf verwenden, um eine Nachricht in eine Warteschlange einzureihen, können Sie angeben, dass der Warteschlangenmanager dem Nachrichtendeskriptor einige

Standard-Kontextinformationen hinzufügen soll. Anwendungen, die über die entsprechende Berechtigungsstufe verfügen, können zusätzliche Kontextinformationen enthalten. Über das Optionenfeld in der MQPMO-Struktur können Sie die Kontextinformationen steuern.

Zugehörige Verweise

„MQOPEN-Optionen für den Nachrichtenkontext" auf Seite 789
Wenn Sie beim Einreihen einer Nachricht in eine Warteschlange die Möglichkeit haben möchten, der Nachricht Kontextinformationen zuzuordnen, müssen Sie beim Öffnen der Warteschlange eine der Nachrichtenkontextoptionen verwenden.

Windows Microsoft Transaction Server-Anwendungen vorbereiten und ausführen

Befolgen Sie die nachstehenden Anweisungen für Ihre Umgebung, um eine MTS-Anwendung für die Ausführung als IBM MQ MQI client-Anwendung vorzubereiten.

Allgemeine Informationen zur Entwicklung von Microsoft Transaction Server (MTS)-Anwendungen, die auf IBM MQ-Ressourcen zugreifen, finden Sie im Kapitel zu MTS im IBM MQ Help Center.

Befolgen Sie die eine der nachstehenden Anweisungen für jede Komponente der Anwendung, um eine MTS-Anwendung für die Ausführung als IBM MQ MQI client-Anwendung vorzubereiten

- Verwendet die Komponente Bindungen der Programmiersprache C für die MQI, müssen Sie entsprechend den Anweisungen im Abschnitt „C-Programme in Windows vorbereiten" auf Seite 1069 vorgehen, die Komponente jedoch mit der Bibliothek 'mqicxa.lib' verbinden, nicht mit 'mqic.lib'.
- Verwendet die Komponente IBM MQ C++-Klassen, befolgen Sie die Anweisungen im Abschnitt „C++-Programme unter Windows erstellen" auf Seite 578 und verbinden Sie die Komponente mit der Bibliothek 'imqx23vn.lib', nicht mit 'imqc23vn.lib'.
- Verwendet die Komponente die Bindungen von Visual Basic für das MQI, müssen Sie entsprechend den Anweisungen im Abschnitt „Visual Basic-Programme in Windows vorbereiten" auf Seite 1073 vorgehen, bei der Definition des Visual Basic-Projekts im Feld **Argumente für bedingte Kompilierung** jedoch `MqType=3` eingeben.

Konzepte für den Entwurf von IBM MQ-Anwendungen

Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

Beim Entwerfen einer IBM MQ-Anwendungen sind folgende Fragen und Optionen zu berücksichtigen:

Anwendungstyp

Was ist der Zweck Ihrer Anwendung? Informationen zu den verschiedenen Arten von Anwendungen, die Sie entwickeln können, finden Sie unter den folgenden Links:

- Server
- Client
- Publish/Subscribe
- Web-Services
- Benutzerexits, API-Exits und installierbare Services

Darüber hinaus können Sie auch eigene Anwendungen für eine automatisierte Verwaltung von IBM MQ schreiben. Weitere Informationen finden Sie in den Abschnitten IBM MQ Administration Interface (MQAI) und Verwaltungsaufgaben automatisieren.

Programmiersprache

IBM MQ unterstützt eine Reihe verschiedener Programmiersprachen für das Schreiben von Anwendungen. Weitere Informationen hierzu finden Sie unter „Anwendungen für IBM MQ entwickeln" auf Seite 5.

Anwendungen für mehrere Plattformen

Soll die Anwendung auf mehreren Plattformen ausgeführt werden? Ist für die Zukunft ein Wechsel auf eine andere Plattform geplant? Lautet die Antwort auf eine dieser Fragen 'Ja', müssen die Programme für Plattformunabhängigkeit codiert werden.

Wenn Sie beispielsweise C verwenden, codieren Sie in ANSI-Standard-C. Verwenden Sie statt einer plattformspezifischen Funktion eine Standardfunktion der C-Bibliothek, selbst wenn die plattformspezifische Funktion schneller oder effizienter ist. Ausgenommen sind Fälle, wenn die Effizienz im Code Vorrang hat und Sie für beide Situationen mittels #ifdef codieren sollten. For example:

```
#ifdef _AIX
    AIX specific code
#else
    generic code
#endif
```

Warteschlangentypen

Möchten Sie immer, wenn Sie eine Warteschlange benötigen, eine erstellen, oder sollen bereits eingerichtete Warteschlangen verwendet werden? Sollen Warteschlangen nach der Verwendung gelöscht oder später erneut verwendet werden? Sollen für die Anwendungsunabhängigkeit Aliaswarteschlangen verwendet werden? Unter Warteschlangen erfahren Sie, welche Arten von Warteschlangen unterstützt werden.

z/OS **Gemeinsam genutzte Warteschlangen, Gruppen mit gemeinsamer Warteschlange und Gruppencluster mit gemeinsamer Warteschlange (nur IBM MQ for z/OS) verwenden**

Vielleicht möchten Sie die Vorzüge erhöhter Verfügbarkeit und Skalierbarkeit sowie eines besseren Lastausgleichs nutzen, die bei Verwendung von gemeinsam genutzten Warteschlangen mit Gruppen mit gemeinsamer Warteschlange möglich sind. Weitere Informationen hierzu finden Sie im Abschnitt Gemeinsam genutzte Warteschlangen und Gruppen mit gemeinsamer Warteschlange.

Vielleicht möchten Sie auch Durchschnitt und Spitzenwert der Nachrichtenflüsse schätzen und überlegen, die Arbeitslast durch den Einsatz von Clustern mit Gruppen mit gemeinsamer Warteschlange zu verteilen. Weitere Informationen hierzu finden Sie im Abschnitt Gemeinsam genutzte Warteschlangen und Gruppen mit gemeinsamer Warteschlange.

Warteschlangenmanagercluster verwenden

Vielleicht möchten Sie die Vorzüge vereinfachter Systemverwaltung sowie erhöhter Verfügbarkeit und Skalierbarkeit und besseren Lastausgleichs nutzen, die bei Verwendung von Clustern möglich sind.

Arten von Nachrichten

Vielleicht möchten Sie für einfache Nachrichten Datagramme verwenden, für andere Situationen jedoch Anforderungsnachrichten (zu denen Antworten erwartet werden) einsetzen. Möglicherweise möchten Sie einigen Ihrer Nachrichten unterschiedliche Prioritäten zuordnen. Weitere Informationen zur Nachrichtenkonzeption finden Sie im Abschnitt „Verfahren zum Entwerfen von Nachrichten” auf Seite 62.

Einsatz von Publish/Subscribe oder Punkt-zu-Punkt-Messaging

Mittels Publish/Subscribe-Messaging sendet eine Anwendung die zur gemeinsamen Nutzung gedachten Informationen in einer IBM MQ-Nachricht an ein Standardziel, das von IBM MQ publish?subscribe verwaltet wird. IBM MQ übernimmt dann die Verteilung dieser Informationen. Die Zielanwendung muss nichts über die Quelle der empfangenen Informationen wissen, sie meldet lediglich Interesse an einem oder mehreren Themen an und erhält die betreffenden Informationen bei Verfügbarkeit. Weitere Informationen zum Publish/Subscribe-Messaging finden Sie unter Publish/Subscribe-Messaging.

Bei Verwendung des Punkt-zu-Punkt-Messaging sendet eine Absenderanwendung eine Nachricht an eine bestimmte Warteschlange, aus der sie erwartungsgemäß von einer empfangenden Anwendung abgerufen wird. Die empfangende Anwendung ruft Nachrichten aus einer bestimmten Warteschlange ab und prüft deren Inhalte. Anwendungen funktionieren oft sowohl als Absender als auch als Empfänger, sie senden also an eine andere Anwendung eine Abfrage und erhalten von dort eine Antwort.

IBM MQ-Programme steuern

Möglicherweise möchten Sie einige Programme automatisch starten oder Programme warten lassen, bis eine bestimmte Nachricht in einer Warteschlange eintrifft (mithilfe der IBM MQ *Triggering* -Funktion, siehe „IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913). Alternativ können Sie eine weitere Instanz einer Anwendung starten, wenn die Nachrichten in einer Warteschlange nicht schnell genug verarbeitet werden (mit der Funktion IBM MQ *Instrumentierungsereignisse* , wie unter Instrumentierungsereignisse beschrieben).

Anwendungen auf einem IBM MQ-Client ausführen

Die Clientumgebung unterstützt die vollständige MQI und nahezu jede in einer prozeduralen Programmiersprache geschriebeneIBM MQ-Anwendung kann zur Ausführung auf einem IBM MQ MQI client erneut verbunden werden. Verbinden Sie die Anwendung auf dem IBM MQ MQI client mit der MQIC-Bibliothek statt mit der MQI-Bibliothek. z/OS 'Get(signal)' wird unter z/OS nicht unterstützt.

Anmerkung: Wird eine Anwendung auf einem IBM MQ-Client ausgeführt, kann sie Verbindungen zu mehreren Warteschlangenmanagern gleichzeitig herstellen oder den Namen eines Warteschlangenmanagers mit einem Stern (*) in einem MQCONN- oder MQCONNX-Aufruf verwenden. Ändern Sie die Anwendung, wenn Sie eine Verbindung zu den Warteschlangenmanagerbibliotheken anstatt zu den Clientbibliotheken herstellen möchten, da diese Funktion nicht verfügbar sein wird.

Weitere Informationen finden Sie unter „Anwendungen in der IBM MQ MQI client-Umgebung ausführen” auf Seite 970.

Anwendungsleistung

Entwurfsentscheidungen können sich auf die Anwendungsleistung auswirken. Vorschläge zur Leistungsoptimierung von IBM MQ-Anwendungen finden Sie unter „Anwendungsdesign und Leistung” auf Seite 63 IBM i und „Entwurfs- und Leistungsaspekte für Anwendungen unter IBM i” auf Seite 67 .

Erweiterte IBM MQ-Verfahren

Für anspruchsvollere Anwendungen empfiehlt sich unter Umständen die Nutzung erweiterter IBM MQ-Verfahren, z. B. das Korrelieren von Antworten sowie das Generieren und Senden von IBM MQ-Kontextinformationen. Weitere Informationen finden Sie in „Verfahren zum Entwerfen von anspruchsvolleren Anwendungen” auf Seite 66.

Datensicherung und Verwaltung der Datenintegrität

Anhand der mit einer Nachricht übergebenen Kontextinformationen können Sie prüfen, ob die Nachricht von einer zulässigen Quelle gesendet wurde. Mit den Synchronisationspunktfunktionen von IBM MQ oder Ihres Betriebssystems können Sie sicherstellen, dass Ihre Daten weiterhin mit anderen Ressourcen übereinstimmen (ausführliche Informationen hierzu finden Sie unter „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901). Mit der *Persistenz*-Funktion von IBM MQ-Nachrichten stellen Sie die Zustellung wichtiger Nachrichten sicher.

IBM MQ-Anwendungen testen

Die Anwendungsentwicklungsumgebung für IBM MQ-Programme unterscheidet sich nicht von denen für andere Anwendungen. Sie können also dieselben Entwicklungstools und auch IBM MQ-Tracefunktionen nutzen.

z/OS Beim Testen von CICS-Anwendungen mit IBM MQ for z/OS können Sie die CICS Execution Diagnostic Facility (CEDF) verwenden. CEDF fängt Eingang und Ausgang jedes MQI-Aufrufs sowie von Aufrufen aller CICS-Dienste ab. Zudem können Sie in der CICS-Umgebung ein API-übergreifendes Exitprogramm schreiben, um Diagnoseinformationen vor und nach jedem MQI-Aufruf bereitzustellen. Weitere Informationen hierzu finden Sie im Abschnitt „Using and writing applications on IBM MQ for z/OS” auf Seite 938.

IBM i Beim Testen von IBM i-Anwendungen können Sie den Standard-Debugger verwenden. Dieser wird mit dem Befehl STRDBG gestartet.

Behandlung von Ausnahmen und Fehlern
Sie müssen sich überlegen, wie nicht zustellbare Nachrichten verarbeitet und vom Warteschlangenmanager gemeldete Fehlersituationen behoben werden sollen. Für einige Berichte müssen in MQPUT Berichtsoptionen definiert werden.

Zugehörige Konzepte
IBM MQ - Technische Übersicht
„Design and performance considerations for z/OS applications" auf Seite 69
Application design is one of the most important factors affecting performance. Use this topic to understand some of the design factors involved in performance.

„Anwendungen für IBM MQ entwickeln" auf Seite 5
Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

„Konzepte für die Anwendungsentwicklung" auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

„Prozedurale Anwendungen für die Warteschlangensteuerung erstellen" auf Seite 758
Dieser Abschnitt enthält Informationen zum Erstellen von Anwendungen für die Warteschlangensteuerung, zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager und zum Öffnen und Schließen von Objekten.

„Prozedurale Clientanwendungen schreiben" auf Seite 961
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„C++-Anwendungen entwickeln" auf Seite 554
In IBM MQ werden C++-Klassen bereitgestellt, die IBM MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen.

„IBM MQ classes for JMS/Jakarta Messaging verwenden" auf Seite 85
IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Java -Messaging-Provider, die mit IBM MQbereitgestellt werden. Neben der Implementierung der in den Spezifikationen JMS und Jakarta Messaging definierten Schnittstellen fügen diese Messaging-Provider zwei Gruppen von Erweiterungen zur Java -Messaging-API hinzu.

„IBM MQ classes for Java verwenden" auf Seite 364
Verwenden Sie 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.

Zugehörige Tasks
„.NET-Anwendungen entwickeln" auf Seite 582
IBM MQ classes for .NET ermöglichen .NET -Anwendungen, eine Verbindung zu IBM MQ als IBM MQ MQI client oder direkt zu einem IBM MQ -Server herzustellen.

Anwendungsname in unterstützten Programmiersprachen angeben

Bereits vor IBM MQ 9.2.0 konnten Sie einen Anwendungsnamen für Java- oder JMS-Clientanwendungen angeben. Ab IBM MQ 9.2.0 wird diese Funktion in IBM MQ for Multiplatforms auf andere Programmiersprachen erweitert.

Verwendung des Anwendungsnamens

Der Anwendungsname wird ausgegeben von:

- runmqsc DISPLAY CONN APPLTAG
- runmqsc DISPLAY QSTATUS TYPE(HANDLE) APPLTAG

- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- Anwendungsaktivitätstrace

Der Anwendungsname wird auch bei der Konfiguration des Anwendungsaktivitätstrace verwendet. Der Standardanwendungsname für Nicht-Java -Anwendungen ist der abgeschnittene Name der ausführbaren Datei, außer unter Windows und IBM i.

Windows Unter Windows handelt es sich bei dem Standardnamen um den vollständig qualifizierten Namen der ausführbaren Datei, der links auf 28 Zeichen abgeschnitten wird.

IBM i Unter IBM i handelt es sich bei dem Standardnamen um den Jobnamen.

Bei Java-Anwendungen ist es der Klassenname mit dem Paketnamen als Präfix, links auf 28 Zeichen abgeschnitten.

Weitere Informationen finden Sie unter PutApplName.

Anwendungen unter IBM MQ for Multiplatforms können ihre Anwendungsnamen entweder administrativ oder mithilfe verschiedener Programmiermethoden festlegen. Dies ermöglicht es Anwendungen, einen aussagekräftigeren und plattformunabhängigen Namen bereitstellen, den Sie bei der Konfiguration eines Anwendungsaktivitätstrace oder bei Ausgaben von verschiedenen **runmqsc**-Befehlen verwenden können.

Sie können Anwendungen in einem einheitlichen Cluster neu verteilen. Um dies zu erreichen, werden aussagekräftige Anwendungsnamen verwendet.

Unterstützte Zeichen

Weitere Informationen zur Angabe des Anwendungsnamens finden Sie im Abschnitt „Empfohlene Zeichen für den Anwendungsnamen” auf Seite 57.

Programmiersprachen

Weitere Informationen dazu, wie Anwendungen, die in die IBM MQ -Bibliotheken in C und anderen Programmiersprachen aufgelöst werden, den Anwendungsnamen bereitstellen können, finden Sie unter „Programmiersprachenverbindungen” auf Seite 59 .

Verwaltete .NET-Anwendungen

Im Abschnitt „Verwaltete .NET-Anwendungen” auf Seite 60 finden Sie Informationen dazu, wie verwaltete .NET-Anwendungen den Anwendungsnamen bereitstellen können.

XMS-Anwendungen

Im Abschnitt „XMS-Anwendungen” auf Seite 61 finden Sie Informationen dazu, wie XMS-Anwendungen den Anwendungsnamen bereitstellen können.

Java- und JMS-Bindings-Anwendungen

ALW

Im Abschnitt „Java- und JMS-Bindings-Anwendungen” auf Seite 61 finden Sie Informationen dazu, wie Java- und JMS-Anwendungen den Anwendungsnamen bereitstellen können.

Zugehörige Konzepte

Anwendungsaktivitätstrace

Informationen zu Uniform-Clustern

Zugehörige Verweise
MQCNO
MQCNO unter IBM i

Anwendungsnamen in unterstützten Programmiersprachen verwenden

In diesem Abschnitt erfahren Sie, wie der Anwendungsname in den verschiedenen von IBM MQ unterstützten Sprachen ausgewählt wird.

Empfohlene Zeichen für den Anwendungsnamen

Anwendungsnamen müssen dem Zeichensatz entsprechen, der durch das Attribut **CodedCharSetId** des Warteschlangenmanagerfelds angegeben wird. Weitere Informationen hierzu finden Sie unter Attribute für den Warteschlangenmanager.

Wird die Anwendung jedoch als IBM MQ MQI client ausgeführt, muss der Anwendungsname dem Zeichensatz und der Codierung des Clients entsprechen.

Um einen reibungslosen Übergang des Anwendungsnamens zwischen Warteschlangenmanagern sicherzustellen und die Anwendungsressourcenüberwachung über die Ressourcenüberwachungsthemen zu ermöglichen, sollten Anwendungsnamen nur druckbare Einzelbytezeichen enthalten.

Anmerkungen:

- Außerdem sollten Sie keinen Schrägstrich und keine Et-Zeichen in Anwendungsnamen verwenden.
- Sie sollten in Anwendungsnamen kein Et-Zeichen verwenden. Metriken des Systemthemas STATAPP für Anwendungsnamen, die ein Et-Zeichen enthalten, werden nicht erstellt.

Dadurch wird der Name begrenzt auf:

- Alphanumerische Zeichen: `A-Z`, `a-z` und `0-9`

  Anmerkung: Auf Systemen, die EBCDIC Katakana verwenden, sollten Sie keine Kleinbuchstaben (`a-z`) in Anwendungsnamen verwenden.
- Leerzeichen
- Unveränderliche druckbare Zeichen in EBCDIC: `+ < = > % * ' ( ) , _ - . : ; ?`
- Das Zeichen '/'. Wenn Sie einen Aktivitätstrace oder Metriken des Systemthemas STATAPP für eine Anwendung subskribieren, deren Name einen Schrägstrich enthält, müssen Sie jeden Schrägstrich durch ein Et-Zeichen ersetzen. Um beispielsweise die Metriken von STATAPP für eine Anwendung namens DEPT1/APPS/STOCKQUOTE zu erhalten, müssen Sie die Themenzeichenfolge $SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE subskribieren. Die Beispielanwendungen 'amqsact' und 'amqsrua' konvertieren bei der Erstellung ihrer Subskriptionen automatisch Schrägstriche in Et-Zeichen.

So legen Sie die Zeichen fest

In der folgenden Tabelle ist zusammenfassend dargestellt, wie der Anwendungsname in den verschiedenen von IBM MQ unterstützten Sprachen ausgewählt wird. Die Methoden für die Auswahl des Namens sind der Reihenfolge nach aufgeführt, die Methode mit höchster Priorität steht dabei an erster Stelle.

	C-Bindings und -Client	Java-Bindings und -Client	JMS-Bindings und -Client	Verwalteter .NET-Client	Nicht verwaltete .NET-Bindings und -Client	Verwalteter XMS-Client	Nicht verwaltete .XMS-Bindings und -Client
Überschreibung der Verbindungseigenschaft		Java-Verbindungseigenschaft überschreiben		Überschreiben der .NET-Verbindungseigenschaft	Überschreiben der .NET-Verbindungseigenschaft		
Überschriebene Eigenschaft		Java überschriebene Eigenschaft		Überschriebene .NET-Eigenschaft	Überschriebene .NET-Eigenschaft		
MQ-Umgebung		Java MQEnvironment		.NET-MQEnvironment	.NET-MQEnvironment		
Eigenschaft für Verbindungsfactory			Eigenschaft für Verbindungsfactory			Eigenschaft für Verbindungsfactory	Eigenschaft für Verbindungsfactory
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	Verbindungsoptionen						
Umgebungsvariable	Umgebungsvariablen				Umgebungsvariablen		Umgebungsvariablen
mqclient.ini (Nur für Clientverbindungen anwendbar)	Clientverbindungen				Clientverbindungen		Clientverbindungen
Java-Klassenname		Java-Klassenname	Java-Klassenname				

	C-Bindings und -Client	Java-Bindings und -Client	JMS-Bindings und -Client	Verwalteter .NET-Client	Nicht verwaltete .NET-Bindings und -Client	Verwalteter XMS-Client	Nicht verwaltete .XMS-Bindings und -Client
Standardname	Standardname			.NET-Standardname	.NET-Standardname	.NET-Standardname	.NET-Standardname

Anmerkung: Die Spalte 'C-Bindings und -Client' gilt auch für die folgenden Programmiersprachen:

- COBOL
- Assembler
- Visual Basic
- RPG

Programmiersprachenverbindungen

Anwendungen, die in die IBM MQ-Bibliotheken in C (und anderen Programmiersprachen) aufgelöst werden, können den Anwendungsnamen auf folgende Arten angeben:

Die Verbindungsmethoden sind der Reihenfolge nach aufgeführt, beginnend mit der Methode mit der höchsten Priorität.

Multi **Verbindungsoptionen**

- ALW MQCNO

  Anmerkung: z/OS Wenn Sie eine Verbindung zu einem IBM MQ for z/OS -Warteschlangenmanager herstellen, können Sie den Anwendungsnamen nur über Clientmodusverbindungen oder über IBM MQ classes for JMS -oder IBM MQ classes for Java -Anwendungen festlegen.
- IBM i MQCNO unter IBM i

ALW **Umgebungsvariablen**

Wenn Sie noch keinen Anwendungsnamen ausgewählt haben, können Sie mit der Umgebungsvariablen **MQAPPLNAME** die Verbindung zum Warteschlangenmanager angeben. For example:

```
export MQAPPLNAME=ExampleApplName
```

Beachten Sie, dass nur die ersten 28 Zeichen verwendet werden und diese nicht nur aus Leerzeichen oder Nullen bestehen dürfen.

Anmerkung: Das Attribut gilt nur für die unterstützten Programmiersprachen, nicht verwaltete .NETund nicht verwaltete XMS -Verbindungen.

ALW **Clientkonfigurationsdatei**

Wenn Sie noch keinen Anwendungsnamen ausgewählt haben und es sich bei der Verbindung um eine Clientverbindung handelt, können Sie die folgenden Informationen in der Clientkonfigurationsdatei (z. B. `mqclient.ini`) angeben, um die Verbindung zum Warteschlangenmanager anzugeben.

```
Connection:
    ApplName=ExampleApplName
```

Anmerkungen:

1. Es werden nur die ersten 28 Zeichen verwendet und diese dürfen nicht nur aus Leerzeichen oder Nullen bestehen.
2. Das Attribut gilt nur für Clientverbindungen in den unterstützten Programmiersprachen, nicht verwaltete .NETund nicht verwaltete XMS -Verbindungen.

Weitere Informationen finden Sie unter IBM MQ MQI client -Konfigurationsdatei `mqclient.ini`.

Standardname

Wenn Sie den Anwendungsnamen immer noch nicht ausgewählt haben, wird weiterhin der Standardname verwendet. Dieser besteht aus dem Pfad und dem Namen der ausführbaren Datei, soweit diese vom Betriebssystem angezeigt werden. Weitere Informationen finden Sie unter PutApplName.

Verwaltete .NET-Anwendungen

Verwaltete .NET-Anwendungen können den Anwendungsnamen auf folgende Arten angeben:

Die Verbindungsmethoden sind der Reihenfolge nach aufgeführt, beginnend mit der Methode mit der höchsten Priorität.

Überschreibung der Verbindungseigenschaft

Sie können wie folgt eine Datei zur Überschreibung von Verbindungsdetails für Anwendungen bereitstellen:

```
<appSettings>
     <add key="overrideConnectionDetails" value="true" />
     <add key="overrideConnectionDetailsFile" value="<location>" />
</appSettings>
```

Die durch `overrideConnectionDetailsFile` angegebene Datei enthält eine Liste von Eigenschaften mit dem Präfix `mqj`. Anwendungen müssen die Eigenschaft **`mqj.APPNAME`** definieren, wobei der Wert der Eigenschaft **`mqj.APPNAME`** den Namen angibt, der zur Angabe der Verbindung zum Warteschlangenmanager verwendet wird.

Es werden nur die ersten 28 Zeichen des Namens verwendet. For example:

```
mqj.APPNAME=ExampleApplName
```

Überschriebene Eigenschaft

Es wurde eine konstante Eigenschaft **`MQC.APPNAME_PROPERTY`** mit dem Wert *APPNAME* definiert. Sie können nun diese Eigenschaft an den **`MQQueueManager`**-Konstruktor übergeben, wobei nur die ersten 28 Zeichen des Namens verwendet werden. For example:

```
Hashtable properties = new Hashtable();
properties.Add( MQC.APPNAME_PROPERTY, "ExampleApplName" );
MQQueueManager qMgr = new MQQueueManager("qmgrname", properties);
```

Weitere Informationen finden Sie unter „Verwaltete und nicht verwaltete Operationen in .NET” auf Seite 665.

MQEnvironment

Die Eigenschaft *AppName* wird der Klasse **MQEnvironment**hinzugefügt und die ersten 28 Zeichen werden nur verwendet. For example:

```
MQEnvironment.AppName = "ExampleApplName";
```

Standardname

Wenn Sie den Anwendungsnamen nicht mit einer der im vorherigen Text beschriebenen Methoden angegeben haben, wird der Anwendungsname automatisch auf den Namen der ausführbaren Datei (und den Pfad, der passt) gesetzt.

XMS-Anwendungen

Die Verbindungsmethoden sind der Reihenfolge nach aufgeführt, beginnend mit der Methode mit der höchsten Priorität.

Eigenschaft für Verbindungsfactory

XMS -Anwendungen können den Anwendungsnamen in der Verbindungsfactory über die **XMSC.WMQ_APPLICATIONNAME** -Eigenschaft ("*XMSC_WMQ_APPNAME*") angeben auf ähnliche Weise wie JMS. Sie können bis zu 28 Zeichen angeben.

Weitere Informationen hierzu finden Sie unter „XMS .NET - Erstellung von verwalteten Objekten” auf Seite 695 und „Eigenschaften einer XMS-Nachricht” auf Seite 703.

JMSAdmin

In den Verwaltungstools wird die Eigenschaft als "**APPLICATIONNAME**" oder "**APPNAME**" bezeichnet.

Java- und JMS-Bindings-Anwendungen

Die Verbindungsmethoden sind der Reihenfolge nach aufgeführt, beginnend mit der Methode mit der höchsten Priorität.

ALW Java- und JMS-Clientanwendungen können bereits einen Anwendungsnamen angeben und in IBM MQ for Multiplatforms wurde dies unter Verwendung des MQCNO-Feldes **ApplName** auf Bindings-Anwendungen erweitert.

Überschreibung der Verbindungseigenschaft

Die Eigenschaft **Application name** wurde zur Liste der Verbindungseigenschaften hinzugefügt, die Sie überschreiben können. Weitere Informationen hierzu finden Sie im Abschnitt IBM MQ -Verbindungseigenschaft überschreiben.

Achtung: Die Verbindungseigenschaften und die Methode, die Datei für das Überschreiben von Verbindungseigenschaften zu verwenden, sind für IBM MQ classes for Java und .NET identisch.

Überschriebene Eigenschaft

Es wurde eine konstante Eigenschaft **MQC.APPNAME_PROPERTY** mit dem Wert *APPNAME* definiert. Sie können nun diese Eigenschaft an den **MQQueueManager**-Konstruktor übergeben, wobei nur die ersten 28 Zeichen des Namens verwendet werden. Weitere Informationen finden Sie im Abschnitt Überschreiben von Verbindungseigenschaften in IBM MQ classes for Javaverwenden.

MQEnvironment

Die Eigenschaft *AppName* wird der Klasse **MQEnvironment** hinzugefügt und die ersten 28 Zeichen werden nur verwendet.

Weitere Informationen finden Sie unter „IBM MQ-Umgebung für IBM MQ classes for Java einrichten” auf Seite 393.

Java-Klassenname

Wenn Sie den Anwendungsnamen noch nicht unter Verwendung einer der bisher beschriebenen Methoden angegeben haben, wird er aus dem Hauptklassennamen abgeleitet.

Weitere Informationen finden Sie unter „IBM MQ-Umgebung für IBM MQ classes for Java einrichten” auf Seite 393.

Achtung: IBM i Unter IBM i kann der Hauptklassenname nicht abgefragt werden, daher wird IBM MQ client für Java stattdessen verwendet.

Zugehörige Konzepte

„IBM MQ-Umgebung für IBM MQ classes for Java einrichten” auf Seite 393
Damit eine Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt, muss die Anwendung den Kanalnamen, den Hostnamen und die Portnummer angeben.

Zugehörige Verweise

MQCNO
MQCNO unter IBM i

Verfahren zum Entwerfen von Nachrichten

Hinweise, die beim Entwerfen von Nachrichten helfen sollen, einschließlich Überlegungen zu Selektoren und Nachrichteneigenschaften.

Was in der Designphase zu beachten ist

Sie erstellen eine Nachricht, wenn Sie sie mit einem MQI-Aufruf in eine Warteschlange einreihen. Als Eingabe für den Aufruf geben Sie einige Steuerinformationen in einem *Nachrichtendeskriptor* (MQMD) sowie die Daten, die Sie an ein anderes Programm senden möchten, an. In der Entwurfsphase müssen Sie jedoch folgende Aspekte berücksichtigen, da sie sich auf die Erstellung Ihrer Nachricht auswirken:

Zu verwendender Nachrichtentyp

Sie entwerfen eine einfache Anwendung zum reinen Senden einer Nachricht? Oder bitten Sie um Antwort auf eine Frage? Wenn Sie eine Frage stellen, könnten Sie den Namen der Warteschlange, in der Sie die Antwort empfangen möchten, in den Nachrichtendeskriptor aufnehmen.

Sollen Ihre Anforderungs- und Antwortnachrichten synchronisiert werden? Dazu müssen Sie ein Zeitlimitintervall für die Antwort auf Ihre Anforderung festlegen. Empfangen Sie in diesem Zeitraum keine Antwort, wird dies als Fehler behandelt.

Oder ziehen Sie es vor, asynchron zu arbeiten, sodass Ihre Prozesse nicht zwingend von konkreten Ereignissen wie beispielsweise allgemeinen Zeitsignalen abhängig sind?

Des Weiteren ist zu überlegen, ob alle Nachrichten in einer Arbeitseinheit enthalten sein sollen.

Nachrichten verschiedene Prioritäten zuweisen

Sie können jeder Nachricht einen Prioritätswert zuweisen und die Warteschlange so definieren, dass sie die Nachrichten in der Reihenfolge ihrer Priorität verwaltet. Wenn in diesem Fall ein anderes Programm eine Nachricht aus der Warteschlange abruft, handelt es sich dabei immer um die Nachricht mit der höchsten Priorität. Wenn die Warteschlange die Nachrichten nicht in der Reihenfolge der Prioritäten verwaltet, ruft ein Programm, das Nachrichten aus der Warteschlange abruft, die Nachrichten in der Reihenfolge ab, in der sie in die Warteschlange gestellt wurden.

Programme können eine Nachricht auch über die ID auswählen, die der Warteschlangenmanager zugewiesen hat, als die Nachricht in die Warteschlange gestellt wurde. Alternativ können Sie für Ihre einzelnen Nachrichten eigene IDs generieren.

Neustart des Warteschlangenmanagers und Auswirkung auf Nachrichten

Der Warteschlangenmanager behält alle persistenten Nachrichten bei. Bei einem Neustart stellt er sie bei Bedarf aus den IBM MQ-Protokolldateien wieder her. Nicht persistente Nachrichten und temporäre dynamische Warteschlangen werden nicht beibehalten. Alle Nachrichten, die nicht gelöscht werden sollen, müssen bei ihrer Erstellung als persistent definiert werden. Stellen Sie beim Schreiben einer Anwendung für IBM MQ for Windows oder IBM MQ auf AIX and Linux-Systemen sicher, dass Ihnen die Zuordnung der Protokolldatei des jeweiligen Systems bekannt ist. So vermeiden Sie, eine Anwendung zu entwerfen, die die Grenzwerte der Protokolldatei erreicht.

z/OS Da Nachrichten in gemeinsam genutzten Warteschlangen (nur verfügbar unter IBM MQ for z/OS) in der Coupling-Facility (CF) gelagert werden, werden nicht persistente Nachrichten bei Neustarts eines Warteschlangenmanagers so lange beibehalten, wie die CF verfügbar ist. Bei einem Ausfall der CF gehen nicht persistente Nachrichten verloren.

Persönliche Informationen an Nachrichtenempfänger übermitteln

Normalerweise legt der Warteschlangenmanagergruppe die Benutzer-ID fest, doch entsprechend berechtigte Anwendungen können dieses Feld ebenfalls festlegen, sodass Sie Ihre eigene Benutzer-ID sowie andere Informationen einschließen können, die das empfangende Programm für Abrechnungen oder zu Sicherheitszwecken verwenden kann.

Anzahl der Empfangswarteschlangen

Multi Wenn eine Nachricht unter Umständen in mehrere Warteschlangen eingereiht werden muss, können Sie die Nachricht in einer Verteilerliste oder einem Thema veröffentlichen.

z/OS Wenn eine Nachricht unter Umständen in mehrere Warteschlangen eingereiht werden muss, können Sie die Nachricht in einem Thema veröffentlichen.

Selektoren und Nachrichteneigenschaften

Neben den eigentlichen Nachrichtennutzdaten können Nachrichten auch Metadaten zugeordnet sein. Diese Nachrichteneigenschaften können für die Bereitstellung zusätzlicher Daten nützlich sein.

Es gibt in Bezug auf diese zusätzlichen Daten zwei Aspekte, die Sie wissen sollten:

- Die Eigenschaften unterliegen nicht dem Schutz von Advanced Message Security (AMS). Wenn Sie möchten, dass Ihre Daten durch AMS geschützt werden, fügen Sie sie in die Nutzdaten ein, nicht in die Nachrichteneigenschaften.
- Die Eigenschaften können zur Auswahl von Nachrichten verwendet werden.

Es ist unbedingt zu beachten, dass die Verwendung von Selektoren mit der Standardnachrichtenkonvention des First In/First Out (FIFO) bricht. Da der Warteschlangenmanager für diesen Verarbeitungsprozess optimiert ist, wird die Verwendung komplexer Selektoren aus Leistungsgründen nicht empfohlen. Der Warteschlangenmanager speichert keine Indizes der Nachrichteneigenschaften, weshalb die Suche nach einer Nachricht eine lineare Suche sein muss. Je größer die Warteschlange, desto komplexer der Selektor und umso höher die Wahrscheinlichkeit, dass der Selektor, der mit einer Nachricht übereinstimmt, die Leistung beeinträchtigt.

Wenn eine komplexe Auswahl erforderlich ist, wird empfohlen, die Nachrichten mithilfe einer Anwendung oder Verarbeitungsengine, z. B. IBM Integration Bus, für verschiedene Ziele zu filtern. Alternativ kann die Verwendung einer Themenhierarchie nützlich sein.

Anmerkung: IBM MQ classes for Java unterstützen nicht die Verwendung von Selektoren. Wenn Sie trotzdem Selektoren verwenden möchten, sollten diese über die JMS-API ausgeführt werden.

Anwendungsdesign und Leistung

Ein schlechtes Programm kann sich vielfältig auf die Leistung auswirken. Dies ist nicht immer sofort erkennbar, weil das Programm selbst scheinbar problemlos läuft, gleichzeitig aber die Leistung anderer Tasks beeinträchtigen kann. Im nachfolgenden Abschnitt werden verschiedene Probleme mit Programmen erläutert, die IBM MQ-Aufrufe tätigen.

Nachfolgend erhalten Sie einige Anregungen für den Entwurf effizienter Anwendungen:

- Entwerfen Sie Ihre Anwendung so, dass die Verarbeitung parallel zur Denkzeit des Benutzers verläuft:
  - Blenden Sie eine Anzeige ein und ermöglichen Sie dem Benutzer, noch während der Initialisierung der Anwendung mit der Eingabe beginnen zu können.
  - Rufen Sie die benötigten Daten parallel von anderen Servern ab.
- Lassen Sie Verbindungen und Warteschlangen offen, wenn Sie sie wiederverwenden, anstatt sie wiederholt zu öffnen, zu schließen, eine Verbindung herzustellen und diese wieder zu trennen.
- Eine Serveranwendung, die nur eine Nachricht einreiht, sollte allerdings MQPUT1 verwenden.
- Warteschlangenmanager sind für Nachrichten mit einer Größe von 4 KB bis 100 KB optimiert. Sehr große Nachrichten sind ineffizient; es ist unter Umständen besser, 100 Nachrichten von je 1 MB anstatt eine Nachricht mit 100 MB zu senden. Ebenfalls ineffizient sind sehr kleine Nachrichten. Der Warteschlangenmanager verwendet auf eine Einzelbyte-Nachricht denselben Arbeitsaufwand wie auf eine Nachricht mit 4 KB.
- Sammeln Sie Ihre Nachrichten in einer Arbeitseinheit, damit sie gleichzeitig festgeschrieben oder zurückgesetzt werden können.
- Verwenden Sie die nicht persistente Option für Nachrichten, die nicht wiederherstellbar sein müssen.
- Wenn Sie eine Nachricht an viele Zielwarteschlangen senden müssen, sollten Sie dafür eine Verteilerliste in Betracht ziehen.

Auswirkung der Nachrichtenlänge

Die Datenmenge in einer Nachricht kann Auswirkungen auf die Leistung der Anwendung haben, von der die Nachricht verarbeitet wird. Um die bestmögliche Leistung der Anwendung zu erreichen, sollten nur die wirklich wesentlichen Daten in einer Nachricht gesendet werden. Beispielsweise müssen in einer Anforderung zur Belastung eines Kontos möglicherweise nur die Kontonummer und der Belastungsbetrag als wirklich benötigte Informationen von der Client- an die Serveranwendung übergeben werden.

Auswirkung der Nachrichtenpersistenz

Persistente Nachrichten werden manuell protokolliert. Da die Leistung Ihrer Anwendung durch dieses Protokollieren beeinträchtigt wird, sollten Sie persistente Nachrichten nur für die wichtigsten Daten verwenden. Wenn die Daten in einer Nachricht im Fall der Beendigung oder des Fehlschlagens eines Warteschlangenmanagers gelöscht werden können, verwenden Sie eine nicht persistente Nachricht.

z/OS MQPUT- und MQGET-Operationen für persistente Nachrichten werden blockiert, wenn der für das Wiederherstellungsprotokoll bereitgestellte Speicher nicht für die Aufzeichnung der Operationen ausreicht. Eine solche Bedingung wird im Jobprotokoll des Warteschlangenmanagers durch die Nachrichten CSQJ110E und CSQJ111A angezeigt. Richten Sie Überwachungsprozesse ein, durch die solche Bedingungen verwaltet und vermieden werden.

Bestimmte Nachrichten suchen

Normalerweise ruft der MQGET-Aufruf die erste Nachricht aus einer Warteschlange ab. Wenn Sie mit den Nachrichten- und Korrelations-IDs (*MsgId* und *CorrelId*) im Nachrichtendeskriptor eine bestimmte Nachricht angeben, durchsucht der Warteschlangenmanager die Warteschlange, bis er die betreffende Nachricht findet. Eine solche Verwendung des Aufrufs MQGET wirkt sich auf die Leistung Ihrer Anwendung aus.

Warteschlangen mit Nachrichten verschiedener Länge

Wenn Ihre Anwendung Nachrichten einer bestimmten Länge nicht verarbeiten kann, vergrößern und verkleinern Sie die Puffer dynamisch entsprechend der typischen Nachrichtengröße. Wenn die Anwendung einen MQGET-Aufruf ausgibt, der aufgrund eines zu kleinen Puffers fehlschlägt, wird die Größe der

Nachrichtendaten zurückgegeben. Ergänzen Sie Ihre Anwendung um entsprechenden Code, damit die Puffergröße angepasst und der MQGET-Aufruf erneut ausgegeben wird.

Anmerkung: Wenn das Attribut **MaxMsgLength** nicht explizit angegeben wird, übernimmt es standardmäßig den Wert 4 MB, was zur Steuerung der Größe des Anwendungspuffers unter Umständen sehr ineffizient ist.

Häufigkeit von Synchronisationspunkten

Programme, die viele MQPUT- oder MQGET-Aufrufe innerhalb des Synchronisationspunkts ausgeben, ohne sie festzuschreiben, können Leistungsprobleme verursachen. Davon betroffene Warteschlangen können mit Nachrichten gefüllt werden, auf die in dieser Zeit nicht zugegriffen werden kann, die aber gleichzeitig möglicherweise von anderen Tasks dringend erwartet werden. Dies wirkt sich nicht nur auf den Speicher aus, sondern auch insofern, als Threads durch Tasks blockiert werden, die versuchen, Nachrichten abzurufen.

MQPUT1-Aufruf verwenden

Verwenden Sie den Aufruf MQPUT1 nur dann, wenn lediglich eine einzelne Nachricht in eine Warteschlange eingereiht werden soll. Wenn Sie mehrere Nachrichten einreihen möchten, verwenden Sie den MQOPEN-Aufruf gefolgt von mehreren MQPUT-Aufrufen und einem einzelnen MQCLOSE-Aufruf.

Anzahl der verwendeten Threads

Windows Unter IBM MQ for Windows benötigt eine Anwendung unter Umständen eine große Anzahl von Threads. Jedem Warteschlangenmanagerprozess ist eine maximal zulässige Anzahl an Anwendungsthreads zugeordnet.

Anwendungen können zu viele Threads verwenden. Stellen Sie fest, ob die Anwendung diese Möglichkeit berücksichtigt und Maßnahmen ergreift, durch die diese Art von Problem beseitigt oder dokumentiert wird.

Persistente Nachrichten unter den Synchronisationspunkt stellen

Persistente Nachrichten sollten unter dem Synchronisationspunkt eingereiht und abgerufen werden. Dies empfiehlt sich, da für die Anwendung, wenn eine persistente Nachricht außerhalb des Synchronisationspunkt abgerufen wird, keine Möglichkeit besteht, festzustellen, ob die Nachricht aus der Warteschlange abgerufen wurde und, wenn dies der Fall ist, ob die Nachricht ebenfalls verloren gegangen ist. Wird eine persistente Nachricht dagegen unter dem Synchronisationspunkt abgerufen, wird die Transaktion bei einem Fehler rückgängig gemacht, weshalb die persistente Nachricht nicht verloren gehen kann, da sie sich in diesem Fall noch in der Warteschlange befindet.

Ebenso sollten Sie persistente Nachrichten unter dem Synchronisationspunkt einreihen. Ein weiterer Grund für das Einreihen und Abrufen persistenter Nachrichten unter dem Synchronisationspunkt ist die in diesem Fall enorme Optimierung des Nachrichtencodes in IBM MQ. Das Einreihen und Abrufen persistenter Nachrichten unter dem Synchronisationspunkt ist daher schneller als außerhalb des Synchronisationspunkts.

Wenn Ihre Anwendung persistente Nachrichten außerhalb des Synchronisationspunkts einreiht, prüft der Warteschlangenmanager, ob er im Auftrag der Anwendung einen impliziten Synchronisationspunkt erstellen kann. Ist eine Erstellung möglich, schließt der Warteschlangenmanager die Einreihung innerhalb dieses Synchronisationspunkts ein und schreibt sie automatisch fest. Ausführliche Informationen hierzu finden Sie unter „Impliziter Synchronisationspunkt unter Multiplatforms” auf Seite 909.

Bei nicht persistenten Nachrichten verhält es sich hingegen umgekehrt. Der Code nicht persistenter Nachrichten wird in IBM MQ für die Verarbeitung außerhalb des Synchronisationspunkts optimiert, weshalb das Einreihen und Abrufen dieser Nachrichten schneller außerhalb des Synchronisationspunkts erfolgt. Wie schnell persistente Nachrichten eingereiht und abgerufen werden, ist von der Plattengeschwindigkeit abhängig, da persistente Nachrichten auf Platte gespeichert werden. Bei nicht persistenten Nachrichten

richtet sich die Geschwindigkeit hingegen nach der CPU-Geschwindigkeit, da selbst beim Einreihen und Abrufen unter dem Synchronisationspunkt keine Plattenzugriffe erforderlich sind.

Wenn bei einer Anwendung Nachrichten eingehen und im Voraus nicht bekannt ist, ob diese persistent oder nicht persistent sind, kann die GMO-Option MQGMO_SYNCPOINT_IF_PERSISTENT verwendet werden.

Verfahren zum Entwerfen von anspruchsvolleren Anwendungen

In diesem Abschnitt werden einige Verfahren erläutert, die Sie beim Entwerfen von anspruchsvolleren Anwendungen einsetzen können, wie z. B. Warten auf Nachrichten, Korrelieren von Antworten, Festlegung und Verwendung von Kontextinformationen, automatisches Starten von Anwendungen, Erstellen von Berichten und Entfernen von Nachrichtenaffinitäten beim Clustering.

Bei einer einfachen IBM MQ-Anwendung müssen Sie festlegen, welche IBM MQ-Objekte und welche Arten von Nachrichten in Ihrer Anwendung verwendet werden sollen. Bei erweiterten Anwendungen sollten Sie einige der nachfolgend vorgestellten Methoden anwenden.

Warten auf Nachrichten

Ein Programm, das für eine Warteschlange zuständig ist, kann wie folgt auf Nachrichten warten:

- Es wartet, bis eine Nachricht eingeht oder bis ein festgelegtes Zeitintervall verstrichen ist (siehe „Warten auf Nachrichten” auf Seite 838).
- z/OS Nur IBM MQ for z/OS: Ein Signal festlegen, mit dem das Programm benachrichtigt wird, dass eine Nachricht eintrifft. Weitere Informationen finden Sie unter „Signaling” auf Seite 839.
- Durch Erstellen eines Callback-Exits, der bei Eingang einer Nachricht ausgelöst wird; siehe „Asynchrone Verarbeitung von IBM MQ-Nachrichten” auf Seite 44.
- Mittels regelmäßiger Abfragen der Warteschlange, um nach eingegangenen Nachrichten zu suchen (*Polling*). Aufgrund möglicher Leistungsbeeinträchtigungen ist diese Methode nicht empfehlenswert.

Antworten korrelieren

Wenn ein Programm in IBM MQ-Anwendungen eine Nachricht empfängt, die mehrere Aktionen erfordert, sendet es mindestens eine Antwortnachricht an den Anforderer.

Damit der Anforderer diese Antworten leichter der jeweils ursprünglichen Anforderung zuordnen kann, kann eine Anwendung im Deskriptor jeder Nachricht das Feld *Korrelations-ID* festlegen. Anschließend kopieren die Programme die Nachrichten-ID der Anforderungsnachricht in das Feld für die Korrelations-ID ihrer Antwortnachrichten.

Kontextinformationen festlegen und verwenden

Mithilfe von *Kontextinformationen* werden Nachrichten dem Benutzer zugeordnet, der sie generiert hat. Außerdem dienen sie zur Ermittlung der Anwendung, mit der jeweilige Nachricht erstellt wurde. Diese Informationen sind bezüglich Sicherheit, Abrechnung, Prüfung sowie bei der Problembestimmung hilfreich.

Wenn Sie eine Nachricht erstellen, können Sie eine Option angeben, mit der angefordert wird, dass der Warteschlangenmanager Ihrer Nachricht Standardkontextinformationen zuordnet.

Weitere Informationen zum Verwenden und Festlegen von Kontextinformationen finden Sie unter „Nachrichtenkontext” auf Seite 50.

IBM MQ-Programme automatisch starten

Wenn eine Nachricht in eine Warteschlange gestellt wird, kann mittels IBM MQ *Triggering* automatisch ein Programm gestartet werden.

Sie können die Triggerbedingungen für eine Warteschlange so festlegen, dass ein Programm zur Verarbeitung dieser Warteschlange gestartet wird:

- Sobald eine Nachricht in die Warteschlange gestellt wird
- Wenn die erste Nachricht in die Warteschlange gestellt wird
- Wenn die Anzahl der Nachrichten in der Warteschlange eine vordefinierte Menge erreicht

Weitere Informationen zum Triggering finden Sie unter„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913. Triggering ist jedoch nur eine Möglichkeit, ein Programm automatisch zu starten. Sie können beispielsweise ein Programm automatisch mittels Nicht-IBM MQ-Funktionen in einem Zeitgeber starten.

Auf Multiplatforms-Systemen kann IBM MQ Serviceobjekte definieren, die IBM MQ-Programme starten, sobald der Warteschlangenmanager gestartet wird. Weitere Informationen hierzu finden Sie im Abschnitt Serviceobjekte.

IBM MQ-Berichte generieren

Sie können die folgenden Berichte in einer Anwendung anfordern:

- Ausnahmeberichte
- Ablaufberichte
- Berichte zur Bestätigung bei Eingang (COA-Berichte)
- Empfangsbestätigungsberichte (COD-Berichte)
- Berichte zur Benachrichtigung über positive Aktionen (PAN-Berichte)
- Berichte zur Benachrichtigung über negative Aktionen (NAN-Berichte)

Diese werden im Abschnitt „Berichtsnachrichten” auf Seite 21 erläutert.

Cluster und Nachrichtenaffinitäten

Bevor Sie Cluster mit mehrere Definitionen für eine einzelne Warteschlange verwenden, stellen Sie fest, ob Ihre Anwendungen den Austausch zusammengehöriger Nachrichten erfordern.

In einem Cluster kann eine Nachricht an jeden Warteschlangenmanager weitergeleitet werden, auf dem sich eine Instanz der entsprechenden Warteschlange befindet. Daher kann die Logik von Anwendungen mit Nachrichtenaffinitäten gestört werden.

Beispiel: Zwei Anwendungen stützen sich auf eine Reihe von Nachrichten, die zwischen ihnen als Fragen und Antworten fließen. Es könnte wichtig sein, dass alle Fragen an den einen und alle Antwortnachrichten an den anderen Warteschlangenmanager gesendet werden. In diesem Fall ist es wichtig, dass die Workload-Management-Routine die Nachrichten nicht an einen Warteschlangenmanager sendet, der auf dem sich zufällig eine Instanz der entsprechenden Warteschlange befindet.

Falls möglich, sollten Sie die Affinitäten entfernen. Durch das Entfernen von Nachrichtenaffinitäten verbessern Sie die Verfügbarkeit und Skalierbarkeit von Anwendungen.

Weitere Informationen hierzu finden Sie im Abschnitt Nachrichtenaffinitäten bearbeiten .

Entwurfs- und Leistungsaspekte für Anwendungen unter IBM i

In diesem Abschnitt erfahren Sie, wie sich Anwendungsdesign, Threads und Speicher auf die Leistung auswirken können.

Diese Informationen sind in zwei Kapitel gegliedert:

- „Überlegungen zum Anwendungsdesign” auf Seite 67
- „Spezielle Leistungsprobleme” auf Seite 69

Überlegungen zum Anwendungsdesign

Ein schlechtes Programm kann sich vielfältig auf die Leistung auswirken. Diese Probleme sind unter Umständen schwer feststellbar, da das Programm scheinbar gut funktioniert, aber die Leistung anderer

Tasks beeinträchtigt. In den nachfolgenden Kapiteln werden verschiedene Probleme mit Programmen erläutert, die IBM MQ for IBM i-Aufrufe tätigen.

Weitere Informationen zum Anwendungsentwurf finden Sie unter „Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 52.

Auswirkung der Nachrichtenlänge
Obwohl die Datenmenge von Nachrichten unter IBM MQ for IBM i bis zu 100 MB betragen kann, wirkt sich die Datenmenge in einer Nachricht auf die Leistung der Anwendung aus, von der die Nachricht verarbeitet wird. Senden Sie zur optimalen Nutzung einer Anwendung nur die wichtigsten Daten in einer Nachricht; beispielsweise müssen in einer Anforderung zur Belastung eines Bankkontos unter Umständen vom Client lediglich die Kontonummer und der zu belastende Betrag an die Serveranwendung übergeben werden.

Auswirkung der Nachrichtenpersistenz
Persistente Nachrichten werden in einem Journal aufgezeichnet. Da durch das Journaling von Nachrichten die Leistung Ihrer Anwendung beeinträchtigt wird, verwenden Sie persistente Nachrichten nur für die wichtigsten Daten. Wenn die Daten in einer Nachricht im Fall der Beendigung oder des Fehlschlagens eines Warteschlangenmanagers gelöscht werden können, verwenden Sie eine nicht persistente Nachricht.

Bestimmte Nachrichten suchen
Normalerweise ruft der MQGET-Aufruf die erste Nachricht aus einer Warteschlange ab. Wenn Sie die Nachrichten-und Korrelations-IDs (*MsgId* und *CorrelId*) im Nachrichtendeskriptor verwenden, um eine bestimmte Nachricht anzugeben, muss der Warteschlangenmanager die Warteschlange durchsuchen, bis er diese Nachricht findet. Diese Verwendung des MQGET-Aufrufs beeinträchtigt die Leistung Ihrer Anwendung.

Warteschlangen mit Nachrichten verschiedener Länge
Wenn die Nachrichten in einer Warteschlange unterschiedlich lang sind, kann Ihre Anwendung den MQGET-Aufruf verwenden, bei dem das Feld *BufferLength* auf null gesetzt ist, damit die Größe der Nachrichtendaten zurückgegeben wird, auch wenn der Aufruf fehlschlägt. Die Anwendung kann den Aufruf anschließend wiederholen, wobei sie die ID der Nachricht, die mit dem ersten Aufruf 'gemessen' wurde, und einen Puffer mit der passenden Größe angibt. Wenn andere Anwendungen auf dieselbe Warteschlange zugreifen, stellen Sie jedoch möglicherweise fest, dass die Leistung Ihrer Anwendung nachlässt, weil der zweite MQGET-Aufruf damit beschäftigt ist, nach einer Nachricht zu suchen, die zwischenzeitlich (zwischen dem ersten und zweiten Aufruf) von einer anderen Anwendung abgerufen wurde.

Wenn Ihre Anwendung keine Nachrichten einer festen Länge verwenden kann, kann dieses Problem auch mit dem MQINQ-Befehl gelöst werden, der die maximale Größe der Nachrichten feststellt, die von der Warteschlange akzeptiert werden kann. Dieser Wert kann dann in Ihrem MQGET-Aufruf verwendet werden. Die maximale Größe von Nachrichten für eine Warteschlange wird im Attribut **MaxMsgLen** der Warteschlange gespeichert. Diese Methode könnte jedoch größere Speichermengen verwenden, da der Wert dieses Warteschlangenattributs die von IBM MQ for IBM i unterstützte maximale Größe (über 2 GB) betragen kann.

Häufigkeit von Synchronisationspunkten
Programme, die viele MQPUT-Aufrufe innerhalb des Synchronisationspunkts ausgeben, ohne sie festzuschreiben, können Leistungsprobleme verursachen. Betroffene Warteschlangen können sich mit Nachrichten füllen, die derzeit nicht nutzbar sind, während andere Tasks möglicherweise auf den Erhalt dieser Nachrichten warten. Dies hat Auswirkungen auf die Speicherbelegung und die Belegung von Threads durch Tasks, mit denen Nachrichten abgerufen werden.

MQPUT1-Aufruf verwenden
Verwenden Sie den Aufruf MQPUT1 nur dann, wenn lediglich eine einzelne Nachricht in eine Warteschlange eingereiht werden soll. Wenn Sie mehrere Nachrichten einreihen möchten, verwenden Sie den MQOPEN-Aufruf gefolgt von mehreren MQPUT-Aufrufen und einem einzelnen MQCLOSE-Aufruf.

Anzahl der verwendeten Threads
Für eine Anwendung sind unter Umständen viele Threads erforderlich. Jedem Warteschlangenmanagerprozess wird eine maximal zulässige Anzahl von Threads zugeordnet. Falls es bei manchen Anwendungen zu Problemen kommt, könnte es daran liegen, dass sie zu viele Threads verwenden.

Stellen Sie fest, ob die Anwendung diese Möglichkeit berücksichtigt und Maßnahmen ergreift, durch die diese Art von Problem beseitigt oder dokumentiert wird. IBM i unterstützt maximal 4095 Threads. Der Standardwert beträgt jedoch 64. IBM MQ stellt seinen Prozessen bis zu 63 Threads bereit.

Spezielle Leistungsprobleme

In diesem Abschnitt werden Speicher- und Leistungsprobleme erläutert.

Speicherprobleme

Wenn Sie die Systemnachricht `CPF0907. Serious storage condition may exist` empfangen, ist es möglich, dass Sie den Speicherbereich füllen, der den IBM MQ for IBM i -Warteschlangenmanagern zugeordnet ist.

Wird Ihre Anwendung oder IBM MQ for IBM i langsam ausgeführt?

Wenn Ihre Anwendung nur langsam ausgeführt wird, könnte dies darauf hinweisen, dass sie sich in einer Schleife befindet oder auf eine Ressource wartet, die nicht verfügbar ist. Dies könnte auch auf ein Leistungsproblem zurückzuführen sein. Es wäre denkbar, dass Ihr System seine Leistungsgrenze nahezu erreicht hat. Dieses Problem tritt wahrscheinlich verstärkt zu Zeiten mit einer hohen Systembelastung auf, also etwa am späten Morgen und am frühen Nachmittag. (Wenn Ihr Netz in mehreren Zeitzonen verwendet wird, treten die Lastspitzen unter Umständen zu anderen Tageszeiten auf.)

Wenn die Leistungseinbußen nicht mit der Systemauslastung zusammenhängen, sondern auch gelegentlich auftreten, wenn das System nur eine geringe Belastung aufweist, ist das Problem wahrscheinlich auf ein schlechtes Anwendungsprogrammdesign zurückzuführen. Es könnte sich darin zeigen, dass es nur auftritt, wenn auf bestimmte Warteschlangen zugegriffen wird.

Sie sollten die Systemwerte QTOTJOB und QADLTOTJ überprüfen.

Die folgenden Symptome können darauf hindeuten, dass IBM MQ for IBM i langsam ausgeführt wird:

- Ihr System reagiert spät auf MQSC-Befehle.
- Wiederholte Anzeigen der Warteschlangenlänge weisen darauf hin, dass die Warteschlange für eine Anwendung, bei der Sie ein hohes Maß an Warteschlangenaktivität erwarten, langsam verarbeitet wird.
- Ist die IBM MQ-Trace aktiv?

Linux Konzepte für den Entwurf von Linux on Power Systems - Little Endian-Anwendungen

Da Linux on Power Systems - Little Endian nur 64-Bit-Anwendungen unterstützt, bietet IBM MQ keine Unterstützung für 32-Bit-Anwendungen.

Zugehörige Konzepte

„Konzepte für den Entwurf von IBM MQ-Anwendungen" auf Seite 52

Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

z/OS Design and performance considerations for z/OS applications

Application design is one of the most important factors affecting performance. Use this topic to understand some of the design factors involved in performance.

There are a number of ways in which poor program design can affect performance. These problems can be difficult to detect because the program can appear to perform well, while affecting the performance of other tasks. Several problems specific to programs making MQI calls are demonstrated in the following sections.

For more information about application design, see "Konzepte für den Entwurf von IBM MQ-Anwendungen" on page 52.

Effect of message length

Although IBM MQ for z/OS allows messages to hold up to 100 MB of data, the amount of data in a message affects the performance of the application that processes the message. To achieve the best performance from your application, send only the essential data in a message. For example, in a request to debit a bank account, the only information that might need to be passed from the client to the server application is the account number and the amount to debit.

Effect of message persistence

Persistent messages are logged. Logging messages reduces the performance of your application, so use persistent messages for essential data only. If the data in a message can be discarded if the queue manager stops or fails, use a nonpersistent message.

Data for persistent messages is written to log buffers. These buffers are written to the log data sets when:

- A commit occurs
- A message is got or put out of syncpoint
- WRTHRSH buffers are filled

Processing many messages in one unit of work can cause less input/output than if the messages were processed one for each unit of work, or out of syncpoint.

Searching for a particular message

The MQGET call typically retrieves the first message from a queue. If you use the message and correlation identifiers (**MsgId** and **CorrelId**) in the message descriptor to specify a particular message, the queue manager searches the queue until it finds that message. Using MQGET in this way affects the performance of your application because, to find a particular message, IBM MQ might have to scan the entire queue.

You can use the **IndexType** queue attribute to specify that you want the queue manager to maintain an index that can be used to increase the speed of MQGET operations on the queue. However, there is a small performance reduction for maintaining an index, so only generate one if you need to use it. You can choose to build an index of message identifiers or of correlation identifiers, or you can choose not to build an index for queues where messages are retrieved sequentially. Try to have many different key values, not lots with the same value. For example Balance1, Balance2, and Balance3, not three with Balance. For shared queues, you must have the correct **IndexType**. For details of the **IndexType** queue attribute, see IndexType.

To avoid affecting queue manager restart time by using indexed queues, use the QINDXBLD(NOWAIT) parameter in the CSQ6SYSP macro. This allows the queue manager restart to complete without waiting for queue index building to complete.

For a full description of the **IndexType** attribute, and other object attributes see Attributes of objects.

Queues that contain messages of different lengths

Get a message, using a buffer size matching the expected size of the message. If you receive the return code indicating that the message is too long, get a bigger buffer. When the get fails in this way, the data length returned is the size of the unconverted message data. If you specify MQGMO_CONVERT on the MQGET call, and the data expands during conversion, it still might not fit in the buffer, in which case you need to further increase the size of the buffer.

If you issue the MQGET with a buffer length of zero, it returns the size of the message and the application can then get a buffer of this size and reissue the get. If you have multiple applications processing the queue, another application might have already processed the message when the original application reissued the get. If you occasionally have large messages, you might need to get a large buffer just for

these messages, and release it after the message has been processed. This should help reduce virtual storage problems if all applications have large buffers.

If your application cannot use messages of a fixed length, another solution to this problem is to use the MQINQ call to find the maximum size of messages that the queue can accept, then use this value in your MQGET call. The maximum size of messages for a queue is stored in the **MaxMsgL** attribute of the queue. This method could use large amounts of storage, however, because the value of **MaxMsgL** could be as high as 100 MB, the maximum allowed by IBM MQ for z/OS.

Note: You can lower the **MaxMsgL** parameter after large messages have been put to the queue. For example you can put a 100 MB message, then set **MaxMsgL** to 50 bytes. This means that it is still possible to get bigger messages than the application expected.

Frequency of syncpoints

Programs that issue many MQPUT calls within syncpoint, without committing them, can cause performance problems. Affected queues can fill up with messages that are currently unusable, while other tasks might be waiting to get these messages. This has implications in terms of storage, and in terms of threads tied up with tasks that are attempting to get messages.

As a rule if you have multiple applications processing a queue you typically get the best performance when you have either

- 100 short messages (less than 1 KB), or
- One message for larger messages (100 KB)

for each syncpoint. If there is only one application processing the queue, you must have more messages for each unit of work.

You can limit the number of messages that a task can get or put within a single unit of recovery with the **MAXUMSGS** queue manager attribute. For information about this attribute, see the **ALTER QMGR** command in MQSC commands.

Advantages of the MQPUT1 call

Use the MQPUT1 call only if you have a single message to put on a queue. If you want to put more than one message, use the MQOPEN call, followed by a series of MQPUT calls and a single MQCLOSE call.

How many messages can a queue manager contain

Local Queues

The number of local messages a queue manager can hold is basically the size of the page sets. You can have up to 100 page sets (though it is recommended page set 0 and page set 1 are for system related objects and queues). You can use a page set with extended format and increase the capacity of a page set.

Shared Queues

The capacity for shared queues depends on the size of the coupling facility (CF). IBM MQ uses CF list structures where fundamental storage units are entries and elements. Each message is stored as 1 entry and multiple elements containing the associated MQMD and other message data. The number of elements consumed by a single message depends on the size of the message and, for CFLEVEL(5), the offload rules in effect at MQPUT time. Fewer elements are needed when message data is offloaded to either Db2 or SMDS. Message data access is slower when the message has been offloaded. See Performance Supportpac MP1H for further comparison of performance and CPU overhead associated with message offload.

What affects performance

Performance can mean how fast messages can be processed, and it can also mean how much CPU is needed per message.

What affects how fast messages can be processed

For persistent messages the biggest impact is the speed of the log data sets. The speed of the log data sets depends on the DASD they are on. Therefore care should be taken to put log data set on low used volumes to reduce contention. Striping the MQ logs improves the log performance when there are multiple pages written per I/O. Z High Performance Fibre connection (zHPF) also has a significant performance to I/O response time when the I/O subsystem is busy.

When there is a request to get and put a message, access to the queue is locked during the request to preserve integrity of the queue. For planning purposes consider the queue locked for the whole request. So if the time for a put is 100 microseconds, and you have more than 10,000 requests a second you might experience delays. You might achieve better than this in practice, but it is a good general rule. You can use different queues to improve performance.

Possible reasons for this can be:

- use a common reply queue which every CICS transaction uses
- each CICS transaction is given a unique reply to queue
- a reply to a queue for CICS region and all transactions in the CICS region use this queue.

The answer depends on the number of requests a second, and the response time of the requests.

If messages have to be read from a page set, they will be slower compared to when the messages are in the buffer pool. If you have more messages than fit into a buffer pool, then they will spill to disk. So you need to ensure that the buffer pool is big enough for your short lived messages. If you have messages that you process many hours later, these are likely to spill to disk, so you should expect a get for these messages to be slower than if they were in the buffer pool.

For a shared queue, the speed of the messages depends on the speed of the Coupling Facility. A CF within the physical processor is likely to be faster than an external CF. The CF response time depends on how busy the CF is. For example on the Hursley systems, when the CF was 17% busy the response time was 14 microseconds. When the CF was 95% busy the response time was 45 microseconds.

If your MQ requests use a lot of CPU, this can affect how fast messages are processed. Because if the Logical Partition (LPAR) is constrained for CPU, applications will be delayed waiting for CPU.

How much CPU per message

In general bigger messages use more CPU, so avoid large (x MB) messages if possible.

When getting specific messages from queues, the queue should be indexed so the queue manager can go directly to the message (and so avoids potentially an entire scan of the queue). If the queue is not indexed then the queue is scanned from the beginning looking for the message. If there are 1000 messages on the queue, it may have to scan all 1000 messages. The result is a lot of unnecessary CPU usage.

Channels using TLS have an additional cost due to the encryption of the message.

In MQ V7 you can select messages by a selector string in addition to the **CORRELID** or **MSGID**. Every message has to be looked in, so if there are many messages on the queue this is expensive.

It is more efficient for an application to do OPEN PUT PUT CLOSE than PUT1 PUT1.

Triggering in CICS

When the message arrival rate of messages for a triggered queue is low, it is efficient to use trigger first. When the message arrival rate is more than 10 messages a second, it is more efficient to trigger the first transaction, then have the transaction process a message and get the next message, and so on. If a message has not arrived in a short period (say between 0.1 and 1 second) the transaction ends. At high throughput you might need multiple transactions running to process the messages and

to prevent a build up of messages. For every trigger message produced, this requires a put and a get of a trigger message, which in effect doubles the cost of the message.

How many connections or concurrent users are supported

Each connection uses virtual storage within the queue manager, so the more concurrent users the more storage used. If you need a very large buffer pool and large number of users, then you might be constrained for virtual storage, and you might need to reduce the size of your buffer pools.

If security is being used, the queue manager caches information within the queue manager for a long period. The amount of virtual storage that is used within the queue manager is affected.

The **CHINIT** can support up to about 10,000 connections. This is limited by virtual storage. If a connection uses more storage, for example using by TLS, the storage per connection increases, which therefore means the **CHINIT** can support less connections. If you are processing large messages, these will require more storage for buffers in the **CHINIT**, so the **CHINIT** can support less messages.

Connections to a remote queue manager are more efficient than client connections. For example, every MQ client requests requires two network flows (one for the request and one for the response). With a channel to a remote queue manager, there may be 50 sends over the network before a response comes back. If you are considering a large client network, it may be more efficient to use a concentrator queue manager on a distributed box, and have one channel coming in and out of the concentrator.

Other things affecting performance

Log data set should be at least 1000 cylinders in size. If the logs are smaller than this, checkpoint activity may be too frequent. On a busy system a checkpoint typically should be every 15 minutes or longer, at very high throughputs it may less than this. When a checkpoint occurs the buffer pools are scanned and 'old' messages and changed pages are written to disk. If checkpoints are too frequent, this can impact performance. The value of LOGLOAD can also affect checkpoint frequency. If the queue manager abnormally ends, then at restart it may have to read back to 3 checkpoints. The best checkpoint interval is a balance between the activity when a checkpoint is taken, and the amount of log data that may need to be read when the queue manager restarts.

There is a significant overhead incurred when starting a channel. It is usually better to start a channel and leave it connected, rather than frequent starts and stops of the channel.

Related information

MP1K: IBM MQ for z/OS 9.0 Performance Report

z/OS IMS and IMS bridge applications on IBM MQ for z/OS

This information helps you to write IMS applications using IBM MQ.

- To use syncpoints and MQI calls in IMS applications, see “Writing IMS applications using IBM MQ” on page 74.
- To write applications that use the IBM MQ - IMS bridge, see “Writing IMS bridge applications” on page 78.

Use the following links to find out more about IMS and IMS bridge applications on IBM MQ for z/OS:

- “Writing IMS applications using IBM MQ” on page 74
- “Writing IMS bridge applications” on page 78

Related concepts

“Message Queue Interface (MQI) - Übersicht” on page 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

“Verbindung zu einem Warteschlangenmanager herstellen und trennen” on page 773

Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

“Objekte öffnen und schließen” on page 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

“Nachrichten in eine Warteschlange einreihen” on page 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

“Nachrichten aus einer Warteschlange abrufen” on page 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

“Objektattribute abfragen und einstellen” on page 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

“Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” on page 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

“IBM MQ-Anwendungen durch Auslöser starten” on page 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

“Mit MQI und Clustern arbeiten” on page 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

“Using and writing applications on IBM MQ for z/OS” on page 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

z/OS Writing IMS applications using IBM MQ

There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

Use the following links to find out more about writing IMS applications on IBM MQ for z/OS:

- “Syncpoints in IMS applications” on page 74
- “MQI calls in IMS applications” on page 75

Restrictions

There are restrictions on which IBM MQ API calls can used by an application using the IMS adapter.

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

Related concepts

“Writing IMS bridge applications” on page 78
This topic contains information about writing applications to use the IBM MQ - IMS bridge.

z/OS *Syncpoints in IMS applications*

In an IMS application, you establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint).

To back out all changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see ROLB call in the IMS documentation.

The queue manager is a participant in a two-phase commit protocol; the IMS syncpoint manager is the coordinator.

All open handles are closed by the IMS adapter at a syncpoint (except in a batch or non-message driven BMP environment). This is because a different user could initiate the next unit of work and IBM MQ security checking is performed when the MQCONN, MQCONNX, and MQOPEN calls are made, not when the MQPUT or MQGET calls are made.

However, in a Wait-for-Input (WFI) or pseudo Wait-for-Input (PWFI) environment IMS does not notify IBM MQ to close the handles until either the next message arrives or a QC status code is returned to the application. If the application is waiting in the IMS region and any of these handles belong to triggered queues, triggering will not occur because the queues are open. For this reason, applications running in a WFI or PWFI environment should explicitly MQCLOSE the queue handles before doing the GU to the IOPCB for the next message.

If an IMS application (either a BMP or an MPP) issues the MQDISC call, open queues are closed but no implicit syncpoint is taken. If the application ends normally, any open queues are closed and an implicit commit occurs. If the application ends abnormally, any open queues are closed and an implicit backout occurs.

z/OS *MQI calls in IMS applications*

Use this information to learn about the use of MQI calls on Server applications and Enquiry applications.

This section covers the use of MQI calls in the following types of IMS applications:

- "Server applications" on page 75
- "Inquiry applications" on page 77

Server applications

Here is an outline of the MQI server application model:

```
Initialize/Connect
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

Sample program CSQ4ICB3 shows the implementation, in C/370, of a BMP using this model. The program establishes communication with IMS first, and then with IBM MQ:

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if
```

```
Return
```

The IMS initialization determines whether the program has been called as a message-driven or a batch-oriented BMP and controls IBM MQ queue manager connection and queue handles accordingly:

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

The IBM MQ initialization connects to the queue manager and opens the queues. In a message-driven BMP this is called after each IMS syncpoint is taken; in a batch-oriented BMP, this is called only during program startup:

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

The implementation of the server model in an MPP is influenced by the fact that the MPP processes a single unit of work per invocation. This is because, when a syncpoint (GU) is taken, the connection and queue handles are closed and the next IMS message is delivered. This limitation can be partially overcome by one of the following:

- **Processing many messages within a single unit-of-work**

  This involves:

- Reading a message
- Processing the required updates
- Putting the reply

in a loop until all messages have been processed or until a set maximum number of messages has been processed, at which time a syncpoint is taken.

Only certain types of application (for example, a simple database update or inquiry) can be approached in this way. Although the MQI reply messages can be put with the authority of the originator of the MQI message being handled, the security implications of any IMS resource updates need to be addressed carefully.

- **Processing one message per invocation of the MPP and ensuring multiple scheduling of the MPP to process all available messages.**

  Use the IBM MQ IMS trigger monitor program (CSQQTRMN) to schedule the MPP transaction when there are messages on the IBM MQ queue and no applications serving it.

  If trigger monitor starts the MPP, the queue manager name and queue name are passed to the program, as shown in the following COBOL code extract:

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

The server model, which is expected to be a long running task, is better supported in a batch processing region, although the BMP cannot be triggered using CSQQTRMN.

Inquiry applications

A typical IBM MQ application initiating an inquiry or update works as follows:

- Gather data from the user
- Put one or more IBM MQ messages
- Get the reply messages (you might have to wait for them)
- Provide a response to the user

Because messages put on to IBM MQ queues do not become available to other IBM MQ applications until they are committed, they must either be put out of syncpoint, or the IMS application must be split into two transactions.

If the inquiry involves putting a single message, you can use the *no syncpoint* option; however, if the inquiry is more complex, or resource updates are involved, you might get consistency problems if failure occurs and you do not use syncpointing.

To overcome this, you can split IMS MPP transactions using MQI calls using a program-to-program message switch; see *IMS Intersystem Communication (ISC)* for information about this. This allows an inquiry program to be implemented in an MPP:

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS Writing IMS bridge applications

This topic contains information about writing applications to use the IBM MQ - IMS bridge.

For information about the IBM MQ - IMS bridge, see The IMS bridge.

Use the following links to find out more about writing IMS bridge applications on IBM MQ for z/OS:

- “How the IMS bridge deals with messages” on page 78
- “Writing IMS transaction programs through IBM MQ” on page 960

Related concepts

“Writing IMS applications using IBM MQ” on page 74
There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

z/OS *How the IMS bridge deals with messages*

When you use the IBM MQ - IMS bridge to send messages to an IMS application, you need to construct your messages in a special format.

You must also put your messages on IBM MQ queues that have been defined with a storage class that specifies the XCF group and member name of the target IMS system. These are known as MQ-IMS bridge queues, or simply **bridge** queues.

The IBM MQ-IMS bridge requires exclusive input access (MQOO_INPUT_EXCLUSIVE) to the bridge queue if it is defined with QSGDISP(QMGR), or if it is defined with QSGDISP(SHARED) together with the NOSHARE option.

A user does not need to sign on to IMS before sending messages to an IMS application. The user ID in the *UserIdentifier* field of the MQMD structure is used for security checking. The level of checking is determined when IBM MQ connects to IMS, and is described in Application access control for the IMS bridge. This enables a pseudo signon to be implemented.

The IBM MQ - IMS bridge accepts the following types of message:

- Messages containing IMS transaction data and an MQIIH structure (described in MQIIH):

```
MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
```

Note:

1. The square brackets, [ ], represent optional multi-segments.
2. Set the *Format* field of the MQMD structure to MQFMT_IMS to use the MQIIH structure.

- Messages containing IMS transaction data but no MQIIH structure:

```
LLZZ<trancode><data> \
[LLZZ<data>][LLZZ<data>]
```

IBM MQ validates the message data to ensure that the sum of the LL bytes plus the length of the MQIIH (if it is present) is equal to the message length.

When the IBM MQ - IMS bridge gets messages from the bridge queues, it processes them as follows:

- If the message contains an MQIIH structure, the bridge verifies the MQIIH (see MQIIH), builds the OTMA headers, and sends the message to IMS. The transaction code is specified in the input message. If this is an LTERM, IMS replies with a DFS1288E message. If the transaction code represents a command, IMS executes the command; otherwise the message is queued in IMS for the transaction.
- If the message contains IMS transaction data, but no MQIIH structure, the IMS bridge makes the following assumptions:
  - The transaction code is in bytes 5 through 12 of the user data
  - The transaction is in nonconversational mode
  - The transaction is in commit mode 0 (commit-then-send)
  - The *Format* in the MQMD is used as the *MFSMapName* (on input)
  - The security mode is MQISS_CHECK

  The reply message is also built without an MQIIH structure, taking the *Format* for the MQMD from the *MFSMapName* of the IMS output.

The IBM MQ - IMS bridge uses one or two Tpipes for each IBM MQ queue:

- A synchronized Tpipe is used for all messages using Commit mode 0 (COMMIT_THEN_SEND) (these show with SYN in the status field of the IMS /DIS TMEMBER client TPIPE xxxx command)
- A non-synchronized Tpipe is used for all messages using Commit mode 1 (SEND_THEN_COMMIT)

The Tpipes are created by IBM MQ when they are first used. A non-synchronized Tpipe exists until IMS is restarted. Synchronized Tpipes exist until IMS is cold started. You cannot delete these Tpipes yourself.

See the following topics for more information about how the IBM MQ - IMS bridge deals with messages:

- “Mapping IBM MQ messages to IMS transaction types” on page 80
- “If the message cannot be put to the IMS queue” on page 80
- “IMS bridge feedback codes” on page 80
- “The MQMD fields in messages from the IMS bridge” on page 81
- “The MQIIH fields in messages from the IMS bridge” on page 82
- “Reply messages from IMS” on page 83
- “Using alternate response PCBs in IMS transactions” on page 83
- “Sending unsolicited messages from IMS” on page 83
- “Message segmentation” on page 83
- “Data conversion for messages to and from the IMS bridge” on page 84

Related concepts

“Writing IMS transaction programs through IBM MQ” on page 960
The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

z/OS *Mapping IBM MQ messages to IMS transaction types*

A table describing the mapping of IBM MQ messages to IMS transaction types.

Table 4. How IBM MQ messages map to IMS transaction types

IBM MQ message type	**Commit-then-send (mode 0) - uses synchronized IMS Tpipes**	**Send-then-commit (mode 1) - uses non-synchronized IMS Tpipes**
Persistent IBM MQ messages	• Recoverable full function transactions • Unrecoverable transactions are rejected by IMS	• Fastpath transactions • Conversational transactions • Full function transactions
Nonpersistent IBM MQ messages	• Unrecoverable full function transactions • Recoverable transactions are permitted with IMS V8 and APAR PQ61404 and all later versions of IMS	• Fastpath transactions • Conversational transactions • Full function transactions

Note: IMS commands cannot use persistent IBM MQ messages with commit mode 0. See Commit mode (commitMode) for more information.

z/OS *If the message cannot be put to the IMS queue*

Learn about actions to take if the message cannot be put to the IMS queue.

If the message cannot be put to the IMS queue, the following action is taken by IBM MQ:

- If a message cannot be put to IMS because the message is invalid, the message is put to the dead-letter queue, and a message is sent to the system console.
- If the message is valid, but is rejected by IMS, IBM MQ sends an error message to the system console, the message includes the IMS sense code, and the IBM MQ message is put to the dead-letter queue. If the IMS sense code is 001A, IMS sends an IBM MQ message containing the reason for the failure to the reply-to queue.

  Note: In the circumstances listed previously, if IBM MQ cannot put the message to the dead-letter queue for any reason, the message is returned to the originating IBM MQ queue. An error message is sent to the system console, and no further messages are sent from that queue.

  To resend the messages, do ***one*** of the following:

  – Stop and restart the Tpipes in IMS corresponding to the queue
  – Alter the queue to GET(DISABLED), and again to GET(ENABLED)
  – Stop and restart IMS or the OTMA
  – Stop and restart your IBM MQ subsystem
- If the message is rejected by IMS for anything other than a message error, the IBM MQ message is returned to the originating queue, IBM MQ stops processing the queue, and an error message is sent to the system console.

  If an exception report message is required, the bridge puts it to the reply-to queue with the authority of the originator. If the message cannot be put to the queue, the report message is put to the dead-letter queue with the authority of the bridge. If it cannot be put to the DLQ, it is discarded.

z/OS *IMS bridge feedback codes*

IMS sense codes are typically output in hexadecimal format in IBM MQ console messages such as CSQ2001I (for example, sense code 0x001F). IBM MQ feedback codes as seen in the dead-letter header of messages put to the dead-letter queue are decimal numbers.

The IMS bridge feedback codes are in the range 301 through 399, or 600 through 855 for NACK sense code 0x001A. They are mapped from the IMS-OTMA sense codes as follows:

1. The IMS-OTMA sense code is converted from a hexadecimal number to a decimal number.
2. 300 is added to the number resulting from the calculation in 1, giving the IBM MQ *Feedback* code.
3. The IMS-OTMA sense code 0x001A, decimal 26 is a special case. A *Feedback* code in the range 600-855 is generated.
   a. The IMS-OTMA reason code is converted from a hexadecimal number to a decimal number.
   b. 600 is added to the number resulting from the calculation in a, giving the IBM MQ *Feedback* code.

For information about IMS-OTMA sense codes, see OTMA sense codes for NAK messages.

The MQMD fields in messages from the IMS bridge

Learn about the MQMD fields in messages from the IMS bridge.

The MQMD of the originating message is carried by IMS in the User Data section of the OTMA headers. If the message originates in IMS, this is built by the IMS Destination Resolution Exit. The MQMD of a message received from IMS is built as follows:

StrucID
"MD "

Version
MQMD_VERSION_1

Report
MQRO_NONE

MsgType
MQMT_REPLY

Expiry
If MQIIH_PASS_EXPIRATION is set in the Flags field of the MQIIH, this field contains the remaining expiry time, else it is set to MQEI_UNLIMITED

Feedback
MQFB_NONE

Encoding
MQENC.Native (the encoding of the z/OS system)

CodedCharSetId
MQCCSI_Q_MGR (the CodedCharSetID of the z/OS system)

Format
MQFMT_IMS if the MQMD.Format of the input message is MQFMT_IMS, otherwise IOPCB.MODNAME

Priority
MQMD.Priority of the input message

Persistence
Depends on commit mode: MQMD.Persistence of the input message if CM-1; persistence matches recoverability of the IMS message if CM-0

MsgId
MQMD.MsgId if MQRO_PASS_MSG_ID, otherwise New MsgId (the default)

CorrelId
MQMD.CorrelId from the input message if MQRO_PASS_CORREL_ID, otherwise MQMD.MsgId from the input message (the default)

BackoutCount
0

ReplyToQ
Blanks

ReplyToQMgr
Blanks (set to local qmgr name by the queue manager during the MQPUT)

UserIdentifier
MQMD.UserIdentifier of the input message

AccountingToken
MQMD.AccountingToken of the input message

ApplIdentityData
MQMD.ApplIdentityData of the input message

PutApplType
MQAT_XCF if no error, otherwise MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
Date when message was put

PutTime
Time when message was put

ApplOriginData
Blanks

z/OS *The MQIIH fields in messages from the IMS bridge*

Learn about the MQIIH fields in messages from the IMS bridge.

The MQIIH of a message received from IMS is built as follows:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat of the input message if MQIIH.ReplyToFormat is not blank, otherwise IOPCB.MODNAME

Flags
0

LTermOverride
LTERM name (Tpipe) from OTMA header

MFSMapName
Map name from OTMA header

ReplyToFormat
Blanks

Authenticator
MQIIH.Authenticator of the input message if the reply message is being put to an MQ-IMS bridge queue, otherwise blanks.

TranInstanceId
Conversation ID / Server Token from OTMA header if in conversation. In versions of IMS prior to V14, this field is always nulls if not in conversation. From IMS V14 onwards, this field may be set by IMS even if not in conversation.

TranState
"C" if in conversation, otherwise blank

CommitMode
Commit mode from OTMA header ("0" or "1")

SecurityScope
Blank

Reserved
Blank

z/OS *Reply messages from IMS*

When an IMS transaction ISRTs to its IOPCB, the message is routed back to the originating LTERM or TPIPE.

These are seen in IBM MQ as reply messages. Reply messages from IMS are put onto the reply-to queue specified in the original message. If the message cannot be put onto the reply-to queue, it is put onto the dead-letter queue using the authority of the bridge. If the message cannot be put onto the dead-letter queue, a negative acknowledgment is sent to IMS to say that the message cannot be received. Responsibility for the message is then returned to IMS. If you are using commit mode 0, messages from that Tpipe are not sent to the bridge, and remain on the IMS queue; that is, no further messages are sent until restart. If you are using commit mode 1, other work can continue.

If the reply has an MQIIH structure, its format type is MQFMT_IMS; if not, its format type is specified by the IMS MOD name used when inserting the message.

z/OS *Using alternate response PCBs in IMS transactions*

When an IMS transaction uses alternate response PCBs (ISRTs to the ALTPCB, or issues a CHNG call to a modifiable PCB), the pre-routing exit (DFSYPRX0) is invoked to determine if the message should be rerouted.

If the message is to be rerouted, the destination resolution exit (DFSYDRU0) is invoked to confirm the destination and prepare the header information See Using OTMA exits in IMS and The pre-routing exit DFSYPRX0 for information about these exit programs.

Unless action is taken in the exits, all output from IMS transactions initiated from an IBM MQ queue manager, whether to the IOPCB or to an ALTPCB, will be returned to the same queue manager.

z/OS *Sending unsolicited messages from IMS*

To send messages from IMS to an IBM MQ queue, you need to invoke an IMS transaction that ISRTs to an ALTPCB.

You need to write pre-routing and destination resolution exits to route unsolicited messages from IMS and build the OTMA user data, so that the MQMD of the message can be built correctly. See The pre-routing exit DFSYPRX0 and The destination resolution user exit for information about these exit programs.

Note: The IBM MQ - IMS bridge does not know whether a message that it receives is a reply or an unsolicited message. It handles the message the same way in each case, building the MQMD and MQIIH of the reply based on the OTMA UserData that arrived with the message

Unsolicited messages can create new Tpipes. For example, if an existing IMS transaction switched to a new LTERM (for example PRINT01), but the implementation requires that the output be delivered through OTMA, a new Tpipe (called PRINT01 in this example) is created. By default, this is a non-synchronized Tpipe. If the implementation requires the message to be recoverable, set the destination resolution exit output flag. See the *IMS Customization Guide* for more information.

z/OS *Message segmentation*

You can define IMS transactions as expecting single- or multi-segment input.

The originating IBM MQ application must construct the user input following the MQIIH structure as one or more LLZZ-data segments. All segments of an IMS message must be contained in a single IBM MQ message sent with a single MQPUT.

The maximum length of an LLZZ-data segment is defined by IMS/OTMA (32767 bytes). The total IBM MQ message length is the sum of the LL bytes, plus the length of the MQIIH structure.

All the segments of the reply are contained in a single IBM MQ message.

There is a further restriction on the 32 KB limitation on messages with format MQFMT_IMS_VAR_STRING. When the data in an ASCII-mixed CCSID message is converted to an EBCDIC-mixed CCSID message, a shift-in byte or a shift-out byte is added every time that there is a transition between SBCS and DBCS characters. The 32 KB restriction applies to the maximum size of the message. That is, because the LL field in the message cannot exceed 32 KB, the message must not exceed 32 KB including all shift-in and shift-out characters. The application building the message must allow for this.

z/OS *Data conversion for messages to and from the IMS bridge*

The data conversion is performed by either the distributed queuing facility (which may call any necessary exits) or by the intra group queuing agent (which does not support the use of exits) when it puts a message to a destination queue that has XCF information defined for its storage class. The data conversion does not occur when a message is delivered to a queue by publish/subscribe.

Any exits needed must be available to the distributed queuing facility in the data set referenced by the CSQXLIB DD statement. This means that you can send messages to an IMS application using the IBM MQ - IMS bridge from any IBM MQ platform.

If there are conversion errors, the message is put to the queue unconverted; this results eventually in it being treated as an error by the IBM MQ - IMS bridge, because the bridge cannot recognize the header format. If a conversion error occurs, an error message is sent to the z/OS console.

See “Datenkonvertierungsexits schreiben” on page 1035 for detailed information about data conversion in general.

Sending messages to the IBM MQ - IMS bridge

To ensure that conversion is performed correctly, you must tell the queue manager what the format of the message is.

If the message has an MQIIH structure, the *Format* in the MQMD must be set to the built-in format MQFMT_IMS, and the *Format* in the MQIIH must be set to the name of the format that describes your message data. If there is no MQIIH, set the *Format* in the MQMD to your format name.

If your data (other than the LLZZs) is all character data (MQCHAR), use as your format name (in the MQIIH or MQMD, as appropriate) the built-in format MQFMT_IMS_VAR_STRING. Otherwise, use your own format name, in which case you must also provide a data-conversion exit for your format. The exit must handle the conversion of the LLZZs in your message, in addition to the data itself (but it does not have to handle any MQIIH at the start of the message).

If your application uses *MFSMapName*, you can use messages with the MQFMT_IMS instead, and define the map name passed to the IMS transaction in the MFSMapName field of the MQIIH.

Receiving messages from the IBM MQ - IMS bridge

If an MQIIH structure is present on the original message that you are sending to IMS, one is also present on the reply message.

To ensure that your reply is converted correctly:

- If you have an MQIIH structure on your original message, specify the format that you want for your reply message in the MQIIH *ReplytoFormat* field of the original message. This value is placed in the MQIIH *Format* field of the reply message. This is particularly useful if all your output data is of the form LLZZ<character data>.

- If you do not have an MQIIH structure on your original message, specify the format that you want for the reply message as the MFS MOD name in the IMS application's ISRT to the IOPCB.

JMS/Jakarta Messaging -und Java -Anwendungen entwickeln

IBM MQ stellt drei Java -Sprachschnittstellen bereit: IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSund IBM MQ classes for Java.

Informationen zu diesem Vorgang

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.

Weitere Informationen hierzu finden Sie unter „Gründe für die Verwendung von IBM MQ classes for Jakarta Messaging” auf Seite 87 und „Zugreifen auf IBM MQ von Java -Auswahl der API” auf Seite 90.

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.

Weitere Informationen hierzu finden Sie unter „Gründe für die Verwendung von IBM MQ classes for JMS” auf Seite 88 und „Zugreifen auf IBM MQ von Java -Auswahl der API” auf Seite 90.

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.

IBM MQ classes for Java kapselt die Message Queue Interface (MQI), die native IBM MQ -API, und verwendet dasselbe Objektmodell wie andere objektorientierte Schnittstellen, während IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging Java Messaging-Schnittstellen von Oracle bzw. Java Community Process implementieren.

Weitere Informationen hierzu finden Sie unter „Gründe für die Verwendung von IBM MQ classes for Java” auf Seite 366 und „Zugreifen auf IBM MQ von Java -Auswahl der API” auf Seite 90.

Anmerkung:

Stabilized IBM wird keine weiteren Erweiterungen für die IBM MQ classes for Java durchführen; sie werden auf der in IBM MQ 8.0 ausgelieferten Stufe stabilisiert. Bestehende Anwendungen, die die IBM MQ classes for Java verwenden, werden weiterhin vollständig unterstützt, aber neue Funktionen werden nicht mehr hinzugefügt und Anforderungen für Erweiterungen abgelehnt. Vollständig unterstützt bedeutet, dass Fehler behoben und daraus erforderliche Änderungen an den IBM MQ-Systemvoraussetzungen vorgenommen werden.

Die IBM MQ classes for Java werden in IMS nicht unterstützt.

Die IBM MQ classes for Java werden in WebSphere Liberty nicht unterstützt. Sie dürfen weder mit der Messaging-Funktion von IBM MQ Liberty noch mit der generischen JCA-Unterstützung verwendet werden. Weitere Informationen finden Sie unter Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

IBM MQ classes for JMS/Jakarta Messaging verwenden

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Java -Messaging-Provider, die mit IBM MQbereitgestellt werden. Neben der Implementierung der in den Spezifikationen JMS und

Jakarta Messaging definierten Schnittstellen fügen diese Messaging-Provider zwei Gruppen von Erweiterungen zur Java -Messaging-API hinzu.

JM 3.0 Ab IBM MQ 9.3.0wird Jakarta Messaging 3.0 für die Entwicklung neuer Anwendungen unterstützt. IBM MQ 9.3.0 unterstützt weiterhin JMS 2.0 für vorhandene Anwendungen. Die Verwendung der JMS 2.0 -API und der Jakarta Messaging 3.0 -API in derselben Anwendung wird nicht unterstützt.

Anmerkung: Für Jakarta Messaging 3.0wird die Steuerung der JMS -Spezifikation von Oracle in Java Community Processverschoben. Oracle behält jedoch die Kontrolle über den "javax" -Namen, der in anderen Java -Technologien verwendet wird, die nicht in den Java Community-Prozess verschoben wurden. Obwohl Jakarta Messaging 3.0 funktional äquivalent zu JMS 2.0 ist, gibt es einige Unterschiede bei der Benennung:

- Der offizielle Name für Version 3.0 ist Jakarta Messaging und nicht Java Message Service.
- Die Paket-und Konstantennamen haben das Präfix `jakarta` und nicht `javax`. In JMS 2.0 ist die einleitende Verbindung zu einem Messaging-Provider beispielsweise ein `javax.jms.Connection` -Objekt und in Jakarta Messaging 3.0 ein `jakarta.jms.Connection` -Objekt.

JMS 2.0 Die javax.jms -Pakete definieren die JMS -Schnittstellen, und ein JMS -Provider implementiert diese Schnittstellen für ein bestimmtes Messaging-Produkt. IBM MQ classes for JMS stellen einen JMS-Provider dar, der die JMS-Schnittstellen für IBM MQ implementiert.

JM 3.0 Die jakarta.jms -Pakete definieren die Jakarta Messaging -Schnittstellen und ein Jakarta Messaging -Provider implementiert diese Schnittstellen für ein bestimmtes Messaging-Produkt. IBM MQ classes for Jakarta Messaging stellen einen Jakarta Messaging-Provider dar, der die Jakarta Messaging-Schnittstellen für IBM MQ implementiert.

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.

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.

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.

Darüber hinaus bieten IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API. 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.

IBM MQ JMS-Erweiterungen
IBM MQ classes for JMS enthält Erweiterungen, die in Objekten wie 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.

IBM JMS-Erweiterungen
IBM MQ classes for JMS stellt auch eine allgemeinere Gruppe von Erweiterungen für die JMS -API bereit, die nicht spezifisch für IBM MQ als Messaging-System oder Java als Programmiersprache sind. 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

Die Erweiterungen stellen ähnliche Funktionen bereit wie IBM MQ Message Service Client (XMS) for C/C++ und IBM MQ Message Service Client (XMS) for .NET.

Zugehörige Konzepte

IBM MQ Java-Sprachenschnittstellen

Zugehörige Tasks

„IBM MQ classes for JMS/Jakarta Messaging -Anwendungen schreiben" auf Seite 147

Nach einer kurzen Einführung in das JMS -Modell finden Sie in diesem Abschnitt ausführliche Anleitungen zum Schreiben von IBM MQ classes for JMS -und IBM MQ classes for Jakarta Messaging -Anwendungen.

JM 3.0 Gründe für die Verwendung von IBM MQ classes for Jakarta Messaging

Die IBM MQ classes for Jakarta Messaging bieten mehrere Vorteile. Unter anderem können vorhandene Jakarta Messaging-Kenntnisse in Ihrem Unternehmen wiederverwendet werden und Anwendungen sind unabhängiger vom Jakarta Messaging-Provider und von der zugrunde liegenden IBM MQ-Konfiguration.

Zusammenfassung der Vorteile der Verwendung von IBM MQ classes for Jakarta Messaging

Die Verwendung von IBM MQ classes for Jakarta Messaging ermöglicht Ihnen die Wiederverwendung vorhandener Jakarta Messaging-Kenntnisse und die Bereitstellung von Anwendungsunabhängigkeit.

- Sie können Jakarta Messaging-Kenntnisse wiederverwenden.

  IBM MQ classes for Jakarta Messaging ist ein Jakarta Messaging-Provider, der die Jakarta Messaging-Schnittstellen für IBM MQ als Messaging-System implementiert. Wenn Ihre Organisation neu in IBM MQist, aber bereits über Jakarta Messaging -(oder JMS) Anwendungsentwicklungskenntnisse verfügt, ist es möglicherweise einfacher, die vertraute Jakarta Messaging -API für den Zugriff auf IBM MQ -Ressourcen zu verwenden, anstatt eine der anderen APIs zu verwenden, die mit IBM MQbereitgestellt werden.
- Jakarta Messaging ist ein integraler Bestandteil von Jakarta EE.

  Jakarta Messaging ist die natürliche API für das Messaging auf der Jakarta EE-Plattform. Jeder Anwendungsserver, der mit Jakarta EE konform ist, muss einen Jakarta Messaging-Provider einschließen. Sie können Jakarta Messaging in Anwendungsclients, Servlets, JavaServer Pages (JSPs), Enterprise Java Beans (EJBs) und in Message-driven Beans (MDBs) verwenden. Es ist vor allem zu beachten, dass Jakarta EE-Anwendungen MDBs für die asynchrone Verarbeitung von Nachrichten verwenden und alle Nachrichten als Jakarta Messaging-Nachrichten an MDBs übermittelt werden.
- Verbindungsfactorys und Ziele können als verwaltete Jakarta Messaging-Objekte in einem zentralen Repository gespeichert werden, statt in einer Anwendung fest codiert zu sein.

  Ein Administrator kann verwaltete Jakarta Messaging-Objekte in einem zentralen Repository erstellen und verwalten und IBM MQ classes for Jakarta Messaging-Anwendungen können diese Objekte mithilfe von Java Naming Directory Interface (JNDI) abrufen. Jakarta Messaging-Verbindungsfactorys und Ziele beinhalten IBM MQ-spezifische Informationen wie Warteschlangenmanagernamen, Kanalnamen, Verbindungsoptionen, Warteschlangennamen und Themennamen. Wenn Verbindungsfactorys und Ziele als verwaltete Objekte gespeichert werden, werden diese Informationen nicht fest in einer Anwendung codiert. Dank dieses Konzepts bleibt die Anwendung bis zu einem gewissen Grad unabhängig von der zugrunde liegenden IBM MQ-Konfiguration.
- Jakarta Messaging ist eine dem Branchenstandard entsprechende API, die eine Portierbarkeit von Anwendungen bereitstellen kann.

Eine Jakarta Messaging -Anwendung kann JNDI verwenden, um Verbindungsfactorys und Ziele abzurufen, die als verwaltete Objekte gespeichert sind, und nur die Schnittstellen verwenden, die im Paket `jakarta.jms` (Jakarta Messaging 3.0) definiert sind, um Messaging-Operationen auszuführen. Die Anwendung ist dann völlig unabhängig von einem Jakarta Messaging-Provider wie etwa IBM MQ classes for Jakarta Messaging und kann von einem Jakarta Messaging-Provider auf einen anderen Provider portiert werden, ohne dass die Anwendung geändert werden muss.

Wenn JNDI in einer bestimmten Anwendungsumgebung nicht verfügbar ist, kann eine IBM MQ classes for Jakarta Messaging -Anwendung Erweiterungen für die Jakarta Messaging -API verwenden, um Verbindungsfactorys und Ziele dynamisch zur Laufzeit zu erstellen. Die Anwendung ist dann völlig eigenständig, ist jedoch an IBM MQ classes for Jakarta Messaging als Jakarta Messaging-Provider gebunden.

- Brückenanwendungen lassen sich mit Jakarta Messaging einfacher schreiben.

  Bei einer Brückenanwendung handelt es sich um eine Anwendung, die Nachrichten von einem Messaging-System erhält und diese an ein anderes Messaging-System sendet. Das Schreiben einer Brückenanwendung kann bei Verwendung produktspezifischer APIs und Nachrichtenformate recht kompliziert sein. Sie haben aber die Möglichkeit, eine Brückenanwendung mithilfe von zwei Jakarta Messaging-Providern zu schreiben, also mit jeweils einem Provider pro Messaging-System. Die Anwendung verwendet dann nur eine API (die Jakarta Messaging-API) und verarbeitet nur Jakarta Messaging-Nachrichten.

Bereitstellbare Umgebungen

Für die Integration mit einem Jakarta EE-Anwendungsserver setzen die Jakarta EE-Standards voraus, dass die Messaging-Provider Ressourcenadapter bereitstellen. Gemäß der Spezifikation Jakarta Connectors Architecture stellt IBM MQ einen Ressourcenadapter bereit, der Jakarta Messaging verwendet, um Messaging-Funktionen in einer zertifizierten Jakarta EE -Umgebung bereitzustellen. Weitere Informationen finden Sie unter „Liberty und der IBM MQ-Ressourcenadapter” auf Seite 463.

Anmerkung: WebSphere Application Server traditional unterstützt derzeit nicht Jakarta EE.

Außerhalb der Jakarta EE-Umgebung werden OSGi- und JAR-Dateien bereitgestellt, was es für Sie einfacher macht, nur die IBM MQ classes for Jakarta Messaging abzurufen. Diese JAR-Dateien sind einfacher implementierbar, entweder eigenständig oder in Software-Management-Frameworks wie Maven. Weitere Informationen finden Sie unter „IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging separat abrufen” auf Seite 133.

Zugehörige Konzepte

IBM MQ Classes for Jakarta Messaging: Übersicht

„Zugreifen auf IBM MQ von Java -Auswahl der API” auf Seite 90
IBM MQ stellt drei Java -Sprachschnittstellen bereit.

JMS 2.0 Gründe für die Verwendung von IBM MQ classes for JMS

Die IBM MQ classes for JMS bieten mehrere Vorteile. Unter anderem können vorhandene JMS-Kenntnisse in Ihrem Unternehmen wiederverwendet werden und Anwendungen sind unabhängiger vom JMS-Provider und von der zugrunde liegenden IBM MQ-Konfiguration.

Zusammenfassung der Vorteile der Verwendung von IBM MQ classes for JMS

Die Verwendung von IBM MQ classes for JMS ermöglicht Ihnen die Wiederverwendung vorhandener JMS-Kenntnisse und die Bereitstellung von Anwendungsunabhängigkeit.

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.

- Sie können JMS-Kenntnisse wiederverwenden.

IBM MQ classes for JMS ist ein JMS-Provider, der die JMS-Schnittstellen für IBM MQ als Messaging-System implementiert. Falls IBM MQ in Ihrem Unternehmen bislang noch nicht eingesetzt wurde, Sie aber Kenntnisse im Bereich der JMS-Anwendungsentwicklung besitzen, ist es unter Umständen für Sie einfacher, die vertraute JMS-API anstelle der anderen, mit IBM MQ bereitgestellten APIs für den Zugriff auf die IBM MQ-Ressourcen zu verwenden.

- JMS ist ein integraler Bestandteil von Java Platform, Enterprise Edition (Java EE).

  JMS ist die natürliche API für das Messaging auf der Java EE-Plattform. Jeder Anwendungsserver, der mit Java EE konform ist, muss einen JMS-Provider einschließen. Sie können JMS in Anwendungsclients, Servlets, JavaServer Pages (JSPs), Enterprise Java Beans (EJBs) und in Message-driven Beans (MDBs) verwenden. Es ist vor allem zu beachten, dass Java EE-Anwendungen MDBs für die asynchrone Verarbeitung von Nachrichten verwenden und alle Nachrichten als JMS-Nachrichten an MDBs übermittelt werden.

- Verbindungsfactorys und Ziele können als verwaltete JMS-Objekte in einem zentralen Repository gespeichert werden, statt in einer Anwendung fest codiert zu sein.

  Ein Administrator kann verwaltete JMS-Objekte in einem zentralen Repository erstellen und verwalten und IBM MQ classes for JMS-Anwendungen können diese Objekte mithilfe von Java Naming Directory Interface (JNDI) abrufen. JMS-Verbindungsfactorys und Ziele beinhalten IBM MQ-spezifische Informationen wie Warteschlangenmanagernamen, Kanalnamen, Verbindungsoptionen, Warteschlangennamen und Themennamen. Wenn Verbindungsfactorys und Ziele als verwaltete Objekte gespeichert werden, werden diese Informationen nicht fest in einer Anwendung codiert. Dank dieses Konzepts bleibt die Anwendung bis zu einem gewissen Grad unabhängig von der zugrunde liegenden IBM MQ-Konfiguration.

- JMS ist eine dem Branchenstandard entsprechende API, die eine Portierbarkeit von Anwendungen bereitstellen kann.

  Eine JMS -Anwendung kann mit JNDI Verbindungsfactorys und Ziele abrufen, die als verwaltete Objekte gespeichert sind, und nur die Schnittstellen verwenden, die im Paket `javax.jms` definiert sind, um Messaging-Operationen auszuführen. Die Anwendung ist dann völlig unabhängig von einem JMS-Provider wie etwa IBM MQ classes for JMS und kann von einem JMS-Provider auf einen anderen Provider portiert werden, ohne dass die Anwendung geändert werden muss.

  Falls JNDI in einer bestimmten Anwendungsumgebung nicht verfügbar ist, kann eine Anwendung der IBM MQ classes for JMS mithilfe der JMS-API-Erweiterungen Verbindungsfactorys und Ziele während der Laufzeit dynamisch erstellen und konfigurieren. Die Anwendung ist dann völlig eigenständig, ist jedoch an IBM MQ classes for JMS als JMS-Provider gebunden.

- Brückenanwendungen lassen sich mit JMS einfacher schreiben.

  Bei einer Brückenanwendung handelt es sich um eine Anwendung, die Nachrichten von einem Messaging-System erhält und diese an ein anderes Messaging-System sendet. Das Schreiben einer Brückenanwendung kann bei Verwendung produktspezifischer APIs und Nachrichtenformate recht kompliziert sein. Sie haben aber die Möglichkeit, eine Brückenanwendung mithilfe von zwei JMS-Providern zu schreiben, also mit jeweils einem Provider pro Messaging-System. Die Anwendung verwendet dann nur eine API (die JMS-API) und verarbeitet nur JMS-Nachrichten.

Bereitstellbare Umgebungen

Für die Integration mit einem Java EE-Anwendungsserver setzen die Java EE-Standards voraus, dass die Messaging-Provider Ressourcenadapter bereitstellen. Gemäß Java EE Connector Architecture-(JCA-)Spezifikation bietet IBM MQ einen Ressourcenadapter, der JMS zur Bereitstellung der Messaging-Funktionen innerhalb einer zertifizierten Java EE-Umgebung verwendet.

Während es möglich war, die IBM MQ classes for Java in Java EE zu verwenden, ist diese API weder für diesen Zweck vorbereitet noch dafür geeignet. Weitere Informationen zu IBM MQ classes for Java in Java EEfinden Sie unter „IBM MQ classes for Java-Anwendungen in Java EE ausführen” auf Seite 367.

Außerhalb der Java EE-Umgebung werden OSGi- und JAR-Dateien bereitgestellt, was es für Sie einfacher macht, nur die IBM MQ classes for JMS abzurufen. Diese JAR-Dateien sind einfacher implementierbar, entweder eigenständig oder in Software-Management-Frameworks wie Maven. Weitere Informationen

finden Sie unter „IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging separat abrufen” auf Seite 133.

Zugehörige Konzepte

IBM MQ Classes for Jakarta Messaging: Übersicht

„Gründe für die Verwendung von IBM MQ classes for Jakarta Messaging” auf Seite 87
Die IBM MQ classes for Jakarta Messaging bieten mehrere Vorteile. Unter anderem können vorhandene Jakarta Messaging-Kenntnisse in Ihrem Unternehmen wiederverwendet werden und Anwendungen sind unabhängiger vom Jakarta Messaging-Provider und von der zugrunde liegenden IBM MQ-Konfiguration.

„Zugreifen auf IBM MQ von Java -Auswahl der API” auf Seite 90
IBM MQ stellt drei Java -Sprachschnittstellen bereit.

Zugreifen auf IBM MQ von Java -Auswahl der API

IBM MQ stellt drei Java -Sprachschnittstellen bereit.

- IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

IBM MQ classes for Jakarta Messaging ermöglicht es Anwendungen, die mit den Jakarta Messaging 3.0 -APIs geschrieben wurden, IBM MQ als Messaging-Provider zu verwenden.

Jakarta Messaging ist die strategische Richtung für das Messaging in Java -Anwendungen.

Jakarta Messaging 3.0 ist funktional äquivalent zu JMS 2.0. Weitere Informationen finden Sie unter „IBM MQ classes for JMS/Jakarta Messaging verwenden” auf Seite 85.

IBM MQ classes for JMS

IBM MQ classes for JMS ermöglicht es Anwendungen, die mit den JMS 2.0 -APIs geschrieben wurden, IBM MQ als Messaging-Provider zu verwenden.

Da Jakarta Messaging JMSersetzt wird, wird IBM MQ classes for JMS für die Verwendung in vorhandenen Anwendungen oder in Umgebungen (z. B. WebSphere Application Server) empfohlen, die Jakarta Messagingnicht unterstützen.

Es wird nicht unterstützt, IBM MQ classes for Jakarta Messaging und IBM MQ classes for JMS in derselben Anwendung zu verwenden.

Weitere Informationen finden Sie unter „IBM MQ classes for JMS/Jakarta Messaging verwenden” auf Seite 85.

IBM MQ classes for Java

Stabilized Die andere API, mit der Java -Anwendungen auf IBM MQ -Ressourcen zugreifen können, ist IBM MQ classes for Java, die eine IBM MQ-orientierte API für Programme bereitstellt, die IBM MQ als Messaging-Provider verwenden können. IBM MQ classes for Java ist jedoch funktional auf der Version stabilisiert, die im Lieferumfang von IBM MQ 8.0enthalten ist. Weitere Informationen finden Sie im Abschnitt „Gründe für die Verwendung von IBM MQ classes for Java” auf Seite 366. Obwohl vorhandene Anwendungen, die IBM MQ classes for Java verwenden, weiterhin vollständig unterstützt werden, sollten neue Anwendungen IBM MQ classes for Jakarta Messagingverwenden.

Allgemeine Features von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging bieten Zugriff auf die Punkt-zu-Punkt- und Publish/Subscribe-Messaging-Funktionen von IBM MQ. Anwendungen können damit nicht nur JMS-Nachrichten senden, die das JMS-Standard-Messaging-Modell unterstützen, sondern auch Nachrichten ohne zusätzliche Header senden und empfangen. Auf diese Weise können die Anwendungen auch mit anderen IBM MQ-Anwendungen (z. B. C MQI-Anwendungen) interagieren. Die vollständige Steuerung der MQMD-und MQ -Nachrichtennutzdaten ist verfügbar.

Darüber hinaus sind IBM MQ-Funktionen wie Nachrichten-Streaming, asynchrone Put-Operationen und Berichtsnachrichten verfügbar.

Mit den bereitgestellten PCF-Helper-Klassen können IBM MQ-PCF-Verwaltungsnachrichten über die JMS-API gesendet und empfangen und zur Verwaltung von Warteschlangenmanagern verwendet werden.

Funktionen, die kürzlich zu IBM MQhinzugefügt wurden, wie z. B. asynchrone Verarbeitung und automatische Verbindungswiederholung, sind in IBM MQ classes for Javanicht verfügbar, aber in IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverfügbar.

Erweiterungen anfordern

Wenn Sie Zugriff auf IBM MQ -Funktionen benötigen, die nicht über IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverfügbar sind, können Sie eine Idee erstellen.

IBM kann dann raten, ob die Implementierung in der Implementierung von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging möglich ist oder ob es ein bewährtes Verfahren gibt, das befolgt werden kann.

Da IBM aktiver Mitgestalter des offenen Standards ist, ist es durchaus auch möglich, dass Vorschläge für neue Messaging-Funktionen im Rahmen des JCP-Prozesses behandelt werden. Diese gelten nur für Jakarta Messaging.

Zugehörige Informationen

Willkommen beim IBM Ideenportal

Prüfprozess für JMS- Java -Spezifikation

PCF-Nachrichten mithilfe von JMS senden

JM 3.0 Voraussetzungen für IBM MQ classes for Jakarta Messaging

In diesem Abschnitt erfahren Sie, was Sie vor der Verwendung von IBM MQ classes for Jakarta Messaging wissen müssen. Für die Entwicklung und Ausführung von IBM MQ classes for Jakarta Messaging werden bestimmte Softwarekomponenten vorausgesetzt.

Informationen zu den Voraussetzungen für IBM MQ classes for Jakarta Messaging finden Sie in den Systemvoraussetzungen für IBM MQ.

Um IBM MQ classes for Jakarta Messaging-Anwendungen zu entwickeln, benötigen Sie ein Java SE Software Development Kit (SDK). Details zu den von Ihrem Betriebssystem unterstützten JDKs finden Sie in den Systemvoraussetzungen für IBM MQ.

Für die Ausführung von Anwendungen der IBM MQ classes for Jakarta Messaging benötigen Sie die folgenden Softwarekomponenten:

- Einen IBM MQ-Warteschlangenmanager.
- Eine Java runtime environment (JRE) für jedes System, auf dem Sie Anwendungen ausführen.
- IBM i Qshell für IBM i (Option 30 des Betriebssystems).
- z/OS Für z/OS, z/OS UNIX System Services (z/OS UNIX).

Der IBM-JSSE-Provider enthält einen FIPS-zertifizierten Verschlüsselungsprovider, kann also programmgesteuert für die FIPS 140-2-Konformität konfiguriert werden, die sofort verwendbar ist. Daher kann die Konformität mit FIPS 140-2 direkt von IBM MQ classes for Jakarta Messagingunterstützt werden.

Der JSSE-Provider von Oracle kann über einen FIPS-zertifizierten Verschlüsselungsprovider verfügen, der in ihm konfiguriert ist, dieser ist jedoch nicht sofort einsatzbereit und für die programmgesteuerte Konfiguration nicht verfügbar. Daher kann IBM MQ classes for Jakarta Messaging in diesem Fall die Konformität mit FIPS 140-2 nicht direkt aktivieren. Möglicherweise können Sie die Konformität manuell aktivieren, IBM kann derzeit jedoch keine Anleitung hierzu zur Verfügung stellen.

Sie können Adressen von Internet Protocol Version 6 (IPv6) in Ihren IBM MQ classes for Jakarta Messaging -Anwendungen verwenden, wenn IPv6 -Adressen von Ihrer Java Virtual Machine (JVM) und der TCP/IP-Implementierung auf Ihrem Betriebssystem unterstützt werden. Das IBM MQ Jakarta Messaging -Verwaltungstool **JMS30Admin**akzeptiert auch IPv6 -Adressen. Weitere Informationen über dieses Tool finden Sie unter JMS -und Jakarta Messaging -Objekte mit den Verwaltungstools konfigurieren.

Das IBM MQ JMS-Verwaltungstool und der IBM MQ Explorer verwenden das Java Naming Directory Interface (JNDI) für den Zugriff auf einen Verzeichnisservice, der verwaltete Objekte speichert. IBM MQ classes for Jakarta Messaging-Anwendungen können JNDI auch zum Abrufen verwalteter Objekte aus einem Verzeichnisservice verwenden.

Anmerkung: 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 Service-Provider ist Code, der den Zugriff auf einen Verzeichnisservice bereitstellt, indem JNDI-Aufrufe dem Verzeichnisservice zugeordnet werden. Ein Dateisystemservice-Provider in den Dateien `fscontext.jar` und `providerutil.jar` wird mit IBM MQ classes for Jakarta Messaging geliefert. Der Service-Provider des Dateisystems ermöglicht den Zugriff auf einen Verzeichnisservice auf der Basis des lokalen Dateisystems.

Falls Sie vorhaben, einen Verzeichnisservice auf Basis eines LDAP-Servers zu verwenden, müssen Sie einen LDAP-Server installieren und konfigurieren oder Zugriff auf einen bereits vorhandenen LDAP-Server besitzen. Insbesondere müssen Sie den LDAP-Server zum Speichern von Java-Objekten konfigurieren. In der Dokumentation des Servers finden Sie Informationen zur Installation und Konfiguration Ihres LDAP-Servers.

JMS 2.0 Voraussetzungen für IBM MQ classes for JMS

In diesem Abschnitt erfahren Sie, was Sie vor der Verwendung von IBM MQ classes for JMS wissen müssen. Für die Entwicklung und Ausführung von IBM MQ classes for JMS werden bestimmte Softwarekomponenten vorausgesetzt.

Informationen zu den Voraussetzungen für IBM MQ classes for JMS finden Sie in den Systemvoraussetzungen für IBM MQ.

Um IBM MQ classes for JMS-Anwendungen zu entwickeln, benötigen Sie ein Java SE Software Development Kit (SDK). Details zu den von Ihrem Betriebssystem unterstützten JDKs finden Sie in den Systemvoraussetzungen für IBM MQ.

Für die Ausführung von Anwendungen der IBM MQ classes for JMS benötigen Sie die folgenden Softwarekomponenten:

- Einen IBM MQ-Warteschlangenmanager.
- Eine Java runtime environment (JRE) für jedes System, auf dem Sie Anwendungen ausführen.
- IBM i Qshell für IBM i (Option 30 des Betriebssystems).
- z/OS Für z/OS, z/OS UNIX System Services (z/OS UNIX).

Der IBM-JSSE-Provider enthält einen FIPS-zertifizierten Verschlüsselungsprovider, kann also programmgesteuert für die FIPS 140-2-Konformität konfiguriert werden, die sofort verwendbar ist. Deshalb kann die Konformität mit FIPS 140-2 direkt von IBM MQ classes for Java und IBM MQ classes for JMS unterstützt werden.

Der JSSE-Provider von Oracle kann über einen FIPS-zertifizierten Verschlüsselungsprovider verfügen, der in ihm konfiguriert ist, dieser ist jedoch nicht sofort einsatzbereit und für die programmgesteuerte Konfiguration nicht verfügbar. Deshalb können die IBM MQ classes for Java und IBM MQ classes for JMS die FIPS 140-2-Konformität in diesem Fall nicht direkt aktivieren. Möglicherweise können Sie die Konformität manuell aktivieren, IBM kann derzeit jedoch keine Anleitung hierzu zur Verfügung stellen.

Sie können Adressen von Internet Protocol Version 6 (IPv6) in Ihren IBM MQ classes for JMS -Anwendungen verwenden, wenn IPv6 -Adressen von Ihrer Java Virtual Machine (JVM) und der TCP/IP-Implementierung auf Ihrem Betriebssystem unterstützt werden. Das IBM MQ-JMS-Verwaltungstool (siehe JMS-Objekte mithilfe des Verwaltungstools konfigurieren) akzeptiert ebenfalls IPv6-Adressen.

Das IBM MQ JMS-Verwaltungstool und der IBM MQ Explorer verwenden das Java Naming Directory Interface (JNDI) für den Zugriff auf einen Verzeichnisservice, der verwaltete Objekte speichert. IBM MQ classes for JMS-Anwendungen können JNDI auch zum Abrufen verwalteter Objekte aus einem Verzeichnisservice verwenden. Ein Service-Provider ist Code, der den Zugriff auf einen Verzeichnisservice bereitstellt, indem JNDI-Aufrufe dem Verzeichnisservice zugeordnet werden. Ein Dateisystemservice-Provider in den Dateien `fscontext.jar` und `providerutil.jar` wird mit IBM MQ classes for JMS geliefert. Der Provider des Dateisystemservice ermöglicht den Zugriff auf einen Verzeichnisservice auf Basis des lokalen Dateisystems.

Falls Sie vorhaben, einen Verzeichnisservice auf Basis eines LDAP-Servers zu verwenden, müssen Sie einen LDAP-Server installieren und konfigurieren oder Zugriff auf einen bereits vorhandenen LDAP-Server besitzen. Insbesondere müssen Sie den LDAP-Server zum Speichern von Java-Objekten konfigurieren. In der Dokumentation des Servers finden Sie Informationen zur Installation und Konfiguration Ihres LDAP-Servers.

IBM MQ classes for JMS/Jakarta Messaging installieren und konfigurieren

In diesem Abschnitt werden die Verzeichnisse und Dateien beschrieben, die bei der Installation von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingerstellt werden. Außerdem erfahren Sie, wie Sie IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging nach der Installation konfigurieren.

Zugehörige Konzepte

„IBM MQ-Ressourcenadapter verwenden” auf Seite 457
Der Ressourcenadapter ermöglicht Anwendungen, die auf einem Anwendungsserver ausgeführt werden, den Zugriff auf IBM MQ-Ressourcen. Er unterstützt sowohl die eingehende Kommunikation als auch die abgehende Kommunikation.

Dies wird für IBM MQ classes for JMS installiert

Bei der Installation von IBM MQ classes for JMS werden verschiedene Dateien und Verzeichnisse installiert. Unter Windows erfolgt ein Teil der Konfiguration während der Installation durch die automatische Festlegung von Umgebungsvariablen. Auf anderen Plattformen und in bestimmten Windows-Umgebungen müssen Sie im Vorfeld Umgebungsvariablen festlegen, damit Sie Anwendungen der IBM MQ classes for JMS ausführen können.

Bei den meisten Betriebssystemen werden die IBM MQ classes for JMS als optionale Komponente bei der Installation von IBM MQ installiert.

Weitere Informationen zur Installation von IBM MQ finden Sie in folgenden Abschnitten:

Multi IBM MQ installieren

z/OS IBM MQ for z/OS installieren

Wichtig: Mit Ausnahme der verschiebbaren JAR-Dateien, die im Abschnitt „IBM MQ classes for JMS/Jakarta Messaging verschiebbare JAR-Dateien” auf Seite 95 beschrieben werden, wird das Kopieren der JAR-Dateien von IBM MQ classes for JMS oder der nativen Bibliotheken auf andere Maschinen oder an eine andere Position auf einer Maschine, auf der die IBM MQ classes for JMS installiert wurden, nicht unterstützt.

Installationsverzeichnisse

In Tabelle 5 auf Seite 94 sehen Sie, wo die Dateien der IBM MQ classes for JMS auf den einzelnen Plattformen installiert werden.

Tabelle 5. Installationsverzeichnisse von IBM MQ classes for JMS

Plattform	Directory
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/java
Windows Windows	*MQ_INSTALLATION_PATH*\java
IBM i IBM i	/QIBM/ProdData/mqm/java
z/OS z/OS	*MQ_INSTALLATION_PATH*/java

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Das Installationsverzeichnis schließt den folgenden Inhalt ein:

- Die JAR-Dateien der IBM MQ classes for JMS, einschließlich der verschiebbaren JAR-Dateien, die sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib befinden.
- Die nativen Bibliotheken von IBM MQ, die von Anwendungen verwendet werden, welche die Java Native Interface nutzen.

  Die nativen 32-Bit-Bibliotheken werden im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib installiert, während die nativen 64-Bit-Bibliotheken im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib64 zu finden sind.

  Weitere Informationen zu den nativen IBM MQ-Bibliotheken finden Sie im Abschnitt „Java Native Interface-(JNI-)Bibliotheken konfigurieren" auf Seite 101.
- Zusätzliche Scripts (diese werden im Abschnitt „Mit IBM MQ classes for JMS/Jakarta Messaging bereitgestellte Scripts" auf Seite 129 beschrieben). Diese Scripts befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\bin.
- Die Spezifikation der API von IBM MQ classes for JMS. Für die Generierung der HTML-Seiten, die die Spezifikationen der API enthalten, wurde das Javadoc-Tool verwendet.

  Die HTML-Seiten befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses:

  – ALW Unter AIX, Linux, and Windows enthält dieses Unterverzeichnis die einzelnen HTML-Seiten.

  – IBM i Unter IBM i befinden sich die HTML-Seiten in einer Datei namens wmqjms_javadoc.jar.

  – z/OS Unter z/OS befinden sich die HTML-Seiten in einer Datei namens wmqjms_javadoc.jar.
- Unterstützung von OGSi. OSGi-Bundles werden im Verzeichnis java\lib\OSGi installiert und sind im Abschnitt „Unterstützung für OSGi mit IBM MQ classes for JMS" auf Seite 130beschrieben.
- Der IBM MQ -Ressourcenadapter, der in jedem Java Platform, Enterprise Edition 7 (Java EE 7) oder Jakarta EE -konformen Anwendungsserver implementiert werden kann

  Der IBM MQ-Ressourcenadapter befindet sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib\jca; weitere Informationen enthält der Abschnitt „IBM MQ-Ressourcenadapter verwenden" auf Seite 457.
- Windows Unter Windows werden Symbole, die für das Debugging verwendet werden können, im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib\symbols installiert.

Darüber hinaus enthält das Installationsverzeichnis einige Dateien, die zu sonstigen IBM MQ-Komponenten gehören.

Beispielanwendungen

JMS 2.0 Mit IBM MQ classes for JMS werden einige Beispielanwendungen bereitgestellt. In Tabelle 6 auf Seite 95 sehen Sie, wo die Beispielanwendungen auf den einzelnen Plattformen installiert werden.

JM 3.0 Für IBM MQ classes for Jakarta Messagingwerden neue Beispiele vorbereitet.

JMS 2.0

Tabelle 6. Beispielverzeichnisse für IBM MQ classes for JMS

Plattform	Directory
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/samp/jms
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms
IBM i IBM i	/QIBM/ProdData/mqm/java/samples/jms
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

In dieser Tabelle steht *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Nach der Installation müssen Sie möglicherweise einige Konfigurationsaufgaben ausführen, um Anwendungen zu kompilieren und auszuführen.

„Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98 beschreibt den Klassenpfad, der für die Ausführung von IBM MQ classes for JMS -Beispielanwendungen erforderlich ist. In diesem Abschnitt werden außerdem zusätzliche JAR-Dateien beschrieben, die unter bestimmten Umständen referenziert werden müssen. Außerdem erfahren Sie, welche Umgebungsvariablen Sie für die Ausführung der Scripts festlegen müssen, die mit IBM MQ classes for JMS bereitgestellt werden.

Zur Steuerung von Eigenschaften wie die Traceverarbeitung und Protokollierung bei einer Anwendung müssen Sie eine Konfigurationseigenschaftendatei bereitstellen. Die Konfigurationseigenschaftendatei der IBM MQ classes for JMS wird im Abschnitt „Die Konfigurationsdatei IBM MQ classes for JMS/Jakarta Messaging” auf Seite 103 beschrieben.

Zugehörige Konzepte

Probleme beim Implementieren des Ressourcenadapters

Zugehörige Tasks

„IBM MQ classes for JMS-Beispielanwendungen verwenden” auf Seite 125

Die Beispielanwendungen der IBM MQ classes for JMS bieten einen Überblick über die allgemeinen Funktionen der JMS-API. Mithilfe der Beispielanwendungen können Sie Ihre Installation und Messaging-Server-Konfiguration überprüfen sowie Ihre eigenen Anwendungen erstellen.

IBM MQ classes for JMS/Jakarta Messaging verschiebbare JAR-Dateien

Die verschiebbaren JAR-Dateien können auf Systeme verschoben werden, auf denen IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingausgeführt werden müssen.

Wichtig:

- Abgesehen von den verlagerbaren JAR-Dateien, die im Abschnitt Verlagerbare JAR-Dateienbeschrieben sind, wird das Kopieren der IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -JAR-Dateien oder nativen Bibliotheken auf andere Maschinen oder an eine andere Position auf einer

Maschine, auf der IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging installiert wurde, nicht unterstützt.

- Schließen Sie die verschiebbaren JAR-Dateien nicht in Anwendungen ein, die auf Java EE-Anwendungsservern, z. B. WebSphere Application Server oder WebSphere Liberty, bereitgestellt werden. In diesen Umgebungen sollte stattdessen der IBM MQ-Ressourcenadapter bereitgestellt und verwendet werden. Beachten Sie, dass WebSphere Application Server den IBM MQ-Ressourcenadapter einbettet, sodass er nicht manuell in dieser Umgebung bereitgestellt werden muss.
- Um Klassenladeprogrammkonflikte zu vermeiden, sollten die verschiebbaren JAR-Dateien nicht in mehreren Anwendungen innerhalb derselben Java-Runtime gebündelt werden. Machen Sie in diesem Szenario die verschiebbaren JAR-Dateien von IBM MQ im Klassenpfad der Java -Laufzeit verfügbar.
- Wenn Sie die verschiebbaren JAR-Dateien in Ihren Anwendungen bündeln, stellen Sie sicher, dass alle vorausgesetzten JAR-Dateien eingeschlossen sind (siehe Beschreibung in Verschiebbare JAR-Dateien). Sie sollten auch sicherstellen, dass Sie über geeignete Prozeduren zum Aktualisieren der gebündelten JAR-Dateien im Rahmen der Anwendungswartung verfügen, um sicherzustellen, dass die IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging immer noch aktuell sind und bekannte Probleme erneut vermittelt werden.

Verschiebbare JAR-Dateien

Innerhalb eines Unternehmens können die folgenden Dateien auf Systeme verschoben werden, auf denen IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingausgeführt wird:

- `bcpkix-jdk15to18.jar` „4" auf Seite 96
- V 9.4.0 `bcpkix-jdk18on.jar` „3" auf Seite 96
- `bcprov-jdk15to18.jar` „4" auf Seite 96
- V 9.4.0 `bcprov-jdk18on.jar` „3" auf Seite 96
- `bcutil-jdk15to18.jar` „4" auf Seite 96
- V 9.4.0 `bcutil-jdk18on.jar` „3" auf Seite 96
- JMS 2.0 `com.ibm.mq.allclient.jar` „1" auf Seite 96
- JM 3.0 `com.ibm.mq.jakarta.client.jar` „2" auf Seite 96
- `fscontext.jar`
- `jakarta.jms-api.jar`
- `jms.jar`
- `org.json.jar`
- `providerutil.jar`

Anmerkungen:

1. JMS 2.0 und JMS 1.1
2. Jakarta Messaging 3.0
3. Von IBM MQ 9.4.0
4. VorherIBM MQ 9.4.0

JMS JAR-Dateien

`jms.jar` enthält die Schnittstellen JMS 1.1 und JMS 2.0 , die als `javax.jms.*`bezeichnet werden.

JM 3.0 `jakarta.jms-api.jar` enthält die Jakarta Messaging 3.0 -Schnittstellen, die als `jakarta.jms.*`bezeichnet werden.

fscontext.jar und providerutil.jar

Die Dateien `fscontext.jar` und `providerutil.jar` sind erforderlich, wenn Ihre Anwendung JNDI-Suchvorgänge unter Verwendung eines Dateisystemkontextes ausführt.

Bouncy Castle-Sicherheitsprovider und JAR-Dateien für CMS -Unterstützung

Der Sicherheitsprovider Bouncy Castle und die JAR-Dateien für CMS-Unterstützung sind erforderlich. Weitere Informationen enthält der Abschnitt Unterstützung für JREs anderer Anbieter als IBM bei AMS.

V 9.4.0 Die folgenden JAR-Dateien sind erforderlich:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

org.json.jar

Die Datei `org.json.jar` ist erforderlich, wenn Ihre IBM MQ classes for JMS-Anwendung eine Clientkanaldefinitionstabelle (Client Canal Definition Table, CCDT) im JSON-Format verwendet.

com.ibm.mq.allclient.jar und com.ibm.mq.jakarta.client.jar

Die Dateien `com.ibm.mq.allclient.jar` und `com.ibm.mq.jakarta.client.jar` enthalten die Klassen IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging, IBM MQ classes for Javasowie die PCF-und Headerklassen. Wenn Sie diese JAR-Datei an eine neue Position verschieben, müssen Sie sicherstellen, dass diese neue Position mit den neuen IBM MQ -Fixpacks beibehalten wird. Stellen Sie außerdem sicher, dass die Verwendung der Dateien dem IBM Support bekannt ist, wenn Sie einen vorläufigen Fix erhalten.

Verwenden Sie den folgenden Befehl, um die Version der Dateien `com.ibm.mq.allclient.jar` und `com.ibm.mq.jakarta.client.jarzu` ermitteln:

JM 3.0
```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0
```
java -jar com.ibm.mq.allclient.jar
```

Im folgenden Beispiel sehen Sie einen Beispielauszug der Ausgabe dieses Befehls:

```
C:\Programme\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
```

```
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen
Damit Sie IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Anwendungen kompilieren und ausführen können, muss die Einstellung für Ihre **CLASSPATH** -Umgebungsvariable die JAR-Datei (IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging Java -Archiv) enthalten. Abhängig von den gegebenen Voraussetzungen müssen Sie Ihrem Klassenpfad möglicherweise weitere JAR-Dateien hinzufügen. Zum Ausführen der Scripts, die mit IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingbereitgestellt werden, müssen weitere Umgebungsvariablen festgelegt werden.

Vorbereitende Schritte

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.

Wichtig: Die Einstellung der Java -Option *-Xbootclasspath* auf IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging wird nicht unterstützt.

Informationen zu diesem Vorgang

Verwenden Sie zum Kompilieren und Ausführen von IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Anwendungen die Einstellung **CLASSPATH** für Ihre Plattform und Java -Messaging-Version, wie in den folgenden Tabellen dargestellt. Alternativ zur Verwendung der Umgebungsvariablen können Sie den Klassenpfad auch im **java**-Befehl angeben.

JMS 2.0 Für IBM MQ classes for JMSenthält die Einstellung das Beispielverzeichnis, damit Sie die IBM MQ classes for JMS -Beispielanwendungen kompilieren und ausführen können.

JM 3.0 Für IBM MQ classes for Jakarta Messagingwerden neue Beispiele vorbereitet.

JM 3.0

*Tabelle 7. **CLASSPATH** -Einstellungen für Jakarta Messaging 3.0 zum Kompilieren und Ausführen von IBM MQ classes for Jakarta Messaging -Anwendungen*

Plattform	CLASSPATH festlegen
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

*Tabelle 8. **CLASSPATH** -Einstellungen für JMS 2.0 zum Kompilieren und Ausführen von IBM MQ classes for JMS -Anwendungen, einschließlich der Beispielanwendungen*	
Plattform	**CLASSPATH-Einstellung**
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/java/samples/jms/samples:

In diesen Tabellen steht *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Das Manifest der JAR-Datei `com.ibm.mq.jakarta.client.jar` oder `com.ibm.mq.allclient.jar` enthält Verweise auf die meisten anderen JAR-Dateien, die für IBM MQ classes for JMS -Anwendungen erforderlich sind, sodass Sie diese JAR-Dateien Ihrem Klassenpfad nicht hinzufügen müssen. Diese JAR-Dateien umfassen diejenigen, die von Anwendungen benötigt werden, welche Java Naming Directory Interface (JNDI) für den Abruf von verwalteten Objekten aus einem Verzeichnisservice verwenden, sowie diejenigen für Anwendungen, die die Java Transaction API (JTA) verwenden.

In folgenden Situationen müssen Sie jedoch zusätzliche JAR-Dateien in Ihren Klassenpfad aufnehmen:

- Wenn Sie Kanalexitschnittstellen verwenden, die anstelle der im Paket `com.ibm.mq.exits` definierten Kanalexitschnittstellen die im Paket `com.ibm.mq` definierten Kanalexitschnittstellen implementieren, müssen Sie dem Klassenpfad die JAR-Datei `com.ibm.mq.jar` von IBM MQ classes for Java hinzufügen.
- Wenn Ihre Anwendung verwaltete Objekte über JNDI aus einem Verzeichnisservice abruft, müssen Sie dem Klassenpfad außerdem noch die folgenden JAR-Dateien hinzufügen:
  - `fscontext.jar`
  - `providerutil.jar`
- Wenn Ihre Anwendung die Java Transaction API nutzt, müssen Sie Ihrem Klassenpfad zusätzlich die JAR-Datei `jta.jar` hinzufügen.

Anmerkung: Diese zusätzlichen JAR-Dateien sind allerdings nur für die Kompilierung Ihrer Anwendungen erforderlich, nicht für deren Ausführung.

Die Scripts, die mit IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging bereitgestellt werden, verwenden die folgenden Umgebungsvariablen:

MQ_JAVA_DATA_PATH
Diese Umgebungsvariable gibt das Verzeichnis für die Protokoll- und Traceausgabe an.

MQ_JAVA_INSTALL_PATH
Diese Umgebungsvariable gibt das Verzeichnis an, in dem IBM MQ classes for JMS installiert ist.

MQ_JAVA_LIB_PATH
Diese Umgebungsvariable gibt das Verzeichnis an, in dem die IBM MQ classes for JMS -Bibliotheken gespeichert sind, wie in den vorherigen Tabellen gezeigt.

Prozedur

- Windows

  Führen Sie unter Windowsnach der Installation von IBM MQden Befehl **setmqenv**aus.

  Wenn Sie diesen Befehl nicht zuerst ausführen, wird möglicherweise die folgende Fehlernachricht angezeigt, wenn Sie einen **dspmqver** -Befehl absetzen:

  ```
  AMQ8351: IBM MQ Java environment has not been configured
  correctly, or the IBM MQ JRE feature has not been installed.
  ```

  Anmerkung: Diese Nachricht ist zu erwarten, wenn Sie IBM MQ Java runtime environment (JRE) nicht installiert haben (siehe Prüfung der Voraussetzungen für zusätzliche Windows-Funktionen).

- Linux AIX

  Legen Sie auf AIX and Linux -Systemen die Umgebungsvariablen selbst fest:

  JMS 2.0 Verwenden Sie für JMS 2.0eines der folgenden Scripts, um die Umgebungsvariablen festzulegen:

  – Wenn Sie eine 32-Bit-JVM verwenden, verwenden Sie das Script `setjmsenv`.
  – Wenn Sie eine 64-Bit-JVM auf einem AIX -oder Linux -System verwenden, verwenden Sie das Script `setjmsenv64`.

  JM 3.0 Verwenden Sie für Jakarta Messaging 3.0eines der folgenden Scripts, um die Umgebungsvariablen festzulegen:

  – Wenn Sie eine 32-Bit-JVM verwenden, verwenden Sie das Script `setjms30env`.
  – Wenn Sie eine 64-Bit-JVM verwenden, verwenden Sie das Script `setjms30env64`.

  Diese Scripts sind im Verzeichnis *MQ_INSTALLATION_PATH*`/java/bin` enthalten; dabei ist *MQ_INSTALLATION_PATH* das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

  Sie können diese Scripts auf verschiedene Arten verwenden. Sie können das Script als Basis für die Festlegung der erforderlichen Umgebungsvariablen verwenden, wie in der Tabelle gezeigt, oder sie mithilfe eines Texteditors zu `.profile` hinzufügen. Handelt es sich um eine vom Standard abweichende Konfiguration, müssen Sie den Inhalt des Scripts entsprechend bearbeiten. Alternativ dazu können Sie auch das Script in jeder Sitzung ausführen, von der aus die JMS-Startscripts ausgeführt werden sollen. Wenn Sie diese Option auswählen, müssen Sie das Script in jedem Shellfenster ausführen, das Sie während des JMS -Prüfprozesses starten:

  – JMS 2.0 Geben Sie für JMS 2.0 `. ./setjmsenv` oder `../setjmsenv64`ein.
  – JM 3.0 Geben Sie für Jakarta Messaging 3.0 `. ./setjms30env` oder `. ./setjms30env64`ein.

  IBM i Unter IBM imüssen Sie die Umgebungsvariable **QIBM_MULTI_THREADED** auf Ysetzen. Sie können dann Multithread-Anwendungen genau wie bei der Ausführung von Einzelthread-Anwendungen ausführen. Weitere Informationen finden Sie unter IBM MQ mit Java und JMS einrichten.

Zugehörige Tasks

„IBM MQ classes for JMS-Beispielanwendungen verwenden” auf Seite 125
Die Beispielanwendungen der IBM MQ classes for JMS bieten einen Überblick über die allgemeinen Funktionen der JMS-API. Mithilfe der Beispielanwendungen können Sie Ihre Installation und Messaging-Server-Konfiguration überprüfen sowie Ihre eigenen Anwendungen erstellen.

Zugehörige Verweise

„Mit IBM MQ classes for JMS/Jakarta Messaging bereitgestellte Scripts” auf Seite 129
Es wird eine Reihe von Scripts bereitgestellt, die Sie bei allgemeinen Tasks unterstützen, die ausgeführt werden müssen, wenn Sie IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverwenden.

Java Native Interface-(JNI-)Bibliotheken konfigurieren
Anwendungen der IBM MQ classes for JMS, die entweder mit dem Bindungstransport oder Clienttransport eine Verbindung zu einem Warteschlangenmanager herstellen und Kanalexitprogramme verwenden, die in anderen Sprachen als Java geschrieben sind, müssen in einer Umgebung ausgeführt werden, die den Zugriff auf die JNI-Bibliotheken (JNI = Java Native Interface) ermöglicht.

Vorbereitende Schritte

Weitere Informationen zur Verwendung der WebSphere Application Server-Umgebung finden Sie unter IBM MQ-Messaging-Provider mit Informationen zu nativen Bibliotheken konfigurieren.

Informationen zu diesem Vorgang

Um diese Umgebung einzurichten, müssen Sie den Bibliothekspfad der Umgebung so konfigurieren, dass die Java Virtual Machine (JVM) die Bibliothek mqjbnd laden kann, bevor Sie die IBM MQ classes for JMS-Anwendung starten.

IBM MQ stellt zwei JNI-Bibliotheken (Java Native Interface) zur Verfügung:

mqjbnd
Diese Bibliothek wird von Anwendungen verwendet, die mithilfe des Bindungstransports eine Verbindung zu einem Warteschlangenmanager herstellen. Sie stellt die Schnittstelle zwischen den IBM MQ classes for JMS und dem Warteschlangenmanager bereit. Die zusammen mit IBM MQ 9.4 installierte Bibliothek 'mqjbnd' kann zum Herstellen einer Verbindung zu jedem beliebigen Warteschlangenmanager für IBM MQ 9.4 (oder früher) verwendet werden.

mqjexitstub02
Die Bibliothek mqjexitstub02 wird von den IBM MQ classes for JMS geladen, wenn eine Anwendung über den Clienttransport eine Verbindung zu einem Warteschlangenmanager herstellt und ein Kanalexitprogramm verwendet, das in einer anderen Sprache als Java geschrieben ist.

Auf bestimmten Plattformen installiert IBM MQ 32-Bit- und 64-Bit-Versionen dieser JNI-Bibliotheken. Die Position der Bibliotheken für die einzelnen Plattformen wird in Tabelle 1 gezeigt.

Tabelle 9. Position der Bibliotheken mit IBM MQ classes for JMS bei den einzelnen Plattformen

Plattform	**Verzeichnis mit den IBM MQ classes for JMS-Bibliotheken**
AIX AIX Linux Linux (Plattformen POWER, x86-64 und zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
Windows Windows	*MQ_INSTALLATION_PATH*\Java\lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*\Java\lib64 (64-Bit-Bibliotheken)
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/lib (31-Bit- und 64-Bit-Bibliotheken)

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Anmerkung: z/OS Unter z/OS können Sie entweder eine 31-Bit- oder eine 64-Bit-JVM (Java Virtual Machine) verwenden. Sie müssen nicht angeben, welche JNI-Bibliotheken verwendet werden sollen; IBM MQ classes for JMS können selbst feststellen, welche JNI-Bibliotheken geladen werden müssen.

Vorgehensweise

1. Konfigurieren Sie die JVM-Eigenschaft **`java.library.path`**, was auf zwei Arten möglich ist:
   - Durch Angabe des JVM-Argument, wie im folgenden Beispiel gezeigt:

     ```
     -Djava.library.path=path_to_library_directory
     ```

     Linux Geben Sie beispielsweise für eine 64-Bit-JVM unter Linux für eine Standardinstallation Folgendes an:

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - Indem Sie die Umgebung der Shell so konfigurieren, dass die JVM einen eigenen Wert für `java.library.path` konfiguriert. Dieser Pfad variiert abhängig von der Plattform und von der Position, an der Sie IBM MQ installiert haben. Für eine 64-Bit-JVM und eine IBM MQ-Standardinstallationsposition können Sie beispielsweise folgende Einstellungen verwenden:

     AIX
     ```
     export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
     ```

     Linux
     ```
     export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH
     ```

     Windows
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

   Es folgt ein Beispiel für den Ausnahmebedingungsstack, den Sie sehen, wenn die Umgebung nicht ordnungsgemäß konfiguriert wurde:

   ```
   Caused by: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
   AMQ8598: Failed to load the WebSphere MQ native JNI library: 'mqjbnd'.
       at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
       at com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
       at java.security.AccessController.doPrivileged(AccessController.java:400)
       at com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
       at com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
       at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
       at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
       at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
       at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.ja-
   va:86)
       at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccesso-
   rImpl.java:58)
       at java.lang.reflect.Constructor.newInstance(Constructor.java:542)
       at com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
       at com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
       at com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQCon-
   nectionFactory.java:8437)
       ... 7 more
   Caused by: java.lang.UnsatisfiedLinkError: mqjbnd (Not found in java.library.path)
       at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
       at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
       at java.lang.System.loadLibrary(System.java:534)
       at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
       ... 20 more
   ```

2. Starten Sie, nachdem entweder die 32-Bit- oder die 64-Bit-Umgebung konfiguriert wurde, die IBM MQ classes for JMS-Anwendung mit folgendem Befehl:

   ```
   java application-name
   ```

   Dabei steht *Anwendungsname* für den Namen der IBM MQ classes for JMS-Anwendung, die ausgeführt werden soll.

In folgenden Fällen wird eine Ausnahmebedingung mit dem IBM MQ-Ursachencode 2495 (MQRC_MODULE_NOT_FOUND) von den IBM MQ classes for JMS ausgelöst:

- Die Anwendung der IBM MQ classes for JMS wird in einer Java runtime environment mit 32 Bit ausgeführt und es wurde eine 64-Bit-Umgebung für die IBM MQ classes for JMS eingerichtet, da die 32-Bit-Version der Java runtime environment die 64-Bit-Version von Java Native Library nicht laden kann.
- Die Anwendung der IBM MQ classes for JMS wird in einer Java runtime environment mit 64 Bit ausgeführt und es wurde eine 32-Bit-Umgebung für die IBM MQ classes for JMS eingerichtet, da die 64-Bit-Version der Java runtime environment die 32-Bit-Version von Java Native Library nicht laden kann.

Die Konfigurationsdatei IBM MQ classes for JMS/Jakarta Messaging
Eine IBM MQ classes for JMS -und IBM MQ classes for Jakarta Messaging -Konfigurationsdatei gibt Eigenschaften an, die für die Konfiguration von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverwendet werden.

Anmerkung: Die in der Konfigurationsdatei definierten Eigenschaften können auch als JVM-Systemeigenschaften festgelegt werden. Wenn eine Eigenschaft sowohl in der Konfigurationsdatei als auch als Systemeigenschaft festgelegt ist, hat die Systemeigenschaft Vorrang. Deshalb können Sie, falls erforderlich, jede Eigenschaft in der Konfigurationsdatei überschreiben, indem Sie sie im **java**-Befehl als Systemeigenschaft angeben.

Das Format einer IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Konfigurationsdatei ist das einer Java -Standardeigenschaftendatei. Eine Beispielkonfigurationsdatei mit dem Namen `jms.config` wird im Unterverzeichnis `bin` des IBM MQ classes for JMS -Installationsverzeichnisses bereitgestellt. In dieser Datei sind alle unterstützten Eigenschaften und ihre Standardwerte dokumentiert.

Sie können den Namen und die Position einer Konfigurationsdatei für IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging auswählen. Verwenden Sie beim Start Ihrer Anwendung einen **java**-Befehl in folgendem Format:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

Im Befehl ist *config_file_url* eine URL (Uniform Resource Locator), die den Namen und die Position der Konfigurationsdatei IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging angibt. Es werden URLs der folgenden Typen unterstützt: HTTP, file, FTP und JAR.

Hier ein Beispiel für einen **java**-Befehl:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

Dieser Befehl gibt die IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Konfigurationsdatei als Datei `D:\mydir\mjms.config` auf dem lokalen Windows -System an.

Wenn eine Anwendung gestartet wird, liest IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging den Inhalt der Konfigurationsdatei und speichert die angegebenen Eigenschaften in einem internen Eigenschaftsspeicher. Wenn der Befehl **java** keine Konfigurationsdatei angibt oder die Konfigurationsdatei nicht gefunden werden kann, verwendet IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging die Standardwerte für alle Eigenschaften.

Eine IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Konfigurationsdatei kann mit allen unterstützten Transporten zwischen einer Anwendung und einem Warteschlangenmanager oder Broker verwendet werden.

In einer IBM MQ MQI client-Konfigurationsdatei angegebene Eigenschaften überschreiben

Eine Konfigurationsdatei IBM MQ MQI client kann auch Eigenschaften angeben, die für die Konfiguration von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingverwendet werden. Die in

einer Konfigurationsdatei für den IBM MQ MQI client angegebenen Eigenschaften finden jedoch nur dann Anwendung, wenn sich eine Anwendung im Clientmodus mit einem Warteschlangenmanager verbindet.

Bei Bedarf können Sie jedes Attribut in einer Konfigurationsdatei IBM MQ MQI client überschreiben, indem Sie es als Eigenschaft in einer Konfigurationsdatei IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging angeben. Wenn Sie ein Attribut in einer IBM MQ MQI client -Konfigurationsdatei überschreiben möchten, verwenden Sie einen Eintrag im folgenden Format in der Konfigurationsdatei IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging :

```
com.ibm.mq.cfg. stanza. propName = propValue
```

Die Variablen in dem Eintrag haben folgende Bedeutungen:

Zeilengruppe
Der Name der Zeilengruppe in der Konfigurationsdatei für den IBM MQ MQI client, die das Attribut enthält.

propName
Der Name des in der Konfigurationsdatei für den IBM MQ MQI client angegebenen Attributs

propValue
Der Wert der Eigenschaft, der den Wert des Attributs überschreibt, das in der Konfigurationsdatei für den IBM MQ MQI client angegeben ist

Alternativ können Sie ein Attribut in einer IBM MQ MQI client-Konfigurationsdatei überschreiben, indem Sie die Eigenschaft als Systemeigenschaft im Befehl **java** angeben. Verwenden Sie das vorherige Format zur Angabe der Eigenschaft als Systemeigenschaft.

Nur die folgenden Attribute in einer IBM MQ MQI client -Konfigurationsdatei sind für IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingrelevant. Wenn Sie andere Attribute angeben oder außer Kraft setzen, hat dies keine Wirkung. Hierbei muss insbesondere beachtet werden, dass die Attribute `ChannelDefinitionFile` und `ChannelDefinitionDirectory` in der Zeilengruppe CHANNELS der Clientkonfigurationsdatei nicht verwendet werden. Details zur Verwendung der CCDT mit IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingfinden Sie unter „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden” auf Seite 296 .

Tabelle 10. Welche Zeilengruppe der Clientkonfigurationsdatei enthält welches Attribut

Zeilengruppe	**Attribut**
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	Put1DefaultAlwaysSync
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	DefRecon
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	ReconDelay
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	PasswordProtection
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	Standardpfad für Exits
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	ExitsDefaultPath64
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	JavaExitsClasspath
Zeilengruppe JMQI der Clientkonfigurationsdatei	useMQCSPauthentication
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	MaximumSize

Tabelle 10. Welche Zeilengruppe der Clientkonfigurationsdatei enthält welches Attribut (Forts.)	
Zeilengruppe	**Attribut**
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	PurgeTime
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	UpdatePercentage
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntRcvBufSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntSndBufSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	Connect_Timeout
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	KeepAlive

Weitere Informationen zur Konfiguration von IBM MQ MQI client finden Sie in der IBM MQ MQI client -Konfigurationsdatei unter `mqclient.ini` .

Java Standard Environment Trace zum Konfigurieren des JMS -Trace verwenden
Mithilfe der Zeilengruppe Java Standard Environment Trace Settings können Sie die Tracefunktion von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging konfigurieren.

`com.ibm.msg.client.commonservices.trace.outputName` = *NameTraceausgabe*
`traceOutputName` ist das Verzeichnis und der Dateiname, an die die Traceausgabe gesendet wird.

Standardmäßig werden Traceinformationen in eine Tracedatei im aktuellen Arbeitsverzeichnis der Anwendung geschrieben. Der Name der Tracedatei ist von der Umgebung abhängig, in der die Anwendung ausgeführt wird:

- JM 3.0 Wenn die Anwendung ab IBM MQ 9.3.0 IBM MQ classes for Jakarta Messaging aus der verschiebbaren JAR-Datei `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) oder IBM MQ classes for JMS aus der verschiebbaren JAR-Datei `com.ibm.mq.allclient.jar` (JMS 2.0) geladen hat, wird der Trace in eine Datei mit dem Namen `mqjavaclient_`*`%PID%`*`.cl`*`%u`*`.trc`geschrieben.
- Wenn die Anwendung die IBM MQ classes for JMS aus der verschiebbaren JAR-Datei `com.ibm.mq.allclient.jar`geladen hat, wird der Trace in eine Datei mit dem Namen `mqjavaclient_`*`%PID%`*`.cl`*`%u`*`.trc`geschrieben.
- Wenn die Anwendung IBM MQ classes for JMS aus der JAR-Datei `com.ibm.mqjms.jar`geladen hat, wird der Trace in eine Datei mit dem Namen `mqjava_`*`%PID%`*`.cl`*`%u`*`.trc`geschrieben.

Dabei steht *%PID%* für die Prozess-ID der Anwendung, für die ein Trace durchgeführt wird, *%u* ist eine eindeutige Zahl zur Unterscheidung der Dateien zwischen Threads, die den Trace unter verschiedenen Java-Klassenladeprogrammen ausführen.

Wenn eine Prozess-ID nicht verfügbar ist, wird eine Zufallszahl generiert und mit dem Buchstaben `f` als Präfix versehen. Wenn Sie die Prozess-ID in einen von Ihnen festgelegten Dateinamen einschließen möchten, verwenden Sie die Zeichenfolge `%PID%`.

Wenn Sie ein anderes Verzeichnis angeben, muss dieses bereits vorhanden sein und Sie müssen über Schreibzugriff auf dieses Verzeichnis verfügen. Wenn Sie nicht über Schreibzugriff verfügen, wird die Traceausgabe in die Datei `System.err` geschrieben.

`com.ibm.msg.client.commonservices.trace.include` = *includeList*
`includeList` ist eine Liste von Paketen und Klassen, für die ein Trace erstellt wird, oder die Sonderwerte `ALL` oder `NONE`.

Die Paket- oder Klassennamen müssen durch ein Semikolon (`;`) getrennt werden. *`includeList`* nimmt standardmäßig den Wert `ALL`an und verfolgt alle Pakete und Klassen in IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging.

Anmerkung: Es ist möglich, ein Paket einzubeziehen, anschließend aber Unterpakete dieses Pakets auszuschließen. Wenn Sie beispielsweise das Paket `a.b` einbeziehen und das Paket `a.b.x` ausschließen, umfasst der Trace alles in `a.b.y` und `a.b.z`, nicht jedoch `a.b.x` oder `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList ist eine Liste von Paketen und Klassen, für die kein Trace erstellt wird, oder die Sonderwerte `ALL` oder `NONE`.

Die Paket- oder Klassennamen müssen durch ein Semikolon (`;`) getrennt werden. *excludeList* nimmt standardmäßig den Wert `NONE`an und schließt daher keine Pakete oder Klassen in IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging von der Traceerstellung aus.

Anmerkung: Sie haben die Möglichkeit, ein Paket auszuschließen, anschließend aber Unterpakete des Pakets einzubeziehen. Wenn Sie beispielsweise das Paket `a.b` ausschließen und das Paket `a.b.x` einbeziehen, umfasst der Trace alles in `a.b.x` und `a.b.x.1`, nicht jedoch `a.b.y` oder `a.b.z`.

Alle Pakete oder Klassen, die auf derselben Ebene sowohl als einbezogen als auch als ausgeschlossen angegeben sind, werden einbezogen.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxBytesFeldgruppe*
maxArrayBytes ist die maximale Anzahl Byte, für die ein Trace von einem beliebigen Byte-Array erstellt wird.

Wenn *maxArrayBytes* auf eine positive ganze Zahl gesetzt wird, begrenzt dies die Anzahl Byte in einem Byte-Array, die in die Tracedatei geschrieben werden. Die Bytefeldgruppe wird abgeschnitten, nachdem *maxArrayBytes* ausgegeben wurde. Die Einstellung *maxArrayBytes* verringert die Größe der resultierenden Tracedatei und die Auswirkungen der Traceerstellung auf die Leistung der Anwendung.

Wenn für diese Eigenschaft der Wert `0` angegeben ist, werden keine Inhalte von Bytefeldgruppen an die Tracedatei gesendet.

Der Standardwert lautet `-1`. Bei dieser Einstellung wird die Anzahl der Bytes in einer Bytefeldgruppe, die an die Tracedatei gesendet werden, nicht begrenzt.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes ist die maximale Anzahl von Bytes, die in eine Traceausgabedatei geschrieben werden.

maxTraceBytes funktioniert mit *traceCycles*. Nähert sich die Anzahl der Bytes des geschriebenen Trace dem Grenzwert, wird die Datei geschlossen und es wird eine neue Traceausgabedatei begonnen.

Bei Angabe des Werts `0` haben Traceausgabedateien die Länge Null. Der Standardwert lautet `-1`. In diesem Fall ist das Datenvolumen, das in eine Traceausgabedatei geschrieben werden kann, nicht begrenzt.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles ist die Anzahl der Traceausgabedateien, die durchlaufen werden sollen.

Wenn die aktuelle Traceausgabedatei den Grenzwert erreicht, der über *maxTraceBytes* angegeben wurde, wird die Datei geschlossen. Die weitere Traceausgabe wird in die nächste Traceausgabedatei in der Folge geschrieben. Jede Traceausgabedatei ist durch ein eigenes numerisches Suffix am Ende des Dateinamens gekennzeichnet. Die aktuelle oder jüngste Traceausgabedatei lautet `mqjms.trc.0`, die unmittelbar vorherige Traceausgabedatei `mqjms.trc.1`. Bis zum Erreichen des Höchstwerts wird dieses Nummerierungsmuster auch auf die älteren Traceausgabedateien angewandt.

Der Standardwert von *traceCycles* ist 1. Wenn *traceCycles* 1 ist und die aktuelle Traceausgabedatei ihre maximale Größe erreicht, wird die Datei geschlossen und gelöscht. Anschließend wird eine neue Traceausgabedatei mit demselben Namen begonnen. Somit ist immer nur eine Traceausgabedatei vorhanden.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters steuert, ob Methodenparameter und Rückgabewerte in den Trace eingeschlossen werden.

traceParameters ist standardmäßig TRUE. Wird *traceParameters* auf FALSE gesetzt, werden nur Methodensignaturen aufgezeichnet.

com.ibm.msg.client.commonservices.trace.startup = *Start*
Es gibt eine Initialisierungsphase von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging , in der Ressourcen zugeordnet werden. Während der Phase der Ressourcenzuordnung wird die Haupttracefunktion initialisiert.

Wenn *startup* auf TRUEgesetzt ist, wird der Starttrace verwendet. Es werden unverzüglich Traceinformationen erstellt, die die Einrichtung aller Komponenten, einschließlich der Tracefunktion selbst, umfassen. Mithilfe der Starttraceinformationen lassen sich Konfigurationsprobleme diagnostizieren. Die Starttraceinformationen werden immer in die Datei `System.err` geschrieben.

startup ist standardmäßig FALSE.

startup wird vor Abschluss der Initialisierung geprüft. Geben Sie die Eigenschaft daher nur in der Befehlszeile als Java-Systemeigenschaft an. Geben Sie sie nicht in der Konfigurationsdatei IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging an.

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Setzen Sie *compressedTrace* auf TRUE, um die Traceausgabe zu komprimieren.

Der Standardwert von *compressedTrace* ist FALSE.

Wenn *compressedTrace* auf TRUEgesetzt ist, wird die Traceausgabe komprimiert. Die standardmäßige Traceausgabedatei hat die Erweiterung `.trz`. Wenn für die Komprimierung der Standardwert FALSE festgelegt ist, hat die Datei die Erweiterung `.trc`. Damit wird angegeben, dass die Datei nicht komprimiert ist. Wenn jedoch der Dateiname für die Traceausgabe in *traceOutputName* angegeben wurde, wird stattdessen dieser Name verwendet; auf die Datei wird kein Suffix angewendet.

Die komprimierte Traceausgabe ist kleiner als die nicht komprimierte Ausgabe. Da weniger Ein-/Ausgaben enthalten sind, ist eine schnellere Ausgabe möglich als beim nicht komprimierten Trace. Die komprimierte Traceerstellung hat weniger Auswirkungen auf die Leistung von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging als die nicht komprimierte Traceerstellung.

Wenn *maxTraceBytes* und *traceCycles* festgelegt sind, werden mehrere komprimierte Tracedateien anstelle mehrerer Flachdateien erstellt.

Wenn IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging unkontrolliert beendet wird, ist eine komprimierte Tracedatei möglicherweise nicht gültig. Aus diesem Grund darf die Tracekomprimierung nur verwendet werden, wenn IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging kontrolliert geschlossen wird. Verwenden Sie die Tracekomprimierung nur, wenn die untersuchten Probleme nicht zu einer unerwarteten Beendigung der JVM selbst führen. Die Tracekomprimierung darf nicht genutzt werden, wenn Probleme diagnostiziert werden, die zu Beendigungen des Typs `System.Halt()` oder zu nicht normalen, unkontrollierten JVM-Beendigungen führen können.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel gibt eine Filterstufe für den Trace an Die definierten Tracestufen lauten wie folgt:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`
- `TRACE_INFO`: `6`
- `TRACE_ENTRYEXIT`: `8`
- `TRACE_DATA`: `9`
- `TRACE_ALL`: `Integer.MAX_VALUE`

Jede Tracestufe umfasst alle niedrigeren Stufen. Ist die Tracestufe beispielsweise auf `TRACE_INFO` gesetzt, werden alle Tracepunkte mit der definierten Stufe `TRACE_EXCEPTION`, `TRACE_WARNING` oder `TRACE_INFO` in den Trace geschrieben. Alle anderen Tracepunkte sind ausgeschlossen.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

Über *standaloneTrace* wird gesteuert, ob der Clienttraceservice der IBM MQ JMS in einer WebSphere Application Server-Umgebung verwendet wird.

Ist *standaloneTrace* auf TRUE gesetzt, wird die Tracekonfiguration anhand der Clienttraceeigenschaften der IBM MQ JMS bestimmt.

Wenn *standaloneTrace* auf FALSE gesetzt ist und der Client der IBM MQ JMS in einem WebSphere Application Server-Container ausgeführt wird, wird der WebSphere Application Server-Trace-Service verwendet. Welche Traceinformationen generiert werden, hängt von den Traceeinstellungen des Anwendungsservers ab.

Der Standardwert von *standaloneTrace* ist FALSE.

Zeilengruppe für die Protokollierung
Über die Zeilengruppe für die Protokollierung können Sie die Protokollierungseinrichtung der IBM MQ classes for JMS konfigurieren.

Folgende Eigenschaften können in die Zeilengruppe für die Protokollierung aufgenommen werden:

com.ibm.msg.client.commonservices.log.outputName = Pfad
Der Name der Protokolldatei, die von der Protokollierungseinrichtung der IBM MQ classes for JMS verwendet wird. Der Standardwert lautet `mqjms.log`, dabei wird die Datei in das aktuelle Arbeitsverzeichnis für die Java Runtime Environment geschrieben, in der die IBM MQ classes for JMS ausgeführt werden.

Für die Eigenschaft sind folgende Werte möglich:

- ein einzelner Pfadname
- eine durch Kommas getrennte Liste von Pfadnamen (alle Daten werden in allen Dateien protokolliert)

Bei den einzelnen Pfadnamen kann es sich um absolute oder relative Pfadnamen handeln, es kommen aber auch folgende Angaben in Frage:

"stderr" oder "System.err"
Steht für den Standardfehlerdatenstrom.

"stdout" oder "System.out"
Steht für den Standardausgabedatenstrom.

com.ibm.msg.client.commonservices.log.maxBytes
Die maximale Anzahl von Bytes, die von jedem Aufruf zur Protokollierung von Nachrichtendaten protokolliert werden.

Positive ganze Zahl
Für jeden Logaufruf werden Daten bis zu der angegebenen Bytezahl geschrieben.

0
Es werden keine Daten geschrieben.

-1
Es werden unbegrenzt Daten geschrieben (Standardeinstellung).

com.ibm.msg.client.commonservices.log.limit
Die maximale Anzahl von Bytes, die in eine einzelne Protokolldatei geschrieben werden (Standardeinstellung 262144).

Positive ganze Zahl
Für jede Protokolldatei werden Daten bis zu der angegebenen Bytezahl geschrieben.

0
Es werden keine Daten geschrieben.

-1
Es werden unbegrenzt Daten geschrieben.

com.ibm.msg.client.commonservices.log.count
Die Anzahl der zu durchlaufenden Protokolldateien. Wenn die einzelnen Dateien den Wert von `com.ibm.msg.client.commonservices.trace.limit` erreichen, beginnt der Trace in der nächsten Datei. Standardwert ist 3.

Positive ganze Zahl
Die Anzahl der zu durchlaufenden Dateien.

0
Eine einzelne Datei.

Zeilengruppe für Java-SE-Spezifikationen
Über die Zeilengruppe für Java-SE-Spezifikationen können Sie Eigenschaften konfigurieren, die verwendet werden, wenn die IBM MQ classes for JMS in einer Java Standard Edition-Umgebung eingesetzt werden.

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE | FALSE
Legt fest, ob eine JavaCore-Datei direkt nach der Generierung einer FDC-Datei durch IBM MQ classes for JMS geschrieben wird. Ist dieser Parameter auf TRUE gesetzt, wird eine JavaCore-Datei im Arbeitsverzeichnis der Java Runtime Environment erstellt, in der die IBM MQ classes for JMS ausgeführt werden.

TRUE
JavaCore erstellen, sofern dies in der Java Runtime Environment möglich ist.

FALSE
Kein JavaCore erstellen; dies ist der Standardwert.

Zeilengruppe für IBM MQ-Eigenschaften
Über die Zeilengruppe für IBM MQ-Eigenschaften können Sie Eigenschaften festlegen, die sich auf die Interaktion der IBM MQ classes for JMS mit IBM MQ beziehen.

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
Wenn eine Anwendung, die die IBM MQ classes for JMS verwendet, über den IBM MQ-Messaging-Provider-Migrationsmodus eine Verbindung zu einem IBM MQ-Warteschlangenmanager herstellt, verwenden die IBM MQ classes for JMS eine Standardpuffergröße von 4 KB, wenn sie Nachrichten erhalten. Wenn die Anwendung versucht, eine Nachricht mit mehr als 4 KB abzurufen, wird die Größe des Puffers von den IBM MQ classes for JMS so angepasst, dass die Nachricht aufgenommen werden kann. Die größere Puffergröße wird dann auch verwendet, wenn nachfolgende Nachrichten erhalten werden.

Über diese Eigenschaft wird gesteuert, wann die Puffergröße wieder auf 4 KB reduziert wird. Standardmäßig wird die Puffergröße wieder auf 4 KB reduziert, wenn zehn aufeinanderfolgende Nachrichten erhalten werden, die kleiner als die größere Puffergröße sind. Legen Sie für die Eigenschaft den Wert 0 fest, wenn die Puffergröße jedes Mal, wenn eine Nachricht erhalten wird, wieder auf 4 KB zurückgesetzt werden soll.

0

Der Puffer wird immer auf die Standardgröße zurückgesetzt.

10

Dies ist der Standardwert. Die Puffergröße wird nach der zehnten Nachricht angepasst.

com.ibm.msg.client.wmq.receiveConversionCCSID
Wenn eine Anwendung, die IBM MQ classes for JMS verwendet, eine Verbindung zu einem IBM MQ-Warteschlangenmanager im normalen Modus des IBM MQ-Messaging-Providers herstellt, kann die Eigenschaft `receiveConversionCCSID` festgelegt werden, um den CCSID-Standardwert in der MQMD-Struktur zu überschreiben, die zum Empfangen von Nachrichten vom Warteschlangenmanager verwendet wird. Standardmäßig enthält die MQMD-Struktur ein auf 1208 gesetztes CCSID-Feld, dieses kann jedoch geändert werden, wenn beispielsweise der Warteschlangenmanager nicht in der Lage ist, Nachrichten in diese Codepage zu konvertieren.

Gültige Werte sind alle gültigen CCSID-Nummern oder einer der folgenden Werte:

-1
Standardwert der Plattform verwenden.

1208
Dies ist der Standardwert.

Zeilengruppe für Spezifikationen des Clientmodus
Über die Zeilengruppe für Spezifikationen des Clientmodus können Sie Eigenschaften angeben, die verwendet werden, wenn die IBM MQ classes for JMS eine Verbindung zu einem Warteschlangenmanager herstellen, der das CLIENT-Transportprotokoll verwendet.

com.ibm.mq.polling.RemoteRequestEntry
Gibt das Abfrageintervall an, das von den IBM MQ classes for JMS zur Prüfung auf unterbrochene Verbindungen verwendet wird, wenn auf eine Antwort eines Warteschlangenmanagers gewartet wird.

Positive ganze Zahl
Die Anzahl der Millisekunden, die vor der Prüfung gewartet werden soll. Der Standardwert lautet 10000, dies entspricht 10 Sekunden. Der Mindestwert ist 3000, niedrigere Werte werden wie dieser Mindestwert behandelt.

Zur Konfiguration des JMS-Clientverhaltens verwendete Eigenschaften
Mit diesen Eigenschaften können Sie das Verhalten des JMS-Clients konfigurieren.

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE
Mit der JMS 2.0-Spezifikation werden Änderungen bestimmter Verhaltensweisen eingeführt. In IBM MQ 8.0 gibt es die Eigenschaft `com.ibm.mq.jms.SupportMQExtensions`, die auf *TRUE* gesetzt werden kann, um diese geänderten Verhaltensweisen wieder auf ihre vorherige Implementierung zurückzusetzen. Das Zurücksetzen des geänderten Verhaltens kann für JMS 2.0 -Anwendungen und für einige Anwendungen, die die JMS 1.1 -API verwenden, aber für IBM MQ 8.0 IBM MQ classes for JMSausgeführt werden, erforderlich sein.

TRUE
Die folgenden drei Funktionalitätsbereiche werden zurückgesetzt, wenn SupportMQExtensions auf TRUE gesetzt wird:

Nachrichtenpriorität
Den Nachrichten kann eine Priorität von *0* - *9* zugewiesen werden. Vor JMS 2.0 konnte für Nachrichten auch der Wert *-1* verwendet werden, der bedeutete, dass die Standardpriorität der Warteschlange verwendet wird. JMS 2.0 lässt eine Nachrichtenpriorität von *-1* nicht zu. Wenn `SupportMQExtensions` aktiviert wird, kann der Wert *-1* verwendet werden.

Client-ID
In der JMS 2.0-Spezifikation müssen Client-IDs ungleich null auf Eindeutigkeit geprüft werden, wenn sie eine Verbindung herstellen. Bei Aktivierung von `SupportMQExtensions` wird diese Anforderung ignoriert, d.h., eine Client-ID kann wiederverwendet werden.

`NoLocal`
Wenn diese Konstante aktiviert ist, darf gemäß der JMS 2.0-Spezifikation ein Konsument keine Nachrichten empfangen, die von derselben Client-ID veröffentlicht werden. Vor JMS 2.0 wurde dieses Attribut für einen Subskribenten definiert, um zu verhindern, dass er Nachrichten empfängt, die von seiner eigenen Verbindung veröffentlicht werden. Durch Aktivierung von `SupportMQExtensions` wird diese Verhaltensweise wieder auf die vorherige Implementierung zurückgesetzt.

FALSE
Die Änderungen der Verhaltensweisen werden beibehalten.

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE
Ab IBM MQ 8.0.0 Fix Pack 2kann IBM MQ classes for JMS , nachdem eine Anwendung eine Byte- oder Datenstromnachricht gesendet hat, den Status der Nachricht, die gerade gesendet wurde, auf "Schreibgeschützt" oder "Schreibgeschützt" setzen.

TRUE
Die Objekte werden nach dem Senden auf schreibgeschützt gesetzt. Bei Einstellung dieses Werts bleibt die Kompatibilität mit der JMS 2.0-Spezifikation erhalten.

FALSE
Die Objekte werden nach dem Senden auf lesegeschützt gesetzt. Dies ist der Standardwert.

Zugehörige Konzepte

„Eigenschaft SupportMQExtensions" auf Seite 344
Die Spezifikation JMS 2.0 hat Änderungen an der Funktionsweise bestimmter Verhaltensweisen eingeführt. IBM MQ 8.0 und höher enthält die Eigenschaft **com.ibm.mq.jms.SupportMQExtensions**, die auf TRUE gesetzt werden kann, um diese geänderten Verhaltensweisen auf frühere Implementierungen zurückzusetzen.

STEPLIB configuration for IBM MQ classes for JMS on z/OS
On z/OS, the STEPLIB used at run time must contain the IBM MQ SCSQAUTH and SCSQANLE libraries. Specify these libraries in the startup JCL or using the `.profile` file.

From z/OS UNIX System Services, you can add these using a line in your `.profile` as shown in the following code snippet, replacing `thlqual` with the high-level data set qualifier that you chose when installing IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In other environments, you typically need to edit the startup JCL to include SCSQAUTH and SCSQANLE on the STEPLIB concatenation:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS und Softwareverwaltungstools
Softwareverwaltungstools wie Apache Maven können mit IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverwendet werden.

Viele große Entwicklungsunternehmen verwenden diese Tools, um Repositorys von Bibliotheken anderer Anbieter zentral zu verwalten.

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging setzen sich aus einer Reihe von JAR-Dateien zusammen. Wenn Sie über diese API Java-Sprachanwendungen entwickeln, muss auf dem System, auf dem die Anwendung entwickelt wird, eine Installation eines IBM MQ-Servers, IBM MQ-Clients oder IBM MQ-Client-SupportPacs vorhanden sein.

Wenn Sie ein solches Tool verwenden und die JAR-Dateien, die die IBM MQ classes for JMS bilden, zu einem zentral verwalteten Repository hinzufügen möchten, müssen folgende Punkte beachtet werden:

- Ein Repository oder Container muss nur für die Entwickler in Ihrem Unternehmen verfügbar gemacht werden. Jegliche Verteilung außerhalb des Unternehmens ist nicht zulässig.
- Das Repository muss eine vollständige und durchgängige Gruppe von JAR-Dateien aus einem einzelnen IBM MQ-Release oder -Fixpack enthalten.
- Es ist Ihre Aufgabe, das Repository mit allen vom IBM Support zur Verfügung gestellten Wartungsreleases zu aktualisieren.

Die folgenden JAR-Dateien müssen im Repository installiert werden:

- JMS 2.0 `com.ibm.mq.allclient.jar` und `jms.jar` sind erforderlich, wenn Sie IBM MQ classes for JMSverwenden.
- JM 3.0 `com.ibm.mq.jakarta.client.jar` und `jakarta.jms-api.jar` sind erforderlich, wenn Sie IBM MQ classes for Jakarta Messagingverwenden.

- `fscontext.jar` ist erforderlich, wenn Sie IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging verwenden und auf verwaltete JMS -Objekte zugreifen, die in einem Dateisystem-JNDI-Kontext gespeichert sind.
- `providerutil.jar` ist erforderlich, wenn Sie IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging verwenden und auf verwaltete JMS -Objekte zugreifen, die in einem Dateisystem-JNDI-Kontext gespeichert sind.
- Der Bouncy Castle-Sicherheitsprovider und die JAR-Dateien für die CMS -Unterstützung sind für die Unterstützung von JREs anderer Anbieter alsIBM erforderlich. Weitere Informationen finden Sie im Abschnitt Unterstützung für JREs anderer Anbieter.

IBM MQ classes for JMS-Anwendungen unter dem Java security manager ausführen

IBM MQ classes for JMS können mit aktiviertem Java security manager ausgeführt werden. Um Anwendungen erfolgreich mit aktiviertem Java security manager ausführen zu können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtlinienkonfigurationsdatei konfigurieren.

Die einfachste Möglichkeit, eine geeignete Richtliniendefinitionsdatei zu erstellen, besteht darin, die mit Ihrer Java runtime environment (JRE) bereitgestellte Richtlinienkonfigurationsdatei zu ändern. Auf den meisten Systemen ist diese Datei im Verzeichnis `lib/security/java.policy` (relativ zum JRE-Verzeichnis) gespeichert. Sie können die Richtlinienkonfigurationsdatei entweder mit Ihrem bevorzugten Editor oder mit dem Programm policytool, das mit Ihrer JRE bereitgestellt wurde, bearbeiten.

Beispiel einer Richtlinienkonfigurationsdatei

Es folgt das Beispiel einer Richtlinienkonfigurationsdatei, die eine erfolgreiche Ausführung der IBM MQ classes for JMS unter dem Standardsicherheitsmanager ermöglicht. Diese Datei muss angepasst werden, um die Positionen bestimmter Dateien und Verzeichnisse anzugeben: *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist, *MQ_DATA_DIRECTORY* steht für die Position des MQ-Datenverzeichnisses und *QM_NAME* für den Namen des Warteschlangenmanagers, für den der Zugriff konfiguriert wird.

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";

  // We'd like to set up an mBean to control trace
  permission javax.management.MBeanServerPermission "createMBeanServer";
  permission javax.management.MBeanPermission "*","*";

  // We need to be able to read manifests etc from the jar files in the installation directory
  permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

  //For the client transport type.
  permission java.net.SocketPermission "*","connect,resolve";

  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";

  //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
```

```
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

   //For applications that use User Exits
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
   permission java.lang.RuntimePermission "createClassLoader";

   //Required for the z/OS platform
   permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

   // Used by the internal ConnectionFactory implementation
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

   // Used for controlled class loading
   permission java.lang.RuntimePermission "setContextClassLoader";

   // Used to default the Application name in Client mode connections
   permission java.util.PropertyPermission "sun.java.command","read";

   // Used by the IBM JSSE classes
   permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

   //Required to determine if an IBM Java Runtime is running in FIPS mode,
   //and to modify the property values status as required.
   permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
   permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
   //Required if an IBM FIPS provider is to be used for SSL communication.
   permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

   // Required for non-IBM Java Runtimes that establish secure client
   // transport mode connections using mutual TLS authentication
   permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
   permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

In diesem Beispiel enthält die Anweisung `grant` die Berechtigungen, die von IBM MQ classes for JMS benötigt werden. Wenn Sie diese grant-Anweisungen in Ihrer Richtlinienkonfigurationsdatei verwenden möchten, müssen Sie möglicherweise die Pfadnamen ändern, um sie an das Installationsverzeichnis Ihrer IBM MQ classes for JMS und an die Speicherposition Ihrer Anwendungen anzupassen.

Die mit den IBM MQ classes for JMS bereitgestellten Beispielanwendungen und die Scripts zu deren Ausführung aktivieren den Sicherheitsmanager nicht.

Wichtig:

Für die Tracefunktion von IBM MQ classes for JMS sind weitere Berechtigungen erforderlich, da sie zusätzliche Abfragen von Systemeigenschaften und auch weitere Dateisystemoperationen ausführt.

Eine geeignete Vorlagendatei für Sicherheitsrichtlinien zur Ausführung unter einem Sicherheitsmanager mit aktivierter Tracefunktion wird im Verzeichnis `samples/wmqjava` der IBM MQ-Installation unter dem Namen `example.security.policy` bereitgestellt.

Konfiguration für Anwendungen der IBM MQ classes for JMS nach der Installation

In diesem Abschnitt erfahren Sie welche Berechtigungen von Anwendungen der IBM MQ classes for JMS benötigt werden, damit sie auf die Ressourcen eines Warteschlangenmanagers zugreifen können. Darüber hinaus werden Verbindungsmodi vorgestellt und es wird beschrieben, wie Sie einen Warteschlangenmanager so konfigurieren, dass Anwendungen im Clientmodus eine Verbindung herstellen können.

Vergessen Sie nicht, die Readme-Datei von IBM MQ zu überprüfen. Sie enthält möglicherweise Informationen, die die Informationen in diesem Abschnitt ersetzen.

Von JMS verwendete Objekte, die eine Berechtigung für nicht privilegierte Benutzer erfordern

Nicht privilegierten Benutzern muss eine Berechtigung erteilt werden, damit sie auf die von JMS verwendeten Warteschlangen zugreifen können. Jede JMS-Anwendung benötigt eine Berechtigung für den Warteschlangenmanager, mit dem sie arbeitet.

Sie finden Details zur Zugriffssteuerung in IBM MQ im Abschnitt Sicherheit einrichten.

Anwendungen der IBM MQ classes for JMS benötigen die Berechtigungen `connect` und `inq` für den Warteschlangenmanager. Mit dem Steuerbefehl **setmqaut** können Sie entsprechende Berechtigungen festlegen. Beispiel:

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

Für die Punkt-zu-Punkt-Domäne werden die folgenden Berechtigungen benötigt:

- Warteschlangen, die von MessageProducer-Objekten verwendet werden, benötigen die Berechtigung `put`.
- Warteschlangen, die von MessageConsumer- und QueueBrowser-Objekten verwendet werden, benötigen die Berechtigungen `get`, `inq` und `browse`.
- Die Methode QueueSession.createTemporaryQueue() benötigt Zugriff auf die Modellwarteschlange, die mit der Eigenschaft TEMPMODEL des QueueConnectionFactory-Objekts angegeben wurde. Diese Modellwarteschlange ist standardmäßig SYSTEM.TEMP.MODEL.QUEUE.

Falls eine dieser Warteschlangen Aliaswarteschlangen sind, benötigen ihre Zielwarteschlangen eine Abfrageberechtigung (inquire). Wenn die Zielwarteschlange eine Clusterwarteschlange ist, benötigt sie auch eine Anzeigeberechtigung (browse).

Für die Publish/Subscribe-Domäne werden die folgenden Warteschlangen verwendet, wenn sich die IBM MQ classes for JMS mit einem IBM MQ-Warteschlangenmanager in einem IBM MQ-Messaging-Provider verbinden, der sich im Migrationsmodus befindet:

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Weitere Informationen zum Migrationsmodus des IBM MQ-Messaging-Providers finden Sie im Abschnitt JMS-Eigenschaft **PROVIDERVERSION** konfigurieren.

Wenn die IBM MQ classes for JMS in diesem Modus eine Verbindung zu einem Warteschlangenmanager herstellen, benötigt außerdem jede Anwendung, die Nachrichten veröffentlicht, Zugriff auf die Datenstromwarteschlange, die durch das TopicConnectionFactory-Objekt oder das Topic-Objekt angegeben ist. Standardmäßig wird für diese Warteschlange SYSTEM.BROKER.DEFAULT.STREAM verwendet.

Wenn Sie ConnectionConsumer, den IBM MQ-Ressourcenadapter oder den WebSphere Application Server-Messaging-Provider von IBM MQ verwenden, sind möglicherweise zusätzliche Berechtigungen erforderlich.

Warteschlangen, die vom ConnectionConsumer gelesen werden sollen, müssen über die Berechtigungen `get`, `inq` und `browse` verfügen. Die Warteschlange für nicht zustellbare Nachrichten des Systems sowie alle Zurückstellungswarteschlangen oder Berichtswarteschlangen, die vom ConnectionConsumer verwendet werden, müssen die Berechtigungen `put` und `passall` haben.

Wenn eine Anwendung den IBM MQ-Messaging-Provider im normalen Modus verwendet, um ein Publish/Subscribe-Messaging durchzuführen, nutzt die Anwendung die integrierte Publish/Subscribe-Funktionalität, die vom Warteschlangenmanager zur Verfügung gestellt wird. Im Abschnitt Publish/Subscribe-Sicherheit finden Sie Informationen zum Schutz der verwendeten Themen und Warteschlangen.

Verbindungsmodi für IBM MQ classes for JMS
Eine Anwendung der IBM MQ classes for JMS kann sich entweder im Client- oder im Bindungsmodus mit einem Warteschlangenmanager verbinden. Im Clientmodus verbinden sich IBM MQ classes for JMS über

TCP/IP mit dem Warteschlangenmanager. Im Bindungsmodus verbinden sich die IBM MQ classes for JMS unter Verwendung von Java Native Interface (JNI) direkt mit dem Warteschlangenmanager.

Unter z/OSkann der Bindungsmodus in jeder Umgebung verwendet werden, der Clientmodus jedoch nur in den folgenden Umgebungen:

- In WebSphere Application Server oder WebSphere Liberty Profile, die eine Verbindung zu einem beliebigen Warteschlangenmanager auf einer beliebigen Plattform, einschließlich z/OS, herstellen
- In Stapelumgebungen, wenn eine Verbindung zu einem IBM MQ for z/OS -Warteschlangenmanager hergestellt wird, der auf einer beliebigen LPAR ausgeführt wird.

Eine Anwendung, die auf einer beliebigen anderen Plattform ausgeführt wird, kann sich entweder im Bindungs- oder im Clientmodus mit einem Warteschlangenmanager verbinden.

Sie können die aktuelle oder eine beliebige frühere unterstützte Version der IBM MQ classes for JMS mit einem aktuellen Warteschlangenmanager verwenden. Ebenso können Sie die aktuelle oder eine frühere unterstützte Version des Warteschlangenmanagers mit der aktuellen Version der IBM MQ classes for JMS verwenden. Wenn Sie verschiedene Versionen kombiniert verwenden, ist die Funktion auf die Stufe der früheren Version begrenzt.

In den folgenden Abschnitten werden die einzelnen Verbindungsmodi ausführlicher beschrieben.

Clientmodus

Um sich im Clientmodus mit einem Warteschlangenmanager zu verbinden, kann eine Anwendung der IBM MQ classes for JMS auf demselben System wie der Warteschlangenmanager ausgeführt werden, es ist aber auch die Ausführung auf einem anderen System möglich. In jedem Fall verbinden sich IBM MQ classes for JMS über TCP/IP mit dem Warteschlangenmanager.

Bindungsmodus

Um sich im Bindungsmodus mit einem Warteschlangenmanager zu verbinden, muss eine Anwendung der IBM MQ classes for JMS auf demselben System wie der Warteschlangenmanager ausgeführt werden.

Die IBM MQ classes for JMS verbinden sich unter Verwendung von Java Native Interface (JNI) direkt mit dem Warteschlangenmanager. Damit der Bindungstransport verwendet werden kann, müssen die IBM MQ classes for JMS in einer Umgebung ausgeführt werden, die auf die Bibliotheken von IBM MQ Java Native Interface zugreifen kann; weitere Informationen hierzu finden Sie im Abschnitt „Java Native Interface-(JNI-)Bibliotheken konfigurieren” auf Seite 101.

Die IBM MQ classes for JMS unterstützen die folgenden Werte für *ConnectOption*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

Wenn Sie die von den IBM MQ classes for JMS verwendeten Verbindungsoptionen ändern möchten, ändern Sie die Eigenschaft CONNOPT der Verbindungsfactory.

Weitere Informationen zu Verbindungsoptionen finden Sie im Abschnitt „Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen” auf Seite 775.

Für die Verwendung des Bindungstransports muss die verwendete Java Runtime Environment die ID des codierten Zeichensatzes (Coded Character Set Identifier, CCSID) des Warteschlangenmanagers unterstützen, zu dem die IBM MQ classes for JMS eine Verbindung herstellen.

Details darüber, wie Sie feststellen können, welche CCSIDs von einer Java Runtime Environment unterstützt werden, finden Sie im Abschnitt IBM MQ FDC with Probe ID 21 generated when using the IBM MQ V7 classes for Java or IBM MQ V7 classes for JMS.

Warteschlangenmanager so konfigurieren, dass sich Anwendungen der IBM MQ classes for JMS im Clientmodus verbinden können
Wenn Sie Ihren Warteschlangenmanager so konfigurieren möchten, dass Anwendungen der IBM MQ classes for JMS im Clientmodus eine Verbindung herstellen können, müssen Sie eine Definition des Serververbindungskanals erstellen und einen Listener starten.

Definition für Serververbindungskanal erstellen

Sie können auf allen Plattformen mit dem MQSC-Befehl DEFINE CHANNEL eine Definition für einen Serververbindungskanal erstellen. Sehen Sie sich das folgende Beispiel an:

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i Unter IBM i können Sie stattdessen wie im folgenden Beispiel den CL-Befehl CRTMQMCHL verwenden:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

In diesem Befehl steht *QMGRNAME* für den Namen Ihres Warteschlangenmanagers.

Windows Linux Unter Linux und Windows können Sie eine Serververbindungskanaldefinition auch mit IBM MQ Explorer erstellen.

z/OS Unter z/OS können Sie eine Serververbindungskanaldefinition über die Betriebs- und Steuerkonsolen erstellen.

Der Name des Kanals (in den vorherigen Beispielen JAVA.CHANNEL) muss mit dem Kanalnamen identisch sein, der mit der Eigenschaft CHANNEL der Verbindungsfactory angegeben wird, die Ihre Anwendung für die Verbindung mit dem Warteschlangenmanager verwendet. Der Standardwert der Eigenschaft CHANNEL ist SYSTEM.DEF.SVRCONN.

Listener starten

Sie müssen einen Listener für Ihren Warteschlangenmanager starten, falls dieser noch nicht gestartet wurde.

Multi Auf Multiplatforms können Sie einen Listener mit dem MQSC-Befehl START LISTENER starten. Hierzu müssen Sie das Listenerobjekt aber zunächst mit dem MQSC-Befehl DEFINE LISTENER erstellen. Beides wird im folgenden Beispiel gezeigt:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS Unter z/OS verwenden Sie wie im folgenden Beispiel nur den Befehl START LISTENER, beachten Sie jedoch, dass der Adressraum des Kanalinitiators gestartet werden muss, damit Sie einen Listener starten können:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i Unter IBM i können Sie wie im folgenden Beispiel den CL-Befehl STRMQMLSR zum Starten eines Listeners verwenden:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

In diesem Befehl steht *QMGRNAME* für den Namen Ihres Warteschlangenmanagers.

ALW Unter AIX, Linux, and Windows können Sie einen Listener auch mit dem Steuerbefehl **runmqlsr** starten, wie das folgende Beispiel zeigt:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

In diesem Befehl steht *Warteschlangenmanagername* für den Namen Ihres Warteschlangenmanagers.

Windows Linux Unter Linux und Windows können Sie einen Listener auch mit IBM MQ Explorer starten.

z/OS Unter z/OS können Sie einen Listener auch über die Betriebs- und Steuerkonsolen starten.

Die Nummer des Ports, an dem der Listener empfangsbereit ist, muss mit der Portnummer identisch sein, die mit der Eigenschaft PORT der Verbindungsfactory angegeben wird, die Ihre Anwendung für die Verbindung mit dem Warteschlangenmanager verwendet. Der Standardwert der Eigenschaft PORT ist 1414.

Punkt-zu-Punkt-Installationsprüftest (IVT) für IBM MQ classes for JMS

Ein Punkt-zu-Punkt-Installationsprüftest (IVT) wird mit IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingbereitgestellt. Das Programm verbindet sich entweder im Bindungs- oder im Clientmodus mit einem Warteschlangenmanager, sendet eine Nachricht an die Warteschlange mit dem Namen SYSTEM.DEFAULT.LOCAL.QUEUE und empfängt dann die Nachricht aus der Warteschlange. Das Programm kann alle Objekte, die es benötigt, dynamisch während der Laufzeit erstellen und konfigurieren oder mithilfe von JNDI verwaltete Objekte aus einem Verzeichnisservice abrufen.

Führen Sie zuerst den Installationsprüftest ohne Verwendung von JNDI aus, da der Test eigenständig ist und keine Verwendung eines Verzeichnisservice erfordert. Eine Beschreibung der verwalteten Objekte finden Sie im Abschnitt JMS-Objekte mithilfe des Verwaltungstools konfigurieren.

Punkt-zu-Punkt-Installationsprüftest ohne Verwendung von JNDI

Bei diesem Test erstellt und konfiguriert das Programm des Installationsprüftests alle von ihm benötigten Objekte dynamisch zur Laufzeit und verwendet kein JNDI.

Multi Auf mehreren Plattformen wird ein Script zum Ausführen des IVT-Programms bereitgestellt. Das Script heißt **IVTRun** auf AIX and Linux -Systemen und **IVTRun.bat** auf Windows. Das Script befindet sich im Unterverzeichnis `bin` des IBM MQ classes for JMS -Installationsverzeichnisses. Der Klassenpfad muss `com.ibm.mqjms.jar`enthalten.

Geben Sie folgenden Befehl ein, um den Test im Bindungsmodus auszuführen:

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Wenn Sie den Test im Clientmodus ausführen möchten, richten Sie zuerst den WS-Manager wie in „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123beschrieben ein. Beachten Sie, dass der zu verwendende Kanal standardmäßig **SYSTEM.DEF.SVRCONN** und die zu verwendende Warteschlange **SYSTEM.DEFAULT.LOCAL.QUEUE**ist. Geben Sie dann den folgenden Befehl ein:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

z/OS Auf z/OS -Systemen wird kein funktional entsprechendes Script bereitgestellt. Stattdessen führen Sie das IVT im Bindungsmodus aus, indem Sie die Klasse Java direkt aufrufen. Unter z/OSkönnen Sie zwischen zwei funktional identischen Instanzen des IVT-Programms wählen:

- `com.ibm.mq.jms.MQJMSIVT`, verfügbar mit IBM MQ classes for JMS (JMS 2.0). Um dieses Programm verwenden zu können, muss der Klassenpfad `com.ibm.mqjms.jar` oder `com.ibm.mq.allclient.jar`enthalten.
- `com.ibm.mq.jakarta.jms.MQJMSIVT`, verfügbar mit IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0). Zur Verwendung dieses Programms muss der Klassenpfad `com.ibm.mq.jakarta.client.jar`enthalten.

Geben Sie den folgenden Befehl ein, um den Test im Bindungsmodus unter z/OSauszuführen:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Die Parameter in den Befehlen haben folgende Bedeutungen:

-m *Warteschlangenmanager*
Der Name des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests eine Verbindung herstellt. Wenn Sie den Test im Bindungsmodus ausführen und diesen Parameter übergehen, verbindet sich das Programm des Installationsprüftests mit dem Standardwarteschlangenmanager.

-host *Hostname*
Der Hostname oder die IP-Adresse des Systems, auf dem der Warteschlangenmanager ausgeführt wird.

-port *Port*
Die Nummer des Ports, an dem der Listener des Warteschlangenmanagers empfangsbereit ist. Der Standardwert ist 1414.

-channel *Kanal*
Der Name des MQI-Kanals, den das Programm des Installationsprüftests verwendet, um sich mit dem Warteschlangenmanager zu verbinden. Der Standardwert ist `SYSTEM.DEF.SVRCONN`.

-v *Providerversion*
Der Release-Level des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests erwartungsgemäß eine Verbindung herstellt.

Mit diesem Parameter wird die Eigenschaft PROVIDERVERSION eines MQQueueConnectionFactory-Objekts festgelegt. Er weist dieselben gültigen Werte wie die Eigenschaft PROVIDERVERSION auf. Deshalb finden Sie weitere Informationen zu diesem Parameter, einschließlich seiner gültigen Werte, im Abschnitt JMS: Änderungen der Eigenschaft PROVIDERVERSION und in der Beschreibung der Eigenschaft PROVIDERVERSION im Abschnitt Eigenschaften von IBM MQ classes for JMS-Objekten.

Der Standardwert ist `unspecified`.

-ccsid *CCSID*
Die ID (CCSID) des codierten Zeichensatzes oder der Codepage, der bzw. die von der Verbindung verwendet werden soll. Der Standardwert ist 819.

-t
Die Traceerstellung ist aktiviert. Standardmäßig ist die Traceerstellung inaktiviert.

Ein erfolgreicher Test erstellt eine Ausgabe, die der folgenden Beispielausgabe ähnelt:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
```

```
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

Punkt-zu-Punkt-Installationsprüftest unter Verwendung von JNDI

Bei diesem Test verwendet das Programm des Installationsprüftests JNDI für den Abruf von verwalteten Objekten aus einem Verzeichnisservice.

Damit Sie den Test ausführen können, müssen Sie zuerst einen Verzeichnisservice konfigurieren, der auf einem LDAP-Server (LDAP = Lightweight Directory Access Protocol) oder auf dem lokalen Dateisystem basiert. Außerdem müssen Sie das IBM MQ-JMS-Verwaltungstool so konfigurieren, dass es den Verzeichnisservice zum Speichern von verwalteten Objekten verwenden kann. Weitere Informationen zu diesen Voraussetzungen finden Sie im Abschnitt „Voraussetzungen für IBM MQ classes for JMS” auf Seite 92. Informationen zum Konfigurieren des IBM MQ-JMS-Verwaltungstools finden Sie im Abschnitt JMS-Verwaltungstool konfigurieren.

Das Programm des Installationsprüftests muss in der Lage sein, JNDI für den Abruf eines MQQueueConnectionFactory-Objekts und eines MQQueue-Objekts aus dem Verzeichnisservice abzurufen. Es wird ein Script bereitgestellt, das diese verwalteten Objekte für Sie erstellt. Das Script heißt IVTSetup auf AIX and Linux-Systemen und IVTSetup.bat auf Windows und befindet sich im Unterverzeichnis `bin` des IBM MQ classes for JMS-Installationsverzeichnisses. Geben Sie folgenden Befehl ein, um das Script auszuführen:

```
IVTSetup
```

Das Script ruft das IBM MQ-JMS-Verwaltungstool auf, um die verwalteten Objekte zu erstellen.

Das MQQueueConnectionFactory-Objekt wird mit dem Namen 'ivtQCF' gebunden und mit den Standardwerten für alle seine Eigenschaften erstellt. Dies bedeutet, dass das Programm des Installationsprüftests im Bindungsmodus ausgeführt wird und sich mit dem Standardwarteschlangenmanager verbindet. Wenn Sie möchten, dass das Programm des Installationsprüftests im Clientmodus ausgeführt wird oder sich mit einem anderen Warteschlangenmanager als dem Standardwarteschlangenmanager verbindet, müssen Sie die entsprechenden Eigenschaften des MQQueueConnectionFactory-Objekts mit dem IBM MQ-JMS-Verwaltungstool oder mit IBM MQ Explorer ändern. Informationen zur Verwendung des IBM MQ Explorer JMS-Verwaltungstools finden Sie im Abschnitt JMS-Objekte mithilfe des Verwaltungstools konfigurieren. Informationen zur Verwendung von IBM MQ Explorer finden Sie im Abschnitt Einführung zu IBM MQ Explorer oder in der Hilfe zu IBM MQ Explorer.

Das MQQueue-Objekt wird mit dem Namen 'ivtQ' gebunden und mit den Standardwerten für alle seine Eigenschaften erstellt. Ausgenommen hiervon ist die Eigenschaft QUEUE, die den Wert SYSTEM.DEFAULT.LOCAL.QUEUE hat.

Sobald Sie die verwalteten Objekte erstellt haben, können Sie das Programm des Installationsprüftests ausführen. Geben Sie folgenden Befehl ein, um den Test unter Verwendung von JNDI auszuführen:

```
IVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Die Parameter in dem Befehl haben folgende Bedeutungen:

-url " *Provider-URL* "
Die URL (Uniform Resource Locator) des Verzeichnisservice. Die URL kann eines der folgenden Formate aufweisen:

- `ldap://hostname/contextName` für einen Verzeichnisservice, der auf einem LDAP-Server basiert
- `file:/directoryPath` für einen Verzeichnisservice, der auf dem lokalen Dateisystem basiert

Beachten Sie, dass Sie die URL in Anführungszeichen (") einschließen müssen.

-icf *Ausgangskontextfactory*
Der Klassenname der Ausgangskontextfactory, der einen der folgenden Werte aufweisen muss:

- `com.sun.jndi.ldap.LdapCtxFactory` für einen Verzeichnisservice, der auf einem LDAP-Server basiert. Dies ist der Standardwert.
- `com.sun.jndi.fscontext.RefFSContextFactory` für einen Verzeichnisservice, der auf dem lokalen Dateisystem basiert

-t
Die Traceerstellung ist aktiviert. Standardmäßig ist die Traceerstellung inaktiviert.

Ein erfolgreicher Test erstellt eine Ausgabe, die derjenigen eines erfolgreichen Tests ohne Verwendung von JNDI ähnelt. Der Hauptunterschied besteht darin, dass die Ausgabe anzeigt, dass der Test JNDI für den Abruf eines MQQueueConnectionFactory-Objekts und eines MQQueue-Objekts verwendet.

Es ist zwar nicht unbedingt erforderlich, wird jedoch empfohlen, nach dem Test eine Bereinigung durchzuführen. Löschen Sie hierfür die verwalteten Objekte, die vom IVTSetup-Script erstellt wurden. Zu diesem Zweck wird ein Script bereitgestellt. Das Script heißt IVTTidy auf AIX and Linux-Systemen und IVTTidy.bat unter Windows und befindet sich im Unterverzeichnis `bin` des IBM MQ classes for JMS-Installationsverzeichnisses.

Problembestimmung beim Punkt-zu-Punkt-Installationsprüftest

Multi

Der Installationsprüftest kann aus folgenden Gründen fehlschlagen:

- Wenn das Programm des Installationsprüftests eine Nachricht schreibt, die angibt, dass eine Klasse nicht gefunden werden kann, prüfen Sie, ob Ihr Klassenpfad wie im Abschnitt „Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98 beschrieben ordnungsgemäß festgelegt ist.
- Der Test kann mit der folgenden Nachricht fehlschlagen:

  ```
  Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
  and host name ' hostname '
  ```

  Außerdem ist dieser Nachricht der Ursachencode 2059 zugeordnet. Die Variablen in der Nachricht haben folgende Bedeutungen:

  qmgr
  Der Name des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests versucht, eine Verbindung herzustellen. Diese Nachrichteneinfügung ist leer, wenn das Programm des

Installationsprüftests versucht, sich im Bindungsmodus mit dem Standardwarteschlangenmanager zu verbinden.

connMode
Der Verbindungsmodus, entweder `Bindings` oder `Client`.

Hostname
Der Hostname oder die IP-Adresse des Systems, auf dem der Warteschlangenmanager ausgeführt wird.

Diese Nachricht bedeutet, dass der Warteschlangenmanager, zu dem das Programm des Installationsprüftests gerade eine Verbindung herstellen möchte, nicht verfügbar ist. Prüfen Sie, ob der Warteschlangenmanager aktiv ist, und falls das Programm des Installationsprüftests versucht, sich mit dem Standardwarteschlangenmanager zu verbinden, stellen Sie sicher, dass der Warteschlangenmanager als Standardwarteschlangenmanager für Ihr System definiert ist.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

Diese Nachricht bedeutet, dass die Warteschlange SYSTEM.DEFAULT.LOCAL.QUEUE nicht in dem Warteschlangenmanager vorhanden ist, mit dem das Programm des Installationsprüftests verbunden ist. Falls die Warteschlange vorhanden ist, ist es auch möglich, dass das Programm des Installationsprüftests die Warteschlange nicht öffnen kann, weil sie nicht für das Einreihen und Abrufen von Nachrichten aktiviert wurde. Prüfen Sie, ob die Warteschlange vorhanden und für das Einreihen und Abrufen von Nachrichten aktiviert ist.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Unable to bind to object
```

Diese Nachricht bedeutet, dass eine Verbindung mit dem LDAP-Server besteht, allerdings ist der LDAP-Server nicht ordnungsgemäß konfiguriert. Der LDAP-Server ist entweder nicht für das Speichern von Java-Objekten konfiguriert oder die Berechtigungen für die Objekte oder das Suffix sind nicht korrekt. Falls Sie in dieser Situation weitere Hilfe benötigen, lesen Sie die Dokumentation Ihres LDAP-Servers.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

Diese Nachricht bedeutet, dass der Warteschlangenmanager nicht ordnungsgemäß für das Akzeptieren einer Clientverbindung von Ihrem System eingerichtet wurde. Ausführliche Informationen finden Sie in „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.

Publish/Subscribe-Installationsprüftest (IVT) für IBM MQ classes for JMS

Mit den IBM MQ classes for JMS wird ein Programm für einen Publish/Subscribe-Installationsprüftest (Installation Verification Test, IVT) bereitgestellt. Das Programm verbindet sich entweder im Bindungs- oder im Clientmodus mit einem Warteschlangenmanager, subskribiert ein Thema, veröffentlicht eine Nachricht im Thema und ruft dann die soeben von ihm veröffentlichte Nachricht ab. Das Programm kann alle Objekte, die es benötigt, dynamisch während der Laufzeit erstellen und konfigurieren oder mithilfe von JNDI verwaltete Objekte aus einem Verzeichnisservice abrufen.

Führen Sie zuerst den Installationsprüftest ohne Verwendung von JNDI aus, da der Test eigenständig ist und keine Verwendung eines Verzeichnisservice erfordert. Eine Beschreibung der verwalteten Objekte finden Sie im Abschnitt JMS-Objekte mithilfe des Verwaltungstools konfigurieren.

Publish/Subscribe-Installationsprüftest ohne Verwendung von JNDI

Bei diesem Test erstellt und konfiguriert das Programm des Installationsprüftests alle von ihm benötigten Objekte dynamisch zur Laufzeit und verwendet kein JNDI.

Für die Ausführung des Programms des Installationsprüftests wird ein Script zur Verfügung gestellt. Das Script wird auf AIX and Linux-Systemen als PSIVTRun und unter Windows als PSIVTRun.bat bezeichnet und befindet sich im Unterverzeichnis `bin` des IBM MQ classes for JMS-Installationsverzeichnisses.

Geben Sie folgenden Befehl ein, um den Test im Bindungsmodus auszuführen:

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

Wenn Sie den Test im Clientmodus ausführen möchten, müssen Sie zuerst den Warteschlangenmanager wie im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123 beschrieben einrichten. Beachten Sie, dass für den zu verwendenden Kanal standardmäßig der Wert SYSTEM.DEF.SVRCONN übernommen. Geben Sie dann folgenden Befehl ein:

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-bqm brokerQmgr ] [-v providerVersion ] [-ccsid ccsid ] [-t]
```

Die Parameter in den Befehlen haben folgende Bedeutungen:

-m *Warteschlangenmanager*
Der Name des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests eine Verbindung herstellt. Wenn Sie den Test im Bindungsmodus ausführen und diesen Parameter übergehen, verbindet sich das Programm des Installationsprüftests mit dem Standardwarteschlangenmanager.

-host *Hostname*
Der Hostname oder die IP-Adresse des Systems, auf dem der Warteschlangenmanager ausgeführt wird.

-port *Port*
Die Nummer des Ports, an dem der Listener des Warteschlangenmanagers empfangsbereit ist. Der Standardwert ist 1414.

-channel *Kanal*
Der Name des MQI-Kanals, den das Programm des Installationsprüftests verwendet, um sich mit dem Warteschlangenmanager zu verbinden. Der Standardwert ist `SYSTEM.DEF.SVRCONN`.

-bqm *Broker-Warteschlangenmanager*
Der Namen des Warteschlangenmanagers, für den der Broker aktiv ist. Der Standardwert ist der Name des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests eine Verbindung herstellt.

Dieser Parameter ist für Warteschlangenmanager mit der Versionsnummer v 7 oder höher nicht relevant.

-v *Providerversion*
Der Release-Level des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests erwartungsgemäß eine Verbindung herstellt.

Mit diesem Parameter wird die Eigenschaft PROVIDERVERSION eines MQTopicConnectionFactory-Objekts festgelegt. Er weist dieselben gültigen Werte wie die Eigenschaft PROVIDERVERSION auf. Daher finden Sie weitere Informationen zu diesem Parameter und seinen gültigen Werten in der Beschreibung der Eigenschaft PROVIDERVERSION im Abschnitt Eigenschaften von IBM MQ classes for JMS-Objekten.

Der Standardwert ist `unspecified`.

-ccsid *CCSID*
Die ID (CCSID) des codierten Zeichensatzes oder der Codepage, der bzw. die von der Verbindung verwendet werden soll. Der Standardwert ist 819.

-t
Die Traceerstellung ist aktiviert. Standardmäßig ist die Traceerstellung inaktiviert.

Ein erfolgreicher Test erstellt eine Ausgabe, die der folgenden Beispielausgabe ähnelt:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

Publish/Subscribe-Installationsprüftest unter Verwendung von JNDI

Bei diesem Test verwendet das Programm des Installationsprüftests JNDI für den Abruf von verwalteten Objekten aus einem Verzeichnisservice.

Damit Sie den Test ausführen können, müssen Sie zuerst einen Verzeichnisservice konfigurieren, der auf einem LDAP-Server (LDAP = Lightweight Directory Access Protocol) oder auf dem lokalen Dateisystem basiert. Außerdem müssen Sie das IBM MQ-JMS-Verwaltungstool so konfigurieren, dass es den Verzeichnisservice zum Speichern von verwalteten Objekten verwenden kann. Weitere Informationen zu diesen Voraussetzungen finden Sie im Abschnitt „Voraussetzungen für IBM MQ classes for JMS” auf Seite 92. Informationen zum Konfigurieren des IBM MQ-JMS-Verwaltungstools finden Sie im Abschnitt JMS-Verwaltungstool konfigurieren.

Das Programm des Installationsprüftests muss in der Lage sein, JNDI für den Abruf eines MQTopicConnectionFactory-Objekts und eines MQTopic-Objekts aus dem Verzeichnisservice abzurufen. Es wird ein Script bereitgestellt, das diese verwalteten Objekte für Sie erstellt. Das Script heißt IVTSetup auf AIX and Linux-Systemen und IVTSetup.bat auf Windows und befindet sich im Unterverzeichnis `bin` des IBM MQ classes for JMS-Installationsverzeichnisses. Geben Sie folgenden Befehl ein, um das Script auszuführen:

```
IVTSetup
```

Das Script ruft das IBM MQ-JMS-Verwaltungstool auf, um die verwalteten Objekte zu erstellen.

Das MQTopicConnectionFactory-Objekt wird mit dem Namen 'ivtTCF' gebunden und mit den Standardwerten für alle seine Eigenschaften erstellt. Dies bedeutet, dass das Programm des Installationsprüftests im Bindungsmodus ausgeführt wird, sich mit dem Standardwarteschlangenmanager verbindet und die eingebettete Publish/Subscribe-Funktion verwendet. Wenn Sie möchten, dass das Programm des Installationsprüftests im Clientmodus ausgeführt wird, sich mit einem anderen Warteschlangenmanager als dem Standardwarteschlangenmanager verbindet oder IBM Integration Bus anstelle der eingebetteten Publish/Subscribe-Funktion verwendet, müssen Sie die entsprechenden Eigenschaften des MQTopicConnectionFactory-Objekts mit dem IBM MQ-JMS-Verwaltungstool oder mit IBM MQ Explorer ändern. Informationen zur Verwendung des IBM MQ-JMS-Verwaltungstools finden Sie im Abschnitt JMS-Objekte mithilfe des Verwaltungstools konfigurieren. Informationen zur Verwendung von IBM MQ Explorer finden Sie im Hilfetext von IBM MQ Explorer.

Das MQTopic-Objekt wird mit dem Namen 'ivtT' gebunden und mit den Standardwerten für alle seine Eigenschaften erstellt. Ausgenommen hiervon ist die Eigenschaft TOPIC, die den Wert MQJMS/PSIVT/Information hat.

Sobald Sie die verwalteten Objekte erstellt haben, können Sie das Programm des Installationsprüftests ausführen. Geben Sie folgenden Befehl ein, um den Test unter Verwendung von JNDI auszuführen:

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Die Parameter in dem Befehl haben folgende Bedeutungen:

-url " *Provider-URL* "
Die URL (Uniform Resource Locator) des Verzeichnisservice. Die URL kann eines der folgenden Formate aufweisen:

- `ldap://hostname/contextName` für einen Verzeichnisservice, der auf einem LDAP-Server basiert
- `file:/directoryPath` für einen Verzeichnisservice, der auf dem lokalen Dateisystem basiert

Beachten Sie, dass Sie die URL in Anführungszeichen (") einschließen müssen.

-icf *Ausgangskontextfactory*
Der Klassenname der Ausgangskontextfactory, der einen der folgenden Werte aufweisen muss:

- `com.sun.jndi.ldap.LdapCtxFactory` für einen Verzeichnisservice, der auf einem LDAP-Server basiert. Dies ist der Standardwert.
- `com.sun.jndi.fscontext.RefFSContextFactory` für einen Verzeichnisservice, der auf dem lokalen Dateisystem basiert

-t
Die Traceerstellung ist aktiviert. Standardmäßig ist die Traceerstellung inaktiviert.

Ein erfolgreicher Test erstellt eine Ausgabe, die derjenigen eines erfolgreichen Tests ohne Verwendung von JNDI ähnelt. Der Hauptunterschied besteht darin, dass die Ausgabe anzeigt, dass der Test JNDI für den Abruf eines MQTopicConnectionFactory-Objekts und eines MQTopic-Objekts verwendet.

Es ist zwar nicht unbedingt erforderlich, wird jedoch empfohlen, nach dem Test eine Bereinigung durchzuführen. Löschen Sie hierfür die verwalteten Objekte, die vom IVTSetup-Script erstellt wurden. Zu diesem Zweck wird ein Script bereitgestellt. Das Script heißt IVTTidy auf AIX and Linux-Systemen und IVTTidy.bat unter Windows und befindet sich im Unterverzeichnis `bin` des IBM MQ classes for JMS-Installationsverzeichnisses.

Problembestimmung beim Publish/Subscribe-Installationsprüftest

Der Installationsprüftest kann aus folgenden Gründen fehlschlagen:

- Wenn das Programm des Installationsprüftests eine Nachricht schreibt, die angibt, dass eine Klasse nicht gefunden werden kann, prüfen Sie, ob Ihr Klassenpfad wie im Abschnitt „Umgebungsvariablen

für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98 beschrieben ordnungsgemäß festgelegt ist.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Failed to connect to queue manager ' qmgr ' with
connection mode ' connMode ' and host name ' hostname '
```

Außerdem ist dieser Nachricht der Ursachencode 2059 zugeordnet. Die Variablen in der Nachricht haben folgende Bedeutungen:

qmgr
Der Name des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests versucht, eine Verbindung herzustellen. Diese Nachrichteneinfügung ist leer, wenn das Programm des Installationsprüftests versucht, sich im Bindungsmodus mit dem Standardwarteschlangenmanager zu verbinden.

connMode
Der Verbindungsmodus, entweder `Bindings` oder `Client`.

Hostname
Der Hostname oder die IP-Adresse des Systems, auf dem der Warteschlangenmanager ausgeführt wird.

Diese Nachricht bedeutet, dass der Warteschlangenmanager, zu dem das Programm des Installationsprüftests gerade eine Verbindung herstellen möchte, nicht verfügbar ist. Prüfen Sie, ob der Warteschlangenmanager aktiv ist, und falls das Programm des Installationsprüftests versucht, sich mit dem Standardwarteschlangenmanager zu verbinden, stellen Sie sicher, dass der Warteschlangenmanager als Standardwarteschlangenmanager für Ihr System definiert ist.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Unable to bind to object
```

Diese Nachricht bedeutet, dass eine Verbindung mit dem LDAP-Server besteht, allerdings ist der LDAP-Server nicht ordnungsgemäß konfiguriert. Der LDAP-Server ist entweder nicht für das Speichern von Java-Objekten konfiguriert oder die Berechtigungen für die Objekte oder das Suffix sind nicht korrekt. Falls Sie in dieser Situation weitere Hilfe benötigen, lesen Sie die Dokumentation Ihres LDAP-Servers.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

Diese Nachricht bedeutet, dass der Warteschlangenmanager nicht ordnungsgemäß eingerichtet ist, um eine Clientverbindung von Ihrem System zu akzeptieren. Weitere Informationen finden Sie unter „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.

JMS 2.0 *IBM MQ classes for JMS-Beispielanwendungen verwenden*

Die Beispielanwendungen der IBM MQ classes for JMS bieten einen Überblick über die allgemeinen Funktionen der JMS-API. Mithilfe der Beispielanwendungen können Sie Ihre Installation und Messaging-Server-Konfiguration überprüfen sowie Ihre eigenen Anwendungen erstellen.

Informationen zu diesem Vorgang

Wenn Sie beim Erstellen Ihrer eigenen Anwendungen Unterstützung benötigen, können Sie die Beispielanwendungen als Ausgangspunkt dafür verwenden. Für jede Anwendung wird eine Quellenversion und ein kompilierte Version bereitgestellt. Prüfen Sie den Quellencode einer Beispielanwendung, um die wichtigsten Schritte zum Erstellen aller für Ihre Anwendung erforderlichen Objekte (ein ConnectionFactory-, Connection-, Session-, Destination- und ein Producer- und/oder ein Consumer-Objekt) zu ermitteln, und um spezielle Eigenschaften festzulegen, mit denen die Funktionsweise Ihrer Anwendung gesteuert wird.

Weitere Informationen finden Sie unter „IBM MQ classes for JMS/Jakarta Messaging -Anwendungen schreiben” auf Seite 147. Die Beispiele könnten sich in zukünftigen Releases von IBM MQ ändern.

Für JMS 2.0zeigt Tabelle 11 auf Seite 126 , wo die IBM MQ classes for JMS -Beispielanwendungen auf den einzelnen Plattformen installiert sind.

Anmerkung:

JM 3.0 Für IBM MQ classes for Jakarta Messagingwerden neue Beispiele vorbereitet.

Tabelle 11. Installationsverzeichnisse für die Beispielanwendungen der IBM MQ classes for JMS

Plattform	Directory
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*/samp/jms/samples
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms\samples
IBM i IBM i	/qibm/proddata/mqm/java/samples/jms/samples
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

In diesem Verzeichnis gibt es Unterverzeichnisse, die eine oder mehrere Beispielanwendungen enthalten, wie in Tabelle 12 auf Seite 126 dargestellt.

Tabelle 12. Beispielanwendungen von IBM MQ classes for JMS

Name der Beispielanwendung	Beschreibung
JmsBrowser.java	Eine JMS-Warteschlangenbrowseranwendung, die alle verfügbaren Nachrichten in der angegebenen Warteschlange in der Reihenfolge, in der sie von einer Konsumentenanwendung empfangen würden, untersucht, ohne sie zu entfernen.
JmsConsumer.java	Eine JMS-Warteschlangenbrowseranwendung, die alle verfügbaren Nachrichten in der angegebenen Warteschlange in der Reihenfolge, in der sie von einer Konsumentenanwendung empfangen würden, untersucht, ohne sie zu entfernen, indem sie die Verbindungsfactory-Instanz und die Zielinstanz in einem Ausgangskontext sucht. (Von diesem Beispiel wird nur Dateisystemkontext unterstützt.)
JmsJndiConsumer.java	Eine JMS-Konsumentenanwendung (Empfänger oder Subskribent), die eine Nachricht von dem angegebenen Ziel (Warteschlange oder Thema) empfängt, indem sie die Verbindungsfactory-Instanz und die Zielinstanz in einem Ausgangskontext sucht. (Von diesem Beispiel wird nur Dateisystemkontext unterstützt.)
JmsJndiProducer.java	Eine JMS-Produzentenanwendung (Sender oder Publisher), die eine einfache Nachricht an das angegebene Ziel (Warteschlange oder Thema) sendet, indem sie die Verbindungsfactory-Instanz und die Zielinstanz in einem Ausgangskontext sucht. (Von diesem Beispiel wird nur Dateisystemkontext unterstützt.)
JmsProducer.java	Eine JMS-Produzentenanwendung (Sender oder Publisher), die eine einfache Nachricht an das angegebene Ziel (Warteschlange oder Thema) sendet.
/interactive/	
SampleConsumerJava.java	Empfangen von Nachrichten aus einem Thema/einer Warteschlange.

Tabelle 12. Beispielanwendungen von IBM MQ classes for JMS (Forts.)	
Name der Beispielanwendung	**Beschreibung**
SampleProducerJava.java	Senden von Nachrichten zu einem Thema/einer Warteschlange.
/interactive/helper/	
BaseOptions.java	Eine abstrakte Klasse, die erweitert werden kann, um Funktionen für Benutzeroptionen bereitzustellen.
IsValidType.java	Eine abstrakte Klasse für Gültigkeitsprüfprogramm-Klassen.
JmsApp.java	Eine abstrakte Klasse, die erweitert werden kann, um Konsumenten-/Produzentenfunktionalität bereitzustellen.
Keys.java	Eine Gruppe von Schlüsseln, die Optionen für die Beispielanwendungen definieren.
Literals.java	Ein Gruppe von Konstantenliteralen.
MyContext.java	Der Kontext, in dem Optionen angezeigt werden.
Options.java	Bietet Funktionalität für (eine) Benutzeroption(en)
OptionsPresenter.java	Der Kontext, in dem aktuelle Optionen angezeigt werden.
/simple/	
SimpleAsyncPutPTP.java	Eine einfache Anwendung für Punkt-zu-Punkt-Messaging; die Nachricht wird asynchron gesendet (auch als *fire-and-forget*-Nachrichtenübertragung bezeichnet). Es werden keine Nachrichten empfangen.
SimpleDurableSub.java	Eine einfache Anwendung zur Demonstration der Funktion für permanente Subskriptionen.
SimpleJNDILookup.java	Eine minimale und einfache Anwendung zur Demonstration der Suche nach JMS-Objekten unter Verwendung des Ausgangskontextes. Es wird keine Verbindung mit dem Warteschlangenmanager hergestellt und es werden keine Nachrichten gesendet oder empfangen.
SimpleMQMDRead.java	Eine einfache Anwendung, die veranschaulicht, wie eine JMS-Anwendung MQMD-Felder (MQ Message Descriptor, MQ-Nachrichtendeskriptor) als JMS-Nachrichteneigenschaften nutzen kann. Es werden keine Nachrichten gesendet. Es wird davon ausgegangen, dass die verwendete Warteschlange mit einigen Nachrichten gefüllt ist.
SimpleMQMDWrite.java	Eine einfache Anwendung, die veranschaulicht, wie eine JMS-Anwendung MQMD-Felder (MQ Message Descriptor, MQ-Nachrichtendeskriptor) schreiben kann. Es werden keine Nachrichten empfangen.
SimplePTP.java	Eine minimale und einfache Anwendung für Punkt-zu-Punkt-Messaging.
SimplePubSub.java	Eine minimale und einfache Anwendung für Publish-Subscribe-Messaging.

Tabelle 12. Beispielanwendungen von IBM MQ classes for JMS (Forts.)	
Name der Beispielanwendung	**Beschreibung**
SimpleReadAheadPTP.java	Eine einfache Anwendung für Punkt-zu-Punkt-Messaging; Nachrichten werden vom Warteschlangenmanager übertragen (auch als Vorauslesefunktion bezeichnet). Es werden keine Nachrichten gesendet. Es wird davon ausgegangen, dass die verwendete Warteschlange mit einigen Nachrichten gefüllt ist.
SimpleRequestor.java	Eine einfache Anwendung, bei der ein Anforderer verwendet wird, um eine Anforderungsnachricht zu senden und dann auf die Antwort zu warten und diese zu empfangen. Hinweis: Es wird davon ausgegangen, dass andere Anwendungen die Anforderungsnachricht verarbeiten und die Antwortnachricht senden.
SimpleResponder.java	Eine einfache Anwendung, die an einem Ziel für eine Nachricht empfangsbereit ist und dann eine Antwort an das Ziel 'replyTo' der Nachricht sendet. Die Anwendung ist für die Verwendung mit dem Beispiel 'SimpleRequestor' geschrieben.
SimpleRetainedPub.java	Eine einfache Anwendung zur Demonstration einer ständigen Veröffentlichung. Es werden keine Nachrichten empfangen.
SimpleWMQJMSPTP.java	Eine minimale und einfache Anwendung für Punkt-zu-Punkt-Messaging.
SimpleWMQJMSPubSub.java	Eine minimale und einfache Anwendung für Publish-Subscribe-Messaging.

Die IBM MQ classes for JMS stellen ein Script namens `runjms` bereit, das zur Ausführung der Beispielanwendungen verwendet werden kann. Dieses Script konfiguriert die IBM MQ-Umgebung so, dass Sie die Beispielanwendungen der IBM MQ classes for JMS ausführen können.

Tabelle 13 auf Seite 128 zeigt den Speicherort des Scripts auf den einzelnen Plattformen:

Tabelle 13. Speicherort des Scripts `runjms`	
Plattform	**Directory**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	oder
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

Führen Sie die folgenden Schritte aus, um eine Beispielanwendung mit dem Script `runjms` aufzurufen:

Vorgehensweise

1. Rufen Sie eine Eingabeaufforderung auf und navigieren Sie zu dem Verzeichnis, das die auszuführende Beispielanwendung enthält.
2. Geben Sie den folgenden Befehl ein:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   Die Beispielanwendung zeigt eine Liste der Parameter an, die sie benötigt.
3. Geben Sie den folgenden Befehl ein, um das Beispiel mit diesen Parametern auszuführen:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Beispiel

Linux Geben Sie beispielsweise die folgenden Befehle ein, um das Beispiel `JmsBrowser` unter Linuxauszuführen:

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

Zugehörige Konzepte

„Dies wird für IBM MQ classes for JMS installiert” auf Seite 93
Bei der Installation von IBM MQ classes for JMS werden verschiedene Dateien und Verzeichnisse installiert. Unter Windows erfolgt ein Teil der Konfiguration während der Installation durch die automatische Festlegung von Umgebungsvariablen. Auf anderen Plattformen und in bestimmten Windows-Umgebungen müssen Sie im Vorfeld Umgebungsvariablen festlegen, damit Sie Anwendungen der IBM MQ classes for JMS ausführen können.

Mit IBM MQ classes for JMS/Jakarta Messaging bereitgestellte Scripts

Es wird eine Reihe von Scripts bereitgestellt, die Sie bei allgemeinen Tasks unterstützen, die ausgeführt werden müssen, wenn Sie IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverwenden.

Tabelle 14 auf Seite 129 listet alle Scripts und ihre Verwendungen auf. Die Scripts befinden sich im Unterverzeichnis `bin` des Installationsverzeichnisses von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging .

Tabelle 14. Scripts, die mit IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging bereitgestellt werden

Dienstprogramm	**Verwenden Sie**
Cleanup [1]	Dieses Script steht aus Gründen der Kompatibilität mit Vorgängerreleases noch zur Verfügung, bleibt jedoch ohne Funktion. Die manuelle Bereinigung von Subskriptionsinformationen ist nicht mehr erforderlich.
DefaultConfiguration	Führt die Standardkonfigurationsanwendung auf anderen Plattformen als Windows aus.
formatLog [1]	Dieses Script steht aus Gründen der Kompatibilität mit Vorgängerreleases noch zur Verfügung, bleibt jedoch ohne Funktion. Die Protokollausgabe wird jetzt in lesbarem Text erstellt.
IVTRun [1] IVTSetup [1] IVTTidy [1]	Wird beim Punkt-zu-Punkt-Installationsprüftest verwendet (siehe Beschreibung unter „Punkt-zu-Punkt-Installationsprüftest (IVT) für IBM MQ classes for JMS” auf Seite 117).
JM 3.0 JMS30Admin [1]	Führt das IBM MQ-Jakarta Messaging-Verwaltungstool aus (siehe Beschreibung im Abschnitt Verwaltungstool starten).

Tabelle 14. Scripts, die mit IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging bereitgestellt werden (Forts.)	
Dienstprogramm	**Verwenden Sie**
JM 3.0 JMS30Admin.config	Die Konfigurationsdatei für das IBM MQ-Jakarta Messaging-Verwaltungstool (siehe Beschreibung im Abschnitt JMS-Verwaltungstool konfigurieren).
JMS 2.0 JMSAdmin [1]	Führt das IBM MQ-JMS-Verwaltungstool aus (siehe Beschreibung im Abschnitt Verwaltungstool starten).
JMS 2.0 JMSAdmin.config	Die Konfigurationsdatei für das IBM MQ-JMS-Verwaltungstool (siehe Beschreibung im Abschnitt JMS-Verwaltungstool konfigurieren).
PSIVTRun [1]	Führt das Programm für den Installationsprüftest von Publish/Subscribe aus (siehe Beschreibung unter „Publish/Subscribe-Installationsprüftest (IVT) für IBM MQ classes for JMS” auf Seite 121).
PSReportDump.class	Diese Klasse steht aus Gründen der Kompatibilität mit Vorgängerreleases noch zur Verfügung, bleibt jedoch ohne Funktion.
JM 3.0 setjms30env „2” auf Seite 130	Linux AIX Für Jakarta Messaging 3.0: Legt die Umgebungsvariablen für die Ausführung einer IBM MQ classes for JMS -Anwendung in einer 32 -Bit- Java Virtual Machine (JVM) auf AIX and Linux -Systemen wie in „Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98beschrieben fest.
JMS 2.0 setjmsenv „2” auf Seite 130	Linux AIX Für JMS 2.0: Legt die Umgebungsvariablen für die Ausführung einer IBM MQ classes for JMS -Anwendung in einer 32 -Bit- Java Virtual Machine (JVM) auf AIX and Linux -Systemen wie in „Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98beschrieben fest.
JM 3.0 setjms30env64 „2” auf Seite 130	Linux AIX Für Jakarta Messaging 3.0werden die Umgebungsvariablen für die Ausführung einer IBM MQ classes for JMS -Anwendung in einer 64-Bit-JVM auf AIX and Linux -Systemen wie in „Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98beschrieben festgelegt.
JMS 2.0 setjmsenv64 „2” auf Seite 130	Linux AIX Für JMS 2.0werden die Umgebungsvariablen für die Ausführung einer IBM MQ classes for JMS -Anwendung in einer 64-Bit-JVM auf AIX and Linux -Systemen wie in „Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98beschrieben festgelegt.

Anmerkung:

1. Unter Windowshat der Dateiname die Erweiterung `.bat`.
2. Diese Scripts sind nur unter AIX and Linux verfügbar. Führen Sie unter Windowsnach der Installation von IBM MQden Befehl **`setmqenv`**aus. Weitere Informationen finden Sie unter „Umgebungsvariablen für IBM MQ classes for JMS/Jakarta Messaging festlegen” auf Seite 98.

Unterstützung für OSGi mit IBM MQ classes for JMS

OSGi stellt ein Framework bereit, das die Implementierung von Anwendungen als Bundles unterstützt. Die OSGi-Bundles werden als Teil von IBM MQ classes for JMSbereitgestellt.

Die IBM MQ classes for JMS enthalten die folgenden OSGi-Bundles.

com.ibm.msg.client.osgi.jms*version_number*.jar
Die allgemeine Codeschicht in den IBM MQ classes for JMS. Informationen zu der Schichtarchitektur von IBM MQ-Klassen für JMS finden Sie im Abschnitt IBM MQ-Klassen für JMS-Architektur.

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
Die vorausgesetzten JAR-Dateien (Java Archive) für die allgemeine Schicht.

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
Allgemeine Services für Java Platform, Standard Edition (Java SE)-Anwendungen.

com.ibm.msg.client.osgi.nls_*version_number*.jar
Nachrichten für die allgemeine Schicht.

com.ibm.msg.client.osgi.wmq_*version_number*.jar
Der IBM MQ-Messaging-Provider in IBM MQ classes for JMS. Informationen zu der Schichtarchitektur von IBM MQ classes for JMS finden Sie im Abschnitt IBM MQ-Klassen für JMS-Architektur.

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
Die vorausgesetzten JAR-Dateien für den IBM MQ-Messaging-Provider.

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
Nachrichten für den IBM MQ-Messaging-Provider.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 Für Jakarta Messaging 3.0ermöglicht diese JAR-Datei Anwendungen die Verwendung von IBM MQ classes for JMS und IBM MQ classes for Javaund enthält auch den Code für die Verarbeitung von PCF-Nachrichten.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
JM 3.0 Für Jakarta Messaging 3.0enthält diese JAR-Datei die Voraussetzungen für `com.ibm.mq.jakarta.osgi.allclient_`*`version_number`*`.jar`.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Für JMS 2.0ermöglicht diese JAR-Datei Anwendungen die Verwendung von IBM MQ classes for JMS und IBM MQ classes for Javasowie den Code für die Verarbeitung von PCF-Nachrichten.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 Für JMS 2.0enthält diese JAR-Datei die Voraussetzungen für `com.ibm.mq.osgi.allclient_`*`version_number`*`.jar`.

Dabei ist *`version_number`* die installierte Versionsnummer von IBM MQ .

Die Bundles werden im Unterverzeichnis `java/lib/OSGi` Ihrer IBM MQ-Installation oder im Ordner `java\lib\OSGi` unter Windows installiert.

Verwenden Sie ab IBM MQ 8.0 für allen neuen Anwendungen die Bundles in `com.ibm.mq.osgi.allclient_8.0.0.0.jar` und `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar`. Durch die Verwendung dieser Bundles besteht die Einschränkung, dass IBM MQ classes for JMS und IBM MQ classes for Java nicht gleichzeitig innerhalb desselben OSGi-Frameworks ausgeführt werden können, nicht mehr. Alle übrigen Einschränkungen bestehen jedoch nach wie vor.

Das Bundle `com.ibm.mq.osgi.java`*`version_number`*`.jar`, das auch im Unterverzeichnis `java/lib/OSGi` Ihrer IBM MQ-Installation oder im Ordner `java\lib\OSGi` unter Windows installiert ist, ist Teil der IBM MQ classes for Java. Dieses Bundle darf nicht in eine OSGi-Laufzeitumgebung geladen werden, in der IBM MQ classes for JMS geladen wurden.

Die OSGi-Bundles für die IBM MQ classes for JMS wurden in die Spezifikation von OSGi Release 4 geschrieben. Sie funktionieren nicht in einer Umgebung mit OSGi Release 3.

Sie müssen Ihren Systempfad oder Bibliothekspfad ordnungsgemäß festlegen, damit die OSGi-Laufzeitumgebung alle erforderlichen DLL-Dateien oder gemeinsam genutzten Bibliotheken finden kann.

Wenn Sie die OSGi-Bundles für die IBM MQ classes for JMS verwenden, können temporäre Themen nicht genutzt werden. Außerdem werden Kanalexitklassen, die in Java geschrieben wurden, aufgrund eines damit verbundenen Problems beim Laden von Klassen in einer Umgebung mit mehreren Klassenla-

deprogrammen wie OSGi nicht unterstützt. Einem Benutzerbundle können zwar die Bundles der IBM MQ classes for JMS bekannt sein, die Benutzerbundles werden jedoch nicht von den Bundles der IBM MQ classes for JMS erkannt. Folglich kann das in einem Bundle der IBM MQ classes for JMS verwendete Klassenladeprogramm keine Kanalexitklasse laden, die sich in einem Benutzerbundle befindet.

Weitere Informationen zu OSGi finden Sie auf der Website OSGi Alliance.

z/OS MQ Adv. VUE JMS/Jakarta Messaging client connectivity to batch applications running on z/OS

Under certain conditions, an IBM MQ classes for JMS/Jakarta Messaging application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

By using a client connection, an IBM MQ classes for JMS/Jakarta Messaging application can connect to a remote z/OS queue manager if the following conditions apply:

- The application is running in a batch environment.
- Der Warteschlangenmanager, zu dem eine Verbindung hergestellt wird, wird mit IBM MQ Advanced for z/OS Value Unit Edition -Berechtigung ausgeführt und der Parameter **ADVCAP** ist daher auf ENABLED gesetzt.

Weitere Informationen zu IBM MQ Advanced for z/OS Value Unit Edition finden Sie unter IBM MQ -Produkt-IDs und -Exportinformationen.

See ANZEIGEN QMGR for more information on **ADVCAP** and QMGR STARTEN for more information on **QMGRPROD**.

Note that batch is the only environment supported; there is no support for JMS/Jakarta Messaging for CICS or JMS/Jakarta Messaging for IMS.

An IBM MQ classes for JMS/Jakarta Messaging application on z/OS cannot use a client mode connection to connect to a queue manager that is not running on z/OS.

If an IBM MQ classes for JMS/Jakarta Messaging application on z/OS attempts to connect using client mode, and is not allowed to do so, exception message JMSFMQ0005 is issued.

Advanced Message Security (AMS) support

IBM MQ classes for JMS/Jakarta Messaging client applications can use AMS when connecting to remote z/OS queue managers, subject to the conditions previously described in this topic.

Um AMS auf diese Weise zu verwenden, müssen die Clientanwendungen den Keystore-Typ `jceracfks` in `keystore.conf`verwenden. Dabei gilt Folgendes:

- Das Eigenschaftsnamenspräfix ist `jceracfks`. Dieses Namenspräfix ist von der Groß-/Kleinschreibung unabhängig.
- Der Keystore ist ein RACF-Schlüsselring.
- Kennwörter sind nicht erforderlich und werden ignoriert. Der Grund dafür ist, dass RACF-Schlüsselringe keine Kennwörter verwenden.
- Wenn Sie den Provider angeben, muss der Provider `IBMJCE` sein.

Wenn Sie `jceracfks` mit AMSverwenden, muss der Keystore das Format `safkeyring://`*`user`*`/`*`keyring`*haben. Dabei gilt Folgendes:

- `safkeyring` ist ein Literal. Dieser Name ist von der Groß-/Kleinschreibung unabhängig.
- *`user`* ist die RACF -Benutzer-ID, die Eigner des Schlüsselrings ist.
- *`keyring`* ist der Name des RACF -Schlüsselrings, bei dem die Groß-/Kleinschreibung beachtet werden muss.

Im folgenden Beispiel wird der Standardschlüsselring von AMS für Benutzer JOHNDOE verwendet:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Related concepts

“Java client connectivity to batch applications running on z/OS” on page 389
Under certain conditions, an IBM MQ classes for Java application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging separat abrufen

Die Bibliotheken und Dienstprogramme, die zum Ausführen von Anwendungen mit IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging erforderlich sind, sind in einer selbstextrahierenden JAR-Datei verfügbar, die Sie von Fix Centralherunterladen. Sie können dies tun, wenn Sie nur diese Dateien abrufen möchten, z. B. für die Implementierung in einem Software-Management-Tool oder für die Verwendung mit eigenständigen Clientanwendungen.

Vorbereitende Schritte

Stellen Sie vor Beginn dieser Task sicher, dass eine Java runtime environment (JRE) auf Ihrem System installiert ist und die JRE dem Systempfad hinzugefügt wurde.

Das Java-Installationsprogramm, das in diesem Installationsprozess verwendet wird, muss nicht von einem Root oder einem bestimmten Benutzer ausgeführt werden. Die einzige Voraussetzung ist, dass der Benutzer, unter dem es ausgeführt wird, Schreibzugriff auf das Verzeichnis hat, in dem die Dateien abgelegt werden sollen.

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.

Informationen zu diesem Vorgang

Eine selbstextrahierende JAR-Datei wird verwendet, um die Größe des Downloads und die für die Extraktion benötigte Zeit zu minimieren. Der genaue Inhalt dieser JAR-Datei und die Unterverzeichnisse, in die die Dateien extrahiert werden, hängen von der Version von IBM MQab.

Wenn Sie die selbstextrahierende JAR-Datei ausführen, wird die IBM MQ -Lizenzvereinbarung angezeigt, die akzeptiert werden muss. Sie können auch das übergeordnete Verzeichnis für die Extraktion ändern.

Ab IBM MQ 9.3 extrahiert die selbstextrahierende JAR-Datei die Dateien in die folgende Verzeichnisstruktur:

wmq/JavaEE
EAR-und RAR-Datei des IBM MQ -Ressourcenadapters.

JMS 2.0 Die folgenden Dateien sind für die Verwendung mit JMS 2.0 -und JMS 1.1 -Objekten vorgesehen:

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 Es gibt auch eine funktional entsprechende Gruppe zur Verwendung mit Jakarta Messaging 3.0 -Objekten:

- In: `wmq.jakarta.jmsra.ivt.ear`. Enthält Installationsprüftestdateien.
- In: `wmq.jakarta.jmsra.rar`. Enthält Ressourcenadapterdateien.

wmq/JavaSE

wmq/JavaSE/bin

JMSAdmin -und **JMS30Admin** -Tools Wird zum Definieren von JNDI -Entitäten verwendet, die JMS -oder Jakarta Messaging -Objekte darstellen.

JMS 2.0 Die folgenden Dateien sind für die Verwendung mit JMS 2.0 -und JMS 1.1 -Objekten vorgesehen:

- `JMSAdmin.bat`
- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 Es gibt auch eine funktional entsprechende Gruppe zur Verwendung mit Jakarta Messaging 3.0 -Objekten:

- In: `JMS30Admin.bat`. Eine Datei, die zum Starten des Tools unter Windowsverwendet wird.
- In: `JMS30Admin`. Ein Script, das zum Starten des Tools auf Linux -und UNIX -Plattformen verwendet wird
- In: `JMS30Admin.config`. Eine Beispielkonfigurationsdatei für das Tool.

Anmerkung:

- Bevor das Tool **JMSAdmin** zur selbstextrahierenden JAR-Datei hinzugefügt wurde, befanden sich die Dateien in diesem Verzeichnis im übergeordneten Verzeichnis `wmq/JavaSE`.
- Ein Client, der mit der selbstextrahierenden JAR-Datei installiert wird, kann das Tool **JMSAdmin** oder **JMS30Admin** verwenden, um verwaltete Java -Messaging-Objekte in einem Dateisystem-kontext (Datei `.bindings`) zu erstellen. Der Client kann auch nach diesen verwalteten Objekten suchen und sie verwenden.
- JMS 2.0 Das Tool **JMSAdmin** für JMS 2.0 -und JMS 1.1 -Objekte wurde der selbstextrahier-enden JAR-Datei unter IBM MQ 9.2.0 Fix Pack 2 und IBM MQ 9.2.2hinzugefügt.
- JM 3.0 Das Tool **JMS30Admin** für die Verwendung mit Jakarta Messaging 3.0 -Objekten wurde der selbstextrahierenden JAR-Datei unter IBM MQ 9.3.0hinzugefügt.

wmq/JavaSE/lib

Advanced Message Security verwendet die folgenden Open-Source- Bouncy Castle -Pakete zur Unterstützung der Syntax für verschlüsselte Nachrichten (CMS). Siehe Support for non-IBM JREs with AMS.

V 9.4.0 Ab IBM MQ 9.4.0:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

Die folgenden Dateien enthalten jeweils die Klassen für ihre jeweilige JMS -oder Jakarta Messa-ging -Version:

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 und JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

Weitere vorausgesetzte JAR-Dateien:

- In: `fscontext.jar`. Erforderlich, wenn Ihre Anwendung JNDI-Suchen mit einem Dateisystem-kontext ausführt.
- JM 3.0 In: `jakarta.jms-api.jar`. Enthält die Jakarta Messaging 3.0 -Schnittstelle und Ausnahmedefinitionen.

- JMS 2.0 In: `jms.jar`. Enthält die JMS 2.0 -Schnittstelle und Ausnahmedefinitionen.
- In: `org.json.jar`. Enthält Klassen, mit denen IBM MQ classes for JMS CCDT-Dateien im JSON-Format interpretieren kann.
- In: `providerutil.jar`. Erforderlich, wenn Ihre Anwendung JNDI-Suchen mit einem Dateisystemkontext ausführt.

Anmerkung: Stabilized `com.ibm.mq.allclient.jar` und `com.ibm.mq.jakarta.client.jar` enthalten beide eine Kopie von IBM MQ classes for Java. In IBM MQ 9.0werden diese Klassen jedoch auf der im Lieferumfang von IBM MQ 8.0enthaltenen Ebene als funktional stabilisiert deklariert. Siehe Einstellungen der Unterstützung, Stabilisierungen und Entfernungen unter IBM MQ 9.0.

`wmq/OSGi`

Die OSGi-Client-Bundles von IBM MQ :

- JM 3.0 `com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar`
- JM 3.0 `com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclient_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar`

Dabei steht *V.R.M.F* für die Nummer der Version, des Release, der Modifikationsstufe und des Fixpacks.

Vorgehensweise

1. Laden Sie die JAR-Datei für den IBM MQ Java-/JMS-Client von Fix Central herunter.
   a) Klicken Sie auf diesen Link: IBM MQ Java-/JMS-Client.
   b) Suchen Sie den Client für Ihre Version von IBM MQ in der angezeigten Liste mit den verfügbaren Fixes.
   For example:

   ```
   release level: 9.3.0.0-IBM-MQ-Install-Java-All
   Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
   ```

   Klicken Sie anschließend auf den Namen der Clientdatei und folgen Sie dem Downloadprozess.
2. Starten Sie die Extraktion aus dem Verzeichnis, in das Sie die Datei heruntergeladen haben.
   Geben Sie zum Starten der Extraktion einen Befehl im folgenden Format ein:

   ```
   java -jar V.R.M.F-IBM-MQ-Install-Java-All.jar
   ```

   Dabei ist *V.R.M.F* die Produktversionsnummer, z. B. `9.3.0.0`, und *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` ist der Name der Datei, die von Fix Centralheruntergeladen wurde.
   Um beispielsweise den JMS -Client für das IBM MQ 9.4.0 -Release zu extrahieren, verwenden Sie den folgenden Befehl:

   ```
   java -jar 9.4.0.0-IBM-MQ-Install-Java-All.jar
   ```

   Anmerkung: Um diese Installation ausführen zu können, muss eine JRE auf Ihrer Maschine installiert und im Systempfad angegeben sein.

   Wenn Sie den Befehl eingeben, werden folgende Informationen angezeigt:

   ```
   Bevor Sie IBM MQ V9.4verwenden, extrahieren oder installieren können, müssen Sie Folgendes
   akzeptieren:
   die Bedingungen von 1. IBM Internationale Nutzungsbedingungen für die Bewertung von
   Programme 2. IBM Internationale Nutzungsbedingungen für Programmpakete und weitere
   license information. Please read the following license agreements carefully.
   ```

```
The license agreement is separately viewable using the
--viewLicenseAgreement option.

Press Enter to display the license terms now, or 'x' to skip.
```

3. Prüfen und akzeptieren Sie die Lizenzbedingungen:
   a) Drücken Sie die Eingabetaste, um die Lizenz anzuzeigen.

      Alternativ können Sie x drücken, um die Anzeige der Lizenz zu überspringen.

      Nachdem die Lizenz angezeigt wird, oder sofort, wenn Sie x drücken, wird die folgende Nachricht angezeigt:

      ```
      Additional license information is separately viewable using the
      --viewLicenseInfo option.

      Press Enter to display additional license information now, or 'x' to skip.
      ```

   b) Drücken Sie die Eingabetaste, um die zusätzlichen Lizenzbedingungen anzuzeigen.

      Alternativ können Sie x drücken, um die Anzeige der zusätzlichen Lizenzbedingungen zu überspringen.

      Nachdem die zusätzlichen Lizenzbedingungen angezeigt werden oder sobald Sie x drücken, wird die folgende Nachricht angezeigt:

      ```
      By choosing the "I Agree" option below, you agree to the terms of the
      license agreement and non-IBM terms, if applicable. Wenn Sie diesen
      agree, select "I do not Agree".

      Select [1] I Agree, or [2] I do not Agree:
      ```

   c) Wählen Sie 1 aus, um die Lizenzvereinbarung zu akzeptieren und mit der Auswahl des Installationsverzeichnisses fortzufahren.

      Wählen Sie alternativ 2 aus, um die Installation sofort zu beenden.

      Wenn Sie 1 auswählen, wird eine Nachricht ähnlich der folgenden angezeigt:

      ```
      Enter directory for product files or leave blank to accept the default value.
      Das Standardzielverzeichnis ist H: \downloads

      Target directory for product files?
      ```

4. Geben Sie das übergeordnete Verzeichnis für die Extraktion an.

   Die Standardposition ist das aktuelle Verzeichnis.

   - Wenn Sie die Produktdateien an die Standardposition extrahieren wollen, drücken Sie die Eingabetaste, ohne einen Wert anzugeben.
   - Wenn Sie die Produktdateien an einer anderen Position extrahieren möchten, müssen Sie den Namen des Verzeichnisses angeben, in das Sie die Dateien extrahieren wollen, und anschließend die Eingabetaste drücken, um die Extraktion zu starten.

   Der von Ihnen angegebene Verzeichnisname darf noch nicht vorhanden sein. Andernfalls wird beim Starten der Extraktion ein Fehler gemeldet und es werden keine Dateien installiert.

   Sofern es noch nicht vorhanden ist, wird das angegebene Verzeichnis erstellt und die Programmdateien werden in dieses Verzeichnis extrahiert. Während der Installation wird ein neues Verzeichnis mit dem Namen `wmq` im angegebenen übergeordneten Verzeichnis erstellt.

   Es werden drei Unterverzeichnisse (`JavaEE`, `JavaSE` und `OSGi`) mit folgenden Inhalten im Verzeichnis `wmq` erstellt:

   JavaEE

   JM 3.0 `wmq.jakarta.jmsra.ivt.ear`

   JM 3.0 `wmq.jakarta.jmsra.rar`

   JMS 2.0 `wmq.jmsra.ivt.ear`

   JMS 2.0 `wmq.jmsra.rar`

   JavaSE

   Dieses Verzeichnis enthält die folgenden Unterverzeichnisse und Dateien:

JavaSE/lib

V 9.4.0 bcpkix-jdk18on.jar

V 9.4.0 bcprov-jdk18on.jar

V 9.4.0 bcutil-jdk18on.jar

JMS 2.0 com.ibm.mq.allclient.jar

JM 3.0 com.ibm.mq.jakarta.client.jar

fscontext.jar

jms.jar

org.json.jar

providerutil.jar

JavaSE/bin

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

OSGi

JM 3.0 com.ibm.mq.jakarta.osgi.allclient_*V.R.M.F*.jar

JM 3.0 com.ibm.mq.jakarta.osgi.allclientprereqs_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclient_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclientprereqs_*V.R.M.F*.jar

Nach Abschluss der Extraktion wird eine Bestätigungsnachricht wie im folgenden Beispiel angezeigt:

```
Dateien werden in H extrahiert: \downloads\wmq
Successfully extracted all product files.
```

Zulassungsliste in IBM MQ classes for JMS/Jakarta Messaging

Der Serialisierungs-und Entserialisierungsmechanismus für Java -Objekte wurde als potenzielles Sicherheitsrisiko identifiziert. Die Zulassungsliste in IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging bietet einen gewissen Schutz vor einigen Serialisierungsrisiken.

Informationen zu diesem Vorgang

Der Mechanismus zur Serialisierung und Deserialisierung von Java-Objekten wurde als potenzielles Sicherheitsrisiko erkannt, weil durch die Deserialisierung Instanzen beliebiger Java-Objekte erstellt werden, was wiederum mit böswilliger Absicht gesendeten Daten die Möglichkeit bietet, verschiedene Probleme zu verursachen. Eine bemerkenswerte Anwendung der Serialisierung ist in Jakarta Messaging 3.0 und Java Message Service 2.0 ObjectMessages , die die Serialisierung verwenden, um beliebige Objekte zu kapseln und zu übertragen.

Eine Zulassungsliste für die Serialisierung stellt eine Möglichkeit dar, einige der durch die Serialisierung entstehenden Risiken zu mindern. Indem explizit angegeben wird, welche Klassen in ObjectMessages eingebunden und welche daraus extrahiert werden können, bietet die Verwendung einer Zulassungsliste einen gewissen Schutz vor einigen Serialisierungsrisiken.

Zugehörige Konzepte

„IBM MQ classes for JMS-Anwendungen unter dem Java security manager ausführen” auf Seite 112
IBM MQ classes for JMS können mit aktiviertem Java security manager ausgeführt werden. Um Anwendungen erfolgreich mit aktiviertem Java security manager ausführen zu können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtlinienkonfigurationsdatei konfigurieren.

Zulassungslistenkonzepte

In IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingwird die Zulassungsliste von Klassen in der Implementierung der JMS -Schnittstelle ObjectMessage unterstützt. Dies bietet eine potenzielle Minderung einiger Sicherheitsrisiken, die sich potenziell auf den Serialisierungs-und Deserialisierungsmechanismus des Java -Objekts beziehen.

Zulassungsliste in IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging

Wichtig:

Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Dies schließt einige Java -Systemeigenschaftsnamen ein, die in diesem Thema erwähnt werden. Sie müssen eine vorhandene Konfiguration nicht ändern. Die vorherigen Systemeigenschaftsnamen funktionieren auch weiterhin.

IBM MQ classes for JMS (JMS 2.0) und IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) unterstützen die Zulassungsliste von Klassen in der Implementierung der JMS -Schnittstelle ObjectMessage .

- JMS 2.0 Für IBM MQ classes for JMSsind die relevanten Eigenschaftsnamen **`com.ibm.mq.jms.allowlist.*`**.
- JM 3.0 Für IBM MQ classes for Jakarta Messaginglauten die relevanten Eigenschaftsnamen **`com.ibm.mq.jakarta.jms.allowlist.*`** .

In der Zulassungsliste wird definiert, welche Java-Klassen mit ObjectMessage.setObject() serialisiert und mit ObjectMessage.getObject() deserialisiert werden können.

- JMS 2.0 Versuche zur Serialisierung oder Deserialisierung einer Instanz einer Klasse, die nicht in der Zulassungsliste für ObjectMessage enthalten ist, führen dazu, dass die Ausnahmebedingung javax.jms.MessageFormatException mit Ursachencode java.io.InvalidClassException ausgelöst wird.
- JM 3.0 Versuche, eine Instanz einer Klasse, die nicht in der Zulassungsliste mit ObjectMessage enthalten ist, zu serialisieren oder zu entserialisieren, führen dazu, dass eine Ausnahme des Typs jakarta.jms.MessageFormatException mit einer Ausnahme des Typs java.io.InvalidClassException ausgelöst wird.

Zulassungsliste erstellen

Wichtig: IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging können nicht mit einer Zulassungsliste verteilt werden. Die Auswahl der Klassen, die mithilfe von ObjectMessages übertragen werden sollen, erfolgt im Anwendungsdesign und kann nicht von IBM MQ vorweggenommen werden.

Aus diesem Grund ermöglicht der Zulassungslistenmechanismus zwei Betriebsarten:

DISCOVERY
: In dieser Betriebsart erstellt der Mechanismus eine Liste mit vollständig qualifizierten Klassennamen, in der alle Klassen aufgeführt sind, die hinsichtlich der Serialisierung oder Deserialisierung in ObjectMessage überprüft wurden.

ENFORCEMENT
: In dieser Betriebsart erzwingt der Mechanismus die Verwendung einer Zulassungsliste und lehnt Versuche ab, Klassen zu serialisieren oder zu deserialisieren, die nicht in der Zulassungsliste enthalten sind.

Wenn Sie diesen Mechanismus verwenden möchten, müssen Sie zunächst die Betriebsart DISCOVERY ausführen, um die Liste mit den aktuell serialisierten und deserialisierten Klassen zusammenzustellen, die Liste dann überprüfen und sie als Grundlage für Ihre Zulassungsliste verwenden. Die Liste kann gegebenenfalls auch unverändert verwendet werden, muss dann aber unbedingt vorher überprüft werden.

Steuerung des Zulassungslistenmechanismus

Es sind drei Systemeigenschaften zur Steuerung des Zulassungslistenmechanismus verfügbar:

com.ibm.mq.jms.allowlist (JMS 2.0) und com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)
Diese Eigenschaft kann auf zwei Arten angegeben werden:

- Pfadname der Datei, die die Zulassungsliste enthält, im URI-Format für Dateien (d. h. beginnend mit `file:`). In der Betriebsart DISCOVERY schreibt der Zulassungslistenmechanismus in diese Datei. Die Datei darf nicht vorhanden sein. Ist die Datei vorhanden, löst der Mechanismus eine Ausnahmebedingung aus, statt sie zu überschreiben. In der Betriebsart ENFORCEMENT liest der Zulassungslistenmechanismus die Datei.
- Durch Kommas getrennte, vollständig qualifizierte Klassennamen, aus denen die Zulassungsliste gebildet wird.

Wenn diese Eigenschaft nicht festgelegt ist, ist der Zulassungslistenmechanismus inaktiv.

Wenn Sie einen Java security manager verwenden, müssen Sie sicherstellen, dass die JAR-Dateien von IBM MQ classes for JMS über Schreib-/Lesezugriff für die Datei verfügen.

com.ibm.mq.jms.allowlist.discover (JMS 2.0) und com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- Wenn diese Eigenschaft nicht festgelegt oder auf 'false' gesetzt ist, wird der Zulassungslistenmechanismus in der Betriebsart ENFORCEMENT ausgeführt.
- Wenn diese Eigenschaft auf 'true' gesetzt ist und die Zulassungsliste als Datei-URI angegeben wurde, wird der Zulassungslistenmechanismus in der Betriebsart DISCOVERY ausgeführt.
- Wenn diese Eigenschaft auf 'true' gesetzt ist und die Zulassungsliste als Liste mit Klassennamen angegeben wurde, löst der Zulassungslistenmechanismus eine geeignete Ausnahmebedingung aus.
- Wenn diese Eigenschaft auf "true" gesetzt ist und die Zulassungsliste nicht mit Eigenschaft com.ibm.mq.jms.allowlist oder com.ibm.mq.jakarta.jms.allowlistangegeben wurde, ist der Zulassungslistenmechanismus inaktiv.
- Wenn diese Eigenschaft auf 'true' gesetzt ist und die Zulassungslistendatei bereits vorhanden ist, löst der Zulassungslistenmechanismus die Ausnahmebedingung java.io.InvalidClassException aus und es werden keine Einträge zur Datei hinzugefügt.

com.ibm.mq.jms.allowlist.mode (JMS 2.0) und com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)
Diese Zeichenfolgeeigenschaft kann auf drei Arten angegeben werden:

- Wenn diese Eigenschaft auf SERIALIZE gesetzt ist, wird in der Betriebsart ENFORCEMENT die Zulassungslistenprüfung nur für die Methode ObjectMessage.setObject() durchgeführt.
- Wenn diese Eigenschaft auf DESERIALIZE gesetzt ist, wird in der Betriebsart ENFORCEMENT die Zulassungslistenprüfung nur für die Methode ObjectMessage.getObject() durchgeführt.
- Wenn diese Eigenschaft nicht festgelegt oder auf einen anderen Wert gesetzt ist, wird in der Betriebsart ENFORCEMENT die Zulassungslistenprüfung sowohl für die Methode ObjectMessage.getObject() als auch für die Methode ObjectMessage.setObject() durchgeführt.

Format der Zulassungslistendatei

Das Format der Zulassungslistendatei hat folgende Hauptmerkmale:

- Die Zulassungslistendatei verwendet die Standarddateicodierung der Plattform mit plattformspezifischen Zeilenenden.

  Anmerkung: Wird eine Zulassungslistendatei verwendet, wird sie immer mit der Standarddateicodierung für die JVM geschrieben und gelesen.

  Dies ist in Ordnung, wenn die Zulassungslistendatei auf eine der folgenden Arten generiert wird:

  - z/OS Generiert von einer eigenständigen Anwendung, die unter z/OS ausgeführt und von anderen eigenständigen Anwendungen, die ebenfalls unter z/OS ausgeführt werden, verwendet wird.
  - Generiert von einer Anwendung, die innerhalb von WebSphere Application Server auf einer beliebigen Plattform ausgeführt und von einer anderen Instanz von WebSphere Application Server verwendet wird.
  - Multi Generiert von einer eigenständigen Anwendung, die in IBM MQ for Multiplatforms ausgeführt und von anderen eigenständigen Anwendungen, die in IBM MQ for Multiplatforms ausgeführt werden, oder von Anwendungen, die innerhalb von WebSphere Application Server auf einer beliebigen Plattform ausgeführt werden, verwendet wird.

  Da WebSphere Application Server ASCII und eine eigenständige JVM EBCDIC verwendet, gibt es jedoch Probleme mit der Dateicodierung, wenn die Zulassungslistendatei auf eine der folgenden Arten generiert wird:

  - Generiert unter z/OS und dann von eigenständigen Anwendungen, die auf einer anderen Plattform als z/OS ausgeführt werden, oder von WebSphere Application Server verwendet.
  - Generiert entweder von WebSphere Application Server oder einer eigenständigen Anwendung, die auf einer anderen Plattform als z/OS ausgeführt wird, und dann von einer eigenständigen Anwendung unter z/OS verwendet.
- Jede nicht leere Zeile enthält einen vollständig qualifizierten Klassennamen. Leere Zeilen werden ignoriert.
- Kommentare können eingefügt werden - alles, was einem Zeichen '#' bis zum Ende der Zeile folgt, wird ignoriert.
- Es gibt einen sehr einfachen Platzhaltermechanismus:
  - '*' kann das **letzte** Element eines Klassennamens sein.
  - '*' entspricht einem **einzelnen** Element eines Klassennamens, d. h. der Klasse, aber keinem Teil des Pakets.

  Zum Beispiel würde `com.ibm.mq.*` mit `com.ibm.mq.MQMessage` übereinstimmen, aber nicht mit `com.ibm.mq.jmqi.remote.api.RemoteFAP`.

  Platzhalter funktionieren nicht für Klassen im Standardpaket, d. h. für Klassen ohne expliziten Paketnamen; der Klassenname "*" wird deshalb zurückgewiesen.
- Falsch formatierte Zulassungslistendateien, z. B. Dateien mit einem Eintrag wie etwa `com.ibm.mq.*.Message`, in dem das Platzhalterzeichen nicht das letzte Element ist, führen dazu, dass die Ausnahmebedingung java.lang.IllegalArgumentException ausgelöst wird.
- Eine leere Zulassungslistendatei hat zur Folge, dass die Verwendung von ObjectMessage vollständig inaktiviert wird.

Format der Zulassungsliste als durch Kommas getrennte Liste

Für eine Zulassungsliste im Format einer durch Kommas getrennten Liste ist der gleiche Platzhaltermechanismus verfügbar.

- Das Zeichen '*' kann vom Betriebssystem erweitert werden, wenn es in einer Befehlszeile oder in einem Shell-Script oder einer Batchdatei angegeben ist, sodass möglicherweise eine Sonderbehandlung nötig ist.
- Das Kommentarzeichen '#' ist nur gültig, wenn eine Datei angegeben ist. Wenn die Zulassungsliste als durch Kommas getrennte Liste von Klassennamen angegeben wird, wird sie unter der Annahme, dass das Betriebssystem oder die Shell sie nicht verarbeitet, als Standardkommentarzeichen in vielen AIX and Linux -Shells als normales Zeichen behandelt.

Wann kommt der Zulassungslistenmechanismus zum Einsatz?

Der Zulassungslistenmechanismus wird initialisiert, wenn die Anwendung das erste Mal die ObjectMessage-Methode setMessage() oder getMessage() ausführt.

Die Systemeigenschaften werden ausgewertet, die Zulassungslistendatei wird geöffnet und in der Betriebsart ENFORCEMENT wird bei der Initialisierung des Mechanismus die Liste der in der Zulassungsliste aufgeführten Klassen geladen. An diesem Punkt wird ein Eintrag in die IBM MQ JMS-Protokolldatei für die Anwendung geschrieben.

Wenn der Mechanismus initialisiert ist, können seine Parameter möglicherweise nicht geändert werden. Der Zeitpunkt der Initialisierung ist schwer vorhersagbar, da er vom Anwendungsverhalten abhängt. Die Systemeigenschafteneinstellungen und der Inhalt der Zulassungslistendatei sollten deshalb ab dem Zeitpunkt des Anwendungsstarts als festgelegt betrachtet werden. Ändern Sie nicht die Eigenschaften oder den Inhalt der Zulassungslistendatei, während die Anwendung aktiv ist, weil die Ergebnisse nicht garantiert sind.

Wichtige Überlegungen

Die beste Strategie zur Minderung der Risiken, die dem Java-Serialisierungsmechanismus innewohnen, besteht darin, alternative Methoden für die Datenübertragung zu untersuchen, z. B. die Verwendung von JSON statt ObjectMessage. Durch die Verwendung von Advanced Message Security-(AMS-)Mechanismen kann die Sicherheit erhöht werden, indem sichergestellt wird, dass Nachrichten aus vertrauenswürdigen Quellen stammen

Wenn Sie den Java security manager-Mechanismus mit Ihrer Anwendung verwenden, müssen Sie folgende Berechtigungen erteilen:

- FilePermission in einer beliebigen Zulassungslistendatei, die Sie verwenden, mit Leseberechtigung für ENFORCEMENT-Modus und Schreibberechtigung für DISCOVER-Modus.
- JMS 2.0 PropertyPermission (read) für die Eigenschaften **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover** und **com.ibm.mq.jms.allowlist.mode**.
- JM 3.0 PropertyPermission (read) für die Eigenschaften **com.ibm.mq.jakarta.jms.allowlist**, **com.ibm.mq.jakarta.jms.allowlist.discover** und **com.ibm.mq.jakarta.jms.allowlist.mode**.

Weitere Informationen

Weitere Informationen zu Zulassungslisten finden Sie in den Abschnitten „JMS -oder Jakarta Messaging -Zulassungsliste einrichten und verwenden” auf Seite 141 und „Verwendung einer Zulassungsliste in WebSphere Application Server” auf Seite 144.

Zugehörige Konzepte

„IBM MQ classes for JMS-Anwendungen unter dem Java security manager ausführen” auf Seite 112
IBM MQ classes for JMS können mit aktiviertem Java security manager ausgeführt werden. Um Anwendungen erfolgreich mit aktiviertem Java security manager ausführen zu können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtlinienkonfigurationsdatei konfigurieren.

JMS -oder Jakarta Messaging -Zulassungsliste einrichten und verwenden

In diesem Abschnitt erfahren Sie, wie eine Zulassungsliste funktioniert und wie Sie eine Zulassungsliste mithilfe der in IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging enthaltenen Funktionalität einrichten, um eine Zulassungslistendatei zu generieren, die eine Liste der Typen von ObjectMessages enthält, die eine Anwendung verarbeiten kann.

Vorbereitende Schritte

Wichtig:

Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Dies schließt einige Java -Systemeigenschaftsnamen ein, die in diesem Thema erwähnt werden. Sie müssen eine vorhandene Konfiguration nicht ändern. Die vorherigen Systemeigenschaftsnamen funktionieren auch weiterhin.

Vor Beginn dieser Task sollten Sie die Informationen im Abschnitt „Zulassungslistenkonzepte” auf Seite 138 gelesen und zur Kenntnis genommen haben.

Informationen zu diesem Vorgang

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.

Wenn Sie die Zulassungslistenfunktion aktiviert haben, verwenden die IBM MQ classes for JMS diese Funktion auf folgende Arten:

- Wenn eine Anwendung eine ObjectMessage senden möchte, kann sie sie auf zwei Arten erstellen, indem sie die folgenden Methoden aufruft:
  - die Methode 'session.createObjectMessage(Serializable)', die das Objekt übergibt, das in der Nachricht enthalten sein soll
  - die Methode 'Session.createObjectMessage()', um eine leere ObjectMessage zu erstellen, mit anschließendem Aufruf von 'ObjectMessage.setObject(Serializable)', um das Objekt zu speichern, das in der ObjectMessage gesendet werden soll

  Wenn die Methode 'Session.createObjectMessage(Serializable)' oder die Methode 'ObjectMessage.setObject(Serializable)' aufgerufen wird, überprüfen die Klassen für JMS, ob das übergebene Objekt einem der Typen entspricht, die in der Zulassungsliste aufgeführt sind.

  Wenn dies der Fall ist, wird das Objekt serialisiert und in der ObjectMessage gespeichert. Ist der Typ des Objekts jedoch nicht in der Zulassungsliste aufgeführt, geben die IBM MQ classes for JMS eine JMS-Ausnahmebedingung (JMSException) mit der Nachricht

  ```
  JMSCC0052: Beim Serialisieren des folgenden Objekts ist eine Ausnahmebedingung aufgetreten:
  'java.io.InvalidClassException: <Objektklasse>; Die Klasse wird möglicherweise nicht seriali-
  siert
  bzw. deserialisiert, da sie nicht in die Zulassungsliste '<Zulassungsliste>' eingeschlossen wur-
  de.
  ```

  an die Anwendung zurück.

  Wichtig: Wenn die Methode 'Session.createObjectMessage(Serializable)' die Ausnahme auslöst, wird die ObjectMessage nicht erstellt. Ähnlich gilt: Wenn die Methode 'ObjectMessage.setObject(Serializable)' die JMS-Ausnahmebedingung (JMSException) auslöst, wird das Objekt der ObjectMessage nicht hinzugefügt.
- Wenn eine Anwendung eine ObjectMessage empfängt, ruft sie die Methode 'ObjectMessage.getObject()' auf, um das darin enthaltene Objekt abzurufen. Wird diese Methode aufgerufen, überprüfen die IBM MQ classes for JMS, ob der Typ des in der ObjectMessage enthaltenen Objekts einem Typ entspricht, der in der Zulassungsliste aufgeführt ist.

  Wenn dies der Fall ist, wird das Objekt deserialisiert und an die Anwendung zurückgegeben. Ist der Typ des Objekts jedoch nicht in der Zulassungsliste aufgeführt, geben die IBM MQ classes for JMS eine JMS-Ausnahmebedingung (JMSException) mit der Nachricht

  ```
  JMSCC0053: An exception occurred while deserializing a message:
  'java.io.InvalidClassException: <object class>; The class may not be
  serialized or deserialized as it has not been included in the
  Zulassungsliste '< Zulassungsliste>'. '.
  ```

  an die Anwendung zurück.

Angenommen, Ihre Anwendung enthält den folgenden Code, um eine ObjectMessage zu senden, die ein Objekt des Typs 'java.net.URI' enthält:

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
```

```
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

Da die Zulassungslistenfunktion nicht aktiviert ist, kann die Anwendung die Nachricht erfolgreich an das gewünschte Ziel übergeben.

Wenn Sie eine Datei mit dem Namen `C:\allowlist.txt` erstellen, die einen einzelnen Eintrag (`java.net.`URL) enthält, und die Anwendung erneut mit der Systemeigenschaft Java starten, gehen Sie wie folgt vor:

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

erneut starten, wird die Zulassungslistenfunktion aktiviert. Die Anwendung kann die ObjectMessage, die ein Objekt des Typs java.net.URI enthält, weiterhin erstellen und senden, da der Typ in der Zulassungsliste aufgeführt ist.

Wenn Sie jedoch die Datei `allowlist.txt` so ändern, dass sie nur den Eintrag `java.util.Calendar` enthält, während die Zulassungslistenfunktion weiter aktiv ist, und die Anwendung dann

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

aufruft, überprüfen die IBM MQ classes for JMS die Zulassungsliste und stellen fest, dass sie keinen Eintrag für java.net.URI enthält.

Folglich wird eine JMS-Ausnahmebedingung (JMSException) mit der Nachricht JMSCC0052 ausgelöst.

Nehmen wir nun einen ähnlichen Fall an: Sie haben eine andere Anwendung, die ObjectMessages mithilfe von folgendem Code empfängt:

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

Wenn die Zulassungslistenfunktion nicht aktiviert ist, kann die Anwendung ObjectMessages empfangen, die ein Objekt eines beliebigen Typs enthalten. Die Anwendung überprüft dann, ob das Objekt den Typ 'java.net.URL' aufweist, bevor die entsprechende Verarbeitung erfolgt.

Wenn Sie die Anwendung nun mit festgelegter Java-Systemeigenschaft:

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

starten, wird die Zulassungslistenfunktion inaktiviert. Wenn die Anwendung:

```
Object messageBody = objectMessage.getObject();
```

aufruft, gibt die Methode 'ObjectMessage.getObject()' nur Objekte des Typs 'java.net.URL' zurück.

Wenn das in der ObjectMessage enthaltene Objekt nicht diesen Typ aufweist, löst die Methode 'ObjectMessage.getObject()' eine JMS-Ausnahmebedingung aus, die die Nachricht JMSCC0053 enthält. Die Anwendung muss dann entscheiden, was mit der Nachricht geschehen soll. Die Nachricht könnte beispielsweise in die Warteschlange für nicht zustellbare Nachrichten für diesen Warteschlangenmanager verschoben werden.

Die Rückgabe der Anwendung erfolgt nur normal, wenn das Objekt in der ObjectMessage den Typ 'java.net.URL' aufweist.

Vorgehensweise

1. Führen Sie die Anwendung, die ObjectMessagesverarbeitet, mit den folgenden Java -Systemeigenschaften aus:

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

Wenn die Anwendung ausgeführt wird, erstellen die IBM MQ classes for JMS eine Datei mit den Objekttypen, die von der Anwendung verarbeitet wurden.

2. Stoppen Sie die Anwendung, nachdem sie über einen bestimmten Zeitraum ein repräsentatives Beispiel der ObjectMessages verarbeitet hat.

   Die Zulassungslistendatei enthält jetzt eine Liste aller Objekttypen, die in den ObjectMessages enthalten waren, die von der Anwendung während ihrer Ausführung verarbeitet wurden.

   Wenn Sie die Anwendung ausreichend lange ausgeführt haben, enthält diese Liste alle möglichen Objekttypen, die in ObjectMessages enthalten sind, die die Anwendung wahrscheinlich verarbeiten wird.

3. Starten Sie die Anwendung erneut, wobei die folgende Systemeigenschaft festgelegt ist:

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Dies aktiviert die Zulassungslistenfunktion, d. h., wenn die IBM MQ classes for JMS eine ObjectMessage eines Typs erkennen, der nicht in der Zulassungsliste aufgeführt ist, wird eine JMS-Ausnahmebedingung (JMSException) ausgelöst, die entweder die Nachricht JMSCC0052 oder JMSCC0053 enthält.

Verwendung einer Zulassungsliste in WebSphere Application Server

Funktionsweise der Zulassungslistenfunktion der IBM MQ classes for JMS in WebSphere Application Server

Wichtig:

Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Dies schließt einige Java -Systemeigenschaftsnamen ein, die in diesem Thema erwähnt werden. Sie müssen eine vorhandene Konfiguration nicht ändern. Die vorherigen Systemeigenschaftsnamen funktionieren auch weiterhin.

Sie müssen sicherstellen, dass Ihre WebSphere Application Server-Installation eine Version des IBM MQ-Ressourcenadapters enthält, die die Zulassungslistenfunktion unterstützt.

Weitere Informationen zur Verwendung der beiden Produkte finden Sie unter „IBM MQ und WebSphere Application Server gemeinsam verwenden" auf Seite 526.

Ab IBM MQ 9.0.0 Fix Pack 1 ist die entsprechende Funktionalität enthalten.

Nach der Aktualisierung des Anwendungsservers können Sie die Java-Systemeigenschaften:

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

verwenden, die im Abschnitt „JMS -oder Jakarta Messaging -Zulassungsliste einrichten und verwenden" auf Seite 141 beschrieben werden.

Anmerkung: Sie müssen die Java-Systemeigenschaften als generische JVM-Argumente auf der Java Virtual Machine festlegen, die zur Ausführung des Anwendungsservers verwendet wird, und den Anwendungsserver erneut starten, damit die Änderungen wirksam werden.

Weitere Informationen finden Sie unter Java Virtual Machine-Einstellungen im Abschnitt *Generische JVM-Argumente*.

Um die Eigenschaften festzulegen, rufen Sie das Java Virtual Machine-Fenster in *Prozessdefinitionen* auf und geben Sie das entsprechende Argument ein.

Die folgende Einstellung:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

bewirkt, dass der Anwendungsserver die Zulassungsliste *IhreBenutzerID_MeinObjekt* verwendet. Nur Objekte des betreffenden Typs werden vom Anwendungsserver verarbeitet.

Mit den folgenden Einstellungen:

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

konfigurieren Sie den Anwendungsserver für die Verwendung des Modus *Discover*. Dabei werden Einzelheiten zu JMS-ObjectMessages, die der Anwendungsserver verarbeitet, in der Datei `C:\allowlist.txt` aufgezeichnet.

Die folgende Einstellung:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

führt dazu, dass der Anwendungsserver die Datei `C:/allowlist.txt` lädt und mit den Informationen in dieser Datei die Zulassungsliste ermittelt.

Zugehörige Konzepte

„IBM MQ classes for JMS-Anwendungen unter dem Java security manager ausführen” auf Seite 112
IBM MQ classes for JMS können mit aktiviertem Java security manager ausgeführt werden. Um Anwendungen erfolgreich mit aktiviertem Java security manager ausführen zu können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtlinienkonfigurationsdatei konfigurieren.

Zeichenfolgekonvertierungen in IBM MQ classes for JMS

Die IBM MQ classes for JMS verwenden CharsetEncoders und CharsetDecoders direkt für die Zeichenfolgenkonvertierung. Das Standardverhalten für die Zeichenfolgenkonvertierung kann mit zwei Systemeigenschaften konfiguriert werden. Die Verarbeitung von Nachrichten, die Zeichen enthalten, die nicht zugeordnet werden können, kann mithilfe Nachrichteneigenschaften zum Festlegen von 'UnmappableCharacterAction' und der Ersetzungsbyte konfiguriert werden.

Vor IBM MQ 8.0 erfolgten Zeichenfolgekonvertierungen in IBM MQ classes for JMS durch Aufrufe der Methoden `java.nio.charset.Charset.decode(ByteBuffer)` und `Charset.encode(CharBuffer)`.

Bei Verwendung einer dieser Methoden werden fehlerhafte oder nicht übersetzbare Daten standardmäßig ersetzt (REPLACE). Durch dieses Verhalten können Fehler in Anwendungen unkenntlich gemacht werden und es kann zu nicht erwarteten Zeichen in übersetzten Daten (z. B. ?) kommen.

Ab IBM MQ 8.0 werden solche Probleme früher und effektiver erkannt, da die IBM MQ classes for JMS-Klassen für Java CharsetEncoders und CharsetDecoders direkt verwenden und die Behandlung von fehlerhaften und nicht umsetzbaren Daten explizit konfigurieren. Das Standardverhalten besteht darin, dass solche Probleme gemeldet werden (REPORT), indem eine entsprechende MQ-Ausnahme (`MQException`) ausgelöst wird.

konfigurieren

Die Umsetzung von UTF-16 (der in Java verwendeten Zeichendarstellung) in einen nativen Zeichensatz wie beispielsweise UTF-8 wird als Codierung (*encoding*) und die Umsetzung in die andere Richtung als Decodierung (*decoding*) bezeichnet.

Derzeit gilt bei der Decodierung das Standardverhalten für `CharsetDecoders`, wobei Fehler durch Auslösen einer Ausnahme gemeldet werden.

Eine Einstellung wird zur Angabe von `java.nio.charset.CodingErrorAction` verwendet, um die Fehlerbehandlung bei der Codierung und Decodierung zu steuern. Eine andere Einstellung wird zur Steuerung des bzw. der Ersetzungsbytes bei der Codierung verwendet. Bei Decodierungsvorgängen wird die standardmäßige Java-Ersetzungszeichenfolge verwendet.

Einstellungen für 'UnmappableCharacterAction' und Ersetzungsbyte in IBM MQ classes for JMS

Ab IBM MQ 8.0 stehen die beiden folgenden Eigenschaften zum Festlegen von 'UnmappableCharacterAction' und Ersetzungsbyte zur Verfügung. Die Definitionen der entsprechenden Konstanten sind in `com.ibm.msg.client.wmq.WMQConstants` enthalten.

JMS_IBM_UNMAPPABLE_ACTION
Zur Festlegung oder zum Abrufen der `CodingErrorAction`, die angewandt werden soll, wenn bei einem Codierungs- oder Decodierungsvorgang ein Zeichen nicht zugeordnet werden kann.

Sie sollten dies wie folgt als `CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()` festlegen:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT
Zur Festlegung oder zum Abrufen der Ersetzungsbytes, die angewandt werden sollen, wenn bei einem Codierungsvorgang ein Zeichen nicht zugeordnet werden kann.

Bei Decodierungsvorgängen wird die standardmäßige Java-Ersetzungszeichenfolge verwendet.

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

Die Eigenschaften `JMS_IBM_UNMAPPABLE_ACTION` und `JMS_IBM_UNMAPPABLE_REPLACEMENT>` können für Ziele oder Nachrichten festgelegt werden. Wenn in einer Nachricht ein Wert festgelegt ist, überschreibt dieser den Wert auf dem Ziel, an das die Nachricht gesendet wird.

Beachten Sie, dass `JMS_IBM_UNMAPPABLE_REPLACEMENT` als einzelnes Byte festgelegt werden muss.

Systemeigenschaften zum Festlegen von Systemstandardwerten

Ab IBM MQ 8.0 sind die folgenden zwei Java-Systemeigenschaften verfügbar, um das Standardverhalten für die Zeichenfolgekonvertierung zu konfigurieren.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Gibt an, welche Aktion für nicht übersetzbare Daten bei der Codierung und Decodierung durchgeführt werden soll. Der Wert kann `REPORT`, `REPLACE` oder `IGNORE` lauten.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Legt die Ersetzungsbytes fest, die angewandt werden sollen, wenn in einem Codierungsvorgang ein Zeichen nicht zugeordnet werden kann, bzw. ruft diese ab. Bei Decodierungsvorgängen wird die standardmäßige Java-Ersetzungszeichenfolge verwendet.

Damit es nicht zu Unklarheiten zwischen den Darstellungen in Form von Java-Zeichen und nativen Bytes kommt, sollten Sie für `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` eine Dezimalzahl angeben, die das Ersetzungsbyte im nativen Zeichensatz darstellt.

Der Dezimalwert von ? als natives Byte lautet beispielsweise 63, wenn der native Zeichensatz auf ASCII basiert (z. B. ISO-8859-1) und 111 bei dem nativen Zeichensatz EBCDIC.

Anmerkung: Falls bei einem MQMD- oder MQMessage-Objekt das Feld **`unmappableAction`** oder **`unMappableReplacement`** festgelegt ist, haben die Werte dieser Felder Vorrang vor den Java-Systemeigenschaften. Auf diese Weise können die Werte der Java-Systemeigenschaften bei Bedarf für jede Nachricht überschrieben werden.

Zugehörige Konzepte

„Zeichenfolgekonvertierungen in IBM MQ classes for Java” auf Seite 369
Die IBM MQ classes for Java verwenden CharsetEncoders und CharsetDecoders direkt für die Zeichenfolgenkonvertierung. Das Standardverhalten für die Zeichenfolgenkonvertierung kann mit zwei Systemeigenschaften konfiguriert werden. Die Verarbeitung von Nachrichten, die Zeichen enthalten, die nicht zugeordnet werden können, kann mithilfe von `com.ibm.mq.MQMD` konfiguriert werden.

IBM MQ classes for JMS/Jakarta Messaging -Anwendungen schreiben

Nach einer kurzen Einführung in das JMS -Modell finden Sie in diesem Abschnitt ausführliche Anleitungen zum Schreiben von IBM MQ classes for JMS -und IBM MQ classes for Jakarta Messaging -Anwendungen.

Informationen zu diesem Vorgang

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.

Zugehörige Konzepte

IBM MQ classes for Jakarta Messaging: Übersicht

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 beide Messaging-Provider. Sie definieren, wie sich JMS -und Jakarta Messaging -Objekte auf IBM MQ -Konzepte beziehen. Die Spezifikationen JMS und Jakarta Messaging erwarten, dass bestimmte JMS -und Jakarta Messaging -Objekte verwaltete Objekte sind.

JMS 2.0 IBM MQ 8.0 hat Unterstützung für die JMS 2.0 -Version des JMS -Standards hinzugefügt, mit der eine vereinfachte API eingeführt wurde, während auch die klassische API beibehalten wurde, aus JMS 1.1.

JM 3.0 Ab IBM MQ 9.3.0wird Jakarta Messaging 3.0 für die Entwicklung neuer Anwendungen unterstützt. IBM MQ 9.3.0 unterstützt weiterhin JMS 2.0 für vorhandene Anwendungen. Die Verwendung der JMS 2.0 -API und der Jakarta Messaging 3.0 -API in derselben Anwendung wird nicht unterstützt.

Anmerkung: Für Jakarta Messaging 3.0wird die Steuerung der JMS -Spezifikation von Oracle in Java Community Processverschoben. Oracle behält jedoch die Kontrolle über den "javax" -Namen, der in anderen Java -Technologien verwendet wird, die nicht in den Java Community-Prozess verschoben wurden. Obwohl Jakarta Messaging 3.0 funktional äquivalent zu JMS 2.0 ist, gibt es einige Unterschiede bei der Benennung:

- Der offizielle Name für Version 3.0 ist Jakarta Messaging und nicht Java Message Service.
- Die Paket-und Konstantennamen haben das Präfix `jakarta` und nicht `javax`. In JMS 2.0 ist die einleitende Verbindung zu einem Messaging-Provider beispielsweise ein `javax.jms.Connection` -Objekt und in Jakarta Messaging 3.0 ein `jakarta.jms.Connection` -Objekt.

JMS 2.0 Die javax.jms -Pakete definieren die JMS -Schnittstellen, und ein JMS -Provider implementiert diese Schnittstellen für ein bestimmtes Messaging-Produkt. IBM MQ classes for JMS stellen einen JMS-Provider dar, der die JMS-Schnittstellen für IBM MQ implementiert.

JM 3.0 Die jakarta.jms -Pakete definieren die Jakarta Messaging -Schnittstellen und ein Jakarta Messaging -Provider implementiert diese Schnittstellen für ein bestimmtes Messaging-Produkt. IBM MQ classes for Jakarta Messaging stellen einen Jakarta Messaging-Provider dar, der die Jakarta Messaging-Schnittstellen für IBM MQ implementiert.

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 9 auf Seite 149 sind diese Objekte und ihre Beziehungen dargestellt.

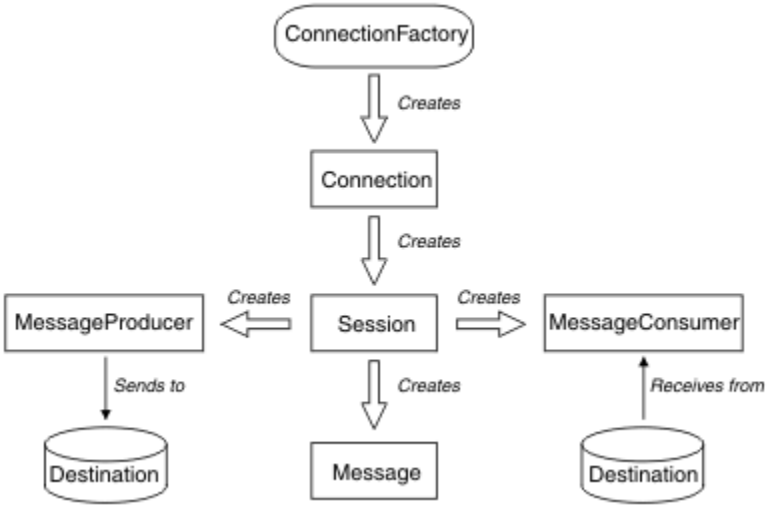


Abbildung 9. JMS-Objekte und ihre Beziehungen

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.

Messaging-Domänen

JMS unterstützt zwei Messaging-Stile:

- 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 15 auf Seite 150 sehen Sie eine Übersicht über die domänenunabhängigen JMS-Schnittstellen und deren domänenspezifischen Entsprechungen.

Tabelle 15. Domänenunabhängige und domänenspezifische JMS-Schnittstellen		
Domänenunabhängige Schnittstellen	**Domänenspezifische Schnittstellen für Punkt-zu-Punkt-Domäne**	**Domänenspezifische Schnittstellen für Publish/Subscribe-Domäne**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
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

Zugehörige Konzepte

IBM MQ Java-Sprachenschnittstellen

„Verbindungsfactorys und Ziele erstellen und konfigurieren” auf Seite 214

Eine IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Anwendung kann Verbindungsfactorys und Ziele erstellen, indem sie sie als verwaltete Objekte aus einem JNDI-Namensbereich (Java Naming and Directory Interface) abruft, die IBM JMS -Erweiterungen oder die IBM MQ JMS -Erweiterungen verwendet. Eine Anwendung kann die IBM JMS-Erweiterungen oder IBM MQ-JMS-Erweiterungen auch zum Festlegen der Eigenschaften von Verbindungsfactorys und Zielen verwenden.

JMS-Nachrichten

JMS-Nachrichten bestehen aus einem Header, aus Eigenschaften und aus einem Hauptteil. JMS definiert fünf Typen des Nachrichtenhauptteils.

JMS-Nachrichten setzen sich aus folgenden Bestandteilen zusammen:

Header
Alle Nachrichten unterstützen dieselbe Gruppe von Headerfeldern. Headerfelder enthalten Werte, die sowohl von Clients als auch Providern zur Identifizierung und Weiterleitung von Nachrichten genutzt werden.

Eigenschaften
Jede Nachricht enthält eine integrierte Funktion zur Unterstützung anwendungsdefinierter Eigenschaftswerte. Die Eigenschaften bieten einen effizienten Mechanismus zum Filtern anwendungsdefinierter Nachrichten.

Hauptteil
JMS definiert fünf Nachrichtenhauptteiltypen, die den größten Teil der derzeit verwendeten Messaging-Stile abdecken:

Datenstrom
Ein Datenstrom mit primitiven Java-Elementwerten. Er wird nacheinander gefüllt und gelesen.

Zuordnung
Eine Gruppe von Name/Wert-Paaren, bei denen die Namen Zeichenfolgen und die Werte primitive Java-Typen sind. Die Einträge können nacheinander oder nach dem Namen aufgerufen werden. Die Reihenfolge der Einträge ist nicht definiert.

Text
Eine Nachricht, die eine 'java.lang.String' enthält.

Objekt
Eine Nachricht, die ein serialisierbares Java-Objekt enthält

Bytes
Ein Datenstrom nicht interpretierter Bytes. Dieser Nachrichtentyp ist für die buchstäbliche Codierung eines Hauptteils vorgesehen, damit dieser einem bestehenden Nachrichtenformat entspricht.

Das Headerfeld 'JMSCorrelationID' wird zum Verknüpfen einer Nachricht mit einer anderen verwendet. Für gewöhnlich verknüpft es eine Antwortnachricht mit der zugehörigen Anforderungsnachricht. Das Feld 'JMSCorrelationID' kann eine providerspezifische Nachrichten-ID, eine anwendungsspezifische Zeichenfolge oder einen providernativen Byte[]-Wert enthalten.

Nachrichtenselektoren in JMS
Nachrichten können anwendungsdefinierte Eigenschaftswerte enthalten. Eine Anwendung kann Nachrichtenselektoren verwenden, damit ein JMS-Provider Nachrichten filtert.

Eine Nachricht enthält eine integrierte Funktion zur Unterstützung anwendungsdefinierter Eigenschaftswerte. Tatsächlich wird damit ein Mechanismus bereitgestellt, mit dem anwendungsspezifische Headerfelder einer Nachricht hinzugefügt werden können. Eigenschaften ermöglichen es einer Anwendung, unter Verwendung von Nachrichtenselektoren einen JMS-Provider zu veranlassen, für sie Nachrichten mithilfe von anwendungsspezifischen Kriterien auszuwählen oder zu filtern. Anwendungsdefinierte Eigenschaften müssen die folgenden Regeln einhalten:

- Eigenschaftsnamen müssen die Regeln für eine Nachrichtenselektor-ID einhalten.
- Für Eigenschaftswerte sind folgende Werte möglich: Boolean, byte, short, int, long, float, double und String.
- Die Präfixe JMSX und JMS_, gefolgt vom Namen, sind reserviert.

Eigenschaftswerte werden vor dem Senden einer Nachricht festgelegt. Wenn ein Client eine Nachricht erhält, sind die Nachrichteneigenschaften schreibgeschützt. Wenn ein Client versucht, zu diesem Zeitpunkt Eigenschaften festzulegen, wird eine Ausnahmebedingung (MessageNotWriteableException) ausgelöst. Wenn 'clearProperties' aufgerufen wird, können die Eigenschaften gelesen und Werte in sie geschrieben werden.

Ein Eigenschaftswert kann unter Umständen einen Wert in einem Nachrichtenhauptteil duplizieren. JMS definiert keine Richtlinie dafür, was als Eigenschaft festgelegt werden kann. Anwendungsentwickler müssen jedoch beachten, dass JMS-Provider Daten in einem Nachrichtenhauptteil wahrscheinlich effizienter verarbeiten als Daten in Nachrichteneigenschaften. Um eine optimale Leistung zu erzielen, dürfen Anwendungen Nachrichteneigenschaften nur verwenden, wenn sie einen Nachrichtenheader anpassen müssen. Der Hauptgrund für diese Aktion ist die Unterstützung einer angepassten Nachrichtenauswahl.

Ein JMS-Nachrichtenselektor ermöglicht es einem Client, unter Verwendung des Nachrichtenheaders die Nachrichten anzugeben, an denen er interessiert ist. Es werden nur Nachrichten zugestellt, die mit dem Selektor übereinstimmen.

Nachrichtenselektoren können keine Nachrichtenhauptteilwerte referenzieren.

Ein Nachrichtenselektor ergibt eine Übereinstimmung mit einer Nachricht, wenn der Selektor beim Ersetzen des Nachrichtenheaderfelds und der Eigenschaftswerte durch ihre entsprechenden IDs im Selektor in 'wahr' ausgewertet wird.

Ein Nachrichtenselektor ist eine Zeichenfolge (String), deren Syntax auf einer Untergruppe der SQL92-Syntax für Bedingungsausdrücke basiert. Die Reihenfolge der Auswertung eines Nachrichtenselektors erfolgt innerhalb einer Rangfolgestufe von links nach rechts. Sie können diese Reihenfolge durch die Verwendung von Klammern ändern. Vordefinierte Selektorliterale und Operatorbezeichnungen werden hier in Großbuchstaben geschrieben, die Groß-/Kleinschreibung muss jedoch nicht beachtet werden.

Inhalt eines Nachrichtenselektors

Ein Nachrichtenselektor kann Folgendes enthalten:

- Literale
  - Ein Zeichenfolgeliteral wird in Anführungszeichen eingeschlossen. Ein doppeltes Anführungszeichen steht für ein Hochkomma. Beispiele: 'literal' und 'literal''s'. Genau wie Java-Zeichenfolgeliterale verwenden sie die Unicode-Zeichencodierung.
  - Ein exaktes numerisches Literal ist ein numerischer Wert ohne Dezimalzeichen, zum Beispiel 57, -957 und +62. Zahlen im Bereich 'Java long' werden unterstützt.
  - Ein näherungsweise berechnetes numerisches Literal ist ein numerischer Wert in Exponentialschreibweise (zum Beispiel 7E3 oder -57.9E2) oder ein numerischer Wert mit einer Dezimalstelle (zum Beispiel 7,, -95,7 oder +6,2). Zahlen im Bereich 'Java double' werden unterstützt.
  - TRUE und FALSE sind boolesche Literale.
- Kennungen:

  - Eine ID ist eine unbegrenzt lange Folge von Java-Buchstaben und Java-Ziffern, wovon die erste ein Java-Buchstabe sein muss. Ein Buchstabe ist ein beliebiges Zeichen, für das die Methode 'Character.isJavaLetter' den Wert 'true' zurückgibt. Dazu zählen auch _ und $. Ein Buchstabe oder eine Ziffer ist ein beliebiges Zeichen, für das die Methode 'Character.isJavaLetterOrDigit' den Wert 'true' zurückgibt.
  - Die Namen NULL, TRUE oder FALSE können nicht als IDs verwendet werden.
  - NOT, AND, OR, BETWEEN, LIKE, IN oder IS können nicht als IDs verwendet werden.
  - Bei Kennungen handelt es sich entweder Verweise auf Headerfelder oder auf Eigenschaften.
  - Bei IDs muss die Groß-/Kleinschreibung beachtet werden.
  - Die Feldreferenzen von Nachrichtenheadern sind auf Folgendes beschränkt:

    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    Die Werte von JMSMessageID, JMSTimestamp, JMSCorrelationID und JMSType können null sein. Ist dies der Fall, werden sie als NULL-Wert behandelt.
  - Ein Name, der mit JMSX beginnt, ist ein von JMS definierter Eigenschaftsname.
  - Ein Name, der mit JMS_ beginnt, ist ein providerspezifischer Eigenschaftsname.
  - Ein Name, der nicht mit JMS beginnt, ist ein anwendungsspezifischer Eigenschaftsname. Falls eine Eigenschaft referenziert wird, die in einer Nachricht nicht vorhanden ist, ist ihr Wert NULL. Ist sie nicht vorhanden, ist ihr Wert der entsprechende Eigenschaftswert.
- Der Leerraum ist mit demjenigen identisch, der für Java definiert ist: Leerzeichen, Horizontaltabulator, Seitenvorschub und Zeilenabschlusszeichen.
- Ausdrücke:
  - Ein Selektor ist ein Bedingungsausdruck. Ein Selektor, der als 'wahr' (true) ausgewertet wird, ist eine Übereinstimmung; ein Selektor, der als 'falsch' (false) oder 'unbekannt' (unknown) ausgewertet wird, ist keine Übereinstimmung.
  - Arithmetische Ausdrücke bestehen aus sich selbst, aus Rechenoperationen, aus IDs (mit einem Wert, der als numerisches Literal behandelt wird) und aus numerischen Literalen.
  - Bedingungsausdrücke bestehen aus sich selbst, aus Vergleichsoperationen und aus logischen Operationen.
- Die Standardverwendung von Klammern () zur Festlegung der Auswertungsreihenfolge von Ausdrücken wird unterstützt.
- Logische Operatoren mit Rangordnung: NOT, AND, OR.
- Vergleichsoperatoren: =, >, >=, <, <=, <> (ungleich).
  - Nur Werte des gleichen Typs können verglichen werden. Hierbei gibt es jedoch eine Ausnahme: Exakte numerische Werte und näherungsweise berechnete numerische Werte können miteinander verglichen werden. (Die erforderliche Typenkonvertierung wird durch die Regeln der numerischen Java-Hochstufung definiert.) Falls versucht wird, verschiedene Typen zu vergleichen, ist der Selektor immer 'false'.
  - Der Vergleich von Zeichenfolgen und booleschen Werten ist auf = und <> beschränkt. Zwei Zeichenfolgen sind nur gleich, wenn sie dieselbe Zeichenfolge enthalten.
- Arithmetische Operatoren mit Rangordnung:
  - +, - (unär).
  - *, / (Multiplikation und Division).

  - +, - (Addition und Subtraktion).
  - Rechenoperationen für einen NULL-Wert werden nicht unterstützt. Bei einem entsprechenden Versuch wird der komplette Selektor immer als 'False' zurückgegeben.
  - Arithmetische Operationen müssen Java Numeric Promotion anwenden.
- Vergleichsoperator arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 und arithmetic-expr3:
  - Age BETWEEN 15 and 19 ist äquivalent zu age >= 15 AND age <= 19.
  - Age NOT BETWEEN 15 and 19 ist äquivalent zu age < 15 OR age > 19.
  - Falls einer der Ausdrücke einer BETWEEN-Operation NULL ist, ist der Wert der Operation 'false'. Falls einer der Ausdrücke einer NOT BETWEEN-Operation NULL ist, ist der Wert der Operation 'true'.
- identifier [NOT] IN (string-literal1, string-literal2, ...) Vergleichsoperator, wobei die Kennung einen Zeichenfolgewert oder den Wert NULL hat.
  - Country IN ('UK', 'US', 'France') ist wahr (true) für 'UK' und falsch (false) für 'Peru'. Dies ist äquivalent zu dem Ausdruck (Country = 'UK') OR (Country = 'US') OR (Country = 'France').
  - Country NOT IN ('UK', 'US', 'France') ist falsch (false) für 'UK' und wahr (true) für 'Peru'. Dies ist äquivalent zu dem Ausdruck NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France')).
  - Wenn die ID einer IN- oder NOT IN-Operation NULL ist, ist der Wert der Operation unbekannt.
- identifier [NOT] LIKE pattern-value [ESCAPE escape-character] - Vergleichsoperator, bei dem die ID einen Zeichenfolgewert aufweist. 'pattern-value' ist ein Zeichenfolgeliteral, wobei _ für ein beliebiges einzelnes Zeichen und % für eine beliebige Zeichenfolge steht (einschließlich der leeren Sequenz). Alle anderen Zeichen stehen für sich. Das optionale Escapezeichen ist ein Zeichenfolgeliteral mit einem einzelnen Zeichen, das als Escapezeichen für die Sonderbedeutung von _ und % in pattern-value verwendet wird.
  - phone LIKE '12%3' ist wahr (true) für 123 sowie 12993 und falsch (false) für 1234.
  - word LIKE 'l_se' ist wahr (true) für "lose" und falsch (false) für "loose".
  - underscored LIKE '_%' ESCAPE '\' ist wahr (true) für "_foo" und falsch (false) für "bar".
  - phone NOT LIKE '12%3' ist falsch (false) für 123 sowie 12993 und wahr (true) für 1234.
  - Wenn die ID einer LIKE- oder NOT LIKE-Operation NULL ist, ist der Wert der Operation unbekannt.
- Der Vergleichsoperator 'identifier IS NULL' testet, ob ein Nullwert für ein Headerfeld oder eine fehlender Eigenschaftswert vorhanden ist.
  - prop_name IS NULL.
- Der Vergleichsoperator 'identifier IS NOT NULL' testet, ob ein Headerfeld mit einem Wert ungleich null oder ein Eigenschaftswert vorhanden ist.
  - prop_name IS NOT NULL.

Beispiel für einen Nachrichtenselektor

Der folgende Nachrichtenselektor wählt Nachrichten mit dem Nachrichtentyp 'car' (Auto), der Farbe 'blue' (Blau) und einer höheren Gewichtsangabe als 2.500 Pfund aus:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

Eigenschaften mit dem Wert NULL

Wie in der obigen Liste angemerkt, können Eigenschaftswerte NULL sein. Die Auswertung von Selektorausdrücken, die NULL-Werte enthalten, wird durch die SQL92-Semantik NULL definiert. Die folgende Liste enthält eine Kurzbeschreibung dieser Semantik:

- SQL behandelt einen NULL-Wert als unbekannt.
- Ein Vergleich oder eine Arithmetik mit einem unbekannten Wert ergibt immer einen unbekannten Wert.

- Der Operator IS NULL konvertiert einen unbekannten Wert in einen TRUE-Wert.
- Der Operator IS NOT NULL konvertiert einen unbekannten Wert in einen FALSE-Wert.

Spezielles Verhalten von JMSMessageID und JMSCorrelationID

Die IBM MQ-Klassen für JMS enthalten Optimierungen, wenn die Auswahl von Nachrichten aus einer Warteschlange entweder auf JMSMessageID oder JMSCorrelationID basiert.

Wenn eine Anwendung einen Selektor mit dem Format

JMSMessageID='ID:*Nachrichten-ID*'

angibt, wobei *Nachrichten-ID* eine Zeichenfolge ist, die eine IBM MQ-Standardnachrichten-ID enthält, dann verwenden die IBM MQ-Klassen für JMS die **MatchOption** MQMO_MATCH_MSG_ID, um die Nachricht mit der angegebenen Nachrichten-ID abzurufen.

Um beispielsweise eine Nachricht mit der Nachrichten-ID 414D51207061756C745639314C545320C57C1A5F25ECE602 aus einer Warteschlange abzurufen, sollte eine Anwendung folgenden Nachrichtenselektor verwenden:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

Entsprechend gilt: Wenn die Anwendung einen Selektor mit dem Format

JMSCorrelationID ='ID:*Korrelations-ID*'

angibt, wobei *Korrelations-ID* eine Zeichenfolge ist, die eine IBM MQ-Standardkorrelations-ID enthält, dann verwenden die IBM MQ-Klassen für JMS die **MatchOption** MQMO_MATCH_CORREL_ID, um die Nachricht mit der angegebenen Korrelations-ID aus der Warteschlange abzurufen.

Im folgenden Beispiel wird ein Nachrichtenselektor verwendet, um eine Nachricht mit der Korrelations-ID 414D51207061756C745639314C545320846E5B5F25B1CC02 abzurufen:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

Wenn ein Nachrichtenselektor entweder für die *Nachrichten-ID* oder die *Korrelations-ID* einen Wert aus lauter Nullen enthält, werden alle Nachrichten aus der Warteschlange abgerufen. Gibt eine Anwendung beispielsweise den Selektor

```
JMSMessageID='ID:00000000000000000000000000000000000000000000000000'
```

an, wird angenommen, dass jede Nachricht in der Warteschlange mit dem Selektor übereinstimmt, und an die Anwendung zurückgegeben.

Weitere Informationen zu den **MatchOptions** MQMO_MATCH_MSG_ID und MQMO_MATCH_CORREL_ID finden Sie im Abschnitt MatchOptions (MQLONG).

Einschränkungen

SQL unterstützt zwar den Vergleich und die Arithmetik mit festen Dezimalstellen, dies wird von den JMS-Nachrichtenselektoren jedoch nicht unterstützt. Daher sind exakte numerische Literale auf diejenigen ohne Dezimalstellen beschränkt. Dies ist auch der Grund, weshalb numerische Werte mit einer Dezimalstelle als alternative Darstellung für einen näherungsweise berechneten numerischen Wert verwendet werden können.

SQL-Kommentare werden nicht unterstützt.

Zuordnung von JMS-Nachrichten zu IBM MQ-Nachrichten
IBM MQ-Nachrichten setzen sich aus einem Nachrichtendeskriptor, einem optionalen MQRFH2-Header und einem Hauptteil zusammen. Der Inhalt einer JMS-Nachricht wird einer IBM MQ-Nachricht teilweise zugeordnet und teilweise in diese kopiert.

In diesem Abschnitt wird beschrieben, wie die im ersten Teil dieses Abschnitts beschriebene JMS-Nachrichtenstruktur einer IBM MQ-Nachricht zugeordnet wird. Diese Informationen sind für Programmierer

interessant, die Nachrichten zwischen JMS-Anwendungen und konventionellen IBM MQ-Anwendungen übertragen möchten. Darüber hinaus sind sie für Personen von Interesse, die Nachrichten, die zwischen zwei JMS-Anwendungen übertragen wurden, bearbeiten möchten. Dies kann beispielsweise in einer IBM Integration Bus-Implementierung der Fall sein.

Die Informationen in diesem Abschnitt gelten nicht, wenn eine Anwendung eine Echtzeitverbindung zu einem Broker nutzt. Wenn eine Anwendung eine Echtzeitverbindung verwendet, erfolgt die gesamte Kommunikation direkt über TCP/IP; es sind keine IBM MQ-Warteschlangen oder -Nachrichten beteiligt.

IBM MQ-Nachrichten setzen sich aus drei Bestandteilen zusammen:

- IBM MQ-Nachrichtendeskriptor (MQMD)
- IBM MQ-MQRFH2-Header
- Nachrichtenhauptteil

Der MQRFH2 ist optional und seine Einbeziehung in eine abgehende Nachricht wird durch das Flag TARGCLIENT in der JMS-Klasse 'Destination' bestimmt. Sie können dieses Flag mit dem IBM MQ-JMS-Verwaltungstool festlegen. Da der MQRFH2 JMS-spezifische Informationen enthält, müssen Sie ihn immer in die Nachricht einschließen, wenn der Absender weiß, dass das empfangende Ziel eine JMS-Anwendung ist. Normalerweise wird der MQRFH2 ausgeschlossen, wenn eine Nachricht direkt an eine JMS-fremde Anwendung gesendet wird. Der Grund hierfür ist, dass eine solche Anwendung keinen MQRFH2 in ihrer IBM MQ-Nachricht erwartet.

Wenn eine eingehende Nachricht keinen MQRFH2-Header hat, wird das entsprechende Flag des Queue- oder Topic-Objekts, das vom Headerfeld 'JMSReplyTo' der Nachricht abgeleitet wird, dieses Flag standardmäßig so gesetzt, dass eine an die Warteschlange oder das Thema gesendete Antwortnachricht ebenfalls keinen MQRFH2-Header hat. Sie können dieses Verhalten, bei dem ein MQRFH2-Header nur in eine Antwortnachricht eingeschlossen wird, wenn die ursprüngliche Nachricht ebenfalls über einen MQRFH2-Header verfügt, inaktivieren, indem Sie die Eigenschaft TARGCLIENTMATCHING der Verbindungsfactory auf NO setzen.

In Abbildung 10 auf Seite 156 ist dargestellt, wie die Struktur einer JMS-Nachricht in eine IBM MQ-Nachricht (und umgekehrt) transformiert wird:

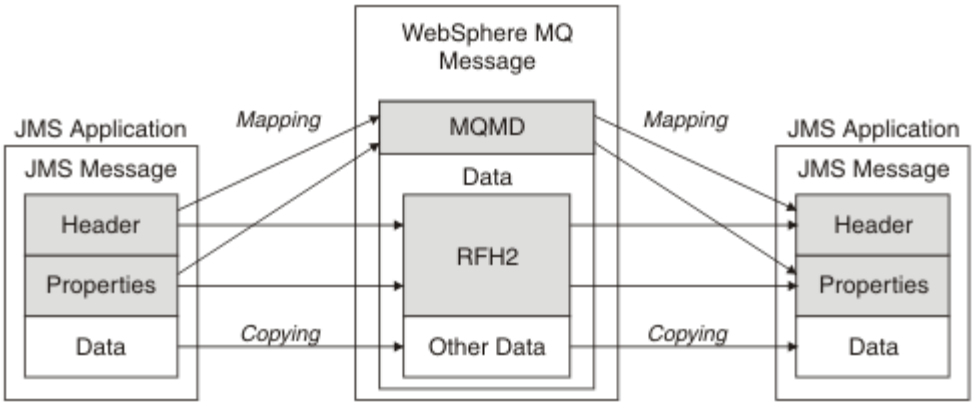


Abbildung 10. Verfahren für die Transformation von Nachrichten zwischen JMS und IBM MQ unter Verwendung des MQRFH2-Headers

Die Strukturen werden auf zwei Arten transformiert:

Zuordnen
Wenn der MQMD ein Feld enthält, das äquivalent zum JMS-Feld ist, wird das JMS-Feld dem MQMD-Feld zugeordnet. Zusätzliche MQMD-Felder sind als JMS-Eigenschaften verfügbar, da eine JMS-Anwendung diese Felder möglicherweise abrufen oder festlegen muss, wenn sie mit einer JMS-fremden Anwendung kommuniziert.

Kopieren
Wenn keine MQMD-Entsprechung vorhanden ist, wird ein JMS-Headerfeld oder eine JMS-Eigenschaft als Feld im MQRFH2 übergeben, wobei möglicherweise eine Transformation stattfindet.

MQRFH2-Header und JMS
In dieser Themensammlung wird der Header MQRFH Version 2 beschrieben, der JMS-spezifische Daten enthält, die dem Nachrichteninhalt zugeordnet sind. Bei MQRFH2 Version 2 handelt es sich um einen erweiterbaren Header, der auch zusätzliche Informationen enthalten kann, die nicht in direktem Zusammenhang mit JMS stehen. In diesem Abschnitt wird jedoch nur seine Verwendung durch JMS behandelt.Sie finden eine ausführliche Beschreibung im Abschnitt MQRFH2 - Rules and Formatting Header 2.

Der Header besteht aus zwei Teilen, und zwar aus einem festen Abschnitt und aus einem variablen Abschnitt.

Fester Teil
Der feste Abschnitt wird im *Standardmuster* des IBM MQ-Headers modelliert und besteht aus folgenden Feldern:

StrucId (MQCHAR4)
Struktur-ID.

Muss MQRFH_STRUC_ID sein (Wert: "RFH") (Anfangswert).

MQRFH_STRUC_ID_ARRAY (Wert: "R", "F", "H", " ") wird ebenfalls definiert.

Version (MQLONG)
Strukturversionsnummer.

Muss MQRFH_VERSION_2 sein (Wert: 2) (Anfangswert).

StrucLength (MQLONG)
Gesamtlänge des MQRFH2, einschließlich der NameValueData-Felder.

Der Wert, der in 'StrucLength' festgelegt ist, muss ein Vielfaches von 4 sein (die Daten in den NameValueData-Feldern können mit Leerzeichen aufgefüllt werden, um dies zu erreichen).

Encoding (MQLONG)
Datencodierung.

Codierung von numerischen Daten im Nachrichtenabschnitt, der dem MQRFH2 folgt (der nächste Header oder die Nachrichtendaten, die diesem Header folgen).

CodedCharSetId (MQLONG)
Die ID des codierten Zeichensatzes.

Darstellung von beliebigen Zeichendaten im Nachrichtenabschnitt, der dem MQRFH2 folgt (der nächste Header oder die Nachrichtendaten, die diesem Header folgen).

Format (MQCHAR8)
Formatname.

Formatname für den Nachrichtenabschnitt, der dem MQRFH2 folgt.

Flags (MQLONG)
Flags.

MQRFH_NO_FLAGS =0. Es sind keine Flags festgelegt.

NameValueCCSID (MQLONG)
Die ID des codierten Zeichensatzes (CCSID) für die NameValueData-Zeichenfolgen, die in diesem Header enthalten sind. Die NameValueData-Zeichenfolgen können in einem Zeichensatz codiert werden, der sich von den anderen Zeichenfolgen unterscheidet, die im Header enthalten sind (StrucID und Format).

Wenn es sich bei der NameValueCCSID um eine 2-Byte-Unicode-CCSID (1200, 13488 oder 17584) handelt, ist die Byteanordnung des Unicode dieselbe wie die Byteanordnung der numerischen Felder im MQRFH2. (Beispiel: Version, StrucLength und NameValueCCSID selbst.)

Tabelle 16. Mögliche Werte für das NameValueCCSID-Feld	
CCSID	**Bedeutung**
1200	UTF-16, die aktuellste unterstützte Unicode-Version
13488	UTF-16, Subset von Unicode Version 2.0
17584	UTF-16, Subset von Unicode Version 3.0 (einschließlich Euro-Symbol)
1208	UTF-8, die aktuellste unterstützte Unicode-Version

Variabler Abschnitt

Der variable Abschnitt folgt dem festen Abschnitt. Der variable Abschnitt enthält eine variable Anzahl an MQRFH2-Ordnern. Jeder Ordner enthält eine variable Anzahl an Elementen oder Eigenschaften. In Ordnern werden zusammengehörige Eigenschaften gruppiert. Die von JMS erstellten MQRFH2-Header können beliebige der folgenden Ordner enthalten:

Ordner `mcd`

`mcd` enthält Eigenschaften, die das Format der Nachricht beschreiben. Beispielsweise identifiziert die Eigenschaft `Msd` (Message Service Domäne = Nachrichtenservicedomäne) eine JMS-Nachricht als `JMSTextMessage`, `JMSBytesMessage`, `JMSStreamMessage`, `JMSMapMessage`, `JMSObjectMessage` oder 0.

Der Ordner `mcd` ist immer in einer JMS-Nachricht mit `MQRFH2` enthalten.

Sie ist immer in einer Nachricht enthalten, die eine von IBM Integration Bus gesendete `MQRFH2` enthält. Er beschreibt die Domäne, das Format, den Typ und die Nachrichtengruppe einer Nachricht.

Tabelle 17. mcd-Eigenschaftsname, -Synonym, -Datentyp und -Ordner			
Eigenschaftssynonym	**Eigenschaftsname**	**Datentyp**	**Ordner**
	`mcd.Msd`	`string`	`<mcd><Msd>`*`messageDomain`*`</Msd></mcd>`
	`mcd.Set`	`string`	`<mcd><Set>`*`messageDomain`*`</Set></mcd>`
	`mcd.Type`	`string`	`<mcd><Type>`*`messageDomain`*`</Type></mcd>`
	`mcd.Fmt`	`string`	`<mcd><Fmt>`*`messageDomain`*`</Fmt></mcd>`

Fügen Sie keine eigenen Eigenschaften zum Ordner `mcd` hinzu.

Ordner `jms`

`jms` enthält JMS-Headerfelder und JMSX-Eigenschaften, die nicht vollständig in `MQMD` ausgedrückt werden können. Der Ordner `jms` ist immer in einem JMS-`MQRFH2` vorhanden.

Ordner `usr`

`usr` enthält anwendungsdefinierte JMS-Eigenschaften, die der Nachricht zugeordnet sind. Der Ordner `usr` ist nur dann vorhanden, wenn eine Anwendung eine anwendungsdefinierte Eigenschaft festgelegt hat.

Ordner `mqext`

`mqext` enthält die folgenden Eigenschaftstypen:

- Eigenschaften, die ausschließlich von WebSphere Application Server verwendet werden.
- Eigenschaften, die sich auf eine verzögerte Nachrichtenübermittlung beziehen.

Der Ordner ist vorhanden, wenn die Anwendung mindestens eine der von IBM definierten Eigenschaften festgelegt oder eine Übermittlungsverzögerung verwendet hat.

Tabelle 18. mqext-Eigenschaftsname, -Synonym, -Datentyp und -Ordner

Eigenschaftssynonym	Eigenschaftsname	Datentyp	Ordner
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

Fügen Sie keine eigenen Eigenschaften zum Ordner `mqext` hinzu.

Ordner `mqps`

`mqps` enthält Eigenschaften, die nur von IBM MQ Publish/Subscribe verwendet werden. Der Ordner ist nur vorhanden, wenn die Anwendung mindestens eine der integrierten Publish/Subscribe-Eigenschaften festgelegt hat.

Tabelle 19. mqps-Eigenschaftsname, -Synonym, -Datentyp und -Ordner

Eigenschaftssynonym	Eigenschaftsname	Datentyp	Ordner
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData...*</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

Fügen Sie keine eigenen Eigenschaften zum Ordner `mqps` hinzu.

In Tabelle 20 auf Seite 160 ist eine vollständige Liste der Eigenschaftsnamen aufgeführt.

Tabelle 20. Von JMS verwendete MQRFH2-Ordner und Eigenschaften

Name des JMS-Felds	Java-Typ	MQRFH2-Ordnername	Eigenschaftsname	Typ/Werte
JMSDestination	Destination	jms	Dst	Zeichenfolge
JMSExpiration	lang	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Zeichenfolge	jms	Cid	Zeichenfolge
JMSReplyTo	Destination	jms	Rto	Zeichenfolge
JMSTimestamp	lang	jms	Tms	i8
JMSType	Zeichenfolge	mcd	Type, Set, Fmt	Zeichenfolge
JMSXGroupID	Zeichenfolge	jms	Gid	Zeichenfolge
JMSXGroupSeq	int	jms	Seq	i4
xxx (benutzerdefiniert)	Alle	usr	xxx	any
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValueLength (MQLONG)
Länge der Zeichenfolge für NameValueData, die unmittelbar auf dieses Längenfeld folgt, in Byte (enthält nicht die eigene Länge).

NameValueData (MQCHARn)
Eine einzelne Zeichenfolge, deren Länge in Byte durch das führende NameValueLength-Feld angegeben wird. Sie enthält einen Ordner mit einer Folge von Eigenschaften. Jede Eigenschaft ist ein Name/Typ/Wert-Triplet, das wie folgt in einem XML-Element enthalten ist, dessen Name der Ordnername ist:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

Auf den abschließenden Tag `</foldername>` können Leerzeichen als Füllzeichen folgen. Jedes Triplet wird mithilfe einer XML-ähnlichen Syntax wie folgt codiert:

```
<name dt='datatype'>value</name>
```

Das Element `dt='datatype'` ist optional und wird für viele Eigenschaften weggelassen, da der Datentyp vordefiniert ist. Wenn sie enthalten ist, muss mindestens ein Leerzeichen vor dem Tag `dt=` stehen.

`name`
Der Name der Eigenschaft; siehe Tabelle 20 auf Seite 160.

`datatype`
Muss nach dem Umsetzen einem der in Tabelle 21 auf Seite 161 aufgeführten Datentypen entsprechen.

value
Eine Zeichenfolgedarstellung des zu übertragenden Werts, wobei die Definitionen in Tabelle 21 auf Seite 161 verwendet werden.

Ein Nullwert wird mit folgender Syntax codiert:

```
<name dt='datatype' xsi:nil='true'></name>
```

Verwenden Sie nicht `xsi:nil='false'`.

Tabelle 21. Eigenschaftsdatentypen

Datentyp	**Definition**
Zeichenfolge	Beliebige Zeichenfolge mit Ausnahme von < und &
boolean	Das Zeichen `0` oder 1 (`0` = false, 1 = true)
bin.hex	Durch Hexadezimalziffern dargestellte Oktette
i1	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -128 bis einschließlich 127 liegen.
i2	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -32768 bis einschließlich 32767 liegen.
i4	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -2147483648 bis einschließlich 2147483647 liegen.
i8	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -9223372036854775808 bis einschließlich 9223372036854775807 liegen.
int	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im selben Bereich wie i8 liegen. Dies kann anstelle eines i*-Typen verwendet werden, wenn der Sender der Eigenschaft keine spezielle Genauigkeit zuordnen will.
r4	Gleitkommazahl, Größe < = 3.40282347E+38, > = 1.175E-37 ausgedrückt mit Ziffern `0..9`, optionales Vorzeichen, optionale Nachkommastellen, optionaler Exponent
r8	Gleitkommazahl, Größe < = 1.7976931348623E+308, > = 2.225E-307 ausgedrückt mit Ziffern `0..9`, optionales Vorzeichen, optionale Nachkommastellen, optionaler Exponent

Ein Zeichenfolgewert kann Leerzeichen enthalten. Sie müssen die folgenden Escapezeichenfolgen in einem Zeichenfolgewert verwenden:

- `&` für das Zeichen &
- `<` für das Zeichen <

Sie können die folgenden Escapezeichenfolgen verwenden, sie sind jedoch nicht erforderlich:

- `>` für das Zeichen >
- `&apos;` für das Zeichen '
- `&quot;` für das Zeichen "

JMS-Felder und -Eigenschaften mit entsprechenden MQMD-Feldern
In diesen Tabellen werden die MQMD-Felder aufgelistet, die äquivalent zu JMS-Headerfeldern, JMS-Eigenschaften und JMS-Provider-spezifischen Eigenschaften sind.

In Tabelle 22 auf Seite 162 werden die JMS-Headerfelder aufgelistet, während in Tabelle 23 auf Seite 162 die JMS-Eigenschaften aufgeführt sind, die den MQMD-Feldern direkt zugeordnet werden. In Tabelle 24 auf Seite 162 werden die providerspezifischen Eigenschaften und die MQMD-Felder aufgelistet, denen sie zugeordnet werden.

Tabelle 22. Zuordnung der JMS-Headerfelder zu MQMD-Feldern

JMS-Headerfeld	**Java-Typ**	**MQMD-Feld**	**C-Typ**
JMSDeliveryMode	int	Permanenz	MQLONG
JMSExpiration	lang	Verfall	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	Zeichenfolge	MsgID	MQBYTE24
JMSTimestamp	lang	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Zeichenfolge	CorrelId	MQBYTE24

Tabelle 23. Zuordnung der JMS-Eigenschaften zu MQMD-Feldern

JMSEigenschaft	**Java-Typ**	**MQMD-Feld**	**C-Typ**
JMSXUserID	Zeichenfolge	UserIdentifier	MQCHAR12
JMSXAppID	Zeichenfolge	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	Zeichenfolge	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Tabelle 24. Zuordnung der JMS-Provider-spezifischen Eigenschaften zu MQMD-Feldern

JMS-Provider-spezifische Eigenschaft	**Java-Typ**	**MQMD-Feld**	**C-Typ**
JMS_IBM_Report_Exception	int	Bericht	MQLONG
JMS_IBM_Report_Expiration	int	Bericht	MQLONG
JMS_IBM_Report_COA	int	Bericht	MQLONG
JMS_IBM_Report_COD	int	Bericht	MQLONG
JMS_IBM_Report_PAN	int	Bericht	MQLONG
JMS_IBM_Report_NAN	int	Bericht	MQLONG

Tabelle 24. Zuordnung der JMS-Provider-spezifischen Eigenschaften zu MQMD-Feldern (Forts.)			
JMS-Provider-spezifische Eigenschaft	**Java-Typ**	**MQMD-Feld**	**C-Typ**
JMS_IBM_Report_Pass_Msg_ID	int	Bericht	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Bericht	MQLONG
JMS_IBM_Report_Discard_Msg	int	Bericht	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	Zeichenfolge	Format „1" auf Seite 163	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	Zeichenfolge	CodedCharacterSetId „2" auf Seite 163	MQLONG
JMS_IBM_PutDate	Zeichenfolge	PutDate	MQCHAR8
JMS_IBM_PutTime	Zeichenfolge	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	boolean	MsgFlags	MQLONG

Anmerkung:

1. JMS_IBM_Format ist das Format des Nachrichtenhauptteils. Es kann durch die anwendungsinterne Einstellung der Nachrichteneigenschaft 'JMS_IBM_Format' (Begrenzung auf 8 Zeichen) definiert werden oder standardmäßig das IBM MQ-Format des Nachrichtenhauptteils entsprechend dem JMS-Nachrichtentyp übernehmen. 'JMS_IBM_Format' wird dem MQMD-Feld 'Format' nur zugeordnet, wenn die Nachricht keine RFH- oder RFH2-Abschnitte enthält. In einer Standardnachricht wird es dem Feld 'Format' des RFH2 direkt vor dem Nachrichtenhauptteil zugeordnet.
2. Der Wert der Eigenschaft 'JMS_IBM_Character_Set' ist ein Zeichenfolgewert (String) mit der Java-Zeichensatzentsprechung für den numerischen CodedCharacterSetId-Wert. Das MQMD-Feld 'CodedCharacterSetId' ist die numerische Entsprechung des in der Eigenschaft 'JMS_IBM_Character_Set' als Zeichenfolge angegebenen Java-Zeichensatzes.

Zuordnung von JMS-Feldern zu IBM MQ-Feldern (abgehende Nachrichten)
In diesen Tabellen ist dargestellt, wie JMS-Headerfelder und JMS-Eigenschaftsfelder beim Senden (send()) oder Veröffentlichen (publish()) zu MQMD- und MQRFH2-Feldern zugeordnet werden.

Tabelle 25 auf Seite 164 zeigt, wie die JMS-Headerfelder beim Senden (send()) oder Veröffentlichen (publish()) zu MQMD-/MQRFH2-Feldern zugeordnet werden. Tabelle 26 auf Seite 164 zeigt, wie JMS-Eigenschaften beim Senden (send()) oder Veröffentlichen (publish()) zu MQMD-/MQRFH2-Feldern zugeordnet werden. Tabelle 27 auf Seite 165 zeigt, wie JMS-Provider-spezifische Eigenschaften beim Senden (send()) oder Veröffentlichen (publish()) zu MQMD-Feldern zugeordnet werden.

Bei Feldern mit dem Hinweis 'Festgelegt durch Nachrichtenobjekt' entspricht der übertragene Wert dem Wert, der direkt vor der send()- oder publish()-Operation in der JMS-Nachricht enthalten ist. Der Wert in der JMS-Nachricht wird durch die Operation nicht geändert.

Bei Feldern mit dem Hinweis 'Festgelegt durch Sendemethode' wird bei der Ausführung der send()- oder publish()-Operation ein Wert zugewiesen (Werte in der JMS-Nachricht werden ignoriert). Der Wert in der JMS-Nachricht wird mit dem verwendeten Wert aktualisiert.

Felder mit dem Hinweis 'Nur Empfang' werden nicht übertragen und bleiben bei send()- oder publish()-Operationen in der Nachricht unverändert.

Tabelle 25. Zuordnung der Felder abgehender Nachrichten

Name des JMS-Headerfelds	MQMD-Feld für Übertragung	Header	Festgelegt durch
JMSDestination		MQRFH2	Sendemethode
JMSDeliveryMode	Permanenz	MQRFH2	Sendemethode
JMSExpiration	Verfall	MQRFH2	Sendemethode
JMSPriority	Priority	MQRFH2	Sendemethode
JMSMessageID	MsgID		Sendemethode
JMSTimestamp	PutDate/PutTime		Sendemethode
JMSCorrelationID	CorrelId	MQRFH2	Nachrichtenobjekt
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Nachrichtenobjekt
JMSType		MQRFH2	Nachrichtenobjekt
JMSRedelivered			Nur Empfang

Anmerkung:

1. Das MQMD-Feld 'CodedCharacterSetId' ist die numerische Entsprechung des in der Eigenschaft 'JMS_IBM_Character_Set' als Zeichenfolge angegebenen Java-Zeichensatzes.

Tabelle 26. Zuordnung der JMS-Eigenschaften abgehender Nachrichten

JMS-Eigenschaftsname	MQMD-Feld für Übertragung	Header	Festgelegt durch
JMSXUserID	UserIdentifier		Sendemethode
JMSXAppID	PutApplName		Sendemethode
JMSXDeliveryCount			Nur Empfang
JMSXGroupID	GroupId	MQRFH2	Nachrichtenobjekt
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Nachrichtenobjekt

Anmerkung:

Diese Eigenschaften sind in der JMS-Spezifikation als schreibgeschützt definiert und sind vom JMS-Provider festgelegt (in einigen Fällen optional).

In IBM MQ classes for JMS können zwei dieser Eigenschaften von der Anwendung überschrieben werden. Stellen Sie dazu sicher, dass das Ziel entsprechend konfiguriert wurde, indem Sie die folgenden Eigenschaften festlegen:

1. Setzen Sie die Eigenschaft `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` auf `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`.
2. Setzen Sie die Eigenschaft `WMQConstants.WMQ_MQMD_WRITE_ENABLED` auf `true`.

Die folgenden Eigenschaften können von der Anwendung überschrieben werden:

JMSXAppID
Diese Eigenschaft kann überschrieben werden, indem die Eigenschaft `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` in der Nachricht festgelegt wird. Der Wert muss eine Java -Zeichenfolge sein.

JMSXGroupID
Diese Eigenschaft kann überschrieben werden, indem die Eigenschaft `WMQConstants.JMS_IBM_MQMD_GROUPID` in der Nachricht festgelegt wird - beim Wert sollte es sich um einen Byte-Array handeln.

Tabelle 27. Zuordnung der JMS-Provider-spezifischen Eigenschaften abgehender Nachrichten

Name der JMS-Provider-spezifischen Eigenschaft	MQMD-Feld für Übertragung	Header	Festgelegt durch
JMS_IBM_Report_Exception	Bericht		Nachrichtenobjekt
JMS_IBM_Report_Expiration	Bericht		Nachrichtenobjekt
JMS_IBM_Report_COA/COD	Bericht		Nachrichtenobjekt
JMS_IBM_Report_NAN/PAN	Bericht		Nachrichtenobjekt
JMS_IBM_Report_Pass_Msg_ID	Bericht		Nachrichtenobjekt
JMS_IBM_Report_Pass_Correl_ID	Bericht		Nachrichtenobjekt
JMS_IBM_Report_Discard_Msg	Bericht		Nachrichtenobjekt
JMS_IBM_MsgType	MsgType		Nachrichtenobjekt
JMS_IBM_Feedback	Feedback		Nachrichtenobjekt
JMS_IBM_Format	Format		Nachrichtenobjekt
JMS_IBM_PutApplType	PutApplType		Sendemethode
JMS_IBM_Encoding	Encoding		Nachrichtenobjekt
JMS_IBM_Character_Set	CodedCharacterSetId		Nachrichtenobjekt
JMS_IBM_PutDate	PutDate		Sendemethode
JMS_IBM_PutTime	PutTime		Sendemethode
JMS_IBM_Last_Msg_In_Group	MsgFlags		Nachrichtenobjekt

JMS-Headerfelder bei send() oder publish() zuordnen
Diese Hinweise beziehen sich auf die Zuordnung von JMS-Feldern bei send() oder publish().

JMSDestination zu MQRFH2
Dies wird als Zeichenfolge gespeichert, die die hervorstechenden Merkmale des Zielobjekts serialisiert, damit ein empfangender JMS ein funktional entsprechendes Zielobjekt wiederherstellen kann. Das Feld MQRFH2 wird als URI codiert (der Abschnitt „Uniform Resource Identifiers (URIs)" auf Seite 232 enthält Details zur URI-Schreibweise).

JMSReplyTo zu MQMD.ReplyToQ, ReplyToQMgr, MQRFH2
Der Warteschlangenname wird in das Feld MQMD.ReplyToQ kopiert, während der Name des Warteschlangenmanagers in die ReplyToQMgr-Felder kopiert wird. Ergänzende Informationen zum Ziel (weitere hilfreiche Details, die im Zielobjekt gespeichert sind) werden in das MQRFH2-Feld kopiert. Das MQRFH2-Feld wird als URI codiert (der Abschnitt „Uniform Resource Identifiers (URIs)" auf Seite 232 enthält Details zur URI-Schreibweise).

JMSDeliveryMode zu MQMD.Persistence
Der Wert von JMSDeliveryMode wird durch die Methode send() oder publish() oder durch MessageProducer festgelegt, es sei denn, er wird vom Destination-Objekt überschrieben. Der JMSDeliveryMode-Wert wird dem Feld MQMD.Persistence wie folgt zugeordnet:

- Der JMS-Wert PERSISTENT entspricht MQPER_PERSISTENT.
- Der JMS-Wert NON_PERSISTENT entspricht MQPER_NOT_PERSISTENT.

Wenn die MQQueue-Persistenzeigenschaft nicht auf WMQConstants.WMQ_PER_QDEF gesetzt ist, wird der Wert des Zustellungsmodus ebenfalls im MQRFH2 codiert.

JMSExpiration zu/aus MQMD.Expiry, MQRFH2
In JMSExpiration wird die Ablaufzeit (die Summe aus der aktuellen Zeit und der Lebensdauer) gespeichert, während der MQMD die Lebensdauer speichert. Außerdem wird JMSExpiration in Millisekunden angegeben, MQMD.Expiry dagegen in Zehntelsekunden.

- Wenn die send()-Methode eine unbegrenzte Lebensdauer festlegt, wird MQMD.Expiry auf MQEI_UNLIMITED gesetzt und im MQRFH2 wird kein Wert für JMSExpiration codiert.
- Wenn die send()-Methode eine Lebensdauer festlegt, die unter 214748364,7 Sekunden (ca. 7 Jahre) liegt, wird die Lebensdauer in MQMD.Expiry gespeichert und die Ablaufzeit (in Millisekunden) wird als i8-Wert im MQRFH2 codiert.
- Wenn die send()-Methode eine Lebensdauer festlegt, die mehr als 214748364,7 Sekunden beträgt, wird MQMD.Expiry auf MQEI_UNLIMITED gesetzt. Die tatsächliche Ablaufzeit in Millisekunden wird als i8-Wert im MQRFH2 codiert.

JMSPriority zu MQMD.Priority
Der Wert von JMSPriority (0-9) wird direkt dem MQMD-Prioritätswert (0-9) zugeordnet. Wenn JMSPriority auf einen vom Standard abweichenden Wert gesetzt wird, wird die Prioritätsstufe ebenfalls im MQRFH2 codiert.

JMSMessageID aus MQMD.MessageID
Allen Nachrichten, die von JMS gesendet werden, werden von IBM MQeindeutige Nachrichten-IDs zugeordnet. Der zugewiesene Wert wird im Feld MQMD.MessageId nach dem MQPUT-Aufruf zurückgegeben und wieder an die Anwendung im Feld JMSMessageID übergeben. Die Nachrichten-ID (messageID) in IBM MQ ist ein Binärwert mit 24 Bytes, während die JMSMessageID eine Zeichenfolge ist. Der Wert von JMSMessageID setzt sich aus dem messageId-Binärwert, der in eine Folge mit 48 Hexadezimalzeichen konvertiert wird, und einem Präfix mit den Zeichen 'ID:' zusammen. JMS stellt einen Hinweis zur Verfügung, der festgelegt werden kann, um die Erstellung von Nachrichten-IDs zu inaktivieren. Dieser Hinweis wird ignoriert und eine eindeutige ID wird in allen Fällen zugewiesen. Jeder Wert, der vor einer send()-Operation im Feld JMSMessageID festgelegt ist, wird überschrieben.

Wenn Sie die Möglichkeit benötigen, MQMD.MessageID: Sie können dies mit einer der im Abschnitt „Nachrichtendeskriptor über eine Anwendung der IBM MQ classes for JMS lesen und schreiben" auf Seite 256beschriebenen IBM MQ JMS -Erweiterungen tun.

JMSTimestamp zu MQRFH2
Beim Senden wird das Feld JMSTimestamp auf einen Wert gesetzt, der von der Systemzeit der JVM abhängt. Dieser Wert wird im MQRFH2 festgelegt. Jeder Wert, der vor einer send()-Operation im Feld JMSTimestamp festgelegt ist, wird überschrieben. Siehe hierzu auch die Eigenschaften JMS_IBM_PutDate und JMS_IBM_PutTime.

JMSType zu MQRFH2
Diese Zeichenfolge wird im MQRFH2-Feld 'mcd.Type' festgelegt. Falls sie im URI-Format angegeben ist, kann sie sich auch auf die Felder 'mcd.Set' und 'mcd.Fmt' auswirken.

JMSCorrelationID zu MQMD.CorrelId, MQRFH2
JMSCorrelationID kann einen der folgenden Werte enthalten:

Eine providerspezifische Nachrichten-ID
Hierbei handelt es sich um eine Nachrichten-ID aus einer zuvor gesendeten oder empfangenen Nachricht. Daher sollte sie eine aus 48 Hexadezimalziffern Zeichenfolge in Kleinschreibung sein, die das Präfix `ID:` enthält. Das Präfix wird entfernt, die verbleibenden Zeichen werden in Binärdaten konvertiert und anschließend im Feld 'MQMD.CorrelId' festgelegt.

Einen providereigenen byte[]-Wert
Der Wert wird in das Feld 'MQMD.CorrelId' kopiert - bei Bedarf wird er mit Nullen aufgefüllt oder abgeschnitten, damit eine Länge von 24 Bytes erreicht wird. Im MQRFH2 wird kein CorrelId-Wert codiert.

Eine anwendungsspezifische Zeichenfolge
Der Wert wird in den MQRFH2 kopiert. Die ersten 24 Bytes der Zeichenfolge im UTF-8-Format werden in die MQMD.CorrelID geschrieben.

JMS-Eigenschaftsfelder zuordnen
Diese Hinweise beziehen sich auf die Zuordnung von JMS-Eigenschaftsfeldern in IBM MQ-Nachrichten.

JMSXUserID aus dem MQMD-Feld UserIdentifier
JMSXUserID wird bei der Rückkehr von einem Sendeaufruf festgelegt.

JMSXAppID aus dem MQMD-Feld PutApplName
JSMXAppID wird bei der Rückkehr von einem Sendeaufruf festgelegt.

JMSXGroupID zu MQRFH2 (Punkt-zu-Punkt)
Bei Punkt-zu-Punkt-Nachrichten wird der Wert von JMSXGroupID in das MQMD-Feld GroupID kopiert. Wenn JMSXGroupID mit dem Präfix 'ID:' beginnt, wird der Wert in einen Binärwert konvertiert. Andernfalls wird er als UTF-8-Zeichenfolge codiert. Falls erforderlich, wird der Wert bis auf eine Länge von 24 Bytes aufgefüllt oder abgeschnitten. Das Flag MQMF_MSG_IN_GROUP wird festgelegt.

JMSXGroupID zu MQRFH2 (Publish/Subscribe)
Bei Publish/Subscribe-Nachrichten wird der Wert von JMSXGroupID als Zeichenfolge in den MQRFH2 kopiert.

JMSXGroupSeq MQMD MsgSeqNumber (Punkt-zu-Punkt)
Bei Punkt-zu-Punkt-Nachrichten wird der Wert von JMSXGroupSeq in das MQMD-Feld MsgSeqNumber kopiert. Das Flag MQMF_MSG_IN_GROUP wird festgelegt.

JMSXGroupSeq MQMD MsgSeqNumber (Publish/Subscribe)
Bei Publish/Subscribe-Nachrichten wird der Wert von JMSXGroupSeq als i4 in den MQRFH2 kopiert.

JMS-Provider-spezifische Felder zuordnen
Die folgenden Hinweise beziehen sich auf die Zuordnung von JMS-Provider-spezifischen Feldern zu IBM MQ-Nachrichten.

JMS_IBM_Report_*XXX* zu MQMD-Bericht
Eine JMS-Anwendung kann die MQMD-Berichtsoptionen mithilfe der folgenden JMS_IBM_Report_*XXX*-Eigenschaften festlegen. Der einzelne MQMD-Wert wird mehreren JMS_IBM_Report_*XXX*-Eigenschaften zugeordnet.

Die Konstanten JMS_IBM_Report_*XXX* befinden sich in `com.ibm.msg.client.jakarta.wmq.WMQConstants` oder `com.ibm.msg.client.wmq.WMQConstants`.

JMS_IBM_Report_Exception

MQRO_EXCEPTION oder
MQRO_EXCEPTION_WITH_DATA oder
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION oder
MQRO_EXPIRATION_WITH_DATA oder
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA oder
MQRO_COA_WITH_DATA oder
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD oder
MQRO_COD_WITH_DATA oder
MQRO_COD_WITH_FULL_DATA

JMS_IBM_Report_PAN
MQRO_PAN

JMS_IBM_Report_NAN
MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID
MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

Die MQRO-Werte befinden sich in `com.ibm.mq.constants.CMQC`.

JMS_IBM_MsgType zu MQMD MsgType
Der Wert wird direkt dem MQMD-Feld 'MsgType' zugeordnet. Wenn die Anwendung keinen expliziten Wert für JMS_IBM_MsgType festgelegt hat, wird ein Standardwert verwendet. Dieser Wert wird wie folgt bestimmt:

- Wenn JMSReplyTo auf ein IBM MQ-Warteschlangenziel gesetzt ist, wird MSGType auf den Wert MQMT_REQUEST gesetzt.
- Wenn JMSReplyTo nicht festgelegt oder auf einen anderen Wert als ein IBM MQ-Warteschlangenziel gesetzt ist, wird MsgType auf den Wert MQMT_DATAGRAM gesetzt.

JMS_IBM_Feedback zu MQMD Feedback
Der Wert wird direkt dem MQMD-Feld 'Feedback' zugeordnet.

JMS_IBM_Format zu MQMD Format
Der Wert wird direkt dem MQMD-Feld 'Format' zugeordnet.

JMS_IBM_Encoding zu MQMD Encoding
Wenn diese Eigenschaft festgelegt ist, überschreibt sie die numerische Codierung der Zielwarteschlange oder des Zielthemas (Queue oder Topic).

JMS_IBM_Character_Set zu MQMD CodedCharacterSetId
Wenn diese Eigenschaft festgelegt ist, überschreibt sie den codierten Zeichensatz der Zielwarteschlange oder des Zielthemas (Queue oder Topic).

JMS_IBM_PutDate aus dem MQMD-Feld 'PutDate'
Der Wert dieser Eigenschaft wird beim Senden direkt auf Basis des PutDate-Felds im MQMD festgelegt. Jeder Wert, der vor einem Sendeaufruf in der Eigenschaft JMSTimestamp festgelegt ist, wird überschrieben. Dieses Feld enthält eine Zeichenfolge mit acht Zeichen im IBM MQ-Datumsformat JJJJMMTT. Diese Eigenschaft kann zusammen mit der Eigenschaft JMS_IBM_PutTime verwendet werden, um die Uhrzeit zu bestimmen, zu der die Nachricht laut Warteschlangenmanager eingereiht wurde.

JMS_IBM_PutTime aus dem MQMD-Feld 'PutTime'
Der Wert dieser Eigenschaft wird beim Senden direkt auf Basis des PutTime-Felds im MQMD festgelegt. Jeder Wert, der vor einem Sendeaufruf in der Eigenschaft JMS_IBM_PutTime festgelegt ist, wird überschrieben. Dieses Feld enthält eine Zeichenfolge mit acht Zeichen im IBM MQ-Zeitformat HHMMSSTH. Diese Eigenschaft kann zusammen mit der Eigenschaft JMS_IBM_PutDate verwendet werden, um die Uhrzeit zu bestimmen, zu der die Nachricht laut Warteschlangenmanager eingereiht wurde.

JMS_IBM_Last_Msg_In_Group zum MQMD-Feld 'MsgFlags'
Beim Punkt-zu-Punkt-Messaging wird dieser boolesche Wert dem Flag MQMF_LAST_MSG_IN_GROUP im MQMD-Feld 'MsgFlags' zugeordnet. Er wird normalerweise zusammen mit den Eigenschaften JMSXGroupID und JMSXGroupSeq verwendet, um einer traditionellen IBM MQ-Anwendung anzuzeigen, dass diese Nachricht die letzte Nachricht in einer Gruppe ist. Beim Publish/Subscribe-Messaging wird diese Eigenschaft ignoriert.

Zuordnung von IBM MQ-Feldern zu JMS-Feldern (eingehende Nachrichten)
In diesen Tabellen ist dargestellt, wie JMS-Headerfelder und JMS-Eigenschaftsfelder beim Abrufen (get()) oder Empfangen (receive()) zu MQMD- und MQRFH2-Feldern zugeordnet werden.

Tabelle 28 auf Seite 169 zeigt, wie JMS-Headerfelder beim Abrufen (get()) oder Empfangen (receive()) zu MQMD-/MQRFH2-Feldern zugeordnet werden. Tabelle 29 auf Seite 170 zeigt, wie JMS-Eigenschaftsfelder beim Abrufen (get()) oder Empfangen (receive()) zu MQMD-/MQRFH2-Feldern zugeordnet werden. In Tabelle 30 auf Seite 170 ist die Zuordnung von JMS-Provider-spezifischen Eigenschaften dargestellt.

Tabelle 28. Zuordnung der JMS-Headerfelder eingehender Nachrichten

Name des JMS-Headerfelds	**MQMD-Feld für Abruf**	**MQRFH2-Feld für Abruf**
JMSDestination		jms.Dst oder mqps.Top „1" auf Seite 170
JMSDeliveryMode	Persistence „2" auf Seite 170	jms.Dlv „2" auf Seite 170
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate „2" auf Seite 170 PutTime „2" auf Seite 170	jms.Tms „2" auf Seite 170
JMSCorrelationID	CorrelId „2" auf Seite 170	jms.Cid „2" auf Seite 170
JMSReplyTo	ReplyToQ „2" auf Seite 170 ReplyToQMgr „2" auf Seite 170	jms.Rto „2" auf Seite 170
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Anmerkung:

1. Wenn sowohl jms.Dst als auch mqps.Top festgelegt sind, wird der Wert in jms.Dst verwendet.
2. Für Eigenschaften, die vom MQRFH2 oder dem MQMD abgerufene Werte haben können, wird - wenn beide verfügbar sind - die Einstellung im MQRFH2 verwendet.
3. Der Wert der Eigenschaft 'JMS_IBM_Character_Set' ist ein Zeichenfolgewert (String) mit der Java-Zeichensatzentsprechung für den numerischen CodedCharacterSetId-Wert.

Tabelle 29. Eigenschaftszuordnung bei eingehenden Nachrichten

JMS-Eigenschaftsname	**MQMD-Feld für Abruf**	**MQRFH2-Feld für Abruf**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId „1" auf Seite 170	jms.Gid „1" auf Seite 170
JMSXGroupSeq	MsgSeqNumber „1" auf Seite 170	jms.Seq „1" auf Seite 170

Anmerkung:

1. Für Eigenschaften, die vom MQRFH2 oder dem MQMD abgerufene Werte haben können, wird - wenn beide verfügbar sind - die Einstellung im MQRFH2 verwendet. Die Eigenschaften werden nur dann auf Basis von MQMD-Werten festgelegt, wenn die Nachrichtenflags MQMF_MSG_IN_GROUP oder MQMF_LAST_MSG_IN_GROUP festgelegt sind.

Tabelle 30. Zuordnung der JMS-Provider-spezifischen Eigenschaften eingehender Nachrichten

JMS-Eigenschaftsname	**MQMD-Feld für Abruf**	**MQRFH2-Feld für Abruf**
JMS_IBM_Report_Exception	Bericht	
JMS_IBM_Report_Expiration	Bericht	
JMS_IBM_Report_COA	Bericht	
JMS_IBM_Report_COD	Bericht	
JMS_IBM_Report_PAN	Bericht	
JMS_IBM_Report_NAN	Bericht	
JMS_IBM_Report_Pass_Msg_ID	Bericht	
JMS_IBM_Report_Pass_Correl_ID	Bericht	
JMS_IBM_Report_Discard_Msg	Bericht	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding „1" auf Seite 171	Encoding	
JMS_IBM_Character_Set „1" auf Seite 171	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. Wird nur festgelegt, wenn die eingehende Nachricht eine Bytenachricht ist.

Nachrichtenaustausch zwischen einer JMS-Anwendung und einer konventionellen IBM MQ-Anwendung
In diesem Abschnitt wird beschrieben, was geschieht, wenn eine JMS-Anwendung Nachrichten mit einer konventionellen IBM MQ-Anwendung austauscht, die den MQRFH2-Header nicht verarbeiten kann.

In Abbildung 11 auf Seite 171 ist die Zuordnung dargestellt.

Der Administrator gibt an, dass die JMS-Anwendung mit einer konventionellen IBM MQ-Anwendung kommuniziert, indem er die Eigenschaft TARGCLIENT des Ziels auf *MQ* setzt. Dies gibt an, dass kein MQRFH2-Header erstellt werden soll. Falls diese Einstellung nicht vorgenommen wird, muss die empfangende Anwendung in der Lage sein, den MQRFH2-Header zu verarbeiten.

Die Zuordnung aus JMS zu einem MQMD, der eine konventionelle IBM MQ-Anwendung als Ziel hat, ist mit der Zuordnung aus JMS zu einem MQMD identisch, der eine JMS-Anwendung als Ziel hat. Wenn IBM MQ classes for JMS eine IBM MQ-Nachricht empfangen, deren MQMD-Feld *Format* auf einen anderen Wert als MQFMT_RFH2 gesetzt ist, werden die Daten aus einer JMS-fremden Anwendung empfangen. Beim Format MQFMT_STRING wird die Nachricht als JMS-Textnachricht empfangen. Andernfalls wird sie als JMS-Bytenachricht empfangen. Da es keinen MQRFH2 gibt, können nur die im MQMD übertragenen JMS-Eigenschaften wiederhergestellt werden.

Wenn IBM MQ classes for JMS eine Nachricht ohne MQRFH2-Header empfangen, wird die Eigenschaft TARGCLIENT des Queue- oder Topic-Objekts, das vom JMSReplyTo-Headerfeld der Nachricht abgeleitet wird, standardmäßig auf 'MQ' gesetzt. Dies bedeutet, dass eine Antwortnachricht, die an die Warteschlange oder an das Thema gesendet wird, ebenfalls keinen MQRFH2-Header enthält. Sie können dieses Verhalten, bei dem ein MQRFH2-Header nur in eine Antwortnachricht eingeschlossen wird, wenn die ursprüngliche Nachricht ebenfalls über einen MQRFH2-Header verfügt, inaktivieren, indem Sie die Eigenschaft TARGCLIENTMATCHING der Verbindungsfactory auf NO setzen.

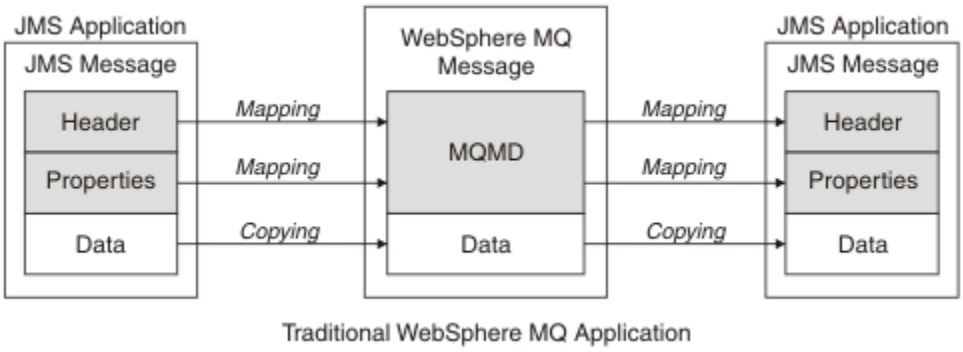


Abbildung 11. JMS-Nachrichtentransformation in IBM MQ-Nachrichten ohne MQRFH2-Header

JMS-Nachrichtenhauptteil
Dieser Abschnitt enthält Informationen zur Codierung des Nachrichtenhauptteils selbst. Die Codierung hängt vom Typ der JMS-Nachricht ab.

ObjectMessage
ObjectMessage ist ein Objekt, das auf die übliche Weise von der Java Runtime serialisiert wird.

TextMessage
TextMessage ist eine codierte Zeichenfolge. Bei einer abgehenden Nachricht wird die Zeichenfolge in dem Zeichensatz codiert, der vom Zielobjekt vorgegeben wird. Standardmäßig wird die UTF-8-Codierung verwendet (die UTF-8-Codierung beginnt beim ersten Zeichen der Nachricht; es gibt kein Längenfeld am Anfang). Sie können jedoch auch beliebige andere Zeichensätze angeben, die von den IBM MQ classes for JMS unterstützt werden. Diese Zeichensätze werden hauptsächlich beim Senden einer Nachricht an eine JMS-fremde Anwendung verwendet.

Falls es sich bei dem Zeichensatz um einen Doppelbytesatz (einschließlich UTF-16) handelt, bestimmt die für Ganzzahlen festgelegte Codierungsspezifikation des Zielobjekts die Reihenfolge der Bytes.

Eine eingehende Nachricht wird unter Verwendung des Zeichensatzes und der Codierung interpretiert, die in der Nachricht selbst angegeben sind. Diese Spezifikationen befinden sich im letzten IBM MQ-Header (bzw. im MQMD, falls keine Header vorhanden sind). Bei JMS-Nachrichten ist der letzte Header in der Regel der MQRFH2.

BytesMessage

BytesMessage ist standardmäßig eine Bytefolge, die durch die Spezifikation JMS 1.0.2 und die zugehörige Java-Dokumentation dokumentiert ist.

Bei einer abgehenden Nachricht, die von der Anwendung selbst zusammengestellt wurde, kann die Codierungseigenschaft des Zielobjekts zum Überschreiben der Codierungen von Feldern mit Ganzzahlen und Gleitkommawerten verwendet werden, die in der Nachricht enthalten sind. (Sie können beispielsweise anfordern, dass Gleitkommawerte nicht im IEEE-Format, sondern im S/390-Format gespeichert werden.)

Eine eingehende Nachricht wird unter Verwendung der numerischen Codierung interpretiert, die in der Nachricht selbst angegeben ist. Diese Spezifikation befindet sich im letzten IBM MQ-Header (bzw. im MQMD, falls keine Header vorhanden sind). Bei JMS-Nachrichten ist der letzte Header in der Regel der MQRFH2.

Wenn eine BytesMessage empfangen und unverändert erneut gesendet wird, wird ihr Hauptteil Byte für Byte so übertragen, wie er empfangen wurde. Die Codierungseigenschaft des Zielobjekts hat keine Auswirkung auf den Hauptteil. Das einzige zeichenfolgeähnliche Element, das explizit in einer BytesMessage gesendet werden kann, ist eine UTF-8-Zeichenfolge. Diese wird im Java-UTF-8-Format codiert. Sie beginnt mit einem aus 2 Bytes bestehenden Längenfeld. Die Zeichensatzeigenschaft des Zielobjekts hat keine Auswirkung auf die Codierung einer abgehenden BytesMessage. Der Zeichensatzwert in einer eingehenden IBM MQ-Nachricht hat keinen Einfluss auf die Interpretation dieser Nachricht als JMS-BytesMessage.

Java-fremde Anwendungen können die Java-UTF-8-Codierung wahrscheinlich nicht erkennen. Damit eine JMS-Anwendung eine BytesMessage mit Textdaten senden kann, muss die Anwendung daher selbst ihre Zeichenfolgen in Byte-Arrays konvertieren und diese Byte-Arrays in die BytesMessage schreiben.

MapMessage

Eine MapMessage ist eine Zeichenfolge, die Triplets aus XML-Name/-Typ/-Wert enthält, die wie folgt codiert sind:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

Dabei steht `datatype` für einen der in Tabelle 21 auf Seite 161 aufgelisteten Datentypen. Da der Standarddatentyp `string` ist, wird das Attribut `dt="string"` bei Zeichenfolgeelementen übergangen.

Der Zeichensatz, der für die Codierung oder Interpretation der XML-Zeichenfolge verwendet wird, welche den Hauptteil einer Zuordnungsnachricht bildet, richtet sich nach den für eine Textnachricht geltenden Regeln.

In Versionen vor Version 5.3 der IBM MQ classes for JMS wurde der Hauptteil einer Zuordnungsnachricht im folgenden Format codiert:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

Die IBM MQ classes for JMS ab Version 5.3 können jedes Format interpretieren, ältere Versionen als Version 5.3 der IBM MQ classes for JMS können das aktuelle Format jedoch nicht interpretieren.

Wenn eine Anwendung Zuordnungsnachrichten an eine andere Anwendung senden muss, die eine Version vor Version 5.3 der IBM MQ classes for JMS verwendet, muss die sendende Anwendung die Verbindungsfactory-Methode `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)` aufrufen, um anzugeben, dass die Zuordnungsnachrichten im früheren Format gesendet werden. Standardmäßig werden alle Zuordnungsnachrichten im aktuellen Format gesendet.

StreamMessage
Eine StreamMessage ähnelt einer Zuordnungsnachricht, enthält jedoch keine Elementnamen:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

Dabei steht `datatype` für einen der in Tabelle 21 auf Seite 161 aufgelisteten Datentypen. Da der Standarddatentyp `string` ist, wird das Attribut `dt="string"` bei Zeichenfolgeelementen übergangen.

Der Zeichensatz, der für die Codierung oder Interpretation der XML-Zeichenfolge verwendet wird, welche den StreamMessage-Hauptteil bildet, richtet sich nach den für eine TextMessage geltenden Regeln.

Das Feld 'MQRFH2.format' wird wie folgt festgelegt:

MQFMT_NONE
für ObjectMessage, BytesMessage oder Nachrichten mit keinem Hauptteil.

MQFMT_STRING
für TextMessage, StreamMessage oder MapMessage.

JMS-Nachrichtenkonvertierung
Die Konvertierung von Nachrichtendaten in JMS erfolgt beim Senden und Empfangen von Nachrichten. IBM MQ führt den Großteil der Datenkonvertierung automatisch durch. Dabei werden Textdaten und numerische Daten beim Übertragen einer Nachricht zwischen JMS-Anwendungen konvertiert. Text wird konvertiert, wenn eine `JMSTextMessage` zwischen einer JMS-Anwendung und einer IBM MQ-Anwendung ausgetauscht wird.

Falls Sie vorhaben, komplexere Nachrichtenaustauschvorgänge durchzuführen, sind die folgenden Abschnitte für Sie interessant. Unter anderem gilt Folgendes als komplexer Nachrichtenaustausch:

- Die Übertragung von Nicht-Textnachrichten zwischen einer IBM MQ-Anwendung und einer JMS-Anwendung.
- Der Austausch von Textdaten im Byteformat.
- Die Konvertierung des Texts in Ihrer Anwendung.

JMS-Nachrichtendaten

Die Datenkonvertierung ist für den Austausch von Textdaten und numerischen Daten zwischen Anwendungen erforderlich. Dies gilt auch dann, wenn der Austausch zwischen zwei JMS-Anwendungen stattfindet. Die interne Darstellung von Text und Zahlen muss codiert werden, damit sie in einer Nachricht übertragen werden können. Die Codierung erzwingt eine Entscheidung hinsichtlich der Darstellungsweise von Zahlen und Text. IBM MQ steuert die Codierung von Text und Zahlen in JMS-Nachrichten, außer bei `JMSObjectMessage` (siehe „JMSObjectMessage” auf Seite 180). Dabei werden drei Nachrichtenattribute verwendet. Die drei Attribute sind `CodedCharacterSetId`,`Encoding` und `Format`.

Diese drei Nachrichtenattribute werden normalerweise in den JMS-Headerfeldern (`MQRFH2`) einer JMS-Nachricht gespeichert. Wenn es sich bei dem Nachrichtentyp um einen MQ-Nachrichtentyp und nicht um einen JMS -Nachrichtentyp handelt, werden die Attribute im Nachrichtendeskriptor `MQMD`gespeichert. Die Attribute werden für die Konvertierung der JMS-Nachrichtendaten verwendet. JMS-Nachrichtendaten werden im Nachrichtendatenteil einer IBM MQ-Nachricht übertragen.

JMS-Nachrichteneigenschaften

JMS-Nachrichteneigenschaften wie z. B. `JMS_IBM_CHARACTER_SET` werden im `MQRFH2`-Headerteil einer JMS-Nachricht ausgetauscht, es sei denn, die Nachricht wurde ohne einen `MQRFH2`-Header gesendet. Nur `JMSTextMessage` und `JMSBytesMessage` können ohne einen `MQRFH2` gesendet werden. Wenn eine JMS-Eigenschaft als IBM MQ-Nachrichteneigenschaft im Nachrichtendeskriptor `MQMD` gespeichert ist, wird sie als Teil der `MQMD`-Konvertierung konvertiert. Wenn eine JMS-Eigenschaft im `MQRFH2` gespeichert wird, wird sie in dem Zeichensatz gespeichert, der durch `MQRFH2.NameValueCCSID` angegeben ist. Wenn eine Nachricht gesendet oder empfangen wird, werden Nachrichteneigenschaft in ihre interne Darstellung in der JVM konvertiert und umgekehrt. Die hierbei verwendete Konvertierung erfolgt in dem Zeichensatz des Nachrichtendeskriptors oder in dem Zeichensatz, der mit `MQRFH2.NameValueCCSID` angegeben wurde. Numerische Daten werden in Text konvertiert.

JMS-Nachrichtenkonvertierung

Die folgenden Abschnitte enthalten Beispiele und Aufgaben, die hilfreich sind, wenn Sie vorhaben, komplexere Nachrichten auszutauschen, die eine Konvertierung erfordern.

Methoden der JMS-Nachrichtenkonvertierung
Den JMS-Anwendungsentwicklern stehen verschiedene Methoden für die Datenkonvertierung zur Verfügung. Diese Methode schließen sich nicht gegenseitig aus; manche Anwendungen werden wahrscheinlich eine Kombination dieser Methoden verwenden. Wenn Ihre Anwendung nur Text oder Nachrichten ausschließlich mit anderen JMS-Anwendungen austauscht, müssen Sie sich normalerweise keine Gedanken um die Datenkonvertierung machen. IBM MQ führt die Datenkonvertierung automatisch für Sie durch.

Sie können die für Sie optimale Methode der Nachrichtenkonvertierung anhand einiger Fragen ermitteln:

Muss ich mich überhaupt um die Nachrichtenkonvertierung kümmern?
In einigen Fällen, wie beispielsweise bei Nachrichtenübertragen von JMS an JMS oder beim Austausch von Textnachrichten mit IBM MQ-Programmen werden alle erforderlichen Konvertierungen automatisch von IBM MQ vorgenommen. Möglicherweise möchten Sie aber aufgrund von Leistungsaspekten die Datenkonvertierung steuern oder müssen komplexe Nachrichten mit einem vordefinierten Format austauschen. In diesen Fällen müssen Sie das Prinzip der Nachrichtenkonvertierung verstehen und die folgenden Abschnitte lesen.

Welche Arten der Konvertierung gibt es?
Es gibt vier Arten der Konvertierung. Diese werden in den folgenden Abschnitten erläutert:

1. „JMS-Client-Datenkonvertierung” auf Seite 174
2. „Anwendungsdatenkonvertierung” auf Seite 175
3. „Warteschlangenmanager-Datenkonvertierung” auf Seite 176
4. „Nachrichtenkanal-Datenkonvertierung” auf Seite 177

Wo sollte die Konvertierung stattfinden?
Im Abschnitt „Methode für die Nachrichtenkonvertierung wählen: receiver makes good” auf Seite 177 wird die übliche Methode beschrieben, bei der der Empfänger die Konvertierung durchführt ("receiver makes good"). "Receiver makes good" gilt auch für die JMS-Datenkonvertierung.

JMS-Client-Datenkonvertierung

Die JMS-Client-Datenkonvertierung[1]Datenkonvertierung ist die Konvertierung von Java -Basiselementen und -Objekten in Byte in einer JMS -Nachricht, wenn sie an ein Ziel gesendet wird, und die Konvertierung wieder zurück, wenn sie empfangen wird. Die JMS-Client-Datenkonvertierung verwendet die Methoden der `JMSMessage`-Klassen. Die Methoden werden nach `JMSMessage`-Klassentyp in Tabelle 31 auf Seite 178 aufgelistet.

[1] "JMS-Client" bezieht sich auf die IBM MQ classes for JMS, die die JMS-Schnittstelle implementieren, welche entweder im Client- oder im Bindungsmodus ausgeführt wird.

Die Konvertierung in die interne JVM-Darstellung von Zahlen und Text und die Konvertierung aus dieser Darstellung wird für Lese-, Abruf-, Festlegungs- und Schreibmethoden (read, get, set und write) durchgeführt. Die Konvertierung erfolgt beim Senden der Nachricht und wenn eine der read- oder get-Methoden für eine empfangene Nachricht aufgerufen wird.

Die Codepage und die numerische Codierung, die zum Schreiben oder Festlegen des Inhalts einer Nachricht verwendet werden, werden als Attribute des Ziels definiert. Die Zielcodepage und die numerische Codierung können im Rahmen einer Verwaltungsaufgabe geändert werden. Eine Anwendung kann ebenfalls die Zielcodepage und die Codierung überschreiben, indem sie die Nachrichteneigenschaften festlegt, die das Schreiben oder Festlegen des Nachrichteninhalts steuern.

Wenn Sie beim Senden einer `JMSBytesMessage`-Nachricht an ein Ziel, dessen Codierung nicht als native Codierung (`Native`) definiert ist, die Zahlencodierung konvertieren möchten, müssen Sie die Nachrichteneigenschaft `JMS_IBM_ENCODING` vor dem Versenden der Nachricht festlegen. Falls Sie das Muster "receiver makes good" verwenden oder Nachrichten zwischen JMS-Anwendungen austauschen, muss die Anwendung die Eigenschaft `JMS_IBM_ENCODING` nicht festlegen. In den meisten Fällen können Sie den Wert `Native` der Eigenschaft `Encoding` übernehmen.

Bei `JMSStreamMessage`-, `JMSMapMessage`- und `JMSTextMessage`-Nachrichten werden die Eigenschaften des Ziels für die Zeichensatzkennung verwendet. Die Codierung wird beim Senden ignoriert, da Zahlen im Textformat ausgegeben werden. Das JMS-Clientanwendungsprogramm muss `JMS_IBM_CHARACTER_SET` vor dem Versenden der Nachricht nicht festlegen, wenn die Zeichensatzeigenschaft des Ziels angewendet werden soll.

Zum Abruf der Daten in einer Nachricht ruft eine Anwendung die JMS-Methoden (read bzw. get) zum Lesen oder Abrufen von Nachrichten auf. Die Methoden beziehen sich auf die Codepage und Codierung, die im vorherigen Nachrichtenheader definiert sind, um die primitiven Java-Elemente und -Objekte ordnungsgemäß zu erstellen.

Die JMS-Client-Datenkonvertierung erfüllt die Anforderungen der meisten JMS-Anwendungen, die Nachrichten zwischen einem JMS-Client und einem anderen austauschen. Sie codieren keine explizite Datenkonvertierung. Sie verwenden nicht die Klasse `java.nio.charset.Charset`, die normalerweise beim Schreiben von Text in eine Datei verwendet wird. Die Methoden `writeString` und `setString` übernehmen die Konvertierung für Sie.

Sie finden weitere Details zur JMS-Client-Datenkonvertierung im Abschnitt „Nachrichtenkonvertierung und -codierung beim JMS-Client” auf Seite 187.

Anwendungsdatenkonvertierung

Eine JMS-Clientanwendung kann mithilfe der Klasse `java.nio.charset.Charset` eine explizite Zeichendatenkonvertierung durchführen; dies wird mit den Beispielen in Abbildung 14 auf Seite 180 und Abbildung 15 auf Seite 180 veranschaulicht. Zeichenfolgedaten werden mit der Methode `getBytes` in Bytes konvertiert und als Bytes gesendet. Die Bytes werden mithilfe eines `String`-Konstruktors, der ein Byte-Array und ein `Charset` abruft, zurück in Text konvertiert. Zeichendaten werden mithilfe der `Charset`-Methoden `encode` und `decode` konvertiert. In der Regel wird die Nachricht als `JMSBytesMessage` gesendet oder empfangen, da der Nachrichtenteil einer `JMSBytesMessage` ausschließlich die Daten enthält, die von der Anwendung geschrieben wurden[2]. Sie können auch unter Verwendung von `JMSStreamMessage`, `JMSMapMessage` oder `JMSObjectMessage` Bytes senden und empfangen.

Es gibt keine Java-Methoden für die Codierung und Decodierung von Bytes, die numerische Daten enthalten, welche in verschiedenen Codierungsformaten dargestellt werden. Numerische Daten werden unter Verwendung der `JMSMessage`-Methoden für das Lesen und Schreiben (read und write) von numerischen Daten automatisch codiert und decodiert. Die Lese- und Schreibmethoden verwenden den Wert des Attributs `JMS_IBM_ENCODING` der Nachrichtendaten.

Eine typische Verwendung der Anwendungsdatenkonvertierung ist, wenn ein JMS-Client eine formatierte Nachricht aus einer JMS-fremden Anwendung sendet oder empfängt. Eine formatierte Nachricht enthält Text, numerische Daten und Bytedaten, die nach der Länge der Datenfelder organisiert sind. Wenn die

[2] Ausnahme: Daten, die mit `writeUTF` geschrieben werden, beginnen mit einem aus 2 Bytes bestehenden Längenfeld.

JMS-fremde Anwendung das Nachrichtenformat nicht als "MQSTR" angegeben hat, wird die Nachricht als `JMSBytesMessage` erstellt. Um formatierte Nachrichtendaten in einer `JMSBytesMessage` zu empfangen, müssen Sie eine Folge von Methoden aufrufen. Die Methoden müssen in der gleichen Reihenfolge aufgerufen werden, in der die Felder in die Nachricht geschrieben wurden. Wenn die Felder numerisch sind, müssen Sie die Codierung und Länge der numerischen Daten kennen. Falls eines der Felder Byte- oder Textdaten enthält, müssen Sie die Länge der Bytedaten in der Nachricht kennen. Eine formatierte Nachricht kann auf zwei benutzerfreundliche Arten in ein Java-Objekt konvertiert werden.

1. Erstellen Sie eine Java-Klasse, die dem Datensatz entspricht, um das Lesen und Schreiben der Nachricht einzubinden. Der Zugriff auf die Daten im Datensatz erfolgt über die get- und set-Methoden der Klasse.
2. Erstellen Sie eine Java-Klasse, die dem Datensatz entspricht, indem Sie die Klasse `com.ibm.mq.headers` erweitern. Der Zugriff auf die Daten in der Klasse erfolgt über typspezifische Zugriffsmethoden der Form `getStringValue(`*`fieldname`*`);`.

Siehe „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 195.

Warteschlangenmanager-Datenkonvertierung

Die Codepagekonvertierung kann vom Warteschlangenmanager ausgeführt werden, wenn ein JMS -Clientprogramm eine Nachricht erhält. Die Konvertierung ist mit derjenigen identisch, die für ein C-Programm durchgeführt wird. Ein C-Programm legt `MQGMO_CONVERT` als `MQGET`-Parameteroption `GetMsgOpts` fest; siehe Abbildung 13 auf Seite 179. Ein Warteschlangenmanager führt die Konvertierung für ein JMS-Clientprogramm durch, das eine Nachricht empfängt, wenn die Zieleigenschaft `WMQ_RECEIVE_CONVERSION` auf `WMQ_RECEIVE_CONVERSION_QMGR` gesetzt ist. Das JMS-Clientprogramm kann die Zieleigenschaft ebenfalls festlegen; siehe Abbildung 12 auf Seite 176.

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Oder

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Abbildung 12. Warteschlangenmanager-Datenkonvertierung aktivieren

Der Hauptvorteil der Konvertierung durch den Warteschlangenmanager wird beim Austausch von Nachrichten mit JMS-fremden Anwendungen ersichtlich. Wenn das Feld `Format` in der Nachricht definiert ist und der Zielzeichensatz oder die Codierung von der Nachricht abweichen, führt der Warteschlangenmanager die Datenkonvertierung für die Zielanwendung durch, falls die Anwendung dies verlangt. Der Warteschlangenmanager konvertiert Nachrichtendaten, die gemäß einem der vordefinierten IBM MQ -Nachrichtentypen formatiert sind, wie z. B. einem CICS bridge -Header (`MQCIH`). Wenn das Feld `Format` benutzerdefiniert ist, sucht der Warteschlangenmanager nach einem Datenkonvertierungsexit mit dem im Feld `Format` angegebenen Namen.

Die Warteschlangenmanager-Datenkonvertierung ist vor allem von Vorteil, wenn das Designmuster "receiver makes good" verwendet wird. Ein sendender JMS-Client muss keine Konvertierung durchführen. Ein empfangendes JMS-fremdes Programm verlässt sich auf den Konvertierungsexit, um sicherzustellen, dass die Nachricht in der erforderlichen Codepage und Codierung zugestellt wird. Bei einem sendenden JMS -Client und einem Empfänger, der keinJMS -Empfänger ist, gilt das Beispiel für IBM MQ.

Sie können mit dem Dienstprogramm **crtmqcvx** für Datenkonvertierungsexits einen Datenkonvertierungsexit erstellen, damit der Warteschlangenmanager Ihr eigenes Datensatzdatenformat konvertieren kann. Sie können Ihr eigenes Datensatzformat erstellen, mit `com.ibm.mq.headers` darauf als Java-Klasse zugreifen und Ihren eigenen Konvertierungsexit für seine Konvertierung verwenden. Unter z/OS

heißt das Dienstprogramm **CSQUCVX** und unter IBM i hat es den Namen **CVTMQMDTA**. Weitere Informationen hierzu finden Sie unter „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 195.

Nachrichtenkanal-Datenkonvertierung

Die Sender-, Server-, Clusterempfänger- und Clustersenderkanäle in IBM MQ verfügen über die Nachrichtenkonvertierungsoption CONVERT. Der Inhalt einer Nachricht kann optional beim Senden der Nachricht konvertiert werden. Die Konvertierung erfolgt auf Sendeseite des Kanals. Die Clusterempfängerdefinition wird verwendet, um den entsprechenden Clustersenderkanal automatisch zu definieren.

Die Datenkonvertierung durch Nachrichtenkanäle wird in der Regel verwendet, wenn keine andere Form der Konvertierung möglich ist.

Methode für die Nachrichtenkonvertierung wählen: "receiver makes good"

Die übliche Methode im IBM MQ-Anwendungsdesign für die Codekonvertierung ist "receiver makes good", bei der der Empfänger für die Konvertierung sorgt. Mit "receiver makes good" lässt sich die Anzahl der Nachrichtenkonvertierungen verringern. Zudem wird das Problem nicht erwarteter Kanalfehler vermieden, wenn die Nachrichtenkonvertierung während der Nachrichtenübertragung bei einem zwischengeschalteten Warteschlangenmanager fehlschlägt. Die Regel "receiver makes good" kann nur dann nicht angewendet werden, wenn der Empfänger aus irgendeinem Grund nicht für die richtige Konvertierung sorgen kann. Dies kann beispielsweise der Fall sein, wenn die empfangende Plattform nicht über den richtigen Zeichensatz verfügt.

Die Regel "receiver makes good" bietet darüber hinaus eine gute allgemeine Anleitung für JMS-Clientanwendungen. In bestimmten Fällen kann die Konvertierung in den richtigen Zeichensatz seitens der Quelle jedoch effizienter sein. Die Konvertierung aus der internen JVM-Darstellung muss stattfinden, wenn eine Nachricht mit Text oder numerischen Typen gesendet wird. Die Konvertierung in den vom Empfänger benötigten Zeichensatz bei einem Empfänger, der kein JMS-Client ist, kann dazu führen, dass der JMS-fremde Empfänger keine Konvertierung mehr durchführen muss. Falls der Empfänger ein JMS-Client ist, führt er die Konvertierung in jedem Fall erneut durch, um Nachrichtendaten zu decodieren und primitive Java-Elemente und -Objekte zu erstellen.

Der Unterschied zwischen JMS-Clientanwendungen und Anwendungen, die in einer Sprache wie C geschrieben wurden, besteht darin, dass Java eine Datenkonvertierung durchführen muss. Eine Java-Anwendung muss Zahlen und Text aus der jeweiligen internen Darstellung in ein codiertes Format konvertieren, das in Nachrichten verwendet wird.

Durch die Festlegung von Ziel- oder Nachrichteneigenschaften können Sie den Zeichensatz und die Codierung festlegen, die von IBM MQ für die Codierung von Zahlen und Text in Nachrichten verwendet werden. Normalerweise übernehmen Sie den Zeichensatz `1208` und die Codierung `Native`.

IBM MQ konvertiert keine Byte-Arrays. Verwenden Sie für die Codierung von Zeichenfolgen und Zeichenbereichen in Byte-Arrays das Paket `java.nio.charset`. `Charset` gibt den Zeichensatz an, der für die Konvertierung einer Zeichenfolge oder eines Zeichenbereichs in ein Byte-Array verwendet wird. Mit `Charset` können Sie auch ein Byte-Array in eine Zeichenfolge oder einen Zeichenbereich decodieren. Es wird davon abgeraten, sich beim Codieren von Zeichenfolgen und Zeichenbereichen auf `java.nio.charset.Charset.defaultCodePage` zu stützen. Der Standardwert von `Charset` ist in der Regel `windows-1252` unter Windows und `UTF-8` unter AIX and Linux. `windows-1252` ist ein Einzelbytezeichensatz und `UTF-8` ist ein Mehrbytezeichensatz.

Behalten Sie im Allgemeinen den Zielzeichensatz und die Codierungseigenschaften bei ihren Standardwerten `UTF-8` und `Native` bei, wenn Sie Nachrichten mit anderen JMS -Anwendungen austauschen. Wenn Sie Nachrichten, die Zahlen oder Text enthalten, mit einer JMS-Anwendung austauschen, wählen Sie den `JMSTextMessage`-, `JMSStreamMessage`-, `JMSMapMessage`- oder `JMSObjectMessage`-Nachrichtentyp aus, der sich am besten für Ihre Zwecke eignet. Es müssen keine weiteren Konvertierungsaufgaben ausgeführt werden.

Wenn Sie Nachrichten mit JMS-fremden Anwendungen austauschen, die ein Datensatzformat verwenden, ist der Vorgang komplizierter. Sie müssen den Text in der Anwendung codieren und decodieren, es sei

denn, der gesamte Datensatz enthält Text und kann als `JMSTextMessage` übertragen werden. Setzen Sie den Zielnachrichtentyp auf `MQ` und verwenden Sie `JMSBytesMessage`, um zu vermeiden, dass IBM MQ classes for JMS den Nachrichtendaten zusätzliche Header- und Taginformationen hinzufügen. Schreiben Sie mit den `JMSBytesMessage`-Methoden Zahlen und Bytes und konvertieren Sie explizit mit der `Charset`-Klasse Text in Byte-Arrays. Ihre Wahl des Zeichensatzes kann durch mehrere Faktoren beeinflusst werden:

- Leistung: Können Sie die Anzahl der Konvertierungen verringern, indem Sie Text in einen Zeichensatz transformieren, der bei den meisten Servern verwendet wird?
- Einheitlichkeit: Übertragen Sie alle Nachrichten in demselben Zeichensatz.
- Reichhaltigkeit: Welche Zeichensätze weisen alle Codepunkte auf, die von Anwendungen verwendet werden müssen?
- Einfachheit: Einzelbytezeichensätze sind einfacher zu verwenden als Zeichensätze mit variabler Länge und Mehrbytezeichensätze.

Siehe „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 195. Dort sind Beispiele für die Konvertierung von Nachrichten enthalten, die mit JMS-fremden Anwendungen ausgetauscht werden.

Beispiele

Tabelle mit Nachrichtentypen und Konvertierungstypen

Tabelle 31. Nachrichtentypen und Konvertierungstypen

	Konvertierungstyp			
Nachrichtentyp	**Text**	**Numerisch**	**Sonstiges**	**--**
`JMSObjectMessage`				`getObject` `setObject`
`JMSTextMessage`	`getText` `setText`			

`JMSBytesMessage`	`readUTF`
	`writeUTF`
		`readDouble`
		`readFloat`
		`readInt`
		`readLong`
		`readShort`
		`readUnsignedShort`
		`writeDouble`
		`writeFloat`
		`writeInt`
		`writeLong`
		`writeShort`
			`readBoolean`
			`readObject`
			`writeBoolean`
			`writeObject`
				`readByte`
				`readUnsignedByte`
				`readBytes`
				`readChar`
				`writeByte`
				`writeBytes`
				`writeChar`

Tabelle 31. Nachrichtentypen und Konvertierungstypen (Forts.)				
	Konvertierungstyp			
Nachrichtentyp	**Text**	**Numerisch**	**Sonstiges**	**--**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Datenkonvertierung über ein C-Programm aufrufen

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                 */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Abbildung 13. Code-Snippet aus `amqsget0.c`

Text in einer JMSBytesMessage senden und empfangen

Der Code in Abbildung 14 auf Seite 180 sendet eine Zeichenfolge in einer BytesMessage. Der Einfachheit halber wird in diesem Beispiel eine einzelne Zeichenfolge gesendet, für die eine JMSTextMessage geeigneter ist. Wenn Sie eine Textzeichenfolge in einer Bytenachricht empfangen möchten, die verschiedene Typen enthält, müssen Sie die Länge der Zeichenfolge in Bytes kennen (*TEXT_LENGTH* in Abbildung 15

auf Seite 180). Selbst bei einer Zeichenfolge mit einer festen Anzahl von Zeichen kann die Länge der Bytedarstellung länger sein.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                    .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Abbildung 14. Zeichenfolge (`String`) in einer `JMSBytesMessage` senden

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Abbildung 15. Zeichenfolge (`String`) von einer `JMSBytesMessage` empfangen

Zugehörige Konzepte

Nachrichtenkonvertierung und -codierung beim JMS-Client
Hier werden die Methoden aufgelistet, die Sie für die Nachrichtenkonvertierung und -codierung beim JMS-Client verwenden. Außerdem werden für jeden Konvertierungstyp Codebeispiele bereitgestellt.

Warteschlangenmanager-Datenkonvertierung
Die Warteschlangenmanager-Datenkonvertierung stand bereits in der Vergangenheit JMS-fremden Anwendungen zur Verfügung, die Nachrichten von JMS-Clients empfangen haben. JMS -Clients, die Nachrichten empfangen, verwenden auch die Warteschlangenmanager-Datenkonvertierung, die optional ist.

Zugehörige Tasks

Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen
Führen Sie die in dieser Aufgabe empfohlenen Schritte aus, um einen Datenkonvertierungsexit sowie eine JMS-Clientanwendung zu entwerfen und zu erstellen, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann. Der Austausch einer formatierten Nachricht mit einer JMS-fremden Anwendung kann mit oder ohne Aufruf eines Datenkonvertierungsexits erfolgen.

Zugehörige Verweise

JMS-Nachrichtentypen und Konvertierung
Die von Ihnen verwendete Methode der Nachrichtenkonvertierung wird durch den jeweiligen Nachrichtentyp bestimmt. Die Interaktion von Nachrichtenkonvertierung und Nachrichtentyp wird für die JMS-Nachrichtentypen `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` und `JMSBytesMessage` beschrieben.

JMS-Nachrichtentypen und Konvertierung
Die von Ihnen verwendete Methode der Nachrichtenkonvertierung wird durch den jeweiligen Nachrichtentyp bestimmt. Die Interaktion von Nachrichtenkonvertierung und Nachrichtentyp wird für die JMS-Nachrichtentypen `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` und `JMSBytesMessage` beschrieben.

`JMSObjectMessage`

`JMSObjectMessage` enthält ein Objekt und alle zugehörigen referenzierten Objekte, die von der JVM in einem Bytestrom serialisiert werden. Der Text wird in UTF-8 serialisiert und auf Zeichenfolgen oder Zeichenbereiche mit höchstens 65534 Bytes begrenzt. Ein Vorteil von `JMSObjectMessage` besteht darin, dass Anwendungen nicht an Datenkonvertierungsproblemen beteiligt sind, solange sie nur die Methoden

und Attribute des Objekts verwenden. JMSObjectMessage stellt eine Datenkonvertierung für komplexe Objekte bereit, bei der sich der Anwendungsprogrammierer keine Gedanken um die Codierung eines Objekts machen muss. Der Nachteil der Verwendung von JMSObjectMessage ist, dass der Austausch nur mit anderen JMS-Anwendungen möglich ist. Wenn Sie einen der anderen JMS-Nachrichtentypen wählen, können JMS-Nachrichten mit JMS-fremden Anwendungen ausgetauscht werden.

„JMSObjectMessage senden und empfangen" auf Seite 183 zeigt ein Zeichenfolgeobjekt (String), das in einer Nachricht ausgetauscht wird.

Eine JMS-Clientanwendung kann eine JMSObjectMessage nur in einer Nachricht empfangen, die einen Hauptteil im JMS-Stil enthält. Das Ziel muss einen Hauptteil im JMS-Stil angeben.

JMSTextMessage

JMSTextMessage enthält eine einzelne Textzeichenfolge. Wenn eine Textnachricht gesendet wird, wird das Textformat (Format) auf "MQSTR ", WMQConstants.MQFMT_STRING, gesetzt. Der Wert für CodedCharacterSetId des Texts wird auf die ID des codierten Zeichensatzes gesetzt, die für das zugehörige Ziel definiert ist. Der Text wird von IBM MQin CodedCharacterSetId codiert. Die Felder CodedCharacterSetId und Format werden entweder im Nachrichtendeskriptor MQMD oder in den JMS-Feldern in einem MQRFH2 festgelegt. Wenn die Nachricht mit dem Nachrichtenhauptteilstil WMQ_MESSAGE_BODY_MQ definiert oder der Hauptteilstil nicht angegeben ist, das Ziel jedoch WMQ_TARGET_DEST_MQ lautet, werden die Nachrichtendeskriptorfelder festgelegt. Andernfalls hat die Nachricht einen JMS-RFH2 und die Felder werden im festen Teil von MQRFH2 festgelegt.

Eine Anwendung kann die ID des codierten Zeichensatzes, die für ein Ziel definiert ist, überschreiben. Sie muss die Nachrichteneigenschaft JMS_IBM_CHARACTER_SET auf eine ID des codierten Zeichensatzes setzen; lesen Sie hierzu das Beispiel in „JMSTextmessage senden und empfangen" auf Seite 183.

Wenn der JMS-Client die Methode consumer.receive aufruft, ist die Warteschlangenmanager-Konvertierung optional. Die Warteschlangenmanager-Konvertierung wird aktiviert, indem die Zieleigenschaft WMQ_RECEIVE_CONVERSION auf WMQ_RECEIVE_CONVERSION_QMGR gesetzt wird. Der Warteschlangenmanager konvertiert die Textnachricht aus dem Wert von JMS_IBM_CHARACTER_SET, der für die Nachricht angegeben ist, bevor die Nachricht an den JMS-Client übertragen wird. Der Zeichensatz der konvertierten Nachricht ist 1208, UTF-8, es sei denn, das Ziel hat einen anderen Wert für WMQ_RECEIVE_CCSID. Der Wert von CodedCharacterSetId in der Nachricht, der sich auf die JMSTextMessage bezieht, wird mit der Zielzeichensatz-ID aktualisiert. Der Text wird von der Methode getText aus dem Zielzeichensatz in Unicode decodiert; lesen Sie hierzu das Beispiel im Abschnitt „JMSTextmessage senden und empfangen" auf Seite 183.

Eine JMSTextMessage kann in einem Nachrichtenhauptteil des MQ-Stils ohne den JMS-Header MQRFH2 gesendet werden. Die Werte der Zielattribute WMQ_MESSAGE_BODY und WMQ_TARGET_DEST bestimmen den Stil des Nachrichtenhauptteils, sofern sie nicht von der Anwendung überschrieben werden. Die Anwendung kann die im Ziel festgelegten Werte überschreiben, indem sie destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) oder destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ) aufruft.

Wenn Sie eine JMSTextMessage mit einem Hauptteil im MQ-Stil senden, indem Sie sie an ein Ziel senden und dabei WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_MQ setzen, können Sie sie nicht als JMSTextMessage von demselben Ziel empfangen. Alle von einem Ziel empfangenen Nachrichten, bei denen WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_MQ gesetzt ist, werden als JMSBytesMessage empfangen. Wenn Sie versuchen, die Nachricht als JMSTextMessage zu empfangen, führt dies zu einer Ausnahmebedingung: ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage

Anmerkung: Der Text in einer JMSBytesMessage wird nicht durch den JMS-Client konvertiert. Der Client kann den Text in der Nachricht nur als Byte-Array empfangen. Wenn die Warteschlangenmanager-Konvertierung aktiviert ist, wird der Text vom Warteschlangenmanager konvertiert, der JMS-Client muss ihn aber dennoch als Byte-Array in einer JMSBytesMessage empfangen.

In der Regel ist es vorteilhafter, mit der Eigenschaft WMQ_TARGET_DEST zu steuern, ob eine JMSTextMessage mit einem Hauptteil im MQ- oder JMS-Stil gesendet wird. Anschließend können Sie die Nach-

richt von einem Ziel empfangen, bei dem WMQ_TARGET_DEST entweder auf WMQ_TARGET_DEST_MQ oder WMQ_TARGET_DEST_JMS gesetzt ist. WMQ_TARGET_DEST hat keine Auswirkung auf dem Empfänger.

JMSMapMessage und JMSStreamMessage

Diese beiden JMS-Nachrichtentypen sind ähnlich. Sie können primitive Datentypen für die Nachrichten mithilfe von Methoden lesen und schreiben, die auf den Schnittstellen DataInputStream und DataOutputStream basieren; siehe „Tabelle mit Nachrichtentypen und Konvertierungstypen" auf Seite 186. Details hierzu finden Sie unter „Nachrichtenkonvertierung und -codierung beim JMS-Client" auf Seite 187. Jedes primitive Element ist mit Tags versehen; siehe „JMS-Nachrichtenhauptteil" auf Seite 171.

Numerische Daten werden gelesen und als Text mit XML-Codierung in die Nachricht geschrieben. Die Zieleigenschaft JMS_IBM_ENCODING wird nicht referenziert. Textdaten werden auf dieselbe Weise wie Text in einer JMSTextMessage behandelt. Wenn Sie den Nachrichteninhalt betrachten, der durch das Beispiel in Abbildung 20 auf Seite 184 erstellt wurde, werden Sie feststellen, dass alle Nachrichtendaten in EBCDIC verfasst sind, da sie mit dem Zeichensatzwert 37 gesendet wurden.

Sie können in einer JMSMapMessage oder JMSStreamMessage mehrere Elemente senden.

Sie können die einzelnen Datenelemente nach Namen aus einer JMSMapMessage oder nach Position aus einer JMSStreamMessage abrufen. Jedes Element wird decodiert, wenn eine get- oder read-Methode aufgerufen wird; dabei wird der in der Nachricht gespeicherte Wert von CodedCharacterSetId verwendet. Wenn die für den Abruf des Elements verwendete Methode einen anderen Typ als den gesendeten Typ zurückgibt, wird der Typ konvertiert. Wenn der Typ nicht konvertiert werden kann, wird eine Ausnahmebedingung ausgelöst. Weitere Informationen finden Sie unter Klasse JMSStreamMessage. Das Beispiel im Abschnitt „Daten in JMSStreamMessage und JMSMapMessage senden" auf Seite 184 veranschaulicht die Typkonvertierung und den Abruf des JMSMapMessage-Inhalts aus der Sequenz.

Das Feld MQRFH2.format für die JMSMapMessage und JMSStreamMessage ist auf "MQSTR " gesetzt. Wenn die Zieleigenschaft WMQ_RECEIVE_CONVERSION auf WMQ_RECEIVE_CONVERSION_QMGR gesetzt ist, werden die Nachrichtendaten vom Warteschlangenmanager konvertiert, bevor sie an den JMS-Client gesendet werden. Die MQRFH2.CodedCharacterSetId der Nachricht ist die WMQ_RECEIVE_CCSID des Ziels. Als MQRFH2.Encoding ist Native festgelegt. Wenn WMQ_RECEIVE_CONVERSION auf WMQ_RECEIVE_CONVERSION_CLIENT_MSG gesetzt ist, wird für CodedCharacterSetId und Encoding des MQRFH2 der vom Sender festgelegte Wert verwendet.

Eine JMS-Clientanwendung kann eine JMSMapMessage oder JMSStreamMessage nur in einer Nachricht empfangen, die einen Hauptteil im JMS-Stil enthält. Außerdem ist der Empfang nur von einem Ziel möglich, das keinen Hauptteil im MQ-Stil angibt.

JMSBytesMessage

Eine JMSBytesMessage kann mehrere primitive Datentypen enthalten. Sie können primitive Datentypen für die Nachrichten mithilfe von Methoden lesen und schreiben, die auf den Schnittstellen DataInputStream und DataOutputStream basieren; siehe „Tabelle mit Nachrichtentypen und Konvertierungstypen" auf Seite 186. Details hierzu finden Sie unter „JMS-Nachrichtentypen und Konvertierung" auf Seite 180.

Die Codierung von numerischen Daten in der Nachricht wird durch den Wert von JMS_IBM_ENCODING gesteuert, der vor dem Schreiben von numerischen Daten in die JMSBytesMessage festgelegt wurde. Eine Anwendung kann die für JMSBytesMessage definierte Standardcodierung Native durch Festlegen der Nachrichteneigenschaft JMS_IBM_ENCODING überschreiben.

Textdaten können unter Verwendung von readUTF und writeUTF in UTF-8 gelesen und geschrieben werden. Für Unicode werden hierfür die Methoden readChar und writeChar verwendet werden. Es gibt keine Methoden, die CodedCharacterSetId verwenden. Alternativ dazu kann der JMS-Client unter Verwendung der Klasse Charset Text in Bytes codieren und decodieren. Dabei werden die Bytes zwischen der JVM und der Nachricht übertragen, ohne dass die IBM MQ classes for JMS eine Konvertierung durchführen; siehe „Text in einer JMSBytesMessage senden und empfangen" auf Seite 184.

Eine an eine MQ-Anwendung gesendete JMSBytesMessage wird in der Regel mit einem Nachrichtenhauptteil des MQ-Stils ohne den JMS-Header MQRFH2 gesendet. Wenn sie an eine JMS-Anwendung gesendet wird, ist der Nachrichtenhauptteilstil normalerweise JMS. Die Werte der Zielattribute WMQ_MESSAGE_BODY und WMQ_TARGET_DEST bestimmen den Stil des Nachrichtenhauptteils, sofern sie nicht von der Anwendung überschrieben werden. Die Anwendung kann die im Ziel festgelegten Werte überschreiben, indem sie `destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ)` oder `destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)` aufruft.

Wenn Sie eine JMSBytesMessage mit einem Hauptteil im MQ-Stil senden, können Sie die Nachricht von einem Ziel empfangen, das einen Nachrichtenhauptteil im MQ-Stil oder im JMS-Stil definiert. Wenn Sie eine JMSBytesMessage mit einem Hauptteil im JMS-Stil senden, müssen Sie die Nachricht von einem Ziel empfangen, das einen Nachrichtenhauptteil im JMS-Stil definiert. Andernfalls wird der MQRFH2 als Teil der Benutzernachrichtendaten behandelt, was möglicherweise nicht Ihren Erwartungen entspricht.

Unabhängig davon, ob eine Nachricht einen MQ -oder einen JMS -Hauptteilstil hat, ist die Art und Weise, wie sie empfangen wird, von der Einstellung WMQ_TARGET_DESTnicht betroffen.

Die Nachricht wird möglicherweise zu einem späteren Zeitpunkt vom Warteschlangenmanager transformiert, wenn ein Format für die Nachrichtendaten bereitgestellt wird und die Warteschlangenmanager-Datenkonvertierung aktiviert ist. Verwenden Sie das Formatfeld ausschließlich für die Angabe des Nachrichtendatenformats oder lassen Sie es leer (MQConstants.MQFMT_NONE).

Sie können mehrere Elemente in einer JMSBytesMessage senden. Jedes numerische Element wird beim Senden der Nachricht unter Verwendung der für die Nachricht definierten Codierung konvertiert.

Sie können die einzelnen Datenelemente aus JMSBytesMessage abrufen. Rufen Sie Lesemethoden in der Reihenfolge auf, in der die Schreibmethoden zur Erstellung der Nachricht aufgerufen wurden. Jedes numerische Element wird beim Aufrufen der Nachricht unter Verwendung des Encoding-Werts konvertiert, der in der Nachricht gespeichert ist.

Im Gegensatz zu JMSMapMessage und JMSStreamMessage enthält JMSBytesMessage nur Daten, die von der Anwendung geschrieben wurden. In den Nachrichtendaten werden keine Zusatzdaten wie beispielsweise XML-Tags gespeichert, mit denen die Elemente in einer JMSMapMessage und JMSStreamMessage definiert werden. Verwenden Sie JMSBytesMessage also für die Übertragung von Nachrichten, die für andere Anwendungen formatiert sind.

Die Konvertierung zwischen JMSBytesMessage und DataInputStream sowie DataOutputStream ist in manchen Anwendungen hilfreich. Code, der auf dem Beispiel „Nachrichten mithilfe von DataInputStream und DataOutputStream lesen und schreiben” auf Seite 185basiert, ist erforderlich, um das Paket com.ibm.mq.header mit JMSzu verwenden.

Beispiele

JMSObjectMessage senden und empfangen

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

Abbildung 16. JMSObjectMessage senden und empfangen

JMSTextmessage senden und empfangen

Eine Textnachricht kann keinen Text in anderen Zeichensätzen enthalten. Das Beispiel zeigt Text in unterschiedlichen Zeichensätzen, der in zwei unterschiedlichen Nachrichten gesendet wird.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Abbildung 17. Textnachricht in dem durch das Ziel definierten Zeichensatz senden

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Abbildung 18. Textnachricht in `ccsid` 37 senden

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Abbildung 19. Textnachricht empfangen

Daten in JMSStreamMessage und JMSMapMessage senden

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                         " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                       " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

Abbildung 20. Daten in JMS`StreamMessage` und JMS`MapMessage` senden

Text in einer JMSBytesMessage senden und empfangen

Der Code in Abbildung 21 auf Seite 185 sendet eine Zeichenfolge in einer BytesMessage. Der Einfachheit halber wird in diesem Beispiel eine einzelne Zeichenfolge gesendet, für die eine JMS`TextMessage` geeigneter ist. Wenn Sie eine Textzeichenfolge in einer Bytenachricht empfangen möchten, die verschiedene Typen enthält, müssen Sie die Länge der Zeichenfolge in Bytes kennen (*TEXT_LENGTH* in Abbildung 22 auf Seite 185). Selbst bei einer Zeichenfolge mit einer festen Anzahl von Zeichen kann die Länge der Bytedarstellung länger sein.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Abbildung 21. Zeichenfolge (`String`) in einer `JMSBytesMessage` senden

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Abbildung 22. Zeichenfolge (`String`) von einer `JMSBytesMessage` empfangen

Nachrichten mithilfe von `DataInputStream` und `DataOutputStream` lesen und schreiben

Der Code in Abbildung 23 auf Seite 185 erstellt eine `JMSBytesMessage` mit einem `DataOutputStream`.

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

Abbildung 23. `JMSBytesMessage` mit `DataOutputStream` senden

Die Anweisung, die die Eigenschaft `JMS_IBM_ENCODING` festlegt, ist auskommentiert. Die Anweisung ist beim direkten Schreiben in eine JMSBytesMessage gültig, hat jedoch keine Auswirkung beim Schreiben in `DataOutputStream`. Die in den `DataOutputStream` geschriebenen Zahlen werden in der Codierung `Native` codiert. Die Festlegung von `JMS_IBM_ENCODING` hat keine Auswirkung.

Der Code in Abbildung 24 auf Seite 186 empfängt eine `JMSBytesMessage` mit einem `DataInput-Stream`.

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

Abbildung 24. JMSBytesMessage mithilfe von DataInputStream empfangen

Die Codepage wird unter Verwendung der in den Eingabenachrichtendaten angegebenen Codepageeigenschaft JMS_IBM_CHARACTER_SET ausgegeben. Bei der Eingabe ist JMS_IBM_CHARACTER_SET eine Java-Codepage und keine numerische ID des codierten Zeichensatzes.

Tabelle mit Nachrichtentypen und Konvertierungstypen

Tabelle 32. Nachrichtentypen und Konvertierungstypen

	Konvertierungstyp			
Nachrichtentyp	**Text**	**Numerisch**	**Sonstiges**	**--**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar

Tabelle 32. Nachrichtentypen und Konvertierungstypen (Forts.)				
	Konvertierungstyp			
Nachrichtentyp	**Text**	**Numerisch**	**Sonstiges**	**--**
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Zugehörige Konzepte

Methoden der JMS-Nachrichtenkonvertierung
Den JMS-Anwendungsentwicklern stehen verschiedene Methoden für die Datenkonvertierung zur Verfügung. Diese Methode schließen sich nicht gegenseitig aus; manche Anwendungen werden wahrscheinlich eine Kombination dieser Methoden verwenden. Wenn Ihre Anwendung nur Text oder Nachrichten ausschließlich mit anderen JMS-Anwendungen austauscht, müssen Sie sich normalerweise keine Gedanken um die Datenkonvertierung machen. IBM MQ führt die Datenkonvertierung automatisch für Sie durch.

Nachrichtenkonvertierung und -codierung beim JMS-Client
Hier werden die Methoden aufgelistet, die Sie für die Nachrichtenkonvertierung und -codierung beim JMS-Client verwenden. Außerdem werden für jeden Konvertierungstyp Codebeispiele bereitgestellt.

Warteschlangenmanager-Datenkonvertierung
Die Warteschlangenmanager-Datenkonvertierung stand bereits in der Vergangenheit JMS-fremden Anwendungen zur Verfügung, die Nachrichten von JMS-Clients empfangen haben. JMS -Clients, die Nachrichten empfangen, verwenden auch die Warteschlangenmanager-Datenkonvertierung, die optional ist.

Zugehörige Tasks

Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen
Führen Sie die in dieser Aufgabe empfohlenen Schritte aus, um einen Datenkonvertierungsexit sowie eine JMS-Clientanwendung zu entwerfen und zu erstellen, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von JMSBytesMessage austauschen kann. Der Austausch einer formatierten Nachricht mit einer JMS-fremden Anwendung kann mit oder ohne Aufruf eines Datenkonvertierungsexits erfolgen.

Nachrichtenkonvertierung und -codierung beim JMS-Client
Hier werden die Methoden aufgelistet, die Sie für die Nachrichtenkonvertierung und -codierung beim JMS-Client verwenden. Außerdem werden für jeden Konvertierungstyp Codebeispiele bereitgestellt.

Die Konvertierung und Codierung finden statt, wenn primitive Java-Elemente oder -Objekte aus einer JMS-Nachricht gelesen oder in diese geschrieben werden. Die Konvertierung wird als 'JMS-Client-Datenkonvertierung' bezeichnet, um sie von der Warteschlangenmanager-Datenkonvertierung und der Anwendungsdatenkonvertierung zu unterscheiden. Die Konvertierung erfolgt strikt, wenn Daten aus einer JMS-Nachricht gelesen oder in diese geschrieben werden. Text wird in die interne 16-Bit-Unicode-Darstellung bzw. aus dieser Darstellung konvertiert.[3]in den Zeichensatz konvertiert, der für Text in Nachrichten verwendet wird. Numerische Daten werden in primitive numerische Java-Typen bzw. aus diesen in die Codierung konvertiert, die für die Nachricht definiert ist. Ob die Konvertierung durchgeführt und welcher Konvertierungstyp dabei verwendet wird, hängt vom JMS-Nachrichtentyp sowie von der Lese- oder Schreiboperation ab.

[3] Bei manchen Unicode-Darstellungen sind mehr als 16 Bit erforderlich. Lesen Sie hierzu die Java SE-Referenzinformationen.

In Tabelle 33 auf Seite 188 sind die Lese- und Schreibmethoden für verschiedene JMS-Nachrichtentypen nach Art der durchgeführten Konvertierung kategorisiert. Die Konvertierungstypen werden im Text unter der Tabelle beschrieben.

Tabelle 33. Nachrichtentypen und Konvertierungstypen

	Konvertierungstyp			
Nachrichtentyp	**Text**	**Numerisch**	**Sonstiges**	**--**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Text

Der Standardwert von `CodedCharacterSetId` für ein Ziel ist 1208, `UTF-8`. Der Text wird standardmäßig aus Unicode konvertiert und als `UTF-8`-Textzeichenfolge gesendet. Beim Empfang wird der Text aus dem codierten Zeichensatz in der vom Client empfangenen Nachricht in Unicode konvertiert.

Die Methoden `setText` und `writeString` konvertieren Text aus Unicode in den Zeichensatz, der für das Ziel definiert ist. Eine Anwendung kann den Zielzeichensatz durch das Festlegen der Nachrichteneigenschaft `JMS_IBM_CHARACTER_SET` überschreiben. `JMS_IBM_CHARACTER_SET` muss beim Senden einer Nachricht eine numerische ID des codierten Zeichensatzes sein.[4].

Bei den Codeausschnitten in „JMSTextmessage senden und empfangen” auf Seite 191 werden zwei Nachrichten gesendet. Die eine Nachricht wird in dem Zeichensatz gesendet, der für das Ziel definiert ist, während die andere Nachricht im Zeichensatz 37 gesendet wird. Dieser wird durch die Anwendung definiert.

Die Methoden `getText` und `readString` konvertieren den Text in der Nachricht aus dem in der Nachricht definierten Zeichensatz in Unicode. Die Methoden verwenden die Codepage, die in der Nachrichteneigenschaft `JMS_IBM_CHARACTER_SET` definiert ist. Die Codepage wird aus `MQRFH2.CodedCharacterSetId` zugeordnet, es sei denn, die Nachricht ist vom Typ 'MQ' und hat keinen `MQRFH2`. Falls es sich bei der Nachricht um eine Nachricht des MQ-Typs ohne `MQRFH2` handelt, wird die Codepage aus `MQMD.CodedCharacterSetId` zugeordnet.

Der Codeausschnitt in Abbildung 29 auf Seite 191 empfängt die Nachricht, die an das Ziel gesendet wurde. Der Text in der Nachricht wird aus der Codepage `IBM037` zurück in Unicode konvertiert.

Anmerkung: Mit dem IBM MQ Explorer können Sie ganz einfach prüfen, ob der Text in den codierten Zeichensatz 37 konvertiert wurde. Durchsuchen Sie die Warteschlange und zeigen Sie die Eigenschaften der Nachrichten vor ihrem Abruf an.

Vergleichen Sie den Codeausschnitt in Abbildung 28 auf Seite 191 mit dem falschen Codeausschnitt in Abbildung 25 auf Seite 189. Im falschen Ausschnitt wird die Textzeichenfolge zweimal konvertiert, und zwar einmal durch die Anwendung und nochmals durch IBM MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

Abbildung 25. Falsche Codepagekonvertierung

Die Methode `writeUTF` konvertiert Text aus Unicode in 1208, UTF-8. Der Textzeichenfolge ist ein Längenfeld mit 2 Bytes vorangestellt. Die maximale Länge der Textzeichenfolge beträgt 65534 Bytes. Die Methode `readUTF` liest ein Element in einer Nachricht, die mit der Methode `writeUTF` geschrieben wurde. Sie liest genau die Anzahl der Bytes, die mit der Methode `writeUTF` geschrieben wurden.

Numerisch

Die standardmäßige numerische Codierung für ein Ziel ist `Native`. Die `Native`-Codierungskonstante für Java hat den Wert 273, `x'00000111'`. Dieser ist bei allen Plattformen gleich. Beim Empfang werden die Zahlen in der Nachricht ordnungsgemäß in numerische primitive Java-Elemente transformiert. Die Transformation verwendet die Codierung, die in der Nachricht definiert ist, und den von der Lesemethode (read) zurückgegebenen Typ.

Die Sendemethode (send) konvertiert Zahlen, die einer Nachricht mit `set` und `write` hinzugefügt wurden, in die numerische Codierung, die für das Ziel definiert ist. Die Zielcodierung kann für eine Nachricht überschrieben werden, indem die Anwendung die Nachrichteneigenschaft `JMS_IBM_ENCODING` festlegt. Beispiel:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING, WMQConstants.WMQ_ENCODING_INTEGER_RE
VERSED);
```

Die numerischen Methoden `get` und `read` konvertieren Zahlen in der Nachricht aus der numerischen Codierung, die in der Nachricht definiert ist. Sie konvertieren die Zahlen in den Typ, der durch die

[4] Beim Empfang einer Nachricht ist `JMS_IBM_CHARACTER_SET` ein Java-`Charset`-Name der Codepage.

Methode `read` oder `get` angegeben wird; siehe Eigenschaft ENCODING. Die Methoden verwenden die Codierung, die in `JMS_IBM_ENCODING` definiert ist. Die Codierung wird aus `MQRFH2.Encoding` zugeordnet, es sei denn, die Nachricht ist vom Typ 'MQ' und hat keinen `MQRFH2`. Falls es sich bei der Nachricht um eine Nachricht des MQ-Typs ohne `MQRFH2` handelt, verwenden die Methoden die Codierung, die in `MQMD.Encoding` definiert ist.

Das Beispiel in Abbildung 30 auf Seite 192 zeigt eine Anwendung, die eine Zahl im Zielformat codiert und diese in einer `JMSStreamMessage` sendet. Vergleichen Sie das Beispiel in Abbildung 30 auf Seite 192 mit dem Beispiel in Abbildung 31 auf Seite 192. Der Unterschied besteht darin, dass `JMS_IBM_ENCODING` in einer `JMSBytesMessage` festgelegt werden muss.

Anmerkung: Mit dem IBM MQ Explorer können Sie ganz einfach prüfen, ob die Zahl ordnungsgemäß codiert wurde. Durchsuchen Sie die Warteschlange und zeigen Sie die Eigenschaften der Nachrichten vor ihrer Verarbeitung an.

Sonstiges

Die Methoden `boolean` (boolesch) codieren die Werte `true` und `false` als `x'01'` und `x'00'` in einer `JMSByteMessage`, `JMSStreamMessage` und `JMSMapMessage`.

Die UTF-Methoden codieren und decodieren Unicode in UTF-8-Textzeichenfolgen. Die Zeichenfolgen sind auf eine Länge mit weniger als 65536 Zeichen begrenzt und ihnen ist ein Längenfeld mit 2 Bytes vorangestellt.

Die Object-Methoden schließen primitive Datentypen als Objekte ein. Numerische Typen und Texttypen werden so codiert oder konvertiert, als ob die primitiven Datentypen unter Verwendung der numerischen Methoden und Textmethoden gelesen oder geschrieben worden wären.

--

Die Methoden `readByte`, `readBytes`, `readUnsignedByte`, `writeByte` und `writeBytes` führen Abruf- oder Einreihungsvorgänge für Einzelbytes oder Byte-Arrays zwischen der Anwendung und der Nachricht ohne Konvertierung durch. Die Methoden `readChar` und `writeChar` führen Abruf- oder Einreihungsvorgänge für 2-Byte-Unicode-Zeichen zwischen der Anwendung und der Nachricht ohne Konvertierung durch.

Mithilfe der Methoden `readBytes` und `writeBytes` kann die Anwendung ihre eigene Codepunktkonvertierung durchführen, wie im Abschnitt „Text in einer JMSBytesMessage senden und empfangen" auf Seite 192 beschrieben.

IBM MQ führt keine Codepagekonvertierung im Client durch, da die Nachricht eine `JMSBytesMessage` ist und die Methoden `readBytes` und `writeBytes` verwendet werden. Wenn die Bytes Text darstellen, müssen Sie dennoch sicherstellen, dass die von der Anwendung verwendete Codepage dem codierten Zeichensatz des Ziels entspricht. Die Nachricht wird möglicherweise erneut von einem Warteschlangenmanager-Konvertierungsexit konvertiert. Eine andere Möglichkeit ist, dass das empfangende JMS-Clientprogramm der Konvention folgt, dass alle Byte-Arrays, die Nachrichtentext darstellen, unter Verwendung der Eigenschaft `JMS_IBM_CHARACTER_SET` in der Nachricht in Zeichenfolgen oder Zeichen konvertiert.

In diesem Beispiel verwendet der Client für seine Konvertierung den codierten Zeichensatz des Ziels:

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Alternativ hätte der Client eine Codepage auswählen und dann den entsprechenden codierten Zeichensatz in der Eigenschaft `JMS_IBM_CHARACTER_SET` der Nachricht festlegen können. Die IBM MQ classes for Java verwenden `JMS_IBM_CHARACTER_SET` für die Festlegung des Felds `CodedCharacterSetId` in den JMS-Eigenschaften im `MQRFH2` oder im Nachrichtendeskriptor `MQMD`:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

Wenn ein Byte-Array in eine JMSStringMessage oder JMSMapMessage geschrieben wird, führen IBM MQ classes for JMS keine Datenkonvertierung durch, da die Bytes in der JMSStringMessage und in der JMSMapMessage nicht als Text, sondern als Hexadezimaldaten typisiert sind.

Wenn die Bytes Zeichen in Ihrer Anwendung darstellen, müssen Sie berücksichtigen, welche Codepunkte gelesen und in die Nachricht geschrieben werden müssen. Der Code in Abbildung 26 auf Seite 191 folgt der Konvention, dass der codierte Zeichensatz des Ziels verwendet wird. Wenn Sie die Zeichenfolge unter Verwendung des Standardzeichensatzes für die JVM erstellen, hängt der Byteinhalt von der jeweiligen Plattform ab. Bei einer JVM unter Windows ist in der Regel für Charset standardmäßig der Wert windows-1252 festgelegt, während unter AIX and Linux UTF-8 verwendet wird. Der Austausch zwischen Windows und AIX and Linux erfordert, dass Sie eine explizite Codepage für den Austausch von Text als Byte auswählen.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Abbildung 26. Bytes, die eine Zeichenfolge in einer JMSStreamMessage darstellen, unter Verwendung des Zielzeichensatzes schreiben

Beispiele

JMSTextmessage senden und empfangen

Eine Textnachricht kann keinen Text in anderen Zeichensätzen enthalten. Das Beispiel zeigt Text in unterschiedlichen Zeichensätzen, der in zwei unterschiedlichen Nachrichten gesendet wird.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Abbildung 27. Textnachricht in dem durch das Ziel definierten Zeichensatz senden

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Abbildung 28. Textnachricht in ccsid 37 senden

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Abbildung 29. Textnachricht empfangen

[5] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

Codierungsbeispiele

Beispiele mit einer Zahl, die in der Codierung gesendet wird, welche für ein Ziel definiert ist. Beachten Sie, dass Sie die Eigenschaft JMS_IBM_ENCODING einer JMSBytesMessage auf den Wert setzen müssen, der für das Ziel angegeben ist.

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

Abbildung 30. Zahl unter Verwendung der Zielcodierung in einer JMSStreamMessage senden

```
BytesMessage bmo = session.createBytesMessage();
bmo.writeInt(256);
int encoding = ((MQDestination) (destination)).getIntProperty
    (WMQConstants.WMQ_ENCODING)
bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
producer.send(bmo);
...
BytesMessage bmi = (BytesMessage)consumer.receive();
System.out.println(bmi.readInt());
...
256
```

Abbildung 31. Zahl unter Verwendung der Zielcodierung in einer JMSBytesMessage senden

Text in einer JMSBytesMessage senden und empfangen

Der Code in Abbildung 32 auf Seite 192 sendet eine Zeichenfolge in einer BytesMessage. Der Einfachheit halber wird in diesem Beispiel eine einzelne Zeichenfolge gesendet, für die eine JMSTextMessage geeigneter ist. Wenn Sie eine Textzeichenfolge in einer Bytenachricht empfangen möchten, die verschiedene Typen enthält, müssen Sie die Länge der Zeichenfolge in Bytes kennen (*TEXT_LENGTH* in Abbildung 33 auf Seite 192). Selbst bei einer Zeichenfolge mit einer festen Anzahl von Zeichen kann die Länge der Bytedarstellung länger sein.

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

Abbildung 32. Zeichenfolge (String) in einer JMSBytesMessage senden

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

Abbildung 33. Zeichenfolge (String) von einer JMSBytesMessage empfangen

Zugehörige Konzepte

Methoden der JMS-Nachrichtenkonvertierung

Den JMS-Anwendungsentwicklern stehen verschiedene Methoden für die Datenkonvertierung zur Verfügung. Diese Methode schließen sich nicht gegenseitig aus; manche Anwendungen werden wahrscheinlich eine Kombination dieser Methoden verwenden. Wenn Ihre Anwendung nur Text oder Nachrichten ausschließlich mit anderen JMS-Anwendungen austauscht, müssen Sie sich normalerweise keine Gedanken um die Datenkonvertierung machen. IBM MQ führt die Datenkonvertierung automatisch für Sie durch.

Warteschlangenmanager-Datenkonvertierung

Die Warteschlangenmanager-Datenkonvertierung stand bereits in der Vergangenheit JMS-fremden Anwendungen zur Verfügung, die Nachrichten von JMS-Clients empfangen haben. JMS -Clients, die Nachrichten empfangen, verwenden auch die Warteschlangenmanager-Datenkonvertierung, die optional ist.

Zugehörige Tasks

Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen

Führen Sie die in dieser Aufgabe empfohlenen Schritte aus, um einen Datenkonvertierungsexit sowie eine JMS-Clientanwendung zu entwerfen und zu erstellen, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann. Der Austausch einer formatierten Nachricht mit einer JMS-fremden Anwendung kann mit oder ohne Aufruf eines Datenkonvertierungsexits erfolgen.

Zugehörige Verweise

JMS-Nachrichtentypen und Konvertierung

Die von Ihnen verwendete Methode der Nachrichtenkonvertierung wird durch den jeweiligen Nachrichtentyp bestimmt. Die Interaktion von Nachrichtenkonvertierung und Nachrichtentyp wird für die JMS-Nachrichtentypen `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` und `JMSBytesMessage` beschrieben.

Warteschlangenmanager-Datenkonvertierung

Die Warteschlangenmanager-Datenkonvertierung stand bereits in der Vergangenheit JMS-fremden Anwendungen zur Verfügung, die Nachrichten von JMS-Clients empfangen haben. JMS -Clients, die Nachrichten empfangen, verwenden auch die Warteschlangenmanager-Datenkonvertierung, die optional ist.

Der Warteschlangenmanager kann Zeichen und numerische Daten in Nachrichtendaten unter Verwendung der Werte von `CodedCharacterSetId`, `Encoding` und `Format`, die für die Nachrichtendaten festgelegt wurden, konvertieren. JMS-fremde Anwendungen konnten die Konvertierungsfunktion schon immer durch Festlegung der `GetMessageOption` `GMO_CONVERT` nutzen.

Der Warteschlangenmanager kann Nachrichten konvertieren, die an JMS -Clients gesendet werden. Die Warteschlangenmanager-Konvertierung wird durch Setzen der Zieleigenschaft `WMQ_RECEIVE_CONVERSION`auf `WMQ_RECEIVE_CONVERSION_QMGR`oder `WMQ_RECEIVE_CONVERSION_CLIENT_MSG`gesteuert. Die Anwendung kann die Zieleinstellung ändern:

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Oder

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Abbildung 34. Warteschlangenmanager-Datenkonvertierung aktivieren

Die Warteschlangenmanager-Datenkonvertierung für einen JMS-Client findet statt, wenn der Client eine `consumer.receive`-Methode aufruft. Die Textdaten werden standardmäßig in UTF-8 (1208) transformiert. Nachfolgende Lese- und Abrufmethoden (read und get) decodieren den Text in den empfangenen Daten aus UTF-8 und erstellen somit primitive Java-Textelemente in ihrer eigenen Unicode-Codierung.

UTF-8 ist nicht der einzige Zielzeichensatz der Warteschlangenmanager-Datenkonvertierung. Sie können eine andere CCSID wählen, indem Sie die Zieleigenschaft `WMQ_RECEIVE_CCSID` festlegen.

Eine Anwendung kann auch die Zieleinstellung ändern, indem sie sie beispielsweise auf 437, DOS-US, setzt:

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

Oder

```
((MQDestination)destination).setReceiveCCSID(437);
```

Abbildung 35. Codierten Zielzeichensatz für Warteschlangenmanager-Konvertierung festlegen

Es besteht ein bestimmter Grund für die Änderung von `WMQ_RECEIVE_CCSID`; die gewählte CCSID wirkt sich nicht auf die Textobjekte aus, die in der JVM erstellt werden. Allerdings können manche JVMs auf bestimmten Plattformen möglicherweise die Konvertierung aus der CCSID des Texts in der Nachricht in Unicode nicht verarbeiten. Die Option bietet Ihnen eine CCSID-Auswahl für jeglichen Text, der in der Nachricht an den Client übermittelt wird. In der Vergangenheit hatten manche JMS-Clientplattformen Probleme, wenn der Nachrichtentext in UTF-8 übermittelt wurde.

Der JMS-Code ist äquivalent zu dem Text in Fettdruck im C-Code in Abbildung 36 auf Seite 194.

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Abbildung 36. Code-Snippet aus `amqsget0.c`

Anmerkung:

Die Warteschlangenmanager-Konvertierung wird nur für die Nachrichtendaten ausgeführt, die ein bekanntes IBM MQ -Format haben. `MQSTR` oder `MQCIH` sind Beispiele bekannter Formate, die vordefiniert sind. Ein bekanntes Format kann auch benutzerdefiniertes Format sein, wenn Sie es in einem Datenkonvertierungsexit bereitgestellt haben.

Als `JMSTextMessage`, `JMSMapMessage` und `JMSStreamMessage` erstellte Nachrichten haben ein `MQSTR`-Format und können vom Warteschlangenmanager konvertiert werden.

Zugehörige Konzepte

Methoden der JMS-Nachrichtenkonvertierung

Den JMS-Anwendungsentwicklern stehen verschiedene Methoden für die Datenkonvertierung zur Verfügung. Diese Methode schließen sich nicht gegenseitig aus; manche Anwendungen werden wahrscheinlich

eine Kombination dieser Methoden verwenden. Wenn Ihre Anwendung nur Text oder Nachrichten ausschließlich mit anderen JMS-Anwendungen austauscht, müssen Sie sich normalerweise keine Gedanken um die Datenkonvertierung machen. IBM MQ führt die Datenkonvertierung automatisch für Sie durch.

Nachrichtenkonvertierung und -codierung beim JMS-Client
Hier werden die Methoden aufgelistet, die Sie für die Nachrichtenkonvertierung und -codierung beim JMS-Client verwenden. Außerdem werden für jeden Konvertierungstyp Codebeispiele bereitgestellt.

„Datenkonvertierungsexit aufrufen" auf Seite 1036
Ein Datenkonvertierungsexit ist ein benutzerdefinierter Exit, der während der Verarbeitung eines MQGET-Aufrufs die Kontrolle übernimmt.

Zugehörige Tasks

Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen
Führen Sie die in dieser Aufgabe empfohlenen Schritte aus, um einen Datenkonvertierungsexit sowie eine JMS-Clientanwendung zu entwerfen und zu erstellen, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann. Der Austausch einer formatierten Nachricht mit einer JMS-fremden Anwendung kann mit oder ohne Aufruf eines Datenkonvertierungsexits erfolgen.

Zugehörige Verweise

JMS-Nachrichtentypen und Konvertierung
Die von Ihnen verwendete Methode der Nachrichtenkonvertierung wird durch den jeweiligen Nachrichtentyp bestimmt. Die Interaktion von Nachrichtenkonvertierung und Nachrichtentyp wird für die JMS-Nachrichtentypen `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` und `JMSBytesMessage` beschrieben.

Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen
Führen Sie die in dieser Aufgabe empfohlenen Schritte aus, um einen Datenkonvertierungsexit sowie eine JMS-Clientanwendung zu entwerfen und zu erstellen, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann. Der Austausch einer formatierten Nachricht mit einer JMS-fremden Anwendung kann mit oder ohne Aufruf eines Datenkonvertierungsexits erfolgen.

Vorbereitende Schritte

Möglicherweise können Sie auch eine einfachere Lösung für den Austausch von Nachrichten mit einer JMS-fremden Anwendung unter Verwendung einer `JMSTextMessage` entwerfen. Prüfen Sie diese Möglichkeit, bevor Sie die Schritte in dieser Aufgabe ausführen.

Informationen zu diesem Vorgang

Ein JMS-Client ist einfacher zu schreiben, wenn er nicht an den einzelnen Schritten der Formatierung von JMS-Nachrichten beteiligt ist, die mit anderen JMS-Clients ausgetauscht werden. Bei den Nachrichtentypen `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` oder `JMSObjectMessage` ist IBM MQ für die Details der Nachrichtenformatierung zuständig. IBM MQ handhabt unterschiedliche Codepages und numerische Codierungen auf verschiedenen Plattformen.

Sie können diese Nachrichtentypen für den Austausch von Nachrichten mit JMS-fremden Anwendungen verwenden. Hierfür müssen Sie verstehen, wie diese Nachrichten von den IBM MQ classes for JMS erstellt werden. Möglicherweise können Sie die JMS-fremde Anwendung ändern, damit sie die Nachrichten interpretieren kann; siehe „Zuordnung von JMS-Nachrichten zu IBM MQ-Nachrichten" auf Seite 155.

Ein Vorteil der Verwendung eines dieser Nachrichtentypen besteht darin, dass die JMS-Clientprogrammierung nicht von dem Anwendungstyp abhängig ist, mit dem der Nachrichtenaustausch stattfindet. Ein Nachteil ist, dass unter Umständen ein anderes Programm geändert werden muss und Sie das andere Programm nicht ändern können.

Ein alternativer Ansatz ist, eine JMS-Clientanwendung zu schreiben, die bestehende Nachrichtenformate handhaben kann. Häufig weisen vorhandene Nachrichten ein festes Format auf und enthalten eine Kombination aus nicht formatierten Daten, Text und Zahlen. Verwenden Sie die Schritte in dieser Aufgabe und den JMS-Beispielclient in „Klassen zum Einbinden eines Satzlayouts in eine JMSBytesMessage schrei-

ben” auf Seite 199 als Ausgangspunkt für die Erstellung eines JMS-Clients, der formatierte Datensätze mit JMS-fremden Anwendungen austauschen kann.

Vorgehensweise

1. Definieren Sie den Satzaufbau oder verwenden Sie eine der vordefinierten IBM MQ-Headerklassen.

   Informationen zur Behandlung von vordefinierten IBM MQ-Headern finden Sie im Abschnitt IBM MQ-Nachrichtenheader handhaben.

   Abbildung 37 auf Seite 197 ist ein Beispiel eines benutzerdefinierten Datensatzaufbaus fester Länge, der vom Konvertierungsdienstprogramm für Daten verarbeitet werden kann.
2. Erstellen Sie den Datenkonvertierungsexit.

   Befolgen Sie die Anweisungen im Abschnitt Datenkonvertierungsexitprogramm schreiben, um einen Datenkonvertierungsexit zu schreiben.

   Versuchen Sie das Beispiel in „Klassen zum Einbinden eines Satzlayouts in eine JMSBytesMessage schreiben” auf Seite 199 und geben Sie dem Datenkonvertierungsexit den Namen `MYRECORD`.
3. Schreiben Sie Java-Klassen, um den Satzaufbau und das Senden und Empfangen eines Datensatzes einzubinden. Sie haben beispielsweise die folgenden beiden Möglichkeiten:
   - Schreiben Sie eine Klasse, die die `JMSBytesMessage` mit dem Datensatz liest und schreibt; siehe „Klassen zum Einbinden eines Satzlayouts in eine JMSBytesMessage schreiben” auf Seite 199.
   - Schreiben Sie eine Klasse, die `com.ibm.mq.header.Header` erweitert und die Datenstruktur des Datensatzes definiert; siehe Klassen für neue Headertypen erstellen.
4. Entscheiden Sie, in welchem codierten Zeichensatz die Nachrichten ausgetauscht werden sollen.

   Lesen Sie den Abschnitt Methode für die Nachrichtenkonvertierung wählen: "receiver makes good".
5. Konfigurieren Sie das Ziel für den Austausch von Nachrichten des MQ-Typs ohne den JMS-Header `MQRFH2`.

   Sowohl das sendende als auch das empfangende Ziel muss für den Austausch von Nachrichten des MQ-Typs konfiguriert sein. Sie können zum Senden und Empfangen dasselbe Ziel verwenden.

   Die Anwendung kann die Zieleigenschaft für den Nachrichtenhauptteil ändern:

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   Bei dem Beispiel in „Klassen zum Einbinden eines Satzlayouts in eine JMSBytesMessage schreiben” auf Seite 199 wird die Zieleigenschaft für den Nachrichtenhauptteil überschrieben, um sicherzustellen, dass eine Nachricht im MQ-Stil gesendet wird.
6. Testen Sie die Lösung mit JMS-Anwendungen und JMS-fremden Anwendungen

   Die folgenden Tools sind beim Testen eines Datenkonvertierungsexits hilfreich:

   - Das Beispielprogramm `amqsgetc0.c` ist hilfreich, wenn Sie den Empfang einer Nachricht testen möchten, die von einem JMS-Client gesendet wurde. Sehen Sie sich die empfohlenen Änderungen für die Verwendung des Beispielheaders `RECORD.h` in Abbildung 38 auf Seite 198 an. Mit den Änderungen empfängt `amqsgetc0.c` eine Nachricht, die vom JMS-Beispielclient gesendet wurde (`TryMyRecord.java`); siehe „Klassen zum Einbinden eines Satzlayouts in eine JMSBytesMessage schreiben” auf Seite 199.
   - Das IBM MQ-Beispielprogramm zum Durchsuchen (`amqsbcg0.c`) ist hilfreich, wenn Sie den Inhalt des Nachrichtenheaders, des JMS-Headers `MQRFH2` und den Nachrichteninhalt überprüfen möchten.
   - Mit dem Programm **`rfhutil`** , das zuvor im SupportPac IH03verfügbar war, können Testnachrichten erfasst und in Dateien gespeichert und anschließend zur Steuerung von Nachrichtenflüssen verwendet werden. Ausgabenachrichten können auch in einer Vielzahl von Formaten gelesen und angezeigt werden. Zu den Formaten gehören zwei Arten von XML sowie ein Abgleich mit einem COBOL-Copybook. Die Daten können in EBCDIC- oder ASCII-Format vorliegen. Bevor die Nachricht gesendet wird, kann ihr ein RFH2-Header hinzugefügt werden.

Wenn Sie den Empfang von Nachrichten mit dem geänderten Beispielprogramm `amqsgetc0.c` ausprobieren und einen Fehler mit dem Ursachencode `2080` erhalten, prüfen Sie, ob die Nachricht einen `MQRFH2` enthält. Bei den Änderungen wird vorausgesetzt, dass die Nachricht an ein Ziel ohne Angabe von `MQRFH2` gesendet wurde.

Beispiele

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                                MQLONG StructLength;
                                MQLONG Encoding;
                                MQLONG CodeCharSetId;
                                MQCHAR Format[8];
                                MQLONG Flags;
                                MQCHAR RecordData[32];
        };
```

Abbildung 37. RECORD.h

- Deklarieren Sie die Datenstruktur von `RECORD.h`.

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- Ändern Sie den Aufruf `MQGET` so, dass `RECORD,` verwendet wird.

  1. Vor der Änderung:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          buflen,          /* buffer length         */
          buffer,          /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

  2. Nach der Änderung:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          sizeof(RECORD),  /* buffer length         */
          pRecord,         /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

- Ändern Sie die Druckanweisung

  1. von:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. Zu:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

Abbildung 38. amqsget0.c ändern

Zugehörige Konzepte

Methoden der JMS-Nachrichtenkonvertierung

Den JMS-Anwendungsentwicklern stehen verschiedene Methoden für die Datenkonvertierung zur Verfügung. Diese Methode schließen sich nicht gegenseitig aus; manche Anwendungen werden wahrscheinlich eine Kombination dieser Methoden verwenden. Wenn Ihre Anwendung nur Text oder Nachrichten ausschließlich mit anderen JMS-Anwendungen austauscht, müssen Sie sich normalerweise keine Gedanken um die Datenkonvertierung machen. IBM MQ führt die Datenkonvertierung automatisch für Sie durch.

Nachrichtenkonvertierung und -codierung beim JMS-Client
Hier werden die Methoden aufgelistet, die Sie für die Nachrichtenkonvertierung und -codierung beim JMS-Client verwenden. Außerdem werden für jeden Konvertierungstyp Codebeispiele bereitgestellt.

Warteschlangenmanager-Datenkonvertierung
Die Warteschlangenmanager-Datenkonvertierung stand bereits in der Vergangenheit JMS-fremden Anwendungen zur Verfügung, die Nachrichten von JMS-Clients empfangen haben. JMS -Clients, die Nachrichten empfangen, verwenden auch die Warteschlangenmanager-Datenkonvertierung, die optional ist.

Dienstprogramm zum Einrichten eines Konvertierungsexitcodes

Zugehörige Verweise

JMS-Nachrichtentypen und Konvertierung
Die von Ihnen verwendete Methode der Nachrichtenkonvertierung wird durch den jeweiligen Nachrichtentyp bestimmt. Die Interaktion von Nachrichtenkonvertierung und Nachrichtentyp wird für die JMS-Nachrichtentypen `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` und `JMSBytesMessage` beschrieben.

Klassen zum Einbinden eines Satzlayouts in eine JMSBytesMessage schreiben
Der Zweck dieser Aufgabe besteht darin, anhand eines Beispiels zu erkunden, wie Sie eine Datenkonvertierung mit einem festen Satzaufbau in einer `JMSBytesMessage` kombinieren können. In dieser Aufgabe erstellen Sie einige Java-Klassen für den Austausch eines Beispielsatzaufbaus in einer `JMSBytesMessage`. Durch die Änderung des Beispiels können Sie Klassen für den Austausch weiterer Datensatzstrukturen schreiben.

Eine `JMSBytesMessage` eignet sich am besten als JMS-Nachrichtentyp, wenn Sie Datensätze mit gemischten Datentypen mit JMS-fremden Programmen austauschen möchten. Bei diesem Typ werden vom JMS-Provider keine Zusatzdaten im Nachrichtenhauptteil eingefügt. Daher ist er der optimale Nachrichtentyp, wenn ein JMS-Clientprogramm mit einem bereits vorhandenen IBM MQ-Programm interagieren muss. Die größte Herausforderung bei Verwendung einer `JMSBytesMessage` besteht im Abgleich der Codierung und des Zeichensatzes, die von dem anderen Programm erwartet werden. Dieses Problem kann durch die Erstellung einer Klasse gelöst werden, die den Datensatz einbindet. Eine Klasse, die das Lesen und Schreiben einer `JMSBytesMessage` für einen bestimmten Datensatztyp einbindet, vereinfacht das Senden und Empfangen von Datensätzen in einem festen Format in einem JMS-Programm. Durch die Erfassung der generischen Aspekte der Schnittstelle in einer abstrakten Klasse kann ein Großteil der Lösung für verschiedene Datensatzformate wiederverwendet werden. Die unterschiedlichen Datensatzformate können in Klassen implementiert werden, die die abstrakte generische Klasse erweitern.

Als Alternative kann die Klasse `com.ibm.mq.headers.Header` erweitert werden. Die `Header`-Klasse verfügt über Methoden wie `addMQLONG` zur deklarativeren Erstellung eines Datensatzformats. Die Verwendung der `Header`-Klasse hat jedoch den Nachteil, dass beim Abrufen und Festlegen von Attributen eine kompliziertere Schnittstelle zur Interpretierung verwendet werden muss. Bei beiden Methoden ist ungefähr dieselbe Menge an Anwendungscode erforderlich.

Eine `JMSBytesMessage` kann neben einem `MQRFH2` nur ein einzelnes Format in einer Nachricht einbinden, es sei denn, jeder Datensatz verwendet dieselben Werte für Format, ID des codierten Zeichensatzes und Codierung. Das Format, die Codierung und der Zeichensatz einer `JMSBytesMessage` sind Eigenschaften der gesamten Nachricht hinter dem `MQRFH2`. Beim Schreiben dieses Beispiels wird vorausgesetzt, dass eine `JMSBytesMessage` nur einen einzigen Benutzerdatensatz enthält.

Vorbereitende Schritte

1. Ihre Kenntnisstufe: Sie müssen mit der Java-Programmierung und mit JMS vertraut sein. Es werden keine Anweisungen für die Einrichtung der Java-Entwicklungsumgebung bereitgestellt. Außerdem ist es von Vorteil, wenn Sie bereits ein Programm für den Austausch einer `JMSTextMessage`, `JMSStreamMessage` oder `JMSMapMessage` geschrieben haben. Sie können dann die Unterschiede verfolgen, die bei einem Nachrichtenaustausch unter Verwendung einer `JMSBytesMessage` bestehen.
2. Für das Beispiel ist IBM WebSphere MQ 7.0 erforderlich.
3. Das Beispiel wurde unter Verwendung der Java-Perspektive der Eclipse-Workbench erstellt. Es erfordert JRE 6.0 oder höher. Sie können die Java-Perspektive in IBM MQ Explorer für die Entwicklung und

Ausführung der Java-Klassen verwenden. Alternativ dazu können Sie auch Ihre eigene Java-Entwicklungsumgebung verwenden.

4. Die Einrichtung der Testumgebung und das Debugging sind in IBM MQ Explorer unkomplizierter als bei Verwendung der Befehlszeilendienstprogramme.

Informationen zu diesem Vorgang

Sie werden durch die Erstellung der folgenden beiden Klassen geführt: RECORD und MyRecord. Gemeinsam binden diese beiden Klassen einen Datensatz mit einem festen Format ein. Sie enthalten Methoden zum Abrufen und Festlegen von Eigenschaften. Die Abrufmethode (get) liest den Datensatz aus einer JMSBytesMessage, während die Einreihungsmethode (put) einen Datensatz in eine JMSBytesMessage schreibt.

In dieser Aufgabe soll keine wiederverwendbare Klasse in Produktionsqualität erstellt werden. Sie können die Beispiele in dieser Aufgabe als Starthilfe für Ihre eigenen Klassen heranziehen. Der Zweck dieser Aufgabe besteht darin, Ihnen einige Anleitungen an die Hand zu geben, die sich vorwiegend auf die Verwendung von Zeichensätzen, Formaten und Codierungen in Verbindung mit einer JMSBytesMessage beziehen. Jeder Schritt bei der Erstellung der Klassen wird erläutert und es werden Aspekte bei der Verwendung einer JMSBytesMessage beschrieben, die gelegentlich übersehen werden.

Die Klasse RECORD ist abstrakt und definiert einige allgemeine Felder für einen Benutzerdatensatz. Die allgemeinen Felder werden mit dem Standardlayout für IBM MQ-Header modelliert, das eine Strukturkennung, eine Version und ein Längenfeld aufweist. Die Codierungs-, Zeichensatz- und Formatfelder, die häufig in IBM MQ-Headern zu finden sind, werden übergangen. Ein anderer Header kann keinem benutzerdefinierten Format folgen. Die Klasse MyRecord, die eine Erweiterung der Klasse RECORD ist, erweitert hierfür buchstäblich den Datensatz um zusätzliche Benutzerfelder. Eine durch die Klassen erstellte JMSBytesMessage kann vom Datenkonvertierungsexit des Warteschlangenmanagers verarbeitet werden.

„Für die Ausführung des Beispiels verwendete Klassen” auf Seite 207 enthält eine umfassende Auflistung von RECORD und MyRecord. Darüber hinaus sind dort zusätzliche "Scaffolding"-Klassen für den Test von RECORD und MyRecord aufgelistet. Es gibt die folgenden zusätzlichen Klassen:

TryMyRecord
Das Hauptprogramm für den Test von RECORD und MyRecord.

EndPoint
Eine abstrakte Klasse, die die Verbindung, das Ziel und die Sitzung von JMS in eine einzelne Klasse einbindet. Ihre Schnittstelle erfüllt lediglich die Mindestanforderungen für das Testen der Klassen RECORD und MyRecord. Es handelt sich bei dieser Klasse nicht um ein etabliertes Designmuster für das Schreiben von JMS-Anwendungen.

Anmerkung: Die Endpoint-Klasse enthält nach der Erstellung eines Ziels folgende Codezeile:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

In V7.0 muss ab V7.0.1.5 die Warteschlangenmanager-Konvertierung aktiviert werden. Standardmäßig ist sie inaktiviert. In V7.0 bis V7.0.1.4 ist die Warteschlangenmanager-Konvertierung standardmäßig aktiviert und diese Codezeile führt zu einem Fehler.

MyProducer und MyConsumer
Klassen, die EndPoint und einen MessageConsumer und MessageProducer erstellen, die verbunden und zum Akzeptieren von Anforderungen bereit sind.

Gemeinsam bilden alle Klassen eine vollständige Anwendung, die Sie erstellen und mit der Sie experimentieren können, um sich mit der Datenkonvertierung in einer JMSBytesMessage vertraut zu machen.

Vorgehensweise

1. Erstellen Sie mit einem Standardkonstruktor eine abstrakte Klasse zur Einbindung der Standardfelder in einen IBM MQ-Header. Später erweitern Sie die Klasse, um den Header genau an Ihre Anforderungen anzupassen.

```
public abstract class RECORD implements Serializable {
   private static final long serialVersionUID = -1616617232750561712L;
   protected final static int UTF8 = 1208;
   protected final static int MQLONG_LENGTH = 4;
   protected final static int RECORD_STRUCT_ID_LENGTH = 4;
   protected final static int RECORD_VERSION_1 = 1;
   protected final String RECORD_STRUCT_ID = "BLNK";
   protected final String RECORD_TYPE = "BLANK   ";
   private String structID = RECORD_STRUCT_ID;
   private int version = RECORD_VERSION_1;
   private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
   private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
   private String headerCharset = "UTF-8";
   private String headerFormat = RECORD_TYPE;

   public RECORD() {
       super();
   }
```

   Anmerkung:

   a. Die Attribute `structID` bis `nextFormat` werden in der Reihenfolge aufgelistet, in der sie in einem IBM MQ-Standardnachrichtenheader zu finden sind.

   b. Die Attribute `format`, `messageEncoding` und `messageCharset` beschreiben den Header selbst und sind nicht Bestandteil des Headers.

   c. Sie müssen entscheiden, ob Sie die ID des codierten Zeichensatzes oder den Zeichensatz des Datensatzes speichern möchten. Java verwendet Zeichensätze, während IBM MQ-Nachrichten IDs des codierten Zeichensatzes verwenden. Der Beispielcode verwendet Zeichensätze.

   d. `int` wird von IBM MQin `MQLONG` serialisiert. `MQLONG` enthält 4 Bytes.

2. Erstellen Sie die Getter und Setter für die privaten Attribute.

   a) Erstellen oder generieren Sie die Getter:

```
    public String getHeaderFormat() { return headerFormat; }
   public int getHeaderEncoding() { return headerEncoding;    }
   public String getMessageCharset() { return headerCharset; }
   public int getMessageEncoding() { return headerEncoding; }
   public String getStructID() { return structID; }
   public int getStructLength() { return structLength; }
   public int getVersion() { return version; }
```

   b) Erstellen oder generieren Sie die Setter:

```
    public void setHeaderCharset(String charset) {
       this.headerCharset = charset; }
   public void setHeaderEncoding(int encoding) {
       this.headerEncoding = encoding; }
   public void setHeaderFormat(String headerFormat) {
       this.headerFormat = headerFormat; }
   public void setStructID(String structID) {
       this.structID = structID; }
   public void setStructLength(int structLength) {
       this.structLength = structLength; }
   public void setVersion(int version) {
       this.version = version; }
}
```

3. Erstellen Sie einen Konstruktor, um eine RECORD-Instanz aus einer `JMSBytesMessage` zu erstellen.

```
public RECORD(BytesMessage message) throws JMSException, IOException,
       MQDataException {
   super();
   setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
   setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
   byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
   message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
   setStructID(new String(structID, getMessageCharset()));
   setVersion(message.readInt());
   setStructLength(message.readInt());
}
```

Anmerkung:

a. `messageCharset` und `messageEncoding` werden aus den Nachrichteneigenschaften erfasst, da sie die für das Ziel festgelegten Werte überschreiben. `format` wird nicht aktualisiert. In diesem Beispiel erfolgt keine Fehlerprüfung. Wenn der Konstruktor `Record(BytesMessage)` aufgerufen wird, wird vorausgesetzt, dass es sich bei `JMSBytesMessage` um eine Nachricht des Typs RECORD handelt. Die Zeile "`setStructID(new String(structID, getMessageCharset()))`" legt die Strukturkennung fest.

b. Die Codezeilen, mit denen die Methode vervollständigt wird, deserialisieren die Felder der Reihe nach in der Nachricht und aktualisieren dabei die Standardwerte, die in der RECORD-Instanz festgelegt sind.

4. Erstellen Sie eine put-Methode, um die Headerfelder in eine `JMSBytesMessage` zu schreiben.

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
       JMSException, UnsupportedEncodingException {
   setHeaderEncoding(myProducer.getEncoding());
   setHeaderCharset(myProducer.getCharset());
   myProducer.setMQClient(true);
   BytesMessage bytes = myProducer.session.createBytesMessage();
   bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
   bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
   bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                        myProducer.getCCSID());
   bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                    + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                    .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
   bytes.writeInt(getVersion());
   bytes.writeInt(getStructLength());
   return bytes;
}
```

Anmerkung:

a. `MyProducer` bindet die JMS-Werte von `Connection`, `Destination`, `Session` und `MessageProducer` in einer einzelnen Klasse ein. 'MyConsumer' wird später verwendet und bindet die JMS-Werte `Connection`, `Destination`, `Session` und `MessageConsumer` in einer einzelnen Klasse ein.

b. Wenn bei einer `JMSBytesMessage` eine andere Codierung als `Native` verwendet wird, muss die Codierung in der Nachricht festgelegt werden. Die Zielcodierung wird in das Nachrichtencodierungsattribut `JMS_IBM_CHARACTER_SET` kopiert und als Attribut der Klasse RECORD gespeichert.

   i) "`setMessageEncoding(myProducer.getEncoding());`" ruft "`(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));`" auf, um die Zielcodierung abzurufen.

   ii) "`Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());`" legt die Nachrichtencodierung fest.

c. Der für die Transformation von Text in Bytes verwendete Zeichensatz wird aus dem Ziel abgerufen und als Attribut der Klasse RECORD gespeichert. Er wird nicht in der Nachricht festgelegt, da er von den IBM MQ classes for JMS nicht beim Schreiben einer `JMSBytesMessage` verwendet wird.

"messageCharset = myProducer.getCharset();"-Aufrufe

```
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());
}
```

Der Java-Zeichensatz wird aus einer ID des codierten Zeichensatzes abgerufen.

"CCSID.getCodepage(ccsid)" ist im Paket com.ibm.mq.headers enthalten. Der Wert von ccsid wird aus einer anderen Methode in MyProducer abgerufen, die das Ziel abfragt:

```
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));
    }
```

d. "myProducer.setMQClient(true);" überschreibt die Zieleinstellung für den Clienttyp und erzwingt einen IBM MQ MQI client. Es kann sinnvoll sein, diese Codezeile wegzulassen, da sie einen administrativen Verwaltungskonfiguration verschleiert.

"myProducer.setMQClient(true);" ruft Folgendes auf:

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

Der Code hat den Nebeneffekt, dass der IBM MQ-Hauptteilstil auf 'unspecified' (nicht angegeben) gesetzt wird, wenn eine Einstellung von JMS überschrieben werden muss.

Anmerkung:

Die IBM MQ classes for JMS schreiben das Format, die Codierung und die Zeichensatzkennung der Nachricht in den Nachrichtendeskriptor MQMD oder in den JMS-Header MQRFH2. Dies hängt davon ab, ob die Nachricht einen Hauptteil im IBM MQ-Stil enthält. Legen Sie die MQMD-Felder nicht manuell fest.

Es gibt eine Methode, mit der die Eigenschaften des Nachrichtendeskriptors manuell festgelegt werden können. Dabei werden die JMS_IBM_MQMD_*-Eigenschaften verwendet. Sie müssen die Zieleigenschaft WMQ_MQMD_WRITE_ENABLED festlegen, um die JMS_IBM_MQMD_*-Eigenschaften festzulegen:

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

Sie müssen die Zieleigenschaft WMQ_MQMD_READ_ENABLED festlegen, damit die Eigenschaften gelesen werden können.

Verwenden Sie JMS_IBM_MQMD_* nur, wenn Sie den uneingeschränkten Zugriff auf die gesamten Nachrichtennutzdaten übernehmen möchten. Anders als die JMS_IBM_*-Eigenschaften steuern die JMS_IBM_MQMD_*-Eigenschaften nicht, wie IBM MQ classes for JMS eine JMS-Nachricht erstellen. Es ist möglich, dass Nachrichtendeskriptoreigenschaften erstellt werden, die mit den Eigenschaften der JMS-Nachricht in Konflikt stehen.

e. Die Codezeilen, die die Methode vervollständigen, serialisieren die Attribute in der Klasse als Felder in der Nachricht.

Die Zeichenfolgeattribute werden mit Leerzeichen aufgefüllt. Die Zeichenfolgen werden unter Verwendung des für den Datensatz definierten Zeichensatzes in Bytes konvertiert und auf die Länge der Nachrichtenfelder abgeschnitten.

5. Schließen Sie die Klasse durch Hinzufügen der Importe ab.

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
```

```
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. Erstellen Sie eine Klasse zur Erweiterung der RECORD-Klasse durch zusätzliche Felder. Schließen Sie einen Standardkonstruktor ein.

```
public class MyRecord extends RECORD {
   private static final long serialVersionUID = -370551723162299429L;
   private final static int FLAGS = 1;
   private final static String STRUCT_ID = "MYRD";
   private final static int DATA_LENGTH = 32;
   private final static String FORMAT = "MYRECORD";
   private int flags = FLAGS;
   private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

   public MyRecord() {
       super();
       super.setStructID(STRUCT_ID);
       super.setHeaderFormat(FORMAT);
       super.setStructLength(super.getStructLength() + MQLONG_LENGTH
               + DATA_LENGTH);
   }
```

Anmerkung:

a. Die RECORD-Unterklasse `MyRecord` passt die Strukturkennung, das Format und die Länge des Headers an.

7. Erstellen oder generieren Sie die Getter und Setter.

a) Erstellen Sie die Getter:

```
    public int getFlags() { return flags; }
   public String getRecordData() { return recordData; } .
```

b) Erstellen Sie die Setter:

```
    public void setFlags(int flags) {
       this.flags = flags; }
   public void setRecordData(String recordData) {
       this.recordData = recordData; }
}
```

8. Erstellen Sie einen Konstruktor, um eine `MyRecord`-Instanz aus einer `JMSBytesMessage` zu erstellen.

```
    public MyRecord(BytesMessage message) throws JMSException, IOException,
           MQDataException {
       super(message);
       setFlags(message.readInt());
       byte[] recordData = new byte[DATA_LENGTH];
       message.readBytes(recordData, DATA_LENGTH);
       setRecordData(new String(recordData, super.getMessageCharset()));
   }
```

Anmerkung:

a. Die Felder, aus denen sich die Standardnachrichtenvorlage zusammensetzt, werden zuerst von der Klasse RECORD gelesen.

b. Der `recordData`-Text wird unter Verwendung der Zeichensatzeigenschaft der Nachricht in eine Zeichenfolge (`String`) konvertiert.

9. Erstellen Sie eine statische Methode, um eine Nachricht aus einem Konsumenten abzurufen und eine neue MyRecord-Instanz zu erstellen.

```
 public static MyRecord get(MyConsumer myConsumer) throws JMSException,
        MQDataException, IOException {
    BytesMessage message = (BytesMessage) myConsumer.receive();
    return new MyRecord(message);
}
```

Anmerkung:

a. In dem Beispiel wird der Konstruktor MyRecord(BytesMessage) der Einfachheit halber über die statische get-Methode aufgerufen. Ansonsten wird häufig der Empfang der Nachricht von der Erstellung einer neuen MyRecord-Instanz getrennt.

10. Erstellen Sie eine put-Methode, um die Kundenfelder an eine JMSBytesMessage anzuhängen, die einen Nachrichtenheader enthält.

```
 public BytesMessage put(MyProducer myProducer) throws JMSException,
        IOException {
    BytesMessage bytes = super.put(myProducer);
    bytes.writeInt(getFlags());
    bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                    + DATA_LENGTH + "s",getRecordData())
                    .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
    myProducer.send(bytes);
    return bytes;
}
```

Anmerkung:

a. Die Methodenaufrufe im Code serialisieren die Attribute in der MyRecord-Klasse als Felder in der Nachricht.

- Das recordData-Attribut String wird mit Leerzeichen aufgefüllt, unter Verwendung des für den Datensatz definierten Zeichensatzes in Bytes konvertiert und auf die Länge der RecordData-Felder abgeschnitten.

11. Schließen Sie die Klasse durch Hinzufügen der Include-Anweisungen ab.

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Ergebnisse

- Die Ausführung der Klasse TryMyRecord führt zu folgenden Ergebnissen:
  - Senden der Nachricht im codierten Zeichensatz 37 und Verwendung eines Warteschlangenmanager-Konvertierungsexits:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

  - Senden der Nachricht im codierten Zeichensatz 37 ohne Verwendung eines Warteschlangenmanager-Konvertierungsexits:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- Die Ergebnisse der Änderung der `TryMyRecord`-Klasse bewirken, dass die Nachricht nicht empfangen, sondern stattdessen mit dem geänderten Beispiel `amqsget0.c` empfangen werden. Das geänderte Beispiel akzeptiert einen formatierten Datensatz; siehe Abbildung 38 auf Seite 198 in „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 195.
  - Senden der Nachricht im codierten Zeichensatz 37 und Verwendung eines Warteschlangenmanager-Konvertierungsexits:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - Senden der Nachricht im codierten Zeichensatz 37 ohne Verwendung eines Warteschlangenmanager-Konvertierungsexits:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
no more messages
Sample AMQSGET0 end
```

Probieren Sie das Beispiel aus und experimentieren Sie mit unterschiedlichen Codepages und einem Datenkonvertierungsexit. Erstellen Sie die Java-Klassen, konfigurieren Sie IBM MQ und führen Sie das Hauptprogramm `TryMyRecord` aus; siehe „#unique_196/unique_196_Connect_42_Try” auf Seite 207.

1. Konfigurieren Sie IBM MQ und JMS für die Ausführung des Beispiels. Die Anweisungen gelten für die Ausführung des Beispiels unter Windows.
   a. Einen WS-Manager erstellen

```
crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

   b. Erstellen Sie eine Warteschlange.

```
echo DEFINE QL('Q1') REPLACE | runmqsc QM1
```

   c. Erstellen Sie ein JNDI-Verzeichnis.

```
cd c:\
md JNDI-Directory
```

   d. Wechseln Sie in das JMS-Verzeichnis 'bin'.

      Das JMS-Verwaltungsprogramm muss von hier aus ausgeführt werden. Der Pfad lautet *MQ_INSTALLATION_PATH*`\java\bin`.

   e. Erstellen Sie die folgenden JMS-Definitionen in einer Datei namens `JMSQM1Q1.txt`.

```
DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ) VER
SION(7)
END
```

   f. Führen Sie das Programm JMSAdmin aus, um die JMS-Ressourcen zu erstellen.

```
JMSAdmin < JMSQM1Q1.txt
```

2. Sie können die von Ihnen erstellten Definitionen in IBM MQ Explorer erstellen, ändern und durchsuchen.
3. Führen Sie `TryMyRecord` aus.

Für die Ausführung des Beispiels verwendete Klassen

Die in den folgenden Codeblöcken aufgelisteten Klassen sind auch in einer komprimierten Datei verfügbar. Laden Sie jm25529_.zip oder jm25529_.tar.gzherunter.

TryMyRecord

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
    }
}
```

RECORD

JM 3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
```

```
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
```

```
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

JM 3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .
```

```
    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

JM 3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
```

```
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
```

```
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

JMS 3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

JMS 3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
```

```
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

Verbindungsfactorys und Ziele erstellen und konfigurieren

Eine IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Anwendung kann Verbindungsfactorys und Ziele erstellen, indem sie sie als verwaltete Objekte aus einem JNDI-Namensbereich (Java Naming and Directory Interface) abruft, die IBM JMS -Erweiterungen oder die IBM MQ JMS -Erweiterungen verwendet. Eine Anwendung kann die IBM JMS-Erweiterungen oder IBM MQ-JMS-Erweiterungen auch zum Festlegen der Eigenschaften von Verbindungsfactorys und Zielen verwenden.

Verbindungsfactorys und Ziele sind Ausgangspunkte im Ablauf der Logik einer JMS -oder Jakarta Messaging -Anwendung. Eine Anwendung verwendet ein ConnectionFactory-Objekt zum Erstellen einer Verbindung mit einem Messaging-Server und sie verwendet ein Queue- oder Topic-Objekt als Ziel für das Versenden von Nachrichten oder als Quelle für den Empfang von Nachrichten. Daher muss eine Anwendung mindestens eine Verbindungsfactory und mindestens ein Ziel erstellen. Nach der Erstellung einer Verbindungsfactory oder eines Ziels muss die Anwendung anschließend möglicherweise das Objekt konfigurieren, indem sie eine oder mehrere der zugehörigen Eigenschaften festlegt.

Zusammenfassend lässt sich sagen, dass eine Anwendung Verbindungsfactorys und Ziele wie folgt erstellen und konfigurieren kann:

Unter Verwendung von JNDI für den Abruf von verwalteten Objekten

Ein Administrator kann das IBM MQ JMS -Verwaltungstool gemäß der Beschreibung im Artikel JMS- und Jakarta Messaging-Objekte mit den Verwaltungstools konfigurierenoder IBM MQ Explorer gemäß der Beschreibung im Abschnitt JMS 2.0 -Objekte mit IBM MQ Explorer konfigurierenverwenden, um Verbindungsfactorys und Ziele als verwaltete Objekte in einem JNDI-Namensbereich zu erstellen und zu konfigurieren. Anschließend kann eine Anwendung die verwalteten Objekte aus dem JNDI-Namensbereich abrufen. Nach dem Abruf eines verwalteten Objekts kann die Anwendung bei Bedarf eine der zugehörigen Eigenschaften entweder mit den IBM JMS-Erweiterungen oder mit den IBM MQ-JMS-Erweiterungen festlegen oder ändern.

Anmerkung: 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.

Verwendung der IBM JMS-Erweiterungen

Eine Anwendung kann mithilfe der IBM JMS-Erweiterungen dynamisch zur Laufzeit Verbindungsfactorys und Ziele erstellen. Die Anwendung erstellt zuerst ein JmsFactoryFactory-Objekt und verwendet dann Methoden dieses Objekts, um Verbindungsfactorys und Ziele zu erstellen. Nach der Erstellung einer Verbindungsfactory oder eines Ziels kann die Anwendung für die Festlegung der zugehörigen Eigenschaften Methoden verwenden, die aus der JmsPropertyContext-Schnittstelle übernommen wurden. Alternativ kann die Anwendung einen Uniform Resource Identifier (URI) zur Angabe einer oder mehrerer Eigenschaften eines Ziels bei der Erstellung des Ziels verwenden.

Verwendung der IBM MQ JMS-Erweiterungen

Eine Anwendung kann auch mithilfe der IBM MQ JMS-Erweiterungen dynamisch zur Laufzeit Verbindungsfactorys und Ziele erstellen. Die Anwendung erstellt Verbindungsfactorys und Ziele mithilfe der bereitgestellten Konstruktoren. Nach der Erstellung einer Verbindungsfactory oder eines Ziels kann die Anwendung für die Festlegung der zugehörigen Eigenschaften Methoden des Objekts verwenden. Alternativ kann die Anwendung einen URI zur Angabe einer oder mehrerer Eigenschaften eines Ziels bei der Erstellung des Ziels verwenden.

Zugehörige Tasks

JMS-und Jakarta Messaging-Ressourcen konfigurieren

JNDI zum Abrufen von verwalteten Objekten in einer JMS -oder Jakarta Messaging -Anwendung verwenden
Zum Abrufen verwalteter Objekte aus einem JNDI-Namensbereich (Java Naming and Directory Interface) muss eine JMS -oder Jakarta Messaging -Anwendung einen Ausgangskontext erstellen und anschließend die Methode "lookup ()" verwenden, um die Objekte abzurufen.

Bevor eine Anwendung verwaltete Objekte aus einem JNDI-Namensbereich abrufen kann, muss ein Administrator diese verwalteten Objekte zuerst erstellen.

JMS 2.0 Für JMS 2.0kann der Administrator das IBM MQ JMS -Verwaltungstool **JMSAmin**oder IBM MQ Explorer verwenden, um verwaltete Objekte in einem JNDI-Namensbereich zu erstellen und zu verwalten. Weitere Informationen finden Sie im Abschnitt Verbindungsfactorys und Ziele in einem JNDI-Namensbereich konfigurieren.

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 sein eigenes Repository für verwaltete Objekte und seine eigenen Tools für das Erstellen und Warten der Objekte zur Verfügung.

Zum Abrufen von verwalteten Objekten aus einem JNDI-Namensbereich muss eine Anwendung zuerst einen Ausgangskontext erstellen, wie im folgenden Beispiel dargestellt:

JM 3.0

```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

JMS 2.0

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

In diesem Code haben die Zeichenfolgevariablen `url` und `icf` die folgenden Bedeutungen:

url
Die URL (Uniform Resource Locator) des Verzeichnisservice. Die URL kann eines der folgenden Formate aufweisen:

- `ldap://`*`hostname/contextName`* für einen Verzeichnisservice, der auf einem LDAP-Server basiert
- `file:/`*`directoryPath`* für einen Verzeichnisservice, der auf dem lokalen Dateisystem basiert

icf
Der Klassenname der Ausgangskontextfactory, der einen der folgenden Werte aufweisen kann:

- `com.sun.jndi.ldap.LdapCtxFactory` für einen Verzeichnisservice, der auf einem LDAP-Server basiert
- `com.sun.jndi.fscontext.RefFSContextFactory` für einen Verzeichnisservice, der auf dem lokalen Dateisystem basiert

Beachten Sie, dass einige Kombinationen aus einem JNDI-Paket und einem LDAP-Service-Provider (LDAP = Lightweight Directory Access Protocol) die Ausgabe des LDAP-Fehlers 84 verursachen können. Um diesen Fehler zu beheben, fügen Sie die folgende Codezeile vor den Aufruf an InitialDirContext() ein:

```
environment.put(Context.REFERRAL, "throw");
```

Nachdem ein Ausgangskontext angefordert wurde, kann die Anwendung verwaltete Objekte aus dem JNDI-Namensbereich mithilfe der Methode lookup() abrufen, wie im folgenden Beispiel dargestellt:

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

Dieser Code ruft die folgenden Objekte aus einem LDAP-basierten Namensbereich ab:

- Ein ConnectionFactory-Objekt, das mit dem Namen 'myCF' gebunden ist
- Ein Queue-Objekt, das mit dem Namen 'myQ' gebunden ist
- Ein Topic-Objekt, das mit dem Namen 'myT' gebunden ist

Weitere Informationen zur Verwendung von JNDI finden Sie in der JNDI-Dokumentation der Oracle Corporation.

Zugehörige Tasks

JMS 2.0 mit IBM MQ Explorer konfigurieren

JMS-und Jakarta-Messaging-Objekte mit den Verwaltungstools konfigurieren

JMS-Ressourcen 2.0 in WebSphere Application Server konfigurieren

Verwendung der IBM JMS-Erweiterungen

IBM MQ classes for JMS (JMS 2.0) und IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) enthalten jeweils eine funktional identische Gruppe von Erweiterungen für die JMS -API, die als IBM JMS -Erweiterungen bezeichnet wird. Eine Anwendung kann diese Erweiterungen verwenden, um Verbindungsfactorys und Ziele dynamisch zur Laufzeit zu erstellen und die Eigenschaften von IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekten festzulegen. Die Erweiterungen können zusammen mit jedem beliebigen Messaging-Provider verwendet werden.

Bei den IBM JMS-Erweiterungen handelt es sich um eine Gruppe von Schnittstellen und Klassen in folgenden Paketen:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Für Jakarta Messaging 3.0befinden sich diese Pakete in `com.ibm.jakarta.client.jar`.

JMS 2.0 Für JMS 2.0befinden sich diese Pakete in `com.ibm.mqjms.jar` oder `com.ibm.mq.allc-lient.jar`.

Diese Erweiterungen stellen die folgende Funktion bereit:

- Einen auf der Factory basierenden Mechanismus, mit dem Verbindungsfactorys und Ziele dynamisch zur Laufzeit erstellt werden können, statt sie als verwaltete Objekte aus einem JNDI-Namensbereich (JNDI = Java Naming and Directory Interface) abzurufen

- Eine Gruppe von Methoden zum Festlegen der Eigenschaften von IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekten
- Eine Gruppe mit Ausnahmebedingungsklassen, die Methoden für den Abruf detaillierter Informationen zu einem Problem enthalten
- Eine Gruppe von Methoden für die Steuerung der Traceverarbeitung
- Eine Gruppe von Methoden zum Abrufen von Versionsinformationen zu IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging

Um Verbindungsfactorys und Ziele dynamisch zur Laufzeit zu erstellen und ihre Eigenschaften festzulegen und abzurufen, stellen die IBM JMS -Erweiterungen eine alternative Gruppe von Schnittstellen zu den IBM MQ JMS -Erweiterungen bereit. Während die IBM MQ-JMS-Erweiterungen jedoch nur für den IBM MQ-Messaging-Provider vorgesehen sind, sind die IBM JMS-Erweiterungen nicht IBM MQ-spezifisch und können mit jedem beliebigen Messaging-Provider innerhalb der im Abschnitt IBM MQ-Klassen für JMS-Architektur beschriebenen Schichtarchitektur verwendet werden.

Die Schnittstelle com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) oder com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) enthält die Definitionen von Konstanten, die eine Anwendung verwenden kann, um die Eigenschaften von IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekten mithilfe der IBM JMS -Erweiterungen festzulegen. Die Schnittstelle enthält Konstanten für den IBM MQ-Messaging-Provider und JMS-Konstanten, die unabhängig von einem bestimmten Messaging-Provider eingesetzt werden können.

Die folgenden Codebeispiele gehen davon aus, dass die folgenden Importanweisungen in der Klasse Java enthalten sind:

JM 3.0
```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0
```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Verbindungsfactorys und Ziele erstellen

Bevor eine Anwendung Verbindungsfactorys und Ziele mithilfe der IBM JMS-Erweiterungen erstellen kann, muss sie zuerst ein JmsFactoryFactory-Objekt erstellen. Um ein JmsFactoryFactory-Objekt zu erstellen, ruft die Anwendung die Methode getInstance() der JmsFactoryFactory-Klasse auf, wie im folgenden Beispiel dargestellt:

JM 3.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

Der Parameter im Aufruf getInstance() ist eine Konstante, die den IBM MQ-Messaging-Provider als gewählten Messaging-Provider identifiziert. Die Anwendung kann dann das JmsFactoryFactory-Objekt verwenden, um Verbindungsfactorys und Ziele zu erstellen.

Um eine Verbindungsfactory zu erstellen, ruft die Anwendung die Methode createConnectionFactory() des JmsFactoryFactory-Objekts auf, wie im folgenden Beispiel dargestellt:

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

Diese Anweisung erstellt ein JmsConnectionFactory-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Dies bedeutet, dass sich die Anwendung im Bindungsmodus mit dem Standardwarteschlangenmanager verbindet. Wenn Sie möchten, dass sich eine Anwendung im Clientmodus verbindet oder eine Verbindung zu einem anderen Warteschlangenmanager als dem Standardwarteschlangenmanager herstellt, muss die Anwendung die entsprechenden Eigenschaften des JmsConnectionFactory-Objekts festlegen, bevor die Verbindung erstellt wird. Weitere Informationen hierzu finden Sie im Abschnitt „Eigenschaften von Objekten der IBM MQ classes for JMS festlegen” auf Seite 219.

Die JmsFactoryFactory-Klasse enthält außerdem Methoden, mit denen Verbindungsfactorys der folgenden Typen erstellt werden können:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

Um ein Queue-Objekt zu erstellen, ruft die Anwendung die Methode createQueue() des JmsFactoryFactory-Objekts auf, wie im folgenden Beispiel dargestellt:

```
JmsQueue q1 = ff.createQueue("Q1");
```

Diese Anweisung erstellt ein JmsQueue-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Das Objekt stellt eine IBM MQ-Warteschlange mit dem Namen Q1 dar, die dem lokalen Warteschlangenmanager gehört. Bei dieser Warteschlange kann es sich um eine lokale Warteschlange, um eine Aliaswarteschlange oder um eine Definition einer fernen Warteschlange handeln.

Die Methode createQueue() kann auch einen Warteschlangen-URI (URI = Uniform Resource Identifier) als Parameter akzeptieren. Ein Warteschlangen-URI ist eine Zeichenfolge, die den Namen einer IBM MQ-Warteschlange und optional den Namen des Warteschlangenmanagers angibt, der Eigner der Warteschlange ist, sowie eine oder mehrere Eigenschaften des JmsQueue-Objekts. Die folgende Anweisung enthält ein Beispiel für einen Warteschlangen-URI:

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

Das durch diese Anweisung erstellte JmsQueue-Objekt stellt eine IBM MQ-Warteschlange mit dem Namen Q2 dar. Der Eigner dieser Warteschlange ist der Warteschlangenmanager QM2; alle Nachrichten, die an dieses Ziel gesendet werden, sind persistent und haben die Priorität 5. Sie finden weitere Informationen zu Warteschlangen-URIs im Abschnitt „Uniform Resource Identifiers (URIs)” auf Seite 232. Eine alternative Methode für das Festlegen der Eigenschaften eines JmsQueue-Objekts ist im Abschnitt „Eigenschaften von Objekten der IBM MQ classes for JMS festlegen” auf Seite 219 beschrieben.

Um ein Topic-Objekt zu erstellen, kann eine Anwendung die Methode createTopic() des JmsFactoryFactory-Objekts verwenden, wie im folgenden Beispiel dargestellt:

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

Diese Anweisung erstellt ein JmsTopic-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Das Objekt stellt ein Thema mit dem Namen Sport/Football/Results dar.

Die Methode createTopic() kann auch einen Themen-URI als Parameter akzeptieren. Ein Themen-URI ist eine Zeichenfolge, die den Namen eines Themas und optional eine oder mehrere Eigenschaften des JmsTopic-Objekts angibt. Die folgenden Anweisungen enthalten ein Beispiel für einen Themen-URI:

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

Das durch diese Anweisungen erstellte JmsTopic-Objekt stellt ein Thema mit dem Namen 'Sport/Tennis/Results' dar. Alle Nachrichten, die an dieses Ziel gesendet werden, sind nicht persistent und haben

die Priorität 0. Weitere Informationen zu Themen-URIs finden Sie unter „Uniform Resource Identifiers (URIs)” auf Seite 232. Eine alternative Methode für das Festlegen der Eigenschaften eines JmsTopic-Objekts ist im Abschnitt „Eigenschaften von Objekten der IBM MQ classes for JMS festlegen” auf Seite 219 beschrieben.

Nachdem eine Anwendung eine Verbindungsfactory oder ein Ziel erstellt hat, kann dieses Objekt nur mit dem ausgewählten Messaging-Provider verwendet werden.

Eigenschaften von Objekten der IBM MQ classes for JMS festlegen

Zur Festlegung der Eigenschaften von Objekten der IBM MQ classes for JMS mithilfe der IBM JMS-Erweiterungen verwendet eine Anwendung die Methoden der Schnittstelle com.ibm.msg.client.JmsPropertyContext. Zum Festlegen der Eigenschaften von IBM MQ classes for Jakarta Messaging -Objekten mit den IBM JMS -Erweiterungen verwendet eine Anwendung die Methoden der Schnittstelle com.ibm.msg.jakarta.client.JmsPropertyContext .

Die JmsPropertyContext-Schnittstelle enthält für jeden Java-Datentyp eine Methode zur Festlegung des Werts einer Eigenschaft mit dem betreffenden Datentyp sowie eine Methode für den Abruf des Werts einer Eigenschaft mit diesem Datentyp. Beispiel: Eine Anwendung ruft die Methode setIntProperty() für die Festlegung einer Eigenschaft mit einem Ganzzahlwert auf und für den Abruf einer Eigenschaft mit einem Ganzzahlwert ruft sie die Methode getIntProperty() auf.

Instanzen von Klassen in den Paketen com.ibm.mq.jms und com.ibm.mq.jakarta.jms übernehmen die Methoden der entsprechenden JmsProperty-Kontextschnittstellen. Daher kann eine Anwendung diese Methoden zum Festlegen der Eigenschaften von MQConnectionFactory-, MQQueue- und MQTopic-Objekten verwenden.

Wenn eine Anwendung ein IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekt erstellt, werden alle Eigenschaften mit Standardwerten automatisch festgelegt. Wenn eine Anwendung eine Eigenschaft festlegt, ersetzt der neue Wert jeden vorherigen Wert, der der Eigenschaft zugewiesen war. Eine Eigenschaft kann nach ihrer Festlegung zwar nicht gelöscht werden, ihr Wert kann jedoch geändert werden.

Wenn eine Anwendung versucht, eine Eigenschaft auf einen Wert zu setzen, der kein gültiger Wert für die Eigenschaft ist, löst IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging eine JMSException-Ausnahmebedingung aus. Falls eine Anwendung versucht, eine nicht festgelegte Eigenschaft abzurufen, entspricht das Verhalten dem in der JMS-Spezifikation beschriebenen Verhalten. IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging lösen eine Ausnahmebedingung NumberFormatfür primitive Datentypen aus und geben für referenzierte Datentypen null zurück.

Zusätzlich zu den vordefinierten Eigenschaften eines IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekts kann eine Anwendung eigene Eigenschaften festlegen. Diese anwendungsdefinierten Eigenschaften werden von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingignoriert.

Weitere Informationen zu den Eigenschaften von IBM MQ classes for JMS -und IBM MQ classes for Jakarta Messaging -Objekten finden Sie unter Eigenschaften von IBM MQ classes for JMS -Objekten.

Der folgende Code ist ein Beispiel für die Festlegung von Eigenschaften mithilfe der IBM JMS-Erweiterungen. Der Code legt fünf Eigenschaften einer Verbindungsfactory fest.

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
WMQConstants.WMQ_CM_CLIENT);
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

Die Festlegung dieser Eigenschaften bewirkt, dass sich die Anwendung im Clientmodus unter Verwendung eines MQI-Kanals mit dem Namen 'QM1.SVR' mit dem Warteschlangenmanager QM1 verbindet. Der Warteschlangenmanager wird auf einem System mit dem Hostnamen HOST1 ausgeführt und der Listener für den Warteschlangenmanager ist an der Portnummer 1415 empfangsbereit. Dieser Verbindung und

sonstigen Warteschlangenmanagerverbindungen, die den damit verbundenen Sitzungen zugeordnet sind, wird der Anwendungsname "My Application" (Meine Anwendung) zugewiesen.

Anmerkung: Da Warteschlangenmanager, die auf z/OS-Plattformen ausgeführt werden, die Festlegung von Anwendungsnamen nicht unterstützen, wird diese Einstellung ignoriert.

Die JmsPropertyContext-Schnittstelle enthält auch die setObjectProperty()-Methode, die eine Anwendung verwenden kann, um Eigenschaften festzulegen. Der zweite Parameter der Methode ist ein Objekt, das den Wert der Eigenschaft einbindet. Der folgende Code erstellt beispielsweise ein Integer-Objekt (Ganzzahlobjekt), das die Ganzzahl 1415 einbindet und anschließend setObjectProperty() aufruft, um die Eigenschaft PORT einer Verbindungsfactory auf den Wert 1415 zu setzen:

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

Daher ist dieser Code äquivalent zu folgender Anweisung:

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

Umgekehrt gibt die Methode getObjectProperty() ein Objekt zurück, das den Wert einer Eigenschaft einbindet.

Implizite Konvertierung eines Eigenschaftswerts von einem Datentyp in einen anderen Datentyp

Wenn eine Anwendung eine Methode der Kontextschnittstelle JmsPropertyverwendet, um die Eigenschaft eines IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekts festzulegen oder abzurufen, kann der Wert der Eigenschaft implizit von einem Datentyp in einen anderen konvertiert werden.

Die folgende Anweisung legt beispielsweise die Eigenschaft PRIORITY des JmsQueue-Objekts 'q1' fest:

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

Da die Eigenschaft PRIORITY einen Ganzzahlwert hat, konvertiert der Aufruf setStringProperty() implizit die Zeichenfolge "5" (den Quellenwert) in die Ganzzahl 5 (den Zielwert), der dann als der Wert der Eigenschaft PRIORITY verwendet wird.

Umgekehrt ruft die folgende Anweisung die Eigenschaft PRIORITY des JmsQueue-Objekts 'q1' ab:

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

Die Ganzzahl 5 (der Quellenwert), die der Wert der Eigenschaft PRIORITY ist, wird implizit vom Aufruf getStringProperty() in die Zeichenfolge "5" konvertiert.

Die von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging unterstützten Konvertierungen werden in Tabelle 34 auf Seite 220gezeigt.

Tabelle 34. Unterstützte Konvertierungen von einem Datentyp in einen anderen Datentyp

Quellendatentyp	**Unterstützte Zieldatentypen**
boolean	Zeichenfolge
Byte	int, long, short, String
char	Zeichenfolge
double	Zeichenfolge
float	double, String

Tabelle 34. Unterstützte Konvertierungen von einem Datentyp in einen anderen Datentyp (Forts.)	
Quellendatentyp	**Unterstützte Zieldatentypen**
int	long, String
lang	Zeichenfolge
short	int, long, String
Zeichenfolge	boolean, byte, double, float, int, long, short

Für die unterstützten Konvertierungen gelten folgende allgemeine Regeln:

- Numerische Werte können von einem Datentyp in einen anderen Datentyp konvertiert werden, vorausgesetzt, bei der Konvertierung gehen keine Daten verloren. So kann beispielsweise ein Wert mit dem Datentyp `int` in einen Wert mit dem Datentyp `long` konvertiert werden, es ist jedoch nicht möglich, ihn in einen Wert mit dem Datentyp `short` zu konvertieren.
- Ein Wert eines beliebigen Datentyps kann in eine Zeichenfolge konvertiert werden.
- Eine Zeichenfolge kann in einen Wert eines beliebigen anderen Datentyps (außer `char`) konvertiert werden, vorausgesetzt, die Zeichenfolge hat das richtige Format für die Konvertierung. Wenn eine Anwendung versucht, eine Zeichenfolge zu konvertieren, die nicht das richtige Format hat, lösen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging die Ausnahmebedingung NumberFormataus.
- Wenn eine Anwendung eine Konvertierung versucht, die nicht unterstützt wird, lösen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging eine Ausnahmebedingung des Typs MessageFormataus.

Es gibt die folgenden spezifischen Regeln für die Konvertierung eines Werts aus einem Datentyp in einen anderen Datentyp:

- Beim Konvertieren eines booleschen Werts in eine Zeichenfolge wird der Wert `true` in die Zeichenfolge "true" und der Wert `false` in die Zeichenfolge "false" konvertiert.
- Beim Konvertieren einer Zeichenfolge in einen booleschen Wert wird die Zeichenfolge "true" (die Groß-/Kleinschreibung muss nicht beachtet werden) in `true` konvertiert. Die Zeichenfolge "false" (die Groß-/Kleinschreibung muss nicht beachtet werden) wird in `false` konvertiert. Alle anderen Zeichenfolgen werden in `false` konvertiert.
- Beim Konvertieren einer Zeichenfolge in einen Wert mit dem Datentyp `byte`, `int`, `long` oder `short` muss die Zeichenfolge folgendes Format haben:

  [*blanks*][*sign*] *digits*

  Die Zeichenfolge hat folgende Komponenten:

  blanks
  Optionale führende Leerzeichen.

  sign
  Ein optionales Pluszeichen (+) oder Minuszeichen (-).

  Ziffern
  Eine zusammenhängende Folge von Ziffern (0-9). Mindestens eine Ziffer muss vorhanden sein.

  Hinter der Ziffernfolge kann die Zeichenfolge sonstige Zeichen enthalten, die keine Ziffern sind. Die Konvertierung wird jedoch gestoppt, sobald das erste dieser Zeichen erreicht wird. Es wird vorausgesetzt, dass die Zeichenfolge eine Ganzzahl im Dezimalformat darstellt.

  Wenn die Zeichenfolge nicht das richtige Format hat, lösen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging eine Ausnahme des Typs NumberFormataus.
- Beim Konvertieren einer Zeichenfolge in einen Wert mit dem Datentyp `double` oder `float` muss die Zeichenfolge folgendes Format haben:

  [*blanks*][*sign*] *digits* [*e_char* [*e_sign*] *e_digits*]

Die Zeichenfolge hat folgende Komponenten:

blanks
Optionale führende Leerzeichen.

sign
Ein optionales Pluszeichen (+) oder Minuszeichen (-).

Ziffern
Eine zusammenhängende Folge von Ziffern (0-9). Mindestens eine Ziffer muss vorhanden sein.

e_Zeich
Ein Exponentenzeichen, entweder *E* oder *e*.

e_Vorz
Ein optionales Pluszeichen (+) oder Minuszeichen (-) für den Exponenten.

e_Ziffern
Eine zusammenhängende Folge von Ziffern (0-9) für den Exponenten. Mindestens eine Ziffer muss vorhanden sein, wenn die Zeichenfolge ein Exponentenzeichen enthält.

Hinter der Ziffernfolge oder den optionalen Zeichen, die einen Exponenten darstellen, kann die Zeichenfolge sonstige Zeichen enthalten, die keine Ziffern sind. Die Konvertierung wird jedoch gestoppt, sobald das erste dieser Zeichen erreicht wird. Es wird davon ausgegangen, dass die Zeichenfolge eine Gleitkommazahl mit einem Exponenten der Potenz 10 darstellt.

Wenn die Zeichenfolge nicht das richtige Format hat, lösen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging eine Ausnahme des Typs NumberFormataus.

- Beim Konvertieren eines numerischen Werts (einschließlich eines Werts mit dem Datentyp `byte`) in eine Zeichenfolge wird der Wert in die Zeichenfolgedarstellung des Werts als Dezimalzahl konvertiert, nicht in die Zeichenfolge, die das ASCII-Zeichen für diesen Wert enthält. Die Ganzzahl 65 wird beispielsweise in die Zeichenfolge "65" konvertiert, nicht in die Zeichenfolge "A".

Mehr als eine Eigenschaft in einem einzelnen Aufruf festlegen

Die JmsPropertyContext-Schnittstelle enthält auch die Methode setBatchProperties(), die eine Anwendung verwenden kann, um mehr als eine Eigenschaft in einem einzelnen Aufruf festzulegen. Der Parameter der Methode ist ein Map-Objekt, das eine Gruppe von Name/Wert-Paaren für Eigenschaften einbindet.

Der folgende Code verwendet beispielsweise die Methode setBatchProperties(), um wie in „Eigenschaften von Objekten der IBM MQ classes for JMS festlegen” auf Seite 219 dargestellt dieselben fünf Eigenschaften einer Verbindungsfactory festzulegen. Der Code erstellt eine Instanz der HashMap-Klasse, die die Map-Schnittstelle implementiert.

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

Beachten Sie, dass der zweite Parameter der Map.put()-Methode ein Objekt sein muss. Daher muss ein Eigenschaftswert mit einem primitiven Datentyp innerhalb eines Objekts eingebunden oder durch eine Zeichenfolge dargestellt werden (siehe Beispiel).

Die Methode setBatchProperties() überprüft jede Eigenschaft. Wenn die Methode setBatchProperties() eine Eigenschaft nicht festlegen kann, da ihr Wert beispielsweise nicht gültig ist, kann keine der angegebenen Eigenschaften festgelegt werden.

Eigenschaftsnamen und Werte

Wenn eine Anwendung die Methoden der entsprechenden JmsProperty-Kontextschnittstelle verwendet, um die Eigenschaften von IBM MQ classes for JMS- oder IBM MQ classes for Jakarta Messaging -Objekten

festzulegen und abzurufen, kann die Anwendung die Namen und Werte von Eigenschaften auf eine der folgenden Arten angeben: Jedes der zugehörigen Beispiele zeigt, wie die Eigenschaft PRIORITY des JmsQueue-Objekts 'q1' so festgelegt wird, dass eine an die Warteschlange gesendete Nachricht die Priorität hat, die im Aufruf send() angegeben ist.

Verwendung der Eigenschaftsnamen und -werte, die als Konstanten in der Schnittstelle com.ibm.msg.client.wmq.WMQConstants definiert sind

Die folgende Anweisung ist ein Beispiel dafür, wie die Namen und Werte von Eigenschaften auf diese Weise angegeben werden:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

Verwendung der Eigenschaftsnamen und -werte, die in Uniform Resource Identifiers (URIs) von Warteschlangen und Themen verwendet werden können

Die folgende Anweisung ist ein Beispiel dafür, wie die Namen und Werte von Eigenschaften auf diese Weise angegeben werden:

```
q1.setIntProperty("priority", -2);
```

Nur die Namen und Werte der Eigenschaften von Zielen können auf diese Art und Weise angegeben werden.

Verwendung der Eigenschaftsnamen und -werte, die vom IBM MQ-JMS-Verwaltungstool erkannt werden

Die folgende Anweisung ist ein Beispiel dafür, wie die Namen und Werte von Eigenschaften auf diese Weise angegeben werden:

```
q1.setStringProperty("PRIORITY", "APP");
```

Die Kurzform des Eigenschaftsnamens, die in der folgenden Anweisung zu sehen ist, ist ebenfalls zulässig:

```
q1.setStringProperty("PRI", "APP");
```

Wenn eine Anwendung eine Eigenschaft abruft, hängt der zurückgegebene Wert von der Art und Weise ab, in der die Anwendung den Namen der Eigenschaft angibt. Wenn eine Anwendung beispielsweise die Konstante WMQConstants.WMQ_PRIORITY als Eigenschaftsnamen angibt, wird als Wert die Ganzzahl -2 zurückgegeben:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

Derselbe Wert wird zurückgegeben, wenn die Anwendung die Zeichenfolge "priority" als Eigenschaftsnamen angibt:

```
int n2 = getIntProperty("priority");
```

Wenn die Anwendung jedoch die Zeichenfolge "PRIORITY" oder "PRI" als Eigenschaftsnamen angibt, wird als Wert die Zeichenfolge "APP" zurückgegeben:

```
String s1 = getStringProperty("PRI");
```

Intern speichern IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging Eigenschaftsnamen und -werte als Literalwerte, die in der übereinstimmenden WMQConstants-Schnittstelle definiert sind. Dies ist das definierte kanonische Format für Eigenschaftsnamen und -werte. Wenn eine Anwendung Eigenschaften unter Verwendung einer der beiden anderen Methoden zur Angabe von Eigenschaftsnamen und -werten festlegt, müssen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging die Namen und Werte aus dem angegebenen Eingabeformat in das kanonische Format konvertieren.

Wenn eine Anwendung Eigenschaften mit einer der beiden anderen Methoden zur Angabe von Eigenschaftsnamen und -werten abruft, müssen IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging die Namen aus dem angegebenen Eingabeformat in das kanonische Format und die Werte aus dem kanonischen Format in das erforderliche Ausgabeformat konvertieren. Wenn diese Konvertierungen durchgeführt werden müssen, kann sich das auf die Leistung auswirken.

Eigenschaftsnamen und -werte, die von Ausnahmen, in Tracedateien oder im IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Protokoll zurückgegeben werden, haben immer das kanonische Format.

Map-Schnittstelle verwenden

Die JmsPropertyContext-Schnittstelle ist eine Erweiterung der java.util.Map-Schnittstelle. Eine Anwendung kann daher die Methoden der Schnittstelle "Map" verwenden, um auf die Eigenschaften eines IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekts zuzugreifen.

Der folgende Code gibt beispielsweise die Namen und Werte von allen Eigenschaften einer Verbindungsfactory aus. Der Code verwendet nur die Methoden der Map-Schnittstelle, um die Namen und Werte der Eigenschaften abzurufen.

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

Prüfungen oder Konvertierungen von Eigenschaften können nicht mit den Methoden der Map-Schnittstelle umgangen werden.

Verwendung der IBM MQ JMS-Erweiterungen
IBM MQ classes for JMS enthalten eine Gruppe von Erweiterungen für die JMS-API. Diese werden als 'IBM MQ JMS-Erweiterungen' bezeichnet. Eine Anwendung kann mithilfe dieser Erweiterungen dynamisch zur Laufzeit Verbindungsfactorys und Ziele erstellen sowie die Eigenschaften der Verbindungsfactorys und Ziele festlegen.

IBM MQ classes for JMS enthalten eine Gruppe von Klassen in den Paketen 'com.ibm.jms' und 'com.ibm.mq.jms'. Diese Klassen implementieren die JMS-Schnittstellen und enthalten die IBM MQ-JMS-Schnittstellen. Bei den nachfolgenden Codebeispielen wird vorausgesetzt, dass diese Pakete mit den folgenden Anweisungen bereits importiert wurden:

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Eine Anwendung kann die IBM MQ-JMS-Erweiterungen verwenden, um die folgenden Funktionen auszuführen:

- Dynamische Erstellung von Verbindungsfactorys und Zielen, statt sie als verwaltete Objekte aus einem JNDI-Namensbereich (JNDI = Java Naming and Directory Interface) abzurufen
- Festlegung der Eigenschaften von Verbindungsfactorys und Zielen

Verbindungsfactorys erstellen

Um eine Verbindungsfactory zu erstellen, kann eine Anwendung den MQConnectionFactory-Konstruktor verwenden, wie im folgenden Beispiel dargestellt:

```
MQConnectionFactory factory = new MQConnectionFactory();
```

Diese Anweisung erstellt ein MQConnectionFactory-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Dies bedeutet, dass sich die Anwendung im Bindungsmodus mit dem Standardwarteschlangenmanager verbindet. Wenn Sie möchten, dass sich eine Anwendung im Clientmodus verbindet oder eine Verbindung zu einem anderen Warteschlangenmanager als dem Standardwarteschlangenmanager herstellt, muss die Anwendung die entsprechenden Eigenschaften des MQConnectionFactory-Objekts festlegen, bevor die Verbindung erstellt wird. Weitere Informationen hierzu finden Sie im Abschnitt „Eigenschaften von Verbindungsfactorys festlegen” auf Seite 225.

Eine Anwendung kann Verbindungsfactorys der folgenden Typen auf ähnliche Weise erstellen:

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

Eigenschaften von Verbindungsfactorys festlegen

Eine Anwendung kann die Eigenschaften einer Verbindungsfactory durch das Aufrufen der entsprechenden Methoden der Verbindungsfactory festlegen. Bei der Verbindungsfactory kann es sich entweder um ein verwaltetes Objekt oder um ein Objekt handeln, das dynamisch zur Laufzeit erstellt wurde.

Betrachten Sie beispielsweise den folgenden Code:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

Dieser Code erstellt ein MQConnectionFactory-Objekt und legt dann fünf Eigenschaften des Objekts fest. Die Festlegung dieser Eigenschaften bewirkt, dass sich die Anwendung im Clientmodus unter Verwendung eines MQI-Kanals mit dem Namen 'QM1.SVR' mit dem Warteschlangenmanager QM1 verbindet. Der Warteschlangenmanager wird auf einem System mit dem Hostnamen HOST1 ausgeführt und der Listener für den Warteschlangenmanager ist an der Portnummer 1415 empfangsbereit.

Eine Anwendung, die eine Echtzeitverbindung mit einem Broker verwendet, kann nur den Publish/Subscribe-Messaging-Stil verwenden. Sie kann nicht den Punkt-zu-Punkt-Stil für das Messaging verwenden.

Nur bestimmte Kombinationen aus Eigenschaften einer Verbindungsfactory sind gültig. Informationen zu den gültigen Kombinationen finden Sie im Abschnitt Abhängigkeiten zwischen Eigenschaften von Objekten der IBM MQ classes for JMS.

Weitere Informationen zu den Eigenschaften einer Verbindungsfactory und den Methoden, die für deren Festlegung verwendet werden, finden Sie im Abschnitt Eigenschaften von IBM MQ classes for JMS-Objekten.

Ziele erstellen

Um ein Queue-Objekt zu erstellen, kann eine Anwendung den MQQueue-Konstruktor verwenden, wie im folgenden Beispiel dargestellt:

```
MQQueue q1 = new MQQueue("Q1");
```

Diese Anweisung erstellt ein MQQueue-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Das Objekt stellt eine IBM MQ-Warteschlange mit dem Namen Q1 dar, die dem lokalen Warteschlangenmanager gehört. Bei dieser Warteschlange kann es sich um eine lokale Warteschlange, um eine Aliaswarteschlange oder um eine Definition einer fernen Warteschlange handeln.

Eine alternative Form des MQQueue-Konstruktors hat zwei Parameter, wie im folgenden Beispiel dargestellt:

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

Das durch diese Anweisung erstellte MQQueue-Objekt stellt eine IBM MQ-Warteschlange mit dem Namen Q2 dar. Der Eigner dieser Warteschlange ist der Warteschlangenmanager QM2. Bei dem Warteschlangenmanager, der auf diese Weise angegeben wird, kann es sich um den lokalen Warteschlangenmanager oder um einen fernen Warteschlangenmanager handeln. Wenn es sich um einen fernen Warteschlangenmanager handelt, muss IBM MQ so konfiguriert sein, dass WebSphere MQ die Nachricht vom lokalen Warteschlangenmanager an den fernen Warteschlangenmanager weiterleiten kann, wenn die Anwendung eine Nachricht an dieses Ziel sendet.

Der MQQueue-Konstruktor kann auch einen Warteschlangen-URI (URI = Uniform Resource Identifier) als einzelnen Parameter akzeptieren. Ein Warteschlangen-URI ist eine Zeichenfolge, die den Namen einer IBM MQ-Warteschlange und optional den Namen des Warteschlangenmanagers angibt, der Eigner der Warteschlange ist, sowie eine oder mehrere Eigenschaften des MQQueue-Objekts. Die folgende Anweisung enthält ein Beispiel für einen Warteschlangen-URI:

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

Das durch diese Anweisung erstellte MQQueue-Objekt stellt eine IBM MQ-Warteschlange mit dem Namen Q3 dar. Der Eigner dieser Warteschlange ist der Warteschlangenmanager QM3; alle Nachrichten, die an dieses Ziel gesendet werden, sind persistent und haben die Priorität 5. Sie finden weitere Informationen zu Warteschlangen-URIs im Abschnitt „Uniform Resource Identifiers (URIs)” auf Seite 232. Eine alternative Methode für das Festlegen der Eigenschaften eines MQQueue-Objekts ist im Abschnitt „Eigenschaften von Zielen festlegen” auf Seite 226 beschrieben.

Um ein Topic-Objekt zu erstellen, kann eine Anwendung den MQTopic-Konstruktor verwenden, wie im folgenden Beispiel dargestellt:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

Diese Anweisung erstellt ein MQTopic-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Das Objekt stellt ein Thema mit dem Namen Sport/Football/Results dar.

Der MQTopic-Konstruktor kann auch einen Themen-URI als Parameter akzeptieren. Ein Themen-URI ist eine Zeichenfolge, die den Namen eines Themas und optional eine oder mehrere Eigenschaften des MQTopic-Objekts angibt. Die folgende Anweisung enthält ein Beispiel für einen Themen-URI:

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

Das durch diese Anweisung erstellte MQTopic-Objekt stellt ein Thema mit dem Namen 'Sport/Tennis/Results' dar. Alle Nachrichten, die an dieses Ziel gesendet werden, sind nicht persistent und haben die Priorität 0. Weitere Informationen zu Themen-URIs finden Sie unter „Uniform Resource Identifiers (URIs)” auf Seite 232. Eine alternative Methode für das Festlegen der Eigenschaften eines MQTopic-Objekts ist im Abschnitt „Eigenschaften von Zielen festlegen” auf Seite 226 beschrieben.

Eigenschaften von Zielen festlegen

Eine Anwendung kann die Eigenschaften eines Zieles durch das Aufrufen der entsprechenden Methoden des Ziels festlegen. Bei dem Ziel kann es sich entweder um ein verwaltetes Objekt oder um ein Objekt handeln, das dynamisch zur Laufzeit erstellt wurde.

Betrachten Sie beispielsweise den folgenden Code:

```
MQQueue q1 = new MQQueue("Q1");
.
```

```
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

Dieser Code erstellt ein MQQueue-Objekt und legt dann zwei Eigenschaften des Objekts fest. Die Festlegung dieser Eigenschaften bewirkt, dass alle Nachrichten, die an das Ziel gesendet werden, persistent sind und die Priorität 5 haben.

Eine Anwendung kann die Eigenschaften des MQTopic-Objekts wie im folgenden Beispiel dargestellt auf ähnliche Weise festlegen:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

Dieser Code erstellt ein MQTopic-Objekt und legt dann zwei Eigenschaften des Objekts fest. Die Festlegung dieser Eigenschaften bewirkt, dass alle Nachrichten, die an das Ziel gesendet werden, nicht persistent sind und die Priorität 0 haben.

Weitere Informationen zu den Eigenschaften eines Ziels und den Methoden, die für deren Festlegung verwendet werden, finden Sie im Abschnitt Eigenschaften von IBM MQ classes for JMS-Objekten.

Linux AIX *Verbindung zu IBM MQ über eine JMS -Anwendung herstellen*

Zum Erstellen einer Verbindung verwendet eine JMS -Anwendung ein **ConnectionFactory** -Objekt, um ein **Connection** -Objekt zu erstellen, und startet dann die Verbindung.

Für JMS 2.0 und höher stellen Anwendungen normalerweise eine Verbindung zu einem Messaging-Provider über ein **ConnectionFactory** -Objekt und die Methode `createContext()` her.

In früheren JMS-Versionen mussten Sie zuerst mit `createConnection` ein **Connection** -Objekt erstellen und dann den Verbindungsaufruf `getSession()` starten, um ein **Session** -Objekt zu erstellen, das Messaging-Operationen ausführen konnte.

Ein **JMSContext** -Objekt kapselt effektiv sowohl die **Connection** -als auch die **Session** -Objekte. Wenn Sie die traditionelle Methode verwenden möchten und die Verbindungs-und Sitzungsobjekte direkt erstellen wollen, lesen Sie die Informationen unter „Verbindung in einer JMS-Anwendung erstellen” auf Seite 227 und „Sitzung in einer JMS-Anwendung erstellen” auf Seite 228.

Zum Erstellen eines **JMSContext** -Objekts verwendet eine Anwendung die Methode `createContext()` eines **ConnectionFactory** -Objekts, wie im folgenden Beispiel gezeigt:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createContext();
```

Wenn eine JMS -Verbindung erstellt wird, erstellt der IBM MQ classes for JMS eine Verbindungskennung (Hconn) und startet einen Dialog mit dem Warteschlangenmanager.

Anmerkung: Beachten Sie, dass die Anwendungsprozess-ID als Standardbenutzer-ID verwendet wird, die an den Warteschlangenmanager übergeben wird. Wenn die Anwendung im Clienttransportmodus ausgeführt wird, muss diese Prozess-ID mit den relevanten Autorisierungen auf dem Server vorhanden sein. Wenn Sie eine andere Identität verwenden möchten, verwenden Sie die Methode `createConnection(username,password)` .

V 9.4.0 Dieser Mechanismus kann auch verwendet werden, um ein Authentifizierungstoken bereitzustellen (siehe Authentifizierungstoken vom ausgewählten Tokenaussteller abrufen).

JMS 1.0 *Verbindung in einer JMS-Anwendung erstellen*

Zum Erstellen einer Verbindung in JMS 1.0verwendet eine JMS -Anwendung ein ConnectionFactory -Objekt, um ein Connection-Objekt zu erstellen, und startet dann die Verbindung.

Um ein Connection-Objekt zu erstellen, verwendet eine Anwendung die Methode createConnection() eines ConnectionFactory-Objekts, wie im folgenden Beispiel dargestellt:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

Wenn eine JMS-Verbindung erstellt wird, erstellen die IBM MQ classes for JMS eine Verbindungskennung (Hconn) und starten einen Dialog mit dem Warteschlangenmanager.

Die QueueConnectionFactory-Schnittstelle und die TopicConnectionFactory-Schnittstelle übernehmen jeweils die Methode createConnection() aus der ConnectionFactory-Schnittstelle. Daher können Sie mit der Methode createConnection() ein domänenspezifisches Objekt erstellen, wie im folgenden Beispiel dargestellt:

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

Dieses Codefragment erstellt ein QueueConnection-Objekt. Eine Anwendung kann jetzt eine domänenunabhängige Operation für dieses Objekt ausführen oder eine Operation, die nur für die Punkt-zu-Punkt-Domäne gilt. Wenn die Anwendung jedoch versucht, eine Operation auszuführen, die nur für die Publish/Subscribe-Domäne gilt, wird die Ausnahmebedingung 'IllegalStateException' mit der folgenden Nachricht ausgelöst:

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

Der Grund hierfür ist, dass die Verbindung über eine domänenspezifische Verbindungsfactory erstellt wurde.

Anmerkung: Beachten Sie, dass die Anwendungsprozess-ID als Standardbenutzer-ID verwendet wird, die an den Warteschlangenmanager übergeben wird. Wenn die Anwendung im Clienttransportmodus ausgeführt wird, muss diese Prozess-ID mit den relevanten Autorisierungen auf dem Server vorhanden sein. Wenn Sie möchten, dass eine andere Identität verwendet wird, verwenden Sie die Methode 'createConnection(Benutzername, Kennwort)'.

Die JMS-Spezifikation gibt an, dass eine Verbindung im Status `stopped` erstellt wird. Ein Nachrichtenkonsument, der der Verbindung zugeordnet ist, kann erst dann Nachrichten empfangen, wenn die Verbindung gestartet wurde. Um eine Verbindung zu starten, verwendet eine Anwendung die Methode start() eines Connection-Objekts, wie im folgenden Beispiel dargestellt:

```
connection.start();
```

V 9.4.0 Dieser Mechanismus kann auch verwendet werden, um ein Authentifizierungstoken bereitzustellen (siehe Authentifizierungstoken vom ausgewählten Tokenaussteller abrufen).

JMS 1.0 *Sitzung in einer JMS-Anwendung erstellen*

Zum Erstellen einer Sitzung in JMS 1.0verwendet eine JMS -Anwendung die Methode createSession() eines Connection-Objekts.

Die Methode createSession() hat zwei Parameter:

1. Einen Parameter, der angibt, ob die Sitzung transaktionsbasiert oder nicht transaktionsbasiert ist
2. Einen Parameter, der den Bestätigungsmodus für die Sitzung angibt

Der folgende Code erstellt beispielsweise eine Sitzung, die nicht transaktionsbasiert ist und den Bestätigungsmodus AUTO_ACKNOWLEDGE aufweist:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Wenn eine JMS-Sitzung erstellt wird, erstellen die IBM MQ classes for JMS eine Verbindungskennung (Hconn) und starten einen Dialog mit dem Warteschlangenmanager.

Ein Session-Objekt (Sitzungsobjekt) und alle daraus erstellten MessageProducer- oder MessageConsumer-Objekte können nicht gleichzeitig von verschiedenen Threads einer Multithread-Anwendung verwendet werden. Die einfachste Methode, um sicherzustellen, dass diese Objekte nicht gleichzeitig verwendet werden, ist die Erstellung eines separaten Session-Objekts für jeden Thread.

V 9.4.0 Dieser Mechanismus kann auch verwendet werden, um ein Authentifizierungstoken bereitzustellen (siehe Authentifizierungstoken vom ausgewählten Tokenaussteller abrufen).

Transaktionsbasierte Sitzungen in JMS-Anwendungen
JMS-Anwendungen können lokale Transaktionen ausführen, indem sie zuerst eine Sitzung mit Transaktionsunterstützung, eine so genannte transaktionsbasierte Sitzung erstellen. Eine Anwendung kann eine Transaktion festschreiben oder rückgängig machen.

JMS-Anwendungen können lokale Transaktionen ausführen. Eine lokale Transaktion ist eine Transaktion, die Änderungen umfasst, welche nur die Ressourcen des Warteschlangenmanagers betreffen, mit dem die Anwendung verbunden ist. Zur Ausführung von lokalen Transaktionen muss eine Anwendung zunächst eine transaktionsbasierte Sitzung erstellen. Hierfür ruft sie die Methode createSession()eines Connection-Objekts auf und gibt als Parameter an, dass die Sitzung transaktionsbasiert ist. Danach werden alle innerhalb der Sitzung gesendeten und empfangenen Nachrichten in einer Folge von Transaktionen gruppiert. Eine Transaktion endet, wenn die Anwendung die Nachrichten, die sie seit Beginn der Transaktion gesendet oder empfangen hat, festschreibt oder rückgängig macht.

Zur Festschreibung einer Transaktion ruft eine Anwendung die Methode commit() des Session-Objekts auf. Wenn eine Transaktion festgeschrieben wird, werden alle Nachrichten, die innerhalb der Transaktion gesendet wurden, für die Zustellung an andere Anwendungen verfügbar. Außerdem werden sämtliche Nachrichten, die innerhalb der Transaktion empfangen wurden, bestätigt, damit der Messaging-Server nicht erneut versucht, diese an die Anwendung zuzustellen. In der Punkt-zu-Punkt-Domäne entfernt der Messaging-Server außerdem die empfangenen Nachrichten aus ihren Warteschlangen.

Um eine Transaktion rückgängig zu machen, ruft eine Anwendung die Methode rollback() des Session-Objekts auf. Wenn eine Transaktion rückgängig gemacht wird, werden alle Nachrichten, die innerhalb der Transaktion gesendet wurden, vom Messaging-Server gelöscht und alle Nachrichten, die innerhalb der Transaktion empfangen wurden, sind erneut für die Zustellung verfügbar. In der Punkt-zu-Punkt-Domäne werden die empfangenen Nachrichten zurück in ihre Warteschlangen gestellt und sind somit wieder für andere Anwendungen sichtbar.

Wenn eine Anwendung eine transaktionsbasierte Sitzung erstellt oder die Methode commit() bzw. rollback() aufruft, wird automatisch eine neue Transaktion gestartet. Daher verfügt eine transaktionsbasierte Sitzung immer über eine aktive Transaktion.

Wenn eine Anwendung eine transaktionsbasierte Sitzung schließt, erfolgt ein impliziter Rollback. Sobald eine Anwendung eine Verbindung schließt, wird für alle transaktionsbasierten Sitzungen der Verbindung ein impliziter Rollback durchgeführt.

Wenn eine Anwendung ohne Schließen einer Verbindung beendet wird, wird ebenfalls für alle transaktionsbasierten Sitzungen der Verbindung ein impliziter Rollback durchgeführt.

Eine Transaktion ist voll und ganz in einer transaktionsbasierten Sitzung enthalten. Eine Transaktion kann sich nicht über verschiedene Sitzungen erstrecken. Dies bedeutet, dass es einer Anwendung nicht möglich ist, Nachrichten in zwei oder mehr transaktionsbasierten Sitzungen zu senden und zu empfangen und alle diese Aktionen danach als einzelne Transaktion festzuschreiben oder rückgängig zu machen.

Bestätigungsmodi von JMS-Sitzungen
Jede Sitzung, die nicht transaktionsbasiert ist, verfügt über einen Bestätigungsmodus, der festlegt, wie die von der Anwendung empfangenen Nachrichten bestätigt werden. Drei Bestätigungsmodi sind verfügbar und die Auswahl des Bestätigungsmodus wirkt sich auf das Design der Anwendung aus.

Wenn eine Sitzung nicht transaktionsbasiert ist, wird die Art des Nachrichtenempfangs durch eine Anwendung durch den Bestätigungsmodus der Sitzung bestimmt. Die drei Bestätigungsmodi werden in den folgenden Absätzen beschrieben:

AUTO_ACKNOWLEDGE

Die Sitzung bestätigt automatisch jede Nachricht, die von der Anwendung empfangen wird.

Wenn Nachrichten synchron an die Anwendung übermittelt werden, bestätigt die Sitzung den Empfang einer Nachricht jedes Mal, wenn ein Receive-Aufruf erfolgreich abgeschlossen wurde. Wenn Nachrichten asynchron an die Anwendung übermittelt werden, bestätigt die Sitzung den Empfang einer Nachricht jedes Mal, wenn ein Aufruf der Methode onMessage() eines Nachrichtenlisteners erfolgreich abgeschlossen wurde.

Wenn die Anwendung eine Nachricht erfolgreich empfängt, ein Fehler jedoch die Bestätigung verhindert, wird die Nachricht wieder für die Übermittlung verfügbar. Daher muss die Anwendung in der Lage sein, eine erneut übermittelte Nachricht zu verarbeiten.

DUPS_OK_ACKNOWLEDGE

Die Sitzung bestätigt den Empfang von Nachrichten durch die Anwendung zu bestimmten Zeiten, die sie auswählt.

Die Verwendung dieses Bestätigungsmodus verringert den Arbeitsaufwand für die Sitzung. Allerdings kann ein Fehler, der die Nachrichtenbestätigung verhindert, dazu führen, dass mehr als eine Nachricht wieder für die Übermittlung verfügbar wird. Daher muss die Anwendung in der Lage sein, erneut übermittelte Nachrichten zu verarbeiten.

Einschränkung: In den Modi AUTO_ACKNOWLEDGE und DUPS_OK_ACKNOWLEDGE unterstützt JMS keine Anwendung, die eine nicht behandelte Ausnahmebedingung in einem Nachrichtenlistener auslöst. Dies bedeutet, dass Nachrichten immer bestätigt werden, wenn der der Nachrichtenlistener zurückkehrt, und zwar unabhängig davon, ob die Verarbeitung erfolgreich war (vorausgesetzt, die aufgetretenen Fehler sind nicht schwerwiegend und verhindern dadurch das Fortsetzen der Anwendung). Wenn Sie eine genauere Kontrolle über die Nachrichtenbestätigung benötigen, verwenden Sie CLIENT_ACKNOWLEDGE oder transaktionsbasierte Modi, die der Anwendung einen uneingeschränkten Zugriff auf die Bestätigungsfunktionen ermöglichen.

CLIENT_ACKNOWLEDGE

Die Anwendung bestätigt die empfangenen Nachrichten, indem sie die Acknowledge-Methode (Bestätigungsmethode) der Message-Klasse (Nachrichtenklasse) aufruft.

Die Anwendung kann den Empfang jeder Nachricht einzeln bestätigen oder einen Nachrichtenstapel empfangen und die Acknowledge-Methode nur für die zuletzt empfangene Nachricht aufrufen. Wenn die Acknowledge-Methode aufgerufen wird, werden alle Nachrichten bestätigt, die seit dem letzten Aufruf der Methode empfangen wurden.

Eine Anwendung kann in Verbindung mit einer dieser Bestätigungsmodi die Nachrichtenübermittlung in einer Sitzung stoppen und erneut starten, indem sie die Recover-Methode der Session-Klasse aufruft. Nachrichten, die empfangen, aber zuvor noch nicht bestätigt wurden, werden erneut übermittelt. Sie werden jedoch möglicherweise nicht in der Reihenfolge übermittelt, in der sie zuvor übermittelt wurden. Es kann vorkommen, dass zwischenzeitlich Nachrichten mit einer höheren Priorität eingetroffen und manche der ursprünglichen Nachrichten abgelaufen sind. In der Punkt-zu-Punkt-Domäne wurden einige der ursprünglichen Nachrichten möglicherweise bereits von einer anderen Anwendung verarbeitet.

Eine Anwendung kann feststellen, ob eine Nachricht erneut übermittelt wird, indem sie den Inhalt des Headerfelds 'JMSRedelivered' der Nachricht überprüft. Hierfür ruft die Anwendung die Methode 'getJMSRedelivered()' der Message-Klasse auf.

Ziele in einer JMS-Anwendung erstellen

Anstelle des Abrufs von Zielen als verwaltete Objekte aus einem JNDI-Namensbereich (JNDI = Java Naming and Directory Interface) kann eine JMS-Anwendung eine Sitzung für die dynamische Erstellung von Zielen zur Laufzeit verwenden. Eine Anwendung kann einen Uniform Resource Identifier (URI) zur Identifizierung einer Warteschlange oder eines Themas von IBM MQ und optional zur Angabe einer oder mehrerer Eigenschaften eines Queue- oder Topic-Objekts verwenden.

Sitzung für die Erstellung von Queue-Objekten verwenden

Um ein Queue-Objekt zu erstellen, kann eine Anwendung die Methode createQueue() eines Session-Objekts verwenden, wie im folgenden Beispiel dargestellt:

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

Dieser Code erstellt ein Queue-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Das Objekt stellt eine IBM MQ-Warteschlange mit dem Namen Q1 dar, die dem lokalen Warteschlangenmanager gehört. Bei dieser Warteschlange kann es sich um eine lokale Warteschlange, um eine Aliaswarteschlange oder um eine Definition einer fernen Warteschlange handeln.

Die Methode createQueue() akzeptiert auch einen Warteschlangen-URI als Parameter. Ein Warteschlangen-URI ist eine Zeichenfolge, die den Namen einer IBM MQ-Warteschlange und optional den Namen des Warteschlangenmanagers angibt, der Eigner der Warteschlange ist, sowie eine oder mehrere Eigenschaften des Queue-Objekts. Die folgende Anweisung enthält ein Beispiel für einen Warteschlangen-URI:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

Das durch diese Anweisung erstellte Queue-Objekt stellt eine IBM MQ-Warteschlange mit dem Namen Q2 dar. Der Eigner dieser Warteschlange ist der Warteschlangenmanager QM2; alle Nachrichten, die an dieses Ziel gesendet werden, sind persistent und haben die Priorität 5. Bei dem Warteschlangenmanager, der auf diese Weise angegeben wird, kann es sich um den lokalen Warteschlangenmanager oder um einen fernen Warteschlangenmanager handeln. Wenn es sich um einen fernen Warteschlangenmanager handelt, muss IBM MQ so konfiguriert sein, dass WebSphere MQ die Nachricht vom lokalen Warteschlangenmanager an den Warteschlangenmanager QM2 weiterleiten kann, wenn die Anwendung eine Nachricht an dieses Ziel sendet. Weitere Informationen zu URIs finden Sie im Abschnitt „Uniform Resource Identifiers (URIs)” auf Seite 232.

Beachten Sie, dass der Parameter in der createQueue()-Methode providerspezifische Informationen enthält. Daher ist die Portabilität Ihrer Anwendung möglicherweise eingeschränkt, wenn Sie zur Erstellung eines Queue-Objekts die Methode createQueue() verwenden, statt ein Queue-Objekt als verwaltetes Objekts aus einem JNDI-Namensbereich abzurufen.

Eine Anwendung kann ein TemporaryQueue-Objekt mithilfe der Methode createTemporaryQueue() eines Session-Objekts erstellen, wie im folgenden Beispiel dargestellt:

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

Auch wenn eine Sitzung zur Erstellung einer temporären Warteschlange verwendet wird, entspricht der Bereich einer temporären Warteschlange der Verbindung, mit der die Sitzung erstellt wurde. Alle Sitzungen der Verbindung können Nachrichtenproduzenten und Nachrichtenkonsumenten für die temporäre Warteschlange erstellen. Die temporäre Warteschlange bleibt erhalten, bis die Verbindung beendet wird oder die Anwendung die temporäre Warteschlange explizit mit der Methode TemporaryQueue.delete() löscht, je nachdem, welches Ereignis früher eintritt.

Wenn eine Anwendung eine temporäre Warteschlange erstellt, erstellen IBM MQ classes for JMS eine dynamische Warteschlange in dem Warteschlangenmanager, mit dem die Anwendung verbunden wird. Die Eigenschaft TEMPMODEL der Verbindungsfactory gibt den Namen der Modellwarteschlange an, mit der

die dynamische Warteschlange erstellt wird, und die Eigenschaft TEMPQPREFIX der Verbindungsfactory gibt das Präfix an, mit dem der Name der dynamischen Warteschlange gebildet wird.

Sitzung für die Erstellung von Topic-Objekten verwenden

Um ein Topic-Objekt zu erstellen, kann eine Anwendung die Methode createTopic() eines Session-Objekts verwenden, wie im folgenden Beispiel dargestellt:

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

Dieser Code erstellt ein Topic-Objekt mit den Standardwerten für alle zugehörigen Eigenschaften. Das Objekt stellt ein Thema mit dem Namen Sport/Football/Results dar.

Die Methode createTopic() akzeptiert auch einen Themen-URI als Parameter. Ein Themen-URI ist eine Zeichenfolge, die den Namen eines Themas und optional eine oder mehrere Eigenschaften des Topic-Objekts angibt. Der folgende Code enthält ein Beispiel für einen Themen-URI:

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

Das durch diesen Code erstellte Topic-Objekt stellt ein Thema mit dem Namen 'Sport/Tennis/Results' dar. Alle Nachrichten, die an dieses Ziel gesendet werden, sind nicht persistent und haben die Priorität 0. Weitere Informationen zu Themen-URIs finden Sie unter „Uniform Resource Identifiers (URIs)” auf Seite 232.

Beachten Sie, dass der Parameter in der createTopic()-Methode providerspezifische Informationen enthält. Daher ist die Portabilität Ihrer Anwendung möglicherweise eingeschränkt, wenn Sie zur Erstellung eines Topic-Objekts die Methode createTopic() verwenden, statt ein Topic-Objekt als verwaltetes Objekts aus einem JNDI-Namensbereich abzurufen.

Eine Anwendung kann ein TemporaryTopic-Objekt mithilfe der Methode createTemporaryTopic() eines Session-Objekts erstellen, wie im folgenden Beispiel dargestellt:

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

Auch wenn eine Sitzung zur Erstellung eines temporären Themas verwendet wird, entspricht der Bereich eines temporären Themas der Verbindung, mit der die Sitzung erstellt wurde. Alle Sitzungen der Verbindung können Nachrichtenproduzenten und Nachrichtenkonsumenten für das temporäre Thema erstellen. Das temporäre Thema bleibt erhalten, bis die Verbindung beendet wird oder die Anwendung das temporäre Thema explizit mit der Methode TemporaryTopic.delete() löscht, je nachdem, welches Ereignis früher eintritt.

Wenn eine Anwendung ein temporäres Thema erstellt, erstellt IBM MQ classes for JMS ein Thema mit einem Namen, der mit den Zeichen TEMP/*tempTopicPräfix*beginnt, wobei *tempTopicPräfix* der Wert der Eigenschaft TEMPTOPICPREFIX der Verbindungsfactory ist.

Uniform Resource Identifiers (URIs)

Ein Warteschlangen-URI ist eine Zeichenfolge, die den Namen einer IBM MQ-Warteschlange und optional den Namen des Warteschlangenmanagers angibt, der Eigner der Warteschlange ist, sowie eine oder mehrere Eigenschaften des von der Anwendung erstellten Queue-Objekts. Ein Themen-URI ist eine Zeichenfolge, die den Namen eines Themas und optional eine oder mehrere Eigenschaften des von der Anwendung erstellten Topic-Objekts angibt.

Ein Warteschlangen-URI hat folgendes Format:

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
```

```
& propertyName2 = propertyValue2
&...]
```

Ein Themen-URI hat folgendes Format:

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Die Variablen in diesen Formaten haben folgende Bedeutungen:

qMgrName
Der Name des Warteschlangenmanagers, der Eigner der Warteschlange ist, die durch den URI angegeben ist.

Bei dem Warteschlangenmanager kann es sich um den lokalen Warteschlangenmanager oder um einen fernen Warteschlangenmanager handeln. Wenn es sich um einen fernen Warteschlangenmanager handelt, muss IBM MQ so konfiguriert sein, dass WebSphere MQ die Nachricht vom lokalen Warteschlangenmanager an den fernen Warteschlangenmanager weiterleiten kann, wenn die Anwendung eine Nachricht an die Warteschlange sendet.

Erfolgt keine Angabe des Namens, wird der lokale Warteschlangenmanager verwendet.

qName
Der Name der IBM MQ-Warteschlange.

Bei der Warteschlange kann es sich um eine lokale Warteschlange, um eine Aliaswarteschlange oder um eine Definition einer fernen Warteschlange handeln.

Die Regeln für die Erstellung von Warteschlangennamen finden Sie unter Regeln für die Benennung von IBM MQ-Objekten.

topicName
Der Name des Themas.

Die Regeln für die Erstellung von Themennamen finden Sie unter Regeln für die Benennung von IBM MQ-Objekten. Vermeiden Sie die Verwendung der Platzhalterzeichen +, #, * und? in Themennamen. Themennamen, die diese Zeichen enthalten, können zu unerwarteten Ergebnissen führen, wenn Sie diese subskribieren. Siehe Themenzeichenfolgen kombinieren.

***propertyName1*, *propertyName2*, ...**
Die Namen der Eigenschaften des Queue- oder Topic-Objekts, das durch die Anwendung erstellt wird. In Tabelle 35 auf Seite 234 sind die gültigen Eigenschaftsnamen aufgelistet, die in einem URI verwendet werden können.

Wenn keine Eigenschaften angegeben werden, werden dem Queue- oder Topic-Objekt für alle seine Eigenschaften Standardwerte zugewiesen.

***propertyValue1*, *propertyValue2*, ...**
Die Werte der Eigenschaften des Queue- oder Topic-Objekts, das durch die Anwendung erstellt wird. In Tabelle 35 auf Seite 234 sind die gültigen Eigenschaftswerte aufgelistet, die in einem URI verwendet werden können.

Klammern ([]) bezeichnen eine optionale Komponente und die Auslassungspunkte (...) bedeuten, dass die Liste der Name/Wert-Paare für Eigenschaften, falls vorhanden, ein oder mehrere Name/Wert-Paare enthalten können.

In Tabelle 35 auf Seite 234 sind die gültigen Eigenschaftsnamen und gültigen Werte aufgelistet, die in Warteschlangen- und Themen-URIs verwendet werden können. Das IBM MQ-JMS-Verwaltungstool verwendet zwar symbolische Konstanten für die Werte von Eigenschaften, URIs können jedoch keine symbolischen Konstanten enthalten.

Tabelle 35. Eigenschaftsnamen und gültige Werte für die Verwendung in Warteschlangen- und Themen-URIs

Eigenschaftsname	Beschreibung	Gültige Werte
CCSID	Gibt an, wie die Zeichendaten im Hauptteil einer Nachricht dargestellt werden, wenn IBM MQ classes for JMS die Nachricht an ein Ziel weiterleiten	• Eine beliebige ID des codierten Zeichensatzes, die von IBM MQ unterstützt wird.
encoding	Gibt an, wie die numerischen Daten im Hauptteil einer Nachricht dargestellt werden, wenn IBM MQ classes for JMS die Nachricht an ein Ziel weiterleiten	• Ein beliebiger gültiger Wert für das Feld *Encoding* in einem IBM MQ-Nachrichtendeskriptor.
expiry	Die Lebensdauer der Nachrichten, die an das Ziel gesendet werden	• -2 - Wie im Aufruf send() angegeben oder, falls keine Angabe im Aufruf send() erfolgt, die Standardlebensdauer des Nachrichtenproduzenten. • 0 - Eine an das Ziel gesendete Nachricht läuft nie ab. • Eine positive Ganzzahl, die die Lebensdauer in Millisekunden angibt.
multicast	Die Multicasteinstellung für ein Thema, wenn eine Echtzeitverbindung mit einem Broker verwendet wird	Die folgende Liste enthält die gültigen Werte. Jedem Wert ist der entsprechende Wert der MULTICAST-Eigenschaft zugeordnet, die im IBM MQ JMS-Verwaltungstool verwendet wird. Sie finden eine Beschreibung der MULTICAST-Eigenschaft und ihrer gültigen Werte im Abschnitt Eigenschaften von IBM MQ classes for JMS-Objekten. • -1 - ASCF • 0 - DISABLED • 3 - NOTR • 5 - RELIABLE • 7 - ENABLED
persistence	Die Persistenz der Nachrichten, die an das Ziel gesendet werden	• -2 - Wie im Aufruf send() angegeben oder, falls keine Angabe im Aufruf send() erfolgt, die Standardpersistenz des Nachrichtenproduzenten. • -1 - Wie im Attribut *DefPersistence* der Warteschlange oder des Themas von IBM MQ angegeben. • 1 - Nicht persistent. • 2 - Persistent. • 3 - Äquivalent zu dem Wert HIGH für die Eigenschaft PERSISTENCE, wie im IBM MQ-JMS-Verwaltungstool verwendet. Sie finden eine Erläuterung dieses Werts im Abschnitt „Persistente JMS-Nachrichten” auf Seite 265.

Tabelle 35. Eigenschaftsnamen und gültige Werte für die Verwendung in Warteschlangen- und Themen-URIs (Forts.)		
Eigenschaftsname	**Beschreibung**	**Gültige Werte**
priority	Die Priorität der Nachrichten, die an das Ziel gesendet werden	• -2 - Wie im Aufruf send() angegeben oder, falls keine Angabe im Aufruf send() erfolgt, die Standardpriorität des Nachrichtenproduzenten. • -1 - Wie im Attribut *DefPriority* der Warteschlange oder des Themas von IBM MQ angegeben. • Eine Ganzzahl im Bereich 0 bis 9, die die Priorität der an das Ziel gesendeten Nachrichten angibt.
targetClient	Gibt an, ob die an das Ziel gesendeten Nachrichten einen MQRFH2-Header enthalten	• 0 - Nachrichten enthalten einen MQRFH2-Header. • 1 - Nachrichten enthalten keinen MQRFH2-Header.

Der folgende URI gibt beispielsweise eine IBM MQ-Warteschlange mit dem Namen Q1 an, deren Eigner der lokale Warteschlangenmanager ist. Ein mit diesem URI erstelltes Queue-Objekt weist für alle seine Eigenschaften Standardwerte auf.

```
queue:///Q1
```

Der folgende URI gibt eine IBM MQ-Warteschlange mit dem Namen Q2 an, deren Eigner ein Warteschlangenmanager mit dem Namen QM2 ist. Alle Nachrichten, die an dieses Ziel gesendet werden, haben die Priorität 6. Die verbleibenden Eigenschaften des Queue-Objekts, das mit diesem URI erstellt wird, haben die jeweiligen Standardwerte.

```
queue://QM2/Q2?priority=6
```

Der folgende URI gibt ein Thema mit dem Namen Sport/Athletics/Results an. Alle Nachrichten, die an dieses Ziel gesendet werden, sind persistent und haben die Priorität 0. Die verbleibenden Eigenschaften des Topic-Objekts, das mit diesem URI erstellt wird, haben die jeweiligen Standardwerte.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

Nachrichten in einer JMS-Anwendung senden

Bevor eine JMS-Anwendung Nachrichten an ein Ziel senden kann, muss sie ein MessageProducer-Objekt für das Ziel erstellen. Um eine Nachricht an das Ziel zu senden, erstellt die Anwendung ein Message-Objekt und ruft dann die send()-Methode des MessageProducer-Objekts auf.

Eine Anwendung verwendet ein MessageProducer-Objekt zum Senden von Nachrichten. Normalerweise erstellt eine Anwendung ein MessageProducer-Objekt für ein bestimmtes Ziel, das eine Warteschlange oder ein Thema sein kann, damit alle Nachrichten, die mit dem Nachrichtenproduzenten gesendet werden, an dasselbe Ziel gesendet werden. Daher muss zuerst ein Queue- oder Topic-Objekt erstellt werden, damit eine Anwendung ein MessageProducer-Objekt erstellen kann. Sie finden Informationen zur Vorgehensweise bei der Erstellung eines Queue- oder Topic-Objekts in den folgenden Abschnitten:

- „JNDI zum Abrufen von verwalteten Objekten in einer JMS -oder Jakarta Messaging -Anwendung verwenden” auf Seite 215
- „Verwendung der IBM JMS-Erweiterungen” auf Seite 216

- „Verwendung der IBM MQ JMS-Erweiterungen” auf Seite 224
- „Ziele in einer JMS-Anwendung erstellen” auf Seite 231

Um ein MessageProducer-Objekt zu erstellen, verwendet eine Anwendung die Methode createProducer() eines Session-Objekts, wie im folgenden Beispiel dargestellt:

```
MessageProducer producer = session.createProducer(destination);
```

Der Parameter `destination` ist ein Queue- oder Topic-Objekt, das die Anwendung zuvor erstellt hat.

Bevor eine Anwendung eine Nachricht senden kann, muss sie ein Message-Objekt erstellen. Der Hauptteil einer Nachricht enthält die Anwendungsdaten und JMS definiert fünf Nachrichtenhauptteiltypen:

- Bytes
- Zuordnung
- Objekt
- Datenstrom
- Text

Jeder Nachrichtenhauptteiltyp hat eine eigene JMS-Schnittstelle, die eine Unterschnittstelle der Message-Schnittstelle (Nachrichtenschnittstelle) ist, und eine Methode in der Session-Schnittstelle für die Erstellung einer Nachricht mit diesem Hauptteiltyp. Die Schnittstelle für eine Textnachricht heißt beispielsweise TextMessage und eine Anwendung verwendet die Methode createTextMessage() eines Session-Objekts für die Erstellung einer Textnachricht, wie in der folgenden Anweisung dargestellt:

```
TextMessage outMessage = session.createTextMessage(outString);
```

Weitere Informationen zu Nachrichten und Nachrichtenhauptteilen finden Sie im Abschnitt „JMS-Nachrichten” auf Seite 151.

Um eine Nachricht zu senden, verwendet eine Anwendung die Methode send() eines MessageProducer-Objekts, wie im folgenden Beispiel dargestellt:

```
producer.send(outMessage);
```

Eine Anwendung kann mit der Methode send() Nachrichten in jeder Messaging-Domäne senden. Die Art des Ziels bestimmt, welche Messaging-Domäne verwendet wird. TopicPublisher (die Unterschnittstelle von MessageProducer, die speziell für die Publish/Subscribe-Domäne vorgesehen ist) verfügt außerdem über die Methode publish(), die anstelle der Methode send() verwendet werden kann. Die Funktionen der beiden Methoden sind identisch.

Eine Anwendung kann ein MessageProducer-Objekt erstellen, ohne dabei eine bestimmtes Ziel anzugeben. In diesem Fall muss die Anwendung das Ziel angeben, wenn sie die Methode send() aufruft.

Falls eine Anwendung eine Nachricht innerhalb einer Transaktion sendet, wird die Nachricht erst dann an das zugehörige Ziel übermittelt, wenn die Transaktion festgeschrieben wurde. Dies bedeutet, dass eine Anwendung nicht innerhalb derselben Transaktion eine Nachricht senden und eine Antwort empfangen kann.

Ein Ziel kann so konfiguriert werden, dass IBM MQ classes for JMS die Nachricht weiterleiten und die Steuerung an die Anwendung zurückgeben, ohne zu ermitteln, ob der Warteschlangenmanager die Nachricht fehlerfrei erhalten hat, wenn eine Anwendung Nachricht an dieses Ziel sendet. Dies wird gelegentlich als *asynchrone Put-Operation* bezeichnet. Weitere Informationen finden Sie in „Nachrichten asynchron in IBM MQ classes for JMS einreihen” auf Seite 336.

Nachrichten in einer JMS-Anwendung empfangen

Eine Anwendung verwendet einen Nachrichtenkonsumenten für das Empfangen von Nachrichten. Ein permanenter Topic-Subskribent ist ein Nachrichtenkonsument, der alle Nachrichten empfängt, die an ein Ziel gesendet werden. Dies betrifft auch Nachrichten, die gesendet werden, solange der Konsument

inaktiv ist. Eine Anwendung kann mithilfe eines Nachrichtenselektors auswählen, welche Nachrichten sie empfangen möchte, und sie kann unter Verwendung eines Nachrichtenlisteners Nachrichten asynchron empfangen.

Eine Anwendung verwendet ein MessageConsumer-Objekt zum Empfangen von Nachrichten. Eine Anwendung erstellt ein MessageConsumer-Objekt für ein bestimmtes Ziel, das eine Warteschlange oder ein Thema sein kann, damit alle Nachrichten, die mit dem Nachrichtenkonsumenten empfangen werden, aus demselben Ziel empfangen werden. Daher muss zuerst ein Queue- oder Topic-Objekt erstellt werden, damit eine Anwendung ein MessageConsumer-Objekt erstellen kann. Sie finden Informationen zur Vorgehensweise bei der Erstellung eines Queue- oder Topic-Objekts in den folgenden Abschnitten:

- „JNDI zum Abrufen von verwalteten Objekten in einer JMS -oder Jakarta Messaging -Anwendung verwenden” auf Seite 215
- „Verwendung der IBM JMS-Erweiterungen” auf Seite 216
- „Verwendung der IBM MQ JMS-Erweiterungen” auf Seite 224
- „Ziele in einer JMS-Anwendung erstellen” auf Seite 231

Um ein MessageConsumerObjekt zu erstellen, verwendet eine Anwendung die Methode createConsumer() eines Session-Objekts, wie im folgenden Beispiel dargestellt:

```
MessageConsumer consumer = session.createConsumer(destination);
```

Der Parameter `destination` ist ein Queue- oder Topic-Objekt, das die Anwendung zuvor erstellt hat.

Die Anwendung verwendet dann die Methode receive() des MessageConsumer-Objekts, um eine Nachricht aus dem Ziel zu empfangen, wie im folgenden Beispiel dargestellt:

```
Message inMessage = consumer.receive(1000);
```

Der Parameter im Aufruf receive() gibt in Millisekunden an, wie lange die Methode auf das Eintreffen einer geeigneten Nachricht wartet, wenn nicht sofort eine Nachricht verfügbar ist. Wird dieser Parameter nicht angegeben, blockiert der Aufruf unendlich lange, bis eine geeignete Nachricht eintrifft. Wenn Sie nicht möchten, dass die Anwendung auf eine Nachricht wartet, verwenden Sie stattdessen die Methode receiveNoWait().

Die Methode receive() gibt eine Nachricht eines bestimmten Typs zurück. Wenn eine Anwendung beispielsweise eine Textnachricht empfängt, handelt es sich bei dem vom Aufruf receive() zurückgegebenen Objekt um ein TextMessage-Objekt.

Der deklarierte Typ des von einem receive()-Aufruf zurückgegebenen Objekts ist jedoch ein Message-Objekt. Um die Daten aus dem Hauptteil einer soeben von der Anwendung empfangenen Nachricht extrahieren zu können, muss die Anwendung daher eine Umsetzung aus der Message-Klasse in eine spezifischere Unterklasse wie TextMessage durchführen. Falls der Nachrichtentyp nicht bekannt ist, kann die Anwendung den Typ mithilfe des Operators `instanceof` ermitteln. Eine Anwendung sollte vor einer Umsetzung immer den Nachrichtentyp ermitteln, damit Fehler ordnungsgemäß behandelt werden können.

Der folgende Code verwendet den Operator `instanceof` und zeigt, wie die Daten aus dem Hauptteil einer Textnachricht extrahiert werden:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

Falls eine Anwendung eine Nachricht innerhalb einer Transaktion sendet, wird die Nachricht erst dann an das zugehörige Ziel übermittelt, wenn die Transaktion festgeschrieben wurde. Dies bedeutet, dass eine

Anwendung nicht innerhalb derselben Transaktion eine Nachricht senden und eine Antwort empfangen kann.

Wenn ein Nachrichtenkonsument Nachrichten aus einem Ziel empfängt, das für das Vorauslesen konfiguriert wurde, werden alle nicht permanenten Nachrichten gelöscht, die sich bei Beendigung der Anwendung im Vorauslesepuffer befinden.

In der Publish/Subscribe-Domäne gibt JMS zwei Arten von Nachrichtenkonsumenten an: den nicht permanenten Topic-Subskribenten und den permanenten Topic-Subskribenten. Diese werden in den folgenden beiden Abschnitten beschrieben.

Nicht permanente Topic-Subskribenten

Ein nicht permanenter Topic-Subskribent empfängt nur die Nachrichten, die veröffentlicht werden, solange der Subskribent aktiv ist. Eine nicht permanente Subskription beginnt, wenn eine Anwendung einen nicht permanenten Topic-Subscriber erstellt. Sie endet, wenn die Anwendung den Subskribenten schließt oder wenn der Subskribent aus dem Geltungsbereich fällt. Als Erweiterung in IBM MQ classes for JMS empfängt ein nicht permanenter Topic-Subskribent auch ständige Veröffentlichungen.

Für die Erstellung eines nicht permanenten Topic-Subskribenten kann eine Anwendung die domänenunabhängige Methode createConsumer() verwenden und dabei ein Topic-Objekt als Ziel angeben. Alternativ dazu kann eine Anwendung wie im folgenden Beispiel die domänenspezifische Methode createSubscriber() verwenden:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

Der Parameter `topic` ist ein Topic-Objekt, das die Anwendung zuvor erstellt hat.

Permanente Topic-Subskribenten

Einschränkung: Wenn eine Anwendung eine Echtzeitverbindung mit einem Broker verwendet, kann sie keine permanenten Topic-Subskribenten erstellen.

Ein permanenter Topic-Subskribent empfängt alle Nachrichten, die während des Lebenszyklus einer permanenten Subskription veröffentlicht werden. Zu diesen Nachrichten gehören auch alle Nachrichten, die veröffentlicht werden, solange der Subskribent nicht aktiv ist. Als Erweiterung in IBM MQ classes for JMS empfängt ein permanenter Topic-Subskribent auch ständige Veröffentlichungen.

Um einen permanenten Topic-Subskribenten zu erstellen, verwendet eine Anwendung die Methode createDurableSubscriber() eines Session-Objekts, wie im folgenden Beispiel dargestellt:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

Beim Aufruf createDurableSubscriber() ist der erste Parameter ein Topic-Objekt, das von der Anwendung zuvor erstellt wurde, und der zweite Parameter ist ein Name, der für die Angabe der permanenten Subskription verwendet wird.

Der Sitzung, die für die Erstellung eines permanenten Topic-Subskribenten verwendet wird, muss eine Client-ID zugeordnet sein. Die Client-ID, die einer Sitzung zugeordnet ist, ist mit derjenigen für die Verbindung identisch, die für die Erstellung der Sitzung verwendet wird. Die Client-ID kann durch das Festlegen der CLIENTID-Eigenschaft des connectionFactory-Objekts angegeben werden. Alternativ kann eine Anwendung die Client-ID auch angeben, indem sie die Methode setClientID() des Connection-Objekts aufruft.

Der Name, der für die Angabe einer permanenten Subskription verwendet wird, muss nur innerhalb der Client-ID eindeutig sein. Daher ist die Client-ID Bestandteil der vollständigen, eindeutigen ID einer permanenten Subskription. Damit eine zuvor erstellte permanente Subskription weiterhin verwendet werden kann, muss eine Anwendung einen permanenten Topic-Subskribenten unter Verwendung einer Sitzung mit der Client-ID erstellen, die der permanenten Subskription zugeordnet wurde. Außerdem muss dabei derselbe Subskriptionsname verwendet werden.

Eine permanente Subskription beginnt, wenn eine Anwendung einen permanenten Topic-Subskribenten mit einer Client-ID und einem Subskriptionsnamen erstellt, für den derzeit noch keine permanente Subskription vorhanden ist. Eine permanente Subskription endet jedoch nicht, wenn die Anwendung den permanenten Topic-Subskribenten schließt. Zur Beendigung einer permanenten Subskription muss eine Anwendung die Methode unsubscribe() eines Session-Objekts aufrufen, dessen Client-ID mit der Client-ID identisch ist, die der permanenten Subskription zugeordnet wurde. Der Parameter im Aufruf unsubscribe() ist der Subskriptionsname, wie im folgenden Beispiel dargestellt:

```
session.unsubscribe("D_SUB_000001");
```

Der Geltungsbereich einer permanenten Subskription ist der Warteschlangenmanager. Wenn eine permanente Subskription auf einem Warteschlangenmanager vorhanden ist und eine Anwendung, die mit einem anderen Warteschlangenmanager verbunden ist, eine permanente Subskription mit derselben Client-ID und demselben Subskriptionsnamen erstellt, sind die beiden permanenten Subskriptionen völlig unabhängig voneinander.

Nachrichtenselektoren

Eine Anwendung kann angeben, dass nur die Nachrichten von nachfolgenden receive()-Aufrufen zurückgegeben werden, die bestimmte Kriterien erfüllen. Wenn sie ein MessageConsumer-Objekt erstellt, kann die Anwendung einen SQL-Ausdruck (SQL = Structured Query Language) angeben, der festlegt, welche Nachrichten abgerufen werden. Dieser SQL-Ausdruck wird als *Nachrichtenselektor* bezeichnet. Der Nachrichtenselektor kann die Namen von JMS-Nachrichtenheaderfeldern und Nachrichteneigenschaften enthalten. Informationen zur Erstellung eines Nachrichtenselektors finden Sie im Abschnitt „Nachrichtenselektoren in JMS” auf Seite 152.

Das folgende Beispiel zeigt, wie eine Anwendung Nachrichten auf Basis einer benutzerdefinierten Eigenschaft mit dem Namen 'myProp' auswählen kann:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

Die JMS-Spezifikation erlaubt einer Anwendung nicht, den Nachrichtenselektor eines Nachrichtenkonsumenten zu ändern. Nachdem eine Anwendung einen Nachrichtenkonsumenten mit einem Nachrichtenselektor erstellt hat, bleibt der Nachrichtenselektor für den Lebenszyklus dieses Konsumenten erhalten. Falls eine Anwendung mehrere Nachrichtenselektoren benötigt, muss die Anwendung für jeden einzelnen Nachrichtenselektor einen Nachrichtenkonsumenten erstellen.

Beachten Sie, dass die Eigenschaft MSGSELECTION der Verbindungsfactory keine Auswirkung hat, wenn eine Anwendung mit einem Warteschlangenmanager der Version 7 verbunden ist. Zur Leistungsoptimierung erfolgt die gesamte Nachrichtenauswahl durch den Warteschlangenmanager.

Lokale Veröffentlichungen unterdrücken

Eine Anwendung kann einen Nachrichtenkonsumenten erstellen, der Veröffentlichungen ignoriert, die in der eigenen Verbindung des Konsumenten veröffentlicht werden. Hierfür setzt die Anwendung den dritten Parameter in einem createConsumer()-Aufruf auf `true`, wie im folgenden Beispiel dargestellt:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

Bei einem createDurableSubscriber()-Aufruf setzt die Anwendung hierfür wie im folgenden Beispiel den vierten Parameter auf`true`:

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

Asynchrone Nachrichtenübermittlung

Eine Anwendung kann Nachrichten asynchron empfangen, indem sie einen Nachrichtenlistener bei einem Nachrichtenkonsumenten registriert. Der Nachrichtenlistener verfügt über eine Methode mit dem Namen 'onMessage', die asynchron aufgerufen wird, sobald eine geeignete Nachricht verfügbar ist. Ihr Zweck besteht in der Verarbeitung der Nachricht. Der folgende Code veranschaulicht das Verfahren:

JM 3.0

```
import jakarta.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

JMS 2.0

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

Eine Anwendung kann eine Sitzung entweder für den synchronen Empfang von Nachrichten mit receive()-Aufrufen oder für den asynchronen Empfang von Nachrichten mit Nachrichtenlistenern verwenden. Sie kann jedoch nicht beide Verfahren nutzen. Wenn eine Anwendung Nachrichten synchron und asynchron empfangen muss, muss sie separate Sitzungen erstellen.

Nachdem eine Sitzung für den asynchronen Empfang von Nachrichten eingerichtet wurde, können die folgenden Methoden nicht mehr für diese Sitzung oder für Objekte, die über diese Sitzung erstellt werden, aufgerufen werden:

- MessageConsumer.receive()
- MessageConsumer.receive(long)
- MessageConsumer.receiveNoWait()
- Session.acknowledge()
- MessageProducer.send(Destination, Message)
- MessageProducer.send(Destination, Message, int, int, long)
- MessageProducer.send(Message)
- MessageProducer.send(Message, int, int, long)
- MessageProducer.send(Destination, Message, CompletionListener)
- MessageProducer.send(Destination, Message, int, int, long, CompletionListener)
- MessageProducer.send(Message, CompletionListener)
- MessageProducer.send(Message, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(Queue)
- Session.createBrowser(Queue, String)
- Session.createBytesMessage()
- Session.createConsumer(Destination)
- Session.createConsumer(Destination, String, boolean)
- Session.createDurableSubscriber(Topic, String)
- Session.createDurableSubscriber(Topic, String, String, boolean)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(Serializable)
- Session.createProducer(Destination)
- Session.createQueue(String)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String)

Falls eine dieser Methoden aufgerufen wird, wird eine Ausnahmebedingung (JMSException) mit der Nachricht

```
JMSCC0033: Ein synchroner Methodenaufruf ist nicht zulässig, wenn eine Sitzung asynchron verwendet wird: 'Methodenname'
```

(Ein synchroner Methodenaufruf ist nicht zulässig, wenn eine Sitzung asynchron verwendet wird: 'Methodenname') ausgelöst.

Nicht verarbeitbare Nachrichten empfangen

Eine Anwendung kann eine Nachricht erhalten, die nicht verarbeitet werden kann. Wenn eine Nachricht nicht verarbeitet werden kann, kann dies mehrere Ursachen haben. So kann es beispielsweise sein, dass die Nachricht ein falsches Format hat. Solche Nachrichten werden als nicht verarbeitbare Nachrichten bezeichnet. Sie müssen auf besondere Weise gehandhabt werden, um zu verhindern, dass die Nachricht rekursiv verarbeitet wird.

Sie finden Details zur Behandlung nicht verarbeitbarer Nachrichten im Abschnitt „Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln” auf Seite 244.

Anpassung der Puffergrößen an die empfangenen Nachrichten

Wenn eine Nachricht von einer Nicht-JMS -Anwendung von IBM MQ empfangen wird, muss von der Anwendung ein Nachrichtenpuffer bereitgestellt werden, in den die Nachricht geschrieben werden kann. JMS -Anwendungen müssen keinen Puffer manuell erstellen. Der IBM MQ classes for JMS erstellt automatisch Nachrichtenpuffer und passt die Größe an die Größe der empfangenen Nachrichten an. Für die meisten Anwendungen bieten automatisch verwaltete Puffer eine angemessene Balance zwischen Leistung und Komfort für den Anwendungsentwickler. Unter bestimmten Umständen kann es sinnvoll sein, die Anfangsgröße des Nachrichtenpuffers manuell anzugeben. Die Standardanfangsgröße eines IBM MQ JMS -Empfangspuffers ist 4 KB. Wenn eine Anwendung immer Nachrichten mit einer Größe von 256 KB empfangen wird, kann es empfehlenswert sein, die anfängliche Puffergröße auf 256 KB zu konfigurieren. Dies kann verhindern, dass IBM MQ classes for JMS versuchen muss, die Nachricht in einem 4-KB-Puffer zu empfangen, bevor die Größe auf 256 KB geändert und erfolgreich empfangen wird. Bei einer mit einem Client verbundenen Anwendung kann dies die Notwendigkeit einer potenziell verschwendeten Netzumlaufzeit vermeiden, während der IBM MQ classes for JMS die richtige zu verwendende Puffergröße ermittelt.

Die ursprüngliche Puffergröße kann konfiguriert werden, indem Sie die Eigenschaft com.ibm.mq.jmqi.defaultMaxMsgSize Java auf den ausgewählten Wert in Byte setzen. Beachten Sie, dass diese Eigenschaft alle IBM MQ JMS -Anwendungen betrifft, die in Java Virtual Machineausgeführt werden. Achten Sie daher darauf, dass andere Nachrichtenkonsumenten, die Nachrichten mit einer anderen Größe empfangen, nicht beeinträchtigt werden.

IBM MQ classes for JMS versucht weiterhin, die Größe des Puffers automatisch zu reduzieren, wenn mehrere Nachrichten empfangen werden, die kleiner als die konfigurierte Größe sind. Dies geschieht standardmäßig, wenn 10 Nachrichten empfangen werden, die alle kleiner als die Puffergröße sind. Wenn beispielsweise 10 Nachrichten in einer Zeile empfangen werden, die 128 KB groß sind, wird der Puffer auf 256 KB auf 128 KB reduziert. Sie wird dann wieder erhöht, wenn größere Nachrichten empfangen werden. Es ist möglich, die Anzahl der Nachrichten zu konfigurieren, die empfangen werden müssen, bevor ein Puffer verkleinert wird. Dies kann beispielsweise nützlich sein, wenn bekannt ist, dass die Anwendung fünf große Nachrichten gefolgt von zehn kleineren Nachrichten und fünf weiteren großen Nachrichten empfängt. Bei den Standardeinstellungen würde der Puffer nach dem Empfang der 10 kleineren Nachrichten reduziert und müsste für die größeren Nachrichten erneut erhöht werden. Die Java -Systemeigenschaft com.ibm.mq.jmqi.smallMsgBufferReductionThreshold kann auf die Anzahl der Nachrichten gesetzt werden, die empfangen werden müssen, bevor die Größe des Puffers reduziert wird. In diesem Beispiel könnte er auf 20 gesetzt werden, um zu verhindern, dass 10 kleinere Nachrichten die Puffergröße verringern.

Die Eigenschaften können unabhängig voneinander festgelegt werden. Sie können beispielsweise die ursprüngliche Puffergröße auf den Standardwert von 4 KB setzen, aber den Wert von com.ibm.mq.jmqi.smallMsgBufferReductionThreshold erhöhen, damit der Puffer länger bleibt, sobald er vergrößert wird.

Wenn eine große Anzahl von Rückgabecodes MQRC_TRUNCATED_MSG_FAILED (2080) für Ihre JMS -Anwendungen in MQI-Statistikdatensätzen angezeigt werden, kann dies ein Hinweis darauf sein, dass Sie eine höhere Anfangspuffergröße für diese Anwendungen konfigurieren oder die Häufigkeit verringern würden, mit der Puffergrößen reduziert werden. Es ist jedoch wichtig zu beachten, dass Sie für eine Anwendung mit langer Laufzeit wahrscheinlich nur eine sehr kleine Anzahl von MQRC_TRUNCATED_MSG_FAILED-Rückgabecodes sehen. Dies liegt daran, dass der Puffer in der Regel unmittelbar nach

dem Empfang der ersten großen Nachricht auf die richtige Größe vergrößert wird und nicht verkleinert wird, wenn nicht mehrere kleinere Nachrichten empfangen werden. Es ist daher möglich, dass eine große Anzahl von MQRC_TRUNCATED_MSG_FAILED andere schlechte Anwendungsverfahren angibt, wie z. B. die Verbindung zu IBM MQ , um nur eine oder zwei Nachrichten zu empfangen, bevor die Verbindung getrennt wird.

Abruf von Subskriptionsbenutzerdaten

Wenn die Nachrichten, die eine IBM MQ classes for JMS-Anwendung aus einer Warteschlange liest, von einer administrativ definierten, permanenten Subskription eingereiht werden, muss die Anwendung auf die Benutzerdateninformationen, die der Subskription zugeordnet sind, zugreifen. Diese Informationen werden als Eigenschaft zur Nachricht hinzugefügt.

Beim Lesen einer Nachricht aus einer Warteschlange, die einen RFH2-Header mit dem Ordner MQPS enthält, wird der Wert, der dem Schlüssel Sud (sofern vorhanden) zugeordnet ist, als Zeichenfolgeeigenschaft zu dem JMS-Nachrichtenobjekt hinzugefügt, das an die IBM MQ classes for JMS-Anwendung zurückgegeben wird. Damit diese Eigenschaft aus der Nachricht abgerufen werden kann, kann die Konstante JMS_IBM_SUBSCRIPTION_USER_DATA in der Schnittstelle JmsConstants mit der folgenden Methode zum Abrufen der Subskriptionsbenutzerdaten verwendet werden:

- JM 3.0 jakarta.jms.Message.getStringProperty(java.lang.String)
- JMS 2.0 javax.jms.Message.getStringProperty(java.lang.String)

Im folgenden Beispiel wird eine permanente Verwaltungssubskription mit dem MQSC-Befehl **DEFINE SUB**definiert:

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

Kopien von Nachrichten, die für die Themenzeichenfolge PUBLIC veröffentlicht werden, werden in die Warteschlange MY.SUBSCRIPTION.Q eingereiht. Die Benutzerdaten, die der permanenten Subskription zugeordnet sind, werden dann als Eigenschaft zu der Nachricht hinzugefügt, die im Ordner MQPS des RFH2-Headers mit dem Schlüssel Sud gespeichert wird.

Die IBM MQ classes for JMS-Anwendung kann folgenden Aufruf ausgeben:

JM 3.0
```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

JMS 2.0
```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

Daraufhin wird folgende Zeichenfolge zurückgegeben:

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

Zugehörige Konzepte

„MQRFH2-Header und JMS” auf Seite 157

Zugehörige Tasks

Verwaltungssubskription definieren

Zugehörige Verweise

SUB DEFINI

Schnittstelle JmsConstants

Anwendung der IBM MQ classes for JMS schließen

Bei einer Anwendung der IBM MQ classes for JMS ist es wichtig, dass bestimmte JMS-Objekte vor dem Stoppen explizit geschlossen werden. Da Beendigungsfunktionen möglicherweise nicht aufgerufen werden, können Sie sich nicht darauf verlassen, dass diese Ressourcen freigeben. Erlauben Sie einer Anwendung keine Beendigung mit aktivem komprimiertem Trace.

Die Garbage-Collection alleine kann nicht alle Ressourcen der IBM MQ classes for JMS und IBM MQ-Ressourcen zeitnah freigeben. Dies gilt vor allem dann, wenn eine Anwendung viele JMS-Objekte mit einer kurzen Lebensdauer auf Sitzungsebene oder auf einer niedrigeren Ebene erstellt. Daher ist es wichtig, dass eine Anwendung ein Connection-, Session-, MessageConsumer- oder MessageProducer-Objekt schließt, wenn es nicht mehr benötigt wird.

Wenn eine Anwendung ohne Schließen einer Verbindung beendet wird, wird für alle transaktionsbasierten Sitzungen der Verbindung ein impliziter Rollback durchgeführt. Um sicherzustellen, dass von der Anwendung vorgenommene Änderungen festgeschrieben werden, muss die Verbindung explizit vor dem Schließen der Anwendung geschlossen werden.

Verwenden Sie keine Beendigungsfunktionen in einer Anwendung, um JMS-Objekte zu schließen. Da Beendigungsfunktionen möglicherweise nicht aufgerufen werden, kann es vorkommen, dass Ressourcen nicht freigegeben werden. Wenn eine Verbindung geschlossen wird, schließt sie alle Sitzungen, die über sie erstellt wurden. Ähnlich gilt, dass MessageConsumers und MessageProducers, die über eine Sitzung erstellt wurden, beim Schließen der Sitzung ebenfalls geschlossen werden. Sie sollten Sitzungen, Nachrichtenkonsumenten und Nachrichtenproduzenten (Sessions, MessageConsumers und MessageProducers) jedoch explizit schließen, um sicherzustellen, dass Ressourcen zeitnah freigegeben werden.

Wenn die Tracekomprimierung aktiviert ist, führt System.Halt() einen Shutdown durch und abnormale, nicht gesteuerte JVM-Beendigungen führen wahrscheinlich zu einer beschädigten Tracedatei. Sofern möglich, inaktivieren Sie die Tracefunktion, sobald Sie die benötigten Traceinformationen gesammelt haben. Wenn Sie für eine Anwendung bis zur abnormalen Beendigung das Tracing nutzen, verwenden Sie eine nicht komprimierte Traceausgabe.

Anmerkung: Um die Verbindung zu einem Warteschlangenamanger zu trennen, ruft eine JMS-Anwendung die Methode 'close()' für das Verbindungsobjekt auf.

Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln

Eine nicht verarbeitbare Nachricht ist eine Nachricht, die von einer empfangenden Anwendung nicht verarbeitet werden kann. Wenn eine nicht verarbeitbare Nachricht an eine Anwendung übermittelt und eine bestimmte Anzahl Male zurückgesetzt wird, können die IBM MQ classes for JMS die Nachricht in eine Rücksetzwarteschlange verschieben.

Eine nicht verarbeitbare Nachricht ist eine Nachricht, die von einer empfangenden Anwendung nicht verarbeitet werden kann. Die Nachricht kann einen unerwarteten Typ aufweisen oder Informationen enthalten, die von der Anwendungslogik nicht verarbeitet werden können. Wenn eine nicht verarbeitbare Nachricht an eine Anwendung übermittelt wird, stellt die Anwendung die Nachricht in die Warteschlange zurück, aus der sie gekommen ist. Standardmäßig ist es so, dass die IBM MQ classes for JMS die Nachricht immer wieder an die Anwendung übermitteln. Dies kann dazu führen, dass die Anwendung in einer Schleife hängen bleibt, da sie jedes Mal vergeblich versucht die Nachricht zu verarbeiten und sie dann zurückstellt.

Um dies zu verhindern, können die IBM MQ classes for JMS nicht verarbeitbare Nachrichten erkennen und sie an ein alternatives Ziel verschieben. Zu diesem Zweck nutzen die IBM MQ classes for JMS folgende Eigenschaften:

- Den Wert des Feldes 'BackoutCount' im MQMD der erkannten Nachricht.
- Die IBM MQ-Warteschlangenattribute **BOTHRESH** (Rücksetzschwellenwert) und **BOQNAME** (Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten) für die Eingabewarteschlange, in der sich die Nachricht befindet.

Immer wenn eine Nachricht von einer Anwendung zurückgesetzt wird, erhöht der Warteschlangenmanager automatisch den Wert des Feldes 'BackoutCount' für die Nachricht.

Wenn die IBM MQ classes for JMS eine Nachricht mit einem BackoutCount-Wert größer als null erkennen, vergleichen sie ihn mit dem Wert des Attributs **BOTHRESH**.

- Wenn der BackoutCount-Wert kleiner als der Wert des Attributs **BOTHRESH** ist, übermitteln die IBM MQ classes for JMS die Nachricht zur Verarbeitung an die Anwendung.
- Ist der BackoutCount-Wert jedoch größer-gleich **BOTHRESH**, wird die Nachricht als nicht verarbeitbare Nachricht betrachtet. In dieser Situation verschieben die IBM MQ classes for JMS die Nachricht dann

in die Warteschlange, die durch das Attribut **BOQNAME** angegeben wird. Wenn die Nachricht nicht in die Rücksetzwarteschlange eingereiht werden kann, wird sie abhängig von den Berichtsoptionen der Nachricht entweder in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers verschoben oder gelöscht.

Anmerkung:

- Wenn für das Attribut **BOTHRESH** der Standardwert `0` beibehalten wird, ist die Behandlung nicht verarbeitbarer Nachrichten inaktiviert. Dies bedeutet, dass alle nicht verarbeitbaren Nachrichten in die Eingabewarteschlange zurückgestellt werden.
- Zu beachten ist darüber hinaus, dass die IBM MQ classes for JMS die Attribute **BOTHRESH** und **BOQNAME** für die Warteschlange abfragen, wenn sie zum ersten Mal eine Nachricht mit einem BackoutCount-Wert größer als null erkennen. Die Werte dieser Attribute werden dann zwischengespeichert und immer dann verwendet, wenn die IBM MQ classes for JMS auf eine Nachricht mit einem BackoutCount-Wert größer als null treffen.

System für Behandlung nicht verarbeitbarer Nachrichten konfigurieren

Welche Warteschlange die IBM MQ classes for JMS beim Abfragen der Attribute **BOTHRESH** und **BOQNAME** verwenden, hängt von der Art der durchgeführten Nachrichtenübermittlung (Messaging) ab:

- Beim Punkt-zu-Punkt-Messaging handelt es sich um die zugrunde liegende lokale Warteschlange. Dies ist wichtig, wenn eine JMS-Anwendung Nachrichten aus Aliaswarteschlangen oder Clusterwarteschlangen verarbeitet.
- Beim Publish/Subscribe-Messaging wird eine verwaltete Warteschlange erstellt, in der die Nachrichten für eine Anwendung aufbewahrt werden. Die IBM MQ classes for JMS fragen die verwaltete Warteschlange ab, um die Werte der Attribute **BOTHRESH** und **BOQNAME** zu ermitteln.

  Die verwaltete Warteschlange wird aus einer Modellwarteschlange erstellt, die dem Topic-Objekt zugeordnet ist, das von der Anwendung subskribiert wurde, und sie übernimmt die Werte der Attribute **BOTHRESH** und **BOQNAME** aus der Modellwarteschlange. Welche Modellwarteschlange verwendet wird, hängt davon ab, ob die empfangende Anwendung eine permanente oder nicht permanente Subskription eingerichtet hat:

  – Die für permanente Subskriptionen verwendete Modellwarteschlange wird durch das Attribut **MDURMDL** des Topics angegeben. Der Standardwert dieses Attributs ist `SYSTEM.DURABLE.MODEL.QUEUE`.

  – Für nicht permanente Subskriptionen wird die verwendete Modellwarteschlange durch das Attribut **MNDURMDL** angegeben. Der Standardwert des Attributs **MNDURMDL** ist `SYSTEM.NDURABLE.MODEL.QUEUE`.

Beim Abfragen der Attribute **BOTHRESH** und **BOQNAME** gehen die IBM MQ classes for JMS wie folgt vor:

- Sie öffnen die lokale Warteschlange oder die Zielwarteschlange für eine Aliaswarteschlange.
- Sie fragen die Attribute **BOTHRESH** und **BOQNAME** ab.
- Sie schließen die lokale Warteschlange oder die Zielwarteschlange für eine Aliaswarteschlange.

Welche Optionen beim Öffnen der lokalen Warteschlange oder der Zielwarteschlange für eine Aliaswarteschlange verwendet werden, hängt von der verwendeten Version der IBM MQ classes for JMS ab:

- Für IBM MQ classes for JMS für IBM MQ 9.1.0 Fix Pack 1 und früher oder IBM MQ 9.1.1 gilt:
  Wenn die lokale Warteschlange oder die Zielwarteschlange für eine Aliaswarteschlange eine Clusterwarteschlange ist, öffnen die IBM MQ classes for JMS die Warteschlange mit den Optionen `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` und `MQOO_FAIL_IF_QUIESCING`. Dies bedeutet, dass der Benutzer, der die empfangende Anwendung ausführt, über Abfrage- und Abrufzugriff auf die lokale Instanz der Clusterwarteschlange verfügen muss.

  Die IBM MQ classes for JMS öffnen alle anderen lokalen Warteschlangentypen mit den Optionen `MQOO_INQUIRE` und `MQOO_FAIL_IF_QUIESCING`. Damit die IBM MQ classes for JMS die Werte der Attribute abfragen können, muss der Benutzer, der die empfangende Anwendung ausführt, über Abfragezugriff auf die lokale Warteschlange verfügen.

- Bei Verwendung von IBM MQ classes for JMS für IBM MQ 9.1.0 Fix Pack 2 und höher oder für IBM MQ 9.1.2 und höher muss der Benutzer, der die empfangende Anwendung ausführt, über Abfragezugriff auf die lokale Warteschlange verfügen, egal um welchen Warteschlangentyp es sich handelt.

Wenn Sie nicht verarbeitbare Nachrichten in eine Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten oder in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers verschieben möchten, müssen Sie dem Benutzer, der die Anwendung ausführt, die Berechtigungen `put` und `passall` erteilen.

Behandlung nicht verarbeitbarer Nachrichten für synchrone Anwendungen

Wenn eine Anwendung Nachrichten synchron empfängt, indem sie eine der folgenden Methoden aufruft, stellen die IBM MQ classes for JMS eine nicht verarbeitbare Nachricht innerhalb der Arbeitseinheit, die beim Versuch der Anwendung, die Nachricht abzurufen, aktiv war, erneut in die Warteschlange:

- JMSConsumer.receive()
- JMSConsumer.receive(long timeout)
- JMSConsumer.receiveBody(Class<T> c)
- JMSConsumer.receiveBody(Class<T> c, long timeout)
- JMSConsumer.receiveBodyNoWait Class<T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive()
- MessageConsumer.receive(long timeout)
- MessageConsumer.receiveNoWait()
- QueueReceiver.receive()
- QueueReceiver.receive(long timeout)
- QueueReceiver.receiveNoWait()
- TopicSubscriber.receive()
- TopicSubscriber.receive(long timeout)
- TopicSubscriber.receiveNoWait()

Falls die Anwendung einen JMS-Kontext oder eine JMS-Sitzung mit Transaktionsunterstützung verwendet, bedeutet dies, dass die Verschiebung der Nachricht in die Rücksetzwarteschlange erst festgeschrieben wird, wenn die Transaktion abgeschlossen ist.

Wenn das Attribut **BOTHRESH** auf einen anderen Wert als null gesetzt wird, sollte auch das Attribut **BOQNAME** festgelegt werden. Wird **BOTHRESH** auf einen Wert größer als null gesetzt, das Attribut **BOQNAME** aber nicht festgelegt, wird das Verhalten durch die Berichtsoptionen der Nachricht bestimmt:

- Hat die Nachricht die Berichtsoption `MQRO_DISCARD_MSG`, wird die Nachricht verworfen.
- Wenn für die Nachricht die Berichtsoption `MQRO_DEAD_LETTER_Q` angegeben ist, versuchen die IBM MQ classes for JMS , die Nachricht in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers zu verschieben.
- Ist für die Nachricht weder `MQRO_DISCARD_MSG` noch `MQRO_DEAD_LETTER_Q` angegeben, versuchen die IBM MQ classes for JMS, die Nachricht in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers zu stellen.

Was mit der Nachricht passiert, falls der Versuch zum Einreihen der Nachricht in die Warteschlange für nicht zustellbare Nachrichten aus irgendeinem Grund fehlschlägt, ist davon abhängig, ob die empfangende Anwendung einen JMS-Kontext oder eine JMS-Sitzung mit oder ohne Transaktionsunterstützung verwendet:

- Wenn die empfangende Anwendung einen JMS-Kontext oder eine JMS-Sitzung mit Transaktionsunterstützung verwendet und die Transaktion festgeschrieben wird, wird die Nachricht verworfen.

- Wenn die empfangende Anwendung einen JMS-Kontext oder eine JMS-Sitzung mit Transaktionsunterstützung verwendet und die Transaktion zurückgesetzt wird, wird die Nachricht an die Eingabewarteschlange zurückgegeben.
- Wenn die empfangende Anwendung einen JMS-Kontext oder eine JMS-Sitzung ohne Transaktionsunterstützung erstellt hat, wird die Nachricht verworfen.

Behandlung nicht verarbeitbarer Nachrichten für asynchrone Anwendungen

Wenn eine Anwendung Nachrichten asynchron über einen Nachrichtenlistener (MessageListener) empfängt, stellen die IBM MQ classes for JMS nicht verarbeitbare Nachrichten erneut in die Warteschlange, ohne dass sich dies auf die Nachrichtenübermittlung auswirkt. Der Prozess des erneuten Einreihens in die Warteschlange findet nicht in einer Arbeitseinheit statt, die der eigentlichen Nachrichtenübermittlung an die Anwendung zugeordnet ist.

Wird **BOTHRESH** auf einen Wert größer als null gesetzt, das Attribut **BOQNAME** aber nicht festgelegt, wird das Verhalten durch die Berichtsoptionen der Nachricht bestimmt:

- Hat die Nachricht die Berichtsoption `MQRO_DISCARD_MSG`, wird die Nachricht verworfen.
- Wenn für die Nachricht die Berichtsoption `MQRO_DEAD_LETTER_Q` angegeben ist, versuchen die IBM MQ classes for JMS , die Nachricht in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers zu verschieben.
- Ist für die Nachricht weder `MQRO_DISCARD_MSG` noch `MQRO_DEAD_LETTER_Q` angegeben, versuchen die IBM MQ classes for JMS, die Nachricht in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers zu stellen.

Wenn der Versuch, die Nachricht in die Warteschlange für nicht zustellbare Nachrichten einzureihen, aus irgendeinem Grund fehlschlägt, geben die IBM MQ classes for JMS die Nachricht an die Eingabewarteschlange zurück.

Informationen dazu, wie Aktivierungsspezifikationen und Verbindungskonsumenten (ConnectionConsumers) nicht verarbeitbare Nachricht behandeln, finden Sie im Abschnitt Nachrichten aus der Warteschlange in ASF entfernen.

Was mit einer Nachricht passiert, wenn sie in die Rücksetzwarteschlange verschoben wird

Wenn eine nicht verarbeitbare Nachricht erneut in die Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten gestellt wird, fügen die IBM MQ classes for JMS einen RFH2-Header zur Nachricht hinzu (falls noch nicht vorhanden) und aktualisieren einige der Felder im Nachrichtendeskriptor (MQMD).

Wenn die nicht verarbeitbare Nachricht bereits einen RFH2-Header enthält (weil es sich beispielsweise um eine JMS-Nachricht handelt), ändern die IBM MQ classes for JMS folgende Felder im MQMD, wenn die Nachricht in die Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten verschoben wird:

- Das Feld 'BackoutCount' wird auf null zurückgesetzt.
- Das Feld 'Expiry' der Nachricht wird aktualisiert, um die verbleibende Ablaufzeit zu dem Zeitpunkt, an dem die nicht verarbeitbare Nachricht von der JMS-Anwendung empfangen wurde, wiederzugeben.

Wenn die nicht verarbeitbare Nachricht keinen RFH2-Header enthält, fügen die IBM MQ classes for JMS einen hinzu und aktualisieren folgende Felder im MQMD im Rahmen der Rücksetzungsverarbeitung:

- Das Feld 'BackoutCount' wird auf null zurückgesetzt.
- Das Feld 'Expiry' der Nachricht wird aktualisiert, um die verbleibende Ablaufzeit zu dem Zeitpunkt, an dem die nicht verarbeitbare Nachricht von der JMS-Anwendung empfangen wurde, wiederzugeben.
- Der Wert des Feldes 'Format' der Nachricht wird in MQHRF2 geändert.
- Der Wert des Feldes 'CCSID' wird in 1208 geändert.
- Der Wert des Feldes 'Encoding' wird in 273 geändert.

Darüber hinaus werden die Felder 'CCSID' und 'Encoding' aus der nicht verarbeitbaren Nachricht in die Felder 'CCSID' und 'Encoding' des RFH2-Headers kopiert, um sicherzustellen, dass die Headerverkettung der Nachricht in der Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten richtig ist.

Zugehörige Konzepte

„Behandlung nicht verarbeitbarer Nachrichten in ASF" auf Seite 354
Innerhalb der Anwendungsserverfunktionen (Application Server Facilities, ASF) weicht die Behandlung nicht verarbeitbarer Nachrichten geringfügig von der sonstigen Vorgehensweise in IBM MQ classes for JMS ab.

Ausnahmebedingungen in IBM MQ classes for JMS

Eine Anwendung der IBM MQ classes for JMS muss Ausnahmebedingungen behandeln, die von einem JMS-API-Aufruf ausgelöst oder an eine Ausnahmebehandlungsroutine übermittelt werden.

IBM MQ classes for JMS melden Laufzeitprobleme durch die Auslösung von Ausnahmebedingungen. Der Typ der Ausnahmebedingungen, die ausgelöst werden, und die Art und Weise, wie diese Ausnahmebedingungen behandelt werden müssen, hängt von der Version der JMS-Spezifikation ab, die von Ihrer Anwendung verwendet wird:

- Methoden in den Schnittstellen, die in JMS 1.1 und früher definiert sind, lösen geprüfte Ausnahmen aus. Die Basisklasse für diese Ausnahmen ist `JMSException`. Weitere Informationen zur Behandlung geprüfter Ausnahmen finden Sie im Abschnitt „Behandlung von geprüften Ausnahmen" auf Seite 248.
- Methoden in den in JMS 2.0 hinzugefügten Schnittstellen lösen ungeprüfte Ausnahmebedingungen aus. Die Basisklasse für diese Ausnahmen ist `JMSRuntimeException`. Weitere Informationen zur Behandlung ungeprüfter Ausnahmen finden Sie im Abschnitt „Behandlung von ungeprüften Ausnahmen" auf Seite 252.

Sie können auch einen `ExceptionListener` bei einem JMS-Objekt `Connection` oder `JMSContext` registrieren. Die MQ-Klassen für JMS benachrichtigen dann den `ExceptionListener`, wenn entweder ein Problem bei einer Verbindung mit dem Warteschlangenmanager erkannt wird oder wenn beim Versuch, eine Nachricht asynchron zuzustellen, ein Problem auftritt. Weitere Informationen finden Sie unter „ExceptionListeners" auf Seite 255.

Zugehörige Konzepte

IBM MQ-Klassen für JMS

Zugehörige Verweise

ASYNCEXCEPTION

Behandlung von geprüften Ausnahmen

Methoden in den Schnittstellen, die in JMS 1.1 oder früher definiert sind, lösen geprüfte Ausnahmen aus. Die Basisklasse für diese Ausnahmen ist `JMSException`. Daher bietet das Abfangen von `JMSExceptions` eine generische Möglichkeit, diese Typen von Ausnahmen zu behandeln.

Jede `JMSException` enthält die folgenden Informationen:

- Eine providerspezifische Ausnahmebedingungsnachricht, die Ihre Anwendung durch Aufrufen der Methode `Throwable.getMessage()` abrufen kann.
- Ein providerspezifischer Fehlercode, den Ihre Anwendung durch Aufrufen der Methode `JMSException.getErrorCode()` abrufen kann.
- Eine verlinkte Ausnahmebedingung. Eine von einem JMS 1.1-API-Aufruf ausgelöste Ausnahmebedingung ist häufig das Ergebnis eines Problems auf niedrigerer Ebene, das von einer anderen Ausnahmebedingung gemeldet wird, die mit dieser Ausnahmebedingung verlinkt ist. Ihre Anwendung kann eine verlinkte Ausnahmebedingung abrufen, indem sie entweder die Methode `JMSException.getLinkedException()` oder die Methode `Throwable.getCause()` aufruft.

Wenn Sie die JMS 1.1-API verwenden, sind die meisten Ausnahmebedingungen, die von IBM MQ classes for JMS ausgelöst werden, Instanzen von Unterklassen von `JMSException`. Diese Unterklassen implementieren die Schnittstelle `com.ibm.msg.client.jms.JmsExceptionDetail`, die die folgenden zusätzlichen Informationen bereitstellt:

- Eine Erläuterung der Ausnahmebedingungsnachricht. Ihre Anwendung kann diese Nachricht durch Aufrufen der Methode `JmsExceptionDetail.getExplanation()` abrufen.
- Eine empfohlene Benutzeraktion für die Ausnahmebedingung. Ihre Anwendung kann diese Nachricht durch Aufrufen der Methode `JmsExceptionDetail.getUserAction()` abrufen.
- Die Schlüssel für die Einfügungen in der Ausnahmebedingungsnachricht. Ihre Anwendung kann einen Iterator für alle Schlüssel durch Aufrufen der Methode `JmsExceptionDetail.getKeys()` abrufen.
- Die Nachrichteneinfügungen in der Ausnahmebedingungsnachricht. Eine Nachrichteneinfügung kann zum Beispiel der Name der Warteschlange sein, die die Ausnahmebedingung verursacht hat, und es kann nützlich für Ihre Anwendung sein, auf diesen Namen zuzugreifen. Ihre Anwendung kann die Nachrichteneinfügung, die einem angegebenen Schlüssel entspricht, durch Aufrufen der Methode `JmsExceptionDetail.getValue()` abrufen.

Alle Methoden in der Schnittstelle `JmsExceptionDetail` geben null zurück, wenn keine Details verfügbar sind.

Wenn eine Anwendung beispielsweise versucht, einen Nachrichtenproduzenten für eine nicht vorhandene IBM MQ-Warteschlange zu erstellen, wird eine Ausnahmebedingung mit folgenden Informationen ausgelöst:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

Die ausgelöste Ausnahmebedingung `com.ibm.msg.client.jms.DetailedInvalidDestinationException`ist eine Unterklasse der folgenden Klasse und implementiert die Schnittstelle `com.ibm.msg.client.jms.JmsExceptionDetail` .

- JM 3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Verlinkte Ausnahmebedingungen

Eine verlinkte Ausnahmebedingung liefert weitere Informationen zu einem Laufzeitproblem. Daher sollte eine Anwendung für jede `JMSException`, die ausgelöst wird, die verknüpfte Ausnahme überprüfen.

Die verlinkte Ausnahmebedingung kann wiederum ihrerseits eine weitere verlinkte Ausnahmebedingung haben, sodass die verlinkten Ausnahmebedingungen eine Kette bilden, die zu dem zugrunde liegenden Problem zurückführt. Eine verlinkte Ausnahmebedingung wird mithilfe des Mechanismus für verkettete Ausnahmebedingungen der Klasse `java.lang.Throwable` implementiert, und Ihre Anwendung kann eine verlinkte Ausnahmebedingung durch Aufrufen der Methode `Throwable.getCause()` abrufen. Bei einem `JMSException` delegiert die Methode `getLinkedException()` an die Methode `Throwable.getCause()`.

Wenn eine Anwendung beispielsweise bei der Verbindung mit einem Warteschlangenmanager eine falsche Portnummer angibt, bilden die Ausnahmebedingungen die folgende Kette:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
```

```
            +--->
                java.net.ConnectionException
```

Normalerweise wird jede Ausnahmebedingung in einer Kette von einer anderen Schicht im Code ausgelöst. In der oben genannten Kette wurden die Ausnahmebedingungen beispielsweise von folgenden Schichten ausgelöst:

- Die erste Ausnahme, eine Instanz einer Unterklasse von `JMSException`, wird von der allgemeinen Schicht in IBM MQ classes for JMSausgelöst.
- Die nächste Ausnahme, eine Instanz von `com.ibm.mq.MQException`, wird vom IBM MQ-Messaging-Provider ausgelöst.
- Die nächsten beiden Ausnahmen, die beide Instanzen von `com.ibm.mq.jmqi.JmqiExceptionsind`, werden von der Java Message Queueing Interface (JMQI) ausgelöst. JMQI ist die Komponente, die von den IBM MQ classes for JMS für die Kommunikation mit einem Warteschlangenmanager verwendet wird.
- Die letzte Ausnahmebedingung, eine Instanz von `java.net.ConnectionException`, wird von der Java-Klassenbibliothek ausgelöst.

Weitere Informationen zu der Schichtarchitektur von IBM MQ classes for JMS finden Sie im Abschnitt IBM MQ-Klassen für JMS-Architektur.

Sie können Ihre Anwendung so codieren, dass diese Kette durchlaufen wird, um alle geeigneten Informationen zu extrahieren, wie im folgenden Beispiel gezeigt wird:

JM 3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
```

```
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Beachten Sie, dass Ihre Anwendung immer den Typ jeder einzelnen Ausnahmebedingung in einer Kette überprüfen sollte, da der Typ variieren kann und verschiedene Ausnahmebedingungstypen verschiedene Informationen einbinden.

IBM MQ-spezifische Informationen zu einem Problem abrufen

Instanzen von `com.ibm.mq.MQException` und `com.ibm.mq.jmqi.JmqiException` kapseln IBM MQ spezifische Informationen zu einem Problem.

Ein `MQException` enthält die folgenden Informationen:

- Einen Beendigungscode, den Ihre Anwendung durch Aufrufen der Methode `getCompCode()` abrufen kann.
- Einen Ursachencode, den Ihre Anwendung durch Aufrufen der Methode `getReason()` abrufen kann.

Beispiele für die Verwendung dieser Methoden finden Sie im Beispielcode in verlinkte Ausnahmebedingungen.

Ein `JmqiException` kapselt auch einen Beendigungscode und einen Ursachencode. Darüber hinaus enthält ein `JmqiException` die Informationen in einer AMQ-*Nnnn* oder CSQ-*Nnnn* Nachricht, wenn der Ausnahmebedingung eine zugeordnet ist. Ihre Anwendung kann die verschiedenen Komponenten dieser Nachricht abrufen, indem Sie folgende Methoden aufruft:

- Die Methode `getWmqMsgExplanation()` gibt die Erläuterung der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zurück.
- Die Methode `getWmqMsgSeverity()` gibt die Wertigkeit der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zurück.
- Die Methode `getWmqMsgSummary()` gibt die Zusammenfassung der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zurück.

- Die Methode `getWmqMsgUserResponse()` gibt die Benutzeraktion zurück, die der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zugeordnet ist.

Behandlung von ungeprüften Ausnahmen
Methoden in den Schnittstellen, die in JMS 2.0 definiert sind, lösen geprüfte Ausnahmen aus. Die Basisklasse für diese Ausnahmen ist `JMSRuntimeException`. Daher bietet das Abfangen von `JMSRuntimeExceptions` eine generische Möglichkeit, diese Typen von Ausnahmen zu behandeln.

Jede `JMSRuntimeException` enthält die folgenden Informationen:

- Eine providerspezifische Ausnahmebedingungsnachricht, die Ihre Anwendung durch Aufrufen der Methode `JMSRuntimeException.getMessage()` abrufen kann.
- Ein providerspezifischer Fehlercode, den Ihre Anwendung durch Aufrufen der Methode `JMSRuntimeException.getErrorCode()` abrufen kann.
- Eine verlinkte Ausnahmebedingung. Eine von einem JMS 2.0-API-Aufruf ausgelöste Ausnahmebedingung ist häufig das Ergebnis eines Problems auf niedrigerer Ebene, das von einer anderen Ausnahmebedingung gemeldet wird, die mit dieser Ausnahmebedingung verlinkt ist. Ihre Anwendung kann eine verknüpfte Ausnahme abrufen, indem sie die Methode `JMSRuntimeException.getCause()` aufruft.

Wenn Sie Methoden in den Schnittstellen aufrufen, die von der JMS 2.0 API bereitgestellt werden, sind die meisten Ausnahmen, die von IBM MQ classes for JMS ausgelöst werden, Instanzen von Unterklassen von `JMSRuntimeException`. Diese Unterklassen implementieren die Schnittstelle `com.ibm.msg.client.jms.JmsExceptionDetail`, die folgende zusätzliche Informationen bereitstellt:

- Eine Erläuterung der Ausnahmebedingungsnachricht. Ihre Anwendung kann diese Nachricht durch Aufrufen der Methode `JmsExceptionDetail.getExplanation()` abrufen.
- Eine empfohlene Benutzeraktion für die Ausnahmebedingung. Ihre Anwendung kann diese Nachricht durch Aufrufen der Methode `JmsExceptionDetail.getUserAction()` abrufen.
- Die Schlüssel für die Einfügungen in der Ausnahmebedingungsnachricht. Ihre Anwendung kann einen Iterator für alle Schlüssel durch Aufrufen der Methode `JmsExceptionDetail.getKeys()` abrufen.
- Die Nachrichteneinfügungen in der Ausnahmebedingungsnachricht. Eine Nachrichteneinfügung kann zum Beispiel der Name der Warteschlange sein, die die Ausnahmebedingung verursacht hat, und es kann nützlich für Ihre Anwendung sein, auf diesen Namen zuzugreifen. Ihre Anwendung kann die Nachrichteneinfügung, die einem angegebenen Schlüssel entspricht, durch Aufrufen der Methode `JmsExceptionDetail.getValue()` abrufen.

Alle Methoden in der Schnittstelle `JmsExceptionDetail` geben null zurück, wenn keine Details verfügbar sind.

Wenn eine Anwendung beispielsweise versucht, eine `JMSProducer` für eine IBM MQ-Warteschlange zu erstellen, die nicht vorhanden ist, wird eine Ausnahme mit den folgenden Informationen ausgelöst:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

Die ausgelöste Ausnahmebedingung `com.ibm.msg.client.jms.DetailedInvalidDestinationException` ist eine Unterklasse der folgenden Klasse und implementiert die Schnittstelle `com.ibm.msg.client.jms.JmsExceptionDetail` .

- JMS 3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Verkettete Ausnahmebedingungen

In der Regel werden Ausnahmebedingungen durch andere Ausnahmebedingungen verursacht. Daher sollte Ihre Anwendung für jede `JMSRuntimeException`, die ausgelöst wird, die verknüpfte Ausnahmebedingung überprüfen.

Die Ursache für `JMSRuntimeException` kann eine weitere Ausnahmebedingung sein. Diese Ausnahmebedingungen bilden eine Kette, die zum ursprünglich zugrunde liegenden Problem zurückführt. Die Ursache einer Ausnahme wird mithilfe des Mechanismus für verkettete Ausnahmen der Klasse `java.lang.Throwable` implementiert und Ihre Anwendung kann eine verknüpfte Ausnahme abrufen, indem sie die Methode `Throwable.getCause()` aufruft.

Wenn eine Anwendung beispielsweise bei der Verbindung mit einem Warteschlangenmanager eine falsche Portnummer angibt, bilden die Ausnahmebedingungen die folgende Kette:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Normalerweise wird jede Ausnahmebedingung in einer Kette von einer anderen Schicht im Code ausgelöst. In der oben genannten Kette wurden die Ausnahmebedingungen beispielsweise von folgenden Schichten ausgelöst:

- Die erste Ausnahme, eine Instanz einer Unterklasse von `JMSRuntimeException`, wird von der allgemeinen Schicht in IBM MQ classes for JMSausgelöst.
- Die nächste Ausnahme, eine Instanz von `com.ibm.mq.MQException`, wird vom IBM MQ-Messaging-Provider ausgelöst.
- Die nächsten beiden Ausnahmen, die beide Instanzen von `com.ibm.mq.jmqi.JmqiExceptionsind`, werden von der Java Message Queueing Interface (JMQI) ausgelöst. JMQI ist die Komponente, die von den IBM MQ classes for JMS für die Kommunikation mit einem Warteschlangenmanager verwendet wird.
- Die letzte Ausnahmebedingung, eine Instanz von `java.net.ConnectionException`, wird von der Java-Klassenbibliothek ausgelöst.

Weitere Informationen zu der Schichtarchitektur von IBM MQ classes for JMS finden Sie im Abschnitt IBM MQ-Klassen für JMS-Architektur.

Sie können Ihre Anwendung so codieren, dass diese Kette durchlaufen wird, um alle geeigneten Informationen zu extrahieren, wie im folgenden Beispiel gezeigt wird:

JM 3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);
```

```
    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Beachten Sie, dass Ihre Anwendung immer den Typ jeder einzelnen Ausnahmebedingung in einer Kette überprüfen sollte, da der Typ variieren kann und verschiedene Ausnahmebedingungstypen verschiedene Informationen einbinden.

IBM MQ-spezifische Informationen zu einem Problem abrufen

Instanzen von `com.ibm.mq.MQException` und `com.ibm.mq.jmqi.JmqiException` kapseln IBM MQ spezifische Informationen zu einem Problem.

Ein `MQException` enthält die folgenden Informationen:

- Einen Beendigungscode, den Ihre Anwendung durch Aufrufen der Methode `getCompCode()` abrufen kann.
- Einen Ursachencode, den Ihre Anwendung durch Aufrufen der Methode `getReason()` abrufen kann.

Beispiele für die Verwendung dieser Methoden finden Sie im Beispielcode in verkettete Ausnahmebedingungen.

Ein `JmqiException` kapselt auch einen Beendigungscode und einen Ursachencode. Darüber hinaus enthält ein `JmqiException` die Informationen in einer AMQ-*Nnnn* oder CSQ-*Nnnn* Nachricht, wenn der Ausnahmebedingung eine zugeordnet ist. Ihre Anwendung kann die verschiedenen Komponenten dieser Nachricht abrufen, indem Sie folgende Methoden aufruft:

- Die Methode `getWmqMsgExplanation()` gibt die Erläuterung der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zurück.
- Die Methode `getWmqMsgSeverity()` gibt die Wertigkeit der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zurück.
- Die Methode `getWmqMsgSummary()` gibt die Zusammenfassung der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zurück.
- Die Methode `getWmqMsgUserResponse()` gibt die Benutzeraktion zurück, die der Nachricht AMQ-*Nnnn* oder CSQ-*Nnnn* zugeordnet ist.

ExceptionListeners
JMS-`Connection` und `JMSContext`-Objekte verfügen über eine zugeordnete Verbindung zu einem Warteschlangenmanager. Ihre Anwendung kann einen `ExceptionListener` bei einem JMS `Connection` oder`JMSContext` registrieren. Wenn ein Problem auftritt, das die Verbindung, die dem `Connection` oder `JMSContext` zugeordnet ist, unbrauchbar macht, stellt der IBM MQ classes for JMS eine Ausnahme an den `ExceptionListener` durch Aufrufen der Methode `onException()` bereit. Ihre Anwendung hat dann Gelegenheit, die Verbindung wiederherzustellen.

IBM MQ classes for JMS können auch eine Ausnahmebedingung an den Listener für Ausnahmebedingungen übermitteln, wenn während des Versuchs, eine Nachricht asynchron zu übermitteln, ein Problem auftritt.

Listener für Ausnahmebedingungen

Ab IBM MQ 8.0.0 Fix Pack 2 wird der Standardwert für die `ConnectionFactory`-Eigenschaft ASYNCHRONE Ausnahmebedingung in ASYNC_EXCEPTIONS_CONNECTIONBROKEN geändert, um das Verhalten von aktuellen JMS-Anwendungen, die einen JMS `MessageListener` und einen JMS `ExceptionListener` konfigurieren, beizubehalten und sicherzustellen, dass die IBM MQ classes for JMS mit der JMS-Spezifikation konsistent sind. Daher werden nur Ausnahmen, die Fehlercodes für unterbrochene Verbindungen entsprechen, an die `ExceptionListener` einer Anwendung zugestellt.

Das APAR IT14820, das ab IBM MQ 9.0.0 Fix Pack 1 enthalten ist, aktualisiert IBM MQ classes for JMS. Damit wird Folgendes erreicht:

- Ein `ExceptionListener`, der von einer Anwendung registriert wird, wird für alle Ausnahmebedingungen des Typs "Verbindung unterbrochen" aufgerufen, unabhängig davon, ob die Anwendung synchrone oder asynchrone Nachrichtenkonsumenten verwendet.
- Ausnahmebedingungen ohne Verbindungsunterbrechung (z. B. MQRC_GET_INHIBITED), die während der Nachrichtenübermittlung auftreten, werden an die `ExceptionListener` einer Anwendung zugestellt, wenn die Anwendung asynchrone Nachrichtenkonsumenten verwendet und die von der Anwendung verwendete JMS-ConnectionFactory die Eigenschaft ASYNC_EXCEPTION auf den Wert ASYNC_EXCEPTIONS_ALL gesetzt hat.

Anmerkung: Ein `ExceptionListener` wird nur einmal für eine Ausnahmebedingung wegen Verbindungsunterbrechung aufgerufen, selbst wenn zwei TCP/IP-Verbindungen (eine von einer JMS-Verbindung und eine von einer JMS-Sitzung) unterbrochen werden.

Bei allen anderen Arten von Problemen löst der aktuelle JMS-API-Aufruf eine Ausnahmebedingung aus. Welcher Ausnahmebedingungstyp ausgelöst wird, hängt von der Version der JMS-API ab, die von der Anwendung verwendet wird:

- Wenn die Anwendung die Schnittstellen verwendet, die von der Spezifikation JMS 1.1 bereitgestellt werden, ist die Ausnahme `JMSException`. Weitere Informationen zur Behandlung dieser Ausnahmebedingungen finden Sie im Abschnitt „Behandlung von geprüften Ausnahmen” auf Seite 248.
- Wenn die Anwendung JMS 2.0-Schnittstellen verwendet, ist die Ausnahme `JMSRuntimeException`. Weitere Informationen zur Behandlung dieser Ausnahmebedingungen finden Sie im Abschnitt „Behandlung von ungeprüften Ausnahmen” auf Seite 252.

Wenn eine Anwendung keinen Listener für Ausnahmebedingungen bei `Connection` oder `JMSContext` registriert, werden alle Ausnahmebedingungen, die an den Listener für Ausnahmebedingungen übermittelt würden, in das Protokoll IBM MQ classes for JMS geschrieben.

Über eine IBM MQ classes for JMS -Anwendung auf IBM MQ -Funktionen zugreifen

IBM MQ classes for JMS stellen Einrichtungen bereit, mit denen mehrere Funktionen von IBM MQ genutzt werden können.

Achtung: Diese Funktionen sind nicht in der JMS-Spezifikation abgedeckt oder verstoßen gegen die JMS-Spezifikation. Wenn Sie sie verwenden, ist Ihre Anwendung wahrscheinlich nicht mit anderen JMS-Providern kompatibel. Funktionen, die die JMS-Spezifikation nicht einhalten, sind mit dem Hinweis 'Achtung' versehen.

Nachrichtendeskriptor über eine Anwendung der IBM MQ classes for JMS lesen und schreiben

Sie steuern die Fähigkeit, auf den Nachrichtendeskriptor (MQMD) zuzugreifen, indem Sie entsprechende Eigenschaften für ein Ziel (Destination) und eine Nachricht (Message) festlegen.

Bei manchen IBM MQ-Anwendungen müssen im MQMD der an sie gesendeten Nachrichten bestimmte Werte festgelegt sein. IBM MQ classes for JMS stellt Nachrichtenattribute zur Verfügung, die es JMS -Anwendungen ermöglichen, MQMD-Felder festzulegen und so JMS -Anwendungen zu ermöglichen, IBM MQ -Anwendungen zu "steuern".

Sie müssen die Eigenschaft WMQ_MQMD_WRITE_ENABLED des Destination-Objekts auf 'true' setzen, damit die Einstellung der MQMD-Eigenschaften wirksam ist. Sie können dann die Methoden für Eigenschafteneinstellungen der Nachricht (zum Beispiel 'setStringProperty') verwenden, um den MQMD-Feldern Werte zuzuweisen. Mit Ausnahme von 'StrucId' und 'Version' sind alle MQMD-Felder verfügbar; 'BackoutCount' kann gelesen werden, es sind jedoch keine Schreibvorgänge darin möglich.

In diesem Beispiel wird eine Nachricht in eine Warteschlange oder ein Thema gestellt, wobei MQMD.UserIdentifier auf "JoeBloggs" gesetzt ist.

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

WMQ_MQMD_MESSAGE_CONTEXT muss vor der Festlegung von JMS_IBM_MQMD_UserIdentifier festgelegt werden. Weitere Informationen zur Verwendung von WMQ_MQMD_MESSAGE_CONTEXT finden Sie im Abschnitt „Eigenschaften von JMS-Nachrichtenobjekten” auf Seite 259.

Auf ähnliche Weise können Sie den Inhalt der MQMD-Felder extrahieren, indem Sie vor dem Empfang einer Nachricht WMQ_MQMD_READ_ENABLED auf 'true' setzen und anschließend die Abrufmethoden der Nachrichten verwenden (beispielsweise 'getStringProperty'). Alle empfangenen Eigenschaften sind schreibgeschützt.

Bei diesem Beispiel wird im Feld *value* der Wert des MQMD.ApplIdentityData-Felds einer Nachricht gespeichert, die aus einer Warteschlange oder aus einem Thema abgerufen wird.

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...

// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

Eigenschaften von JMS-Zielobjekten
Zwei Eigenschaften des Zielobjekts (Destination) steuern den Zugriff auf den MQMD über JMS, während eine dritte Eigenschaft den Nachrichtenkontext steuert.

Tabelle 36. Eigenschaftsnamen und Beschreibungen

Eigenschaft	**Kurzform**	**Beschreibung**
WMQ_MQMD_WRITE_ENABLED	MDW	Gibt an, ob eine JMS-Anwendung die Werte von MQMD-Feldern festlegen kann
WMQ_MQMD_READ_ENABLED	MDR	Gibt an, ob eine JMS-Anwendung die Werte von MQMD-Feldern extrahieren kann
WMQ_MQMD_MESSAGE _CONTEXT	MDCTX	Gibt an, welche Ebene von Nachrichtenkontext durch die JMS-Anwendung gesetzt werden soll. Die Anwendung muss mit entsprechender Kontextberechtigung ausgeführt werden, damit diese Eigenschaft in Kraft treten kann.

Tabelle 37. Eigenschaftsnamen, Werte und Festlegungsmethoden

Eigenschaft	Gültige Werte im Verwaltungstool (Standardwerte sind fett dargestellt)	Gültige Werte in Programmen	Festlegungsmethode
WMQ_MQMD_WRITE_ENABLED	• **NO** Alle JMS_IBM_MQMD*-Eigenschaften werden ignoriert; ihre Werte werden nicht in die zugrunde liegende MQMD-Struktur kopiert. • JA Die JMS_IBM_MQMD*-Eigenschaften werden verarbeitet. Ihre Werte werden in die zugrunde liegende MQMD-Struktur kopiert.	• **False** • True	setMQMDWriteEnabled
WMQ_MQMD_READ_ENABLED	• **NO** Beim Senden von Nachrichten werden die JMS_IBM_MQMD*-Eigenschaften einer gesendeten Nachricht nicht mit den aktualisierten Feldwerten in der MQMD-Struktur aktualisiert. Beim Empfangen von Nachrichten ist keine der JMS_IBM_MQMD*-Eigenschaften in einer empfangenen Nachricht verfügbar, auch wenn der Absender einige oder alle dieser Eigenschaften festgelegt hatte. • JA Beim Senden von Nachrichten werden alle der JMS_IBM_MQMD*-Eigenschaften in einer gesendeten Nachricht mit den aktualisierten Feldwerten im MQMD aktualisiert. Dies gilt auch für Werte, die nicht explizit vom Absender festgelegt wurden. Beim Empfangen von Nachrichten sind alle JMS_IBM_MQMD-Eigenschaften in einer empfangenen Nachricht verfügbar. Dies gilt auch für diejenigen, die nicht explizit vom Absender festgelegt wurden.	• **False** • True	setMQMDReadEnabled

Tabelle 37. Eigenschaftsnamen, Werte und Festlegungsmethoden (Forts.)

Eigenschaft	Gültige Werte im Verwaltungstool (Standardwerte sind fett dargestellt)	Gültige Werte in Programmen	Festlegungsmethode
WMQ_MQMD _MESSAGE_CONTEXT	• **Standard** Der API-Aufruf MQOPEN und die MQPMO-Struktur geben keine expliziten Nachrichtenkontextoptionen an. • SET_IDENTITY_CONTEXT Der API-Aufruf MQOPEN gibt die Nachrichtenkontextoption MQOO_SET_IDENTITY_CONTEXT an. Die MQPMO-Struktur gibt MQPMO_SET_IDENTITY_CONTEXT an. • SET_ALL_CONTEXT Der API-Aufruf MQOPEN gibt die Nachrichtenkontextoption MQOO_SET_ALL_CONTEXT an. Die MQPMO-Struktur gibt MQPMO_SET_ALL_CONTEXT an.	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessageContext

Eigenschaften von JMS-Nachrichtenobjekten
Mit den Nachrichtenobjekteigenschaften mit dem Präfix JMS_IBM_MQMD können Sie den Wert des entsprechenden MQMD-Felds festlegen oder anzeigen.

Nachrichten senden

Mit Ausnahme von StrucId und Version werden alle MQMD-Felder dargestellt. Diese Eigenschaften beziehen sich nur auf die MQMD-Felder; kommt eine Eigenschaft sowohl im MQMD- als auch im MQRFH2-Header vor, wird die Version im MQRFH2 nicht festgelegt oder extrahiert.

Mit Ausnahme von JMS_IBM_MQMD_BackoutCount kann jede dieser Eigenschaften festgelegt werden. Ein für JMS_IBM_MQMD_BackoutCount festgelegter Wert wird ignoriert.

Wenn eine Eigenschaft eine maximale Länge hat und Sie einen zu langen Wert angeben, wird der Wert abgeschnitten.

Bei bestimmten Eigenschaften müssen Sie auch die Eigenschaft WMQ_MQMD_MESSAGE_CONTEXT im Zielobjekt (Destination) festlegen. Die Anwendung muss mit entsprechender Kontextberechtigung ausgeführt werden, damit diese Eigenschaft in Kraft treten kann. Wenn Sie WMQ_MQMD_MESSAGE_CONTEXT nicht auf einen geeigneten Wert setzen, wird der Eigenschaftswert ignoriert. Wenn Sie für WMQ_MQMD_MESSAGE_CONTEXT einen geeigneten Wert festlegen, Ihre Kontextberechtigung für den Warteschlangenmanager jedoch nicht ausreicht, wird die Ausnahmebedingung JMSException ausgegeben. Bei folgenden Eigenschaften sind bestimmte Werte für WMQ_MQMD_MESSAGE_CONTEXT erforderlich.

Bei den folgenden Eigenschaften muss WMQ_MQMD_MESSAGE_CONTEXT auf WMQ_MDCTX_SET_IDENTITY_CONTEXT oder WMQ_MDCTX_SET_ALL_CONTEXT gesetzt werden:

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

Bei den folgenden Eigenschaften muss WMQ_MQMD_MESSAGE_CONTEXT auf WMQ_MDCTX_SET_ALL_CONTEXT gesetzt werden:

- JMS_IBM_MQMD_PutApplType
- JMS_IBM_MQMD_PutApplName
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- JMS_IBM_MQMD_ApplOriginData

Nachrichten empfangen

Alle diese Eigenschaften sind in einer empfangenen Nachricht verfügbar, wenn WMQ_MQMD_READ_ENABLED auf 'true' gesetzt ist, und zwar unabhängig von den tatsächlichen Eigenschaften, die bei der erstellenden Anwendung festgelegt sind. Gemäß JMS-Spezifikation kann eine Anwendung die Eigenschaften einer empfangenen Nachricht nur dann ändern, wenn zuvor die Werte aller Eigenschaften gelöscht wurden. Die empfangene Nachricht kann ohne Änderung der Eigenschaften weitergeleitet werden.

Achtung: Wenn Ihre Anwendung eine Nachricht von einem Ziel empfängt, bei dem die Eigenschaft WMQ_MQMD_READ_ENABLED auf 'true' gesetzt ist, und diese an ein Ziel weiterleitet, bei dem WMQ_MQMD_WRITE_ENABLED auf 'true' gesetzt ist, führt dies dazu, dass alle MQMD-Feldwerte der empfangenen Nachricht in die weitergeleitete Nachricht kopiert werden.

Eigenschaftentabelle

In dieser Tabelle sind die Eigenschaften des Nachrichtenobjekts (Message) aufgelistet, die die MQMD-Felder darstellen. Über die Links können Sie die ausführlichen Beschreibungen der Felder und ihre zulässigen Werte aufrufen.

Tabelle 38. Eigenschaftsnamen, Beschreibungen und Typen

Eigenschaft	Beschreibung	Java-Typ	Link zur ausführlichen Beschreibung
JMS_IBM_MQMD_Report	Optionen für Berichtsnachrichten	Integer	Bericht
JMS_IBM_MQMD_MsgType	Nachrichtentyp	Integer	MsgType
JMS_IBM_MQMD_Expiry	Nachrichtenlebensdauer	Integer	Verfall
JMS_IBM_MQMD_Feedback	Rückmeldung oder Ursachencode	Integer	Feedback
JMS_IBM_MQMD_Encoding	Numerische Codierung von Nachrichtendaten	Integer	Encoding
JMS_IBM_MQMD_CodedCharSetId	Zeichensatzkennung von Nachrichtendaten	Integer	CodedCharSetId
JMS_IBM_MQMD_Format	Name des Formats von Nachrichtendaten	Zeichenfolge	Format
JMS_IBM_MQMD_Priority [1]	Nachrichtenpriorität	Integer	Priorität
JMS_IBM_MQMD_Persistence	Nachrichtenpersistenz	Integer	Persistenz
JMS_IBM_MQMD_MsgId [2]	Nachrichten-ID	Object (byte[]) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	Korrelations-ID	Object (byte[]) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	Zurücksetzungszähler	Integer	BackoutCount

Tabelle 38. Eigenschaftsnamen, Beschreibungen und Typen (Forts.)

Eigenschaft	Beschreibung	Java-Typ	Link zur ausführlichen Beschreibung
JMS_IBM_MQMD_ReplyToQ	Name der Antwortwarteschlange	Zeichenfolge	ReplyToQ
JMS_IBM_MQMD_ReplyToQMgr	Name des Antwortwarteschlangenmanagers	Zeichenfolge	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	Benutzer-ID	Zeichenfolge	UserIdentifier
JMS_IBM_MQMD_AccountingToken	Abrechnung	Object (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	Anwendungsdaten zur Identität	Zeichenfolge	ApplIdentityData
JMS_IBM_MQMD_PutApplType	Typ der Anwendung, die die Nachricht eingereiht hat	Integer	PutApplType
JMS_IBM_MQMD_PutApplName	Name der Anwendung, die die Nachricht eingereiht hat	Zeichenfolge	PutApplName
JMS_IBM_MQMD_PutDate	Datum, an dem die Nachricht eingereiht wurde	Zeichenfolge	PutDate
JMS_IBM_MQMD_PutTime	Uhrzeit, zu der die Nachricht eingereiht wurde	Zeichenfolge	PutTime
JMS_IBM_MQMD_ApplOriginData	Anwendungsdaten zum Ursprung	Zeichenfolge	ApplOriginData
JMS_IBM_MQMD_GroupId	Gruppen-ID	Object (byte[]) [4]	GroupId
JMS_IBM_MQMD_MsgSeqNumber	Folgenummer einer logischen Nachricht innerhalb einer Gruppe	Integer	MsgSeqNumber
JMS_IBM_MQMD_Offset	Relative Adresse von Daten in einer physischen Nachricht ab dem Anfang der logischen Nachricht	Integer	Offset
JMS_IBM_MQMD_MsgFlags	Nachrichtenmarkierungen	Integer	MsgFlags
JMS_IBM_MQMD_OriginalLength	Länge der ursprünglichen Nachricht	Integer	OriginalLength

1. **Achtung:** Wenn Sie für JMS_IBM_MQMD_Priority einen Wert zuweisen, der nicht im Bereich von 0 bis 9 liegt, verstößt dies gegen die JMS-Spezifikation.
2. **Achtung:** Die JMS-Spezifikation legt fest, dass die Nachrichten-ID vom JMS-Provider festgelegt werden muss. Sie muss entweder eindeutig oder null sein. Wenn Sie für JMS_IBM_MQMD_MsgId einen Wert zuweisen, wird dieser Wert in die JMSMessageID kopiert. Er wird also nicht vom JMS-Provider festgelegt und ist möglicherweise nicht eindeutig: Dies verstößt gegen die JMS-Spezifikation.

3. **Achtung:** Wenn Sie für JMS_IBM_MQMD_CorrelId einen Wert zuweisen, der mit der Zeichenfolge 'ID:' beginnt, verstößt dies gegen die JMS-Spezifikation.

4. **Achtung:** Die Verwendung von Byte-Array-Eigenschaften für eine Nachricht verstößt gegen die JMS-Spezifikation.

Mit IBM MQ classes for JMS auf IBM MQ -Nachrichtendaten aus einer Anwendung zugreifen
Sie können auf die vollständigen IBM MQ -Nachrichtendaten in einer Anwendung mit IBM MQ classes for JMSzugreifen. Damit auf alle Daten zugegriffen werden kann, muss die Nachricht eine JMSBytesMessage sein. Der Hauptteil der JMSBytesMessage enthält einen beliebigen MQRFH2-Header sowie sonstige IBM MQ-Header und die folgenden Nachrichtendaten.

Setzen Sie die Eigenschaft WMQ_MESSAGE_BODY des Ziels auf WMQ_MESSAGE_BODY_MQ, um den gesamten Nachrichteninhalt in der JMSBytesMessage zu empfangen.

Wenn WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_JMS oder WMQ_MESSAGE_BODY_UNSPECIFIED gesetzt ist, wird der Nachrichtenhauptteil ohne den JMS-Header MQRFH2 zurückgegeben und die Eigenschaften der JMSBytesMessage reflektieren die im RFH2 festgelegten Eigenschaften.

Einige Anwendungen können die in diesem Abschnitt beschriebenen Funktionen nicht verwenden. Wenn eine Anwendung mit einem IBM MQ-Warteschlangenmanager der Version 6 verbunden ist oder für sie PROVIDERVERSION auf 6 gesetzt ist, sind die Funktionen nicht verfügbar.

Nachricht senden

Beim Senden von Nachrichten hat die Zieleigenschaft WMQ_MESSAGE_BODY Vorrang vor WMQ_TARGET_CLIENT.

Wenn WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_JMS gesetzt ist, generieren IBM MQ classes for JMS automatisch einen MQRFH2-Header, der auf den Einstellungen der JMSMessage-Eigenschaften und Headerfelder basiert.

Wenn WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_MQ gesetzt ist, wird dem Nachrichtenhauptteil kein zusätzlicher Header hinzugefügt.

Wenn WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_UNSPECIFIED gesetzt ist, senden IBM MQ classes for JMS einen MQRFH2-Header, es sei denn, WMQ_TARGET_CLIENT wurde auf WMQ_TARGET_DEST_MQ gesetzt. Beim Empfang bewirkt das Setzen von WMQ_TARGET_CLIENT auf WMQ_TARGET_DEST_MQ, dass jeder MQRFH2 aus dem Nachrichtenhauptteil entfernt wird.

Anmerkung: JMSBytesMessage und JMSTextMessage benötigen keinen MQRFH2, während er von JMSStreamMessage, JMSMapMessage und JMSObjectMessage benötigt wird.

WMQ_MESSAGE_BODY_UNSPECIFIED ist die Standardeinstellung für WMQ_MESSAGE_BODY und WMQ_TARGET_DEST_JMS ist die Standardeinstellung für WMQ_TARGET_CLIENT.

Wenn Sie eine JMSBytesMessagesenden, können Sie die Standardeinstellungen für den JMS -Nachrichtenhauptteil überschreiben, wenn die Nachricht IBM MQ erstellt wird. Verwenden Sie die folgenden Eigenschaften:

- JMS_IBM_Format oder JMS_IBM_MQMD_Format: Diese Eigenschaft gibt das Format des IBM MQ-Headers oder der Anwendungsnutzdaten an, die mit dem JMS-Nachrichtenhauptteil beginnen, wenn kein WebSphere MQ-Header vorangestellt ist.
- JMS_IBM_Character_Set oder JMS_IBM_MQMD_CodedCharSetId: Diese Eigenschaft gibt die CCSID des IBM MQ-Headers oder der Anwendungsnutzdaten an, die mit dem JMS-Nachrichtenhauptteil beginnen, wenn kein WebSphere MQ-Header vorangestellt ist.
- JMS_IBM_Encoding oder JMS_IBM_MQMD_Encoding: Diese Eigenschaft gibt die Codierung des IBM MQ-Headers oder der Anwendungsnutzdaten an, die mit dem JMS-Nachrichtenhauptteil beginnen, wenn kein WebSphere MQ-Header vorangestellt ist.

Wenn beide Eigenschaftstypen angegeben sind, überschreiben die JMS_IBM_MQMD_*-Eigenschaften die entsprechenden JMS_IBM_*-Eigenschaften, sofern die Zieleigenschaft WMQ_MQMD_WRITE_ENABLED auf true gesetzt ist.

Hinsichtlich der Auswirkung der Festlegung von Nachrichteneigenschaften mit JMS_IBM_MQMD_* und JMS_IBM_* bestehen entscheidende Unterschiede:

1. Die JMS_IBM_MQMD_*-Eigenschaften sind IBM MQ JMS-Provider-spezifisch.
2. Die JMS_IBM_MQMD_*-Eigenschaften werden nur im MQMD festgelegt. JMS_IBM_*-Eigenschaften werden nur dann im MQMD festgelegt, wenn die Nachricht keinen MQRFH2-JMS-Header aufweist. Andernfalls werden sie im JMS-Header RFH2 festgelegt.
3. Die JMS_IBM_MQMD_*-Eigenschaften wirken sich nicht auf die Codierung von Text und Zahlen aus, die in eine JMSMessage geschrieben werden.

   Eine empfangende Anwendung nimmt wahrscheinlich die Werte von MQMD.Encoding und MQMD.CodedCharSetId an, die der Codierung und dem Zeichensatz der Zahlen und des Textes im Nachrichtenhauptteil entsprechen. Falls JMS_IBM_MQMD_*-Eigenschaften verwendet werden sollen, liegt dies in der Zuständigkeit der sendenden Anwendung. Codierung und Zeichensatz von Zahlen und Text im Nachrichtenhauptteil werden durch die JMS_IBM_*-Eigenschaften festgelegt.

   Der fehlerhaft codierte Ausschnitt in Abbildung 39 auf Seite 263 sendet eine Nachricht, die im Zeichensatz 1208 codiert ist. Dabei ist MQMD.CodedCharSetId auf 37 gesetzt.

   a. Senden einer falsch codierten Nachricht

   ```
   TextMessage tmo = session.createTextMessage();
   ((MQDestination) destination).setMessageBodyStyle
                  (WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQDestination)destination).setMQMDWriteEnabled(true);
   tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
   tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
   tmo.setText("String one");
   producer.send(tmo);
   ```

   b. Empfang der Nachricht unter Berücksichtigung des Werts von JMS_IBM_CHARACTER_SET, der über den Wert von MQMD.CodedCharSetId festgelegt wird:

   ```
   TextMessage tmi = (TextMessage) cons.receive();
   System.out.println("Message is \"" + tmi.getText() + "\"");
   ```

   c. Folgende Ausgabe wird zurückgegeben:

   ```
   Message is "éÈÊ'>...??>?"
   ```

Abbildung 39. Inkonsistente Codierung von MQMD und Nachrichtendaten

Beide Codeausschnitte in Abbildung 40 auf Seite 264 führen dazu, dass eine Nachricht in eine Warteschlange oder in ein Thema gestellt wird. Der zugehörige Hauptteil enthält die Anwendungsnutzdaten ohne Hinzufügung eines automatisch generierten MQRFH2-Headers.

1. Einstellung von WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. Einstellung von WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

Abbildung 40. Senden einer Nachricht mit einem MQ-Nachrichtenhauptteil

Nachricht empfangen

Wenn WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_JMS gesetzt ist, werden der Typ und Hauptteil der eingehenden JMS-Nachricht durch den Inhalt der empfangenen WebSphere MQ-Nachricht bestimmt. Der Nachrichtentyp und der Hauptteil richten sich nach den Feldern im MQRFH2-Header bzw. im MQMD, falls kein MQRFH2 vorhanden ist.

Wenn WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_MQ gesetzt ist, ist der Typ der eingehenden JMS-Nachricht JMSBytesMessage. Bei dem JMS-Nachrichtenhauptteil handelt es sich um die Nachrichtendaten, die vom zugrunde liegenden API-Aufruf MQGET zurückgegeben werden. Die Länge des Nachrichtenhauptteils entspricht der vom MQGET-Aufruf zurückgegebenen Länge. Der Zeichensatz und die Codierung der Daten im Nachrichtenhauptteil werden durch die Felder CodedCharSetId und Encoding des MQMD bestimmt. Das Format der Daten im Nachrichtenhauptteil richtet sich nach dem Feld Format des MQMD.

Wenn WMQ_MESSAGE_BODY auf den Standardwert WMQ_MESSAGE_BODY_UNSPECIFIED gesetzt ist, setzen IBM MQ classes for JMS diesen Wert auf WMQ_MESSAGE_BODY_JMS.

Wenn Sie eine JMSBytesMessage empfangen, können Sie sie decodieren, indem Sie die folgenden Eigenschaften referenzieren:

- JMS_IBM_Format oder JMS_IBM_MQMD_Format: Diese Eigenschaft gibt das Format des IBM MQ-Headers oder der Anwendungsnutzdaten an, die mit dem JMS-Nachrichtenhauptteil beginnen, wenn kein WebSphere MQ-Header vorangestellt ist.
- JMS_IBM_Character_Set oder JMS_IBM_MQMD_CodedCharSetId: Diese Eigenschaft gibt die CCSID des IBM MQ-Headers oder der Anwendungsnutzdaten an, die mit dem JMS-Nachrichtenhauptteil beginnen, wenn kein WebSphere MQ-Header vorangestellt ist.
- JMS_IBM_Encoding oder JMS_IBM_MQMD_Encoding: Diese Eigenschaft gibt die Codierung des IBM MQ-Headers oder der Anwendungsnutzdaten an, die mit dem JMS-Nachrichtenhauptteil beginnen, wenn kein WebSphere MQ-Header vorangestellt ist.

Der folgende Codeausschnitt führt zum Empfang einer Nachricht, die eine JMSBytesMessage ist. Ungeachtet des Inhalts der empfangenen Nachricht und des Formatfelds des empfangenen MQMD ist die Nachricht eine JMSBytesMessage.

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

Zieleigenschaft WMQ_MESSAGE_BODY
WMQ_MESSAGE_BODY bestimmt, ob eine JMS-Anwendung den MQRFH2 einer IBM MQ-Nachricht als Teil der Nachrichtennutzdaten verarbeitet (also als Teil des JMS-Nachrichtenhauptteils).

Tabelle 39. Eigenschaftsnamen und Beschreibungen

Eigenschaft	Kurzform	Beschreibung
WMQ_MESSAGE_BODY	MBODY	Bestimmt, ob eine JMS-Anwendung den MQRFH2 einer IBM MQ-Nachricht als Teil der Nachrichtennutzdaten verarbeitet (also als Teil des JMS-Nachrichtenhauptteils).

Tabelle 40. Eigenschaftsnamen, Werte und Festlegungsmethoden

Eigenschaft	Gültige Werte im Verwaltungstool (Standardwerte sind fett dargestellt)	Gültige Werte in Programmen	Festlegungsmethode
WMQ_MESSAGE_BODY	• **UNSPECIFIED** Beim Senden von Nachrichten generiert IBM MQ classes for JMS je nachdem, welcher Wert für WMQ_TARGET_CLIENT angegeben ist, einen MQRFH2-Header und fügt diesen ein oder auch nicht. Beim Empfang agiert dies als Wert JMS. • JMS Beim Senden generiert IBM MQ classes for JMS automatisch einen MQRFH2-Header und fügt diesen in die IBM MQ-Nachricht ein. Beim Empfang legt IBM MQ classes for JMS die Eigenschaften der JMS-Nachricht entsprechend den Werten im MQRFH2 fest (sofern vorhanden); der MQRFH2 wird nicht als Teil des JMS-Nachrichtentexts dargestellt. • MQ Beim Senden generiert IBM MQ classes for JMS keinen MQRFH2. Beim Empfang stellt IBM MQ classes for JMS den MQRFH2 als Teil des JMS-Nachrichtentexts dar.	• **WMQ_MESSAGE_ BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

Persistente JMS-Nachrichten
Anwendungen der IBM MQ classes for JMS können mithilfe des Warteschlangenattributs **NonPersistentMessageClass** die Leistung für persistente JMS-Nachrichten verbessern. Dadurch wird jedoch bis zu einem gewissen Grad die Zuverlässigkeit beeinträchtigt.

Eine IBM MQ-Warteschlange verfügt über ein Attribut mit dem Namen **NonPersistentMessageClass**. Der Wert dieses Attributs bestimmt, ob nicht persistente Nachrichten in der Warteschlange beim Neustart des Warteschlangenmanagers gelöscht werden.

Sie können das Attribut für eine lokale Warteschlange festlegen, indem Sie den IBM MQ-Scriptbefehl (MQSC) DEFINE QLOCAL mit einem der folgenden Parameter verwenden:

NPMCLASS(NORMAL)
Nicht persistente Nachrichten in der Warteschlange werden beim Neustart des Warteschlangenmanagers gelöscht. Dies ist der Standardwert.

NPMCLASS(HIGH)
Nicht persistente Nachrichten in der Warteschlange werden nicht gelöscht, wenn der Warteschlangenmanager nach einer Beendigung im Quiescemodus oder nach einer sofortigen Beendigung erneut gestartet wird. Nicht persistente Nachrichten werden möglicherweise jedoch nach einem präventiven Abschluss oder einem Fehler gelöscht.

In diesem Abschnitt wird beschrieben, wie Anwendungen der IBM MQ classes for JMS mithilfe dieses Warteschlangenattributs die Leistung für persistente JMS-Nachrichten verbessern können.

Die Eigenschaft PERSISTENCE eines Queue- oder Topic-Objekts kann den Wert HIGH haben. Diesen Wert können Sie mit dem IBM MQ-JMS-Verwaltungstool einstellen, er kann aber auch durch eine Anwendung mit der Methode 'Destination.setPersistence()' eingestellt werden, die den Wert 'WMQConstants.WMQ_PER_NPHIGH als Parameter übergibt.

Wenn eine Anwendung eine persistente JMS-Nachricht oder eine nicht persistente JMS-Nachricht an ein Ziel sendet, bei dem für die Eigenschaft PERSISTENCE der Wert HIGH festgelegt ist, und die zugrunde liegende IBM MQ-Warteschlange auf NPMCLASS(HIGH) gesetzt ist, wird die Nachricht als nicht persistente IBM MQ-Nachricht in die Warteschlange gestellt. Wenn die Eigenschaft PERSISTENCE des Ziels nicht den Wert HIGH hat oder die zugrunde liegende Warteschlange auf NPMCLASS(NORMAL) gesetzt ist, wird eine persistente JMS-Nachricht als persistente IBM MQ-Nachricht in die Warteschlange gestellt und eine nicht persistente JMS-Nachricht wird als nicht persistente IBM MQ-Nachricht in die Warteschlange gestellt.

Wenn eine persistente JMS-Nachricht als nicht persistente IBM MQ-Nachricht in eine Warteschlange gestellt wird und Sie sicherstellen möchten, dass die Nachricht nach einer Beendigung eines Warteschlangenmanagers im Quiescemodus oder nach einer sofortigen Beendigung eines Warteschlangenmanagers nicht gelöscht wird, müssen alle Warteschlangen, durch die die Nachricht möglicherweise geleitet wird, auf NPMCLASS(HIGH) gesetzt werden. In der Publish/Subscribe-Domäne gehören zu diesen Warteschlangen auch Warteschlangen für Subskribenten. Als Unterstützung bei der Umsetzung dieser Konfiguration lösen IBM MQ classes for JMS eine Ausnahmebedingung des Typs 'InvalidDestinationException' aus, wenn eine Anwendung versucht, einen Nachrichtenkonsumenten für ein Ziel zu erstellen, dessen PERSISTENCE- Eigenschaft auf den Wert HIGH gesetzt ist, und bei dem die IBM MQ-Warteschlange den Wert NPMCLASS(NORMAL) hat.

Wenn die PERSISTENCE-Eigenschaft eines Ziels auf HIGH gesetzt wird, hat dies keine Auswirkung darauf, wie eine Nachricht von diesem Ziel empfangen wird. Eine als persistente JMS-Nachricht gesendete Nachricht wird als persistente JMS-Nachricht empfangen, während eine als nicht persistente JMS-Nachricht als nicht persistente JMS-Nachricht empfangen wird.

Wenn eine Anwendung die erste Nachricht an ein Ziel sendet und dabei die Eigenschaft PERSISTENCE den Wert HIGH aufweist oder wenn eine Anwendung den ersten Nachrichtenkonsumenten für ein Ziel erstellt und die Eigenschaft PERSISTENCE den Wert HIGH hat, geben IBM MQ classes for JMS einen MQINQ-Aufruf aus, um festzustellen, ob NPMCLASS(HIGH) in der zugrunde liegenden IBM MQ-Warteschlange festgelegt ist. Daher muss die Anwendung zur Abfrage der Warteschlange berechtigt sein. Außerdem speichern die IBM MQ classes for JMS das Ergebnis des MQINQ-Aufrufs, bis das Ziel gelöscht wird. Es werden keine weiteren MQINQ-Aufrufe ausgegeben. Wenn Sie also die NPMCLASS-Einstellung für die zugrunde liegende Warteschlange ändern, solange die Anwendung das Ziel noch verwendet, wird die neue Einstellung nicht von den IBM MQ classes for JMS bemerkt.

Wenn Sie zulassen, dass persistente JMS-Nachrichten als nicht persistente IBM MQ-Nachrichten in IBM MQ-Warteschlangen gestellt werden, können Sie die Leistung verbessern, allerdings wird dadurch die Zuverlässigkeit beeinträchtigt. Wenn Sie eine maximale Zuverlässigkeit für persistente JMS-Nachrichten

benötigen, sollten Sie die Nachrichten nicht an ein Ziel senden, bei dem die Eigenschaft PERSISTENCE den Wert HIGH hat.

Die JMS -Schicht kann SYSTEM.JMS.TEMPQ.MODELanstelle von SYSTEM.DEFAULT.MODEL.QUEUE. Mit SYSTEM.JMS.TEMPQ.MODEL werden permanente dynamische Warteschlangen erzeugt, die persistente Nachrichten akzeptieren, da persistente Nachrichten nicht von SYSTEM.DEFAULT.MODEL.QUEUE akzeptiert werden können. Wenn Sie temporäre Warteschlangen verwenden möchten, die persistente Nachrichten akzeptieren, müssen Sie also SYSTEM.JMS.TEMPQ.MODEL verwenden oder die Modellwarteschlange in eine alternative Warteschlange Ihrer Wahl ändern.

TLS mit IBM MQ classes for JMS verwenden
IBM MQ classes for JMS-Anwendungen können die TLS-Verschlüsselung (Transport Layer Security) verwenden. Hierfür benötigen sie einen JSEE-Provider.

IBM MQ classes for JMS-Verbindungen, die TRANSPORT(CLIENT) verwenden, unterstützen die TLS-Verschlüsselung. TLS stellt eine Kommunikationsverschlüsselung, Authentifizierung und Nachrichtenintegrität bereit. In der Regel wird damit die Kommunikation zwischen zwei Peers im Internet oder innerhalb eines Intranets geschützt.

IBM MQ classes for JMS nutzen Java Secure Socket Extension (JSSE) für die Handhabung der TLS-Verschlüsselung und benötigen daher einen JSSE-Provider. JVMs mit JSE v1.4 verfügen bereits über einen integrierten JSSE-Provider. Die Details der Verwaltung und Speicherung von Zertifikaten kann je nach Provider variieren. Informationen hierzu finden Sie in der Dokumentation Ihres JSSE-Providers.

In diesem Abschnitt wird vorausgesetzt, dass Ihr JSSE-Provider ordnungsgemäß installiert und konfiguriert wurde. Außerdem müssen geeignete Zertifikate installiert worden sein, die Ihrem JSEE-Provider zur Verfügung stehen. Sie können jetzt mit JMSAdmin mehrere Verwaltungseigenschaften festlegen.

Wenn Ihre Anwendung der IBM MQ classes for JMS für die Verbindung mit einem Warteschlangenmanager eine Definitionstabelle für den Clientkanal (CCDT) verwendet, lesen Sie den Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden” auf Seite 296.

Objekteigenschaft SSLCIPHERSUITE
Legen Sie SSLCIPHERSUITE fest, um die TLS-Verschlüsselung für ein ConnectionFactory-Objekt zu aktivieren.

Wenn Sie die TLS-Verschlüsselung für ein ConnectionFactory-Objekt aktivieren möchten, setzen Sie die Eigenschaft SSLCIPHERSUITE über JMSAdmin auf eine von Ihrem JSSE-Provider unterstützte Cipher-Suite. Dieser Wert muss mit der für den Zielkanal festgelegten CipherSpec übereinstimmen. Da sich CipherSuites jedoch von CipherSpecs unterscheiden, haben sie unterschiedliche Namen. „TLS CipherSpecs und Cipher-Suites in IBM MQ classes for JMS” auf Seite 271 enthält eine Tabellenzuordnung zwischen den von IBM MQ unterstützten CipherSpecs und ihren Cipher-Suite-Entsprechungen, die in JSSE bekannt sind. Weitere Informationen zu CipherSpecs und Cipher-Suites bei IBM MQ finden Sie im Abschnitt IBM MQ schützen.

Wenn Sie beispielsweise ein ConnectionFactory-Objekt einrichten möchten, das für die Herstellung einer Verbindung über einen TLS-fähigen MQI-Kanal mit der CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA verwendet werden kann, geben Sie folgenden Befehl an JMSAdmin aus:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

Dies kann auch von einer Anwendung aus festgelegt werden, indem die Methode setSSLCipherSuite() für ein MQConnectionFactory-Objekt verwendet wird.

Wenn eine CipherSpec in der Eigenschaft SSLCIPHERSUITE angegeben ist, versucht JMSAdmin zur Entlastung des Benutzers, die CipherSpec einer entsprechenden Cipher-Suite zuzuordnen, und gibt eine Warnung aus. Dieser Zuordnungsversuch wird nicht unternommen, wenn die Eigenschaft durch eine Anwendung angegeben wird.

Alternativ dazu können Sie die Definitionstabelle für den Clientkanal (Client Channel Definition Table, CCDT) verwenden. Weitere Informationen finden Sie unter „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden” auf Seite 296.

SSLFIPSREQUIRED, Objekteigenschaft
Wenn eine Verbindung zum Verwenden einer Cipher-Suite, die vom IBM Java-JSSE-FIPS-Provider (IBMJSSEFIPS) unterstützt wird, erforderlich ist, legen Sie für die Eigenschaft SSLFIPSREQUIRED der Verbindungsfactory den Wert YES fest.

Anmerkung: Unter AIX, Linux, and Windowsstellt IBM MQ die Konformität mit FIPS 140-2 über das Verschlüsselungsmodul IBM Crypto for C (ICC) bereit. Das Zertifikat für dieses Modul wurde in den Langzeitstatus versetzt. Kunden sollten das IBM Crypto for C (ICC) -Zertifikat anzeigen und sich über alle Empfehlungen von NIST im Klaren sein. Ein Ersatz-FIPS 140-3-Modul ist derzeit in Bearbeitung und sein Status kann angezeigt werden, indem in der NIST-CMVP-Module in der Prozesslistenach ihm gesucht wird.

IBM MQ Operator 3.2.0 und das Container-Image des Warteschlangenmanagers ab 9.4.0.0 basieren auf UBI 9. Die Konformität mit FIPS 140-3 steht derzeit an und ihr Status kann angezeigt werden, indem Sie in der NIST CMVP-Module in der Prozesslistenach "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" suchen.

Der Standardwert dieser Eigenschaft ist NO. Dies bedeutet, dass eine Verbindung jede beliebige Cipher-Suite verwenden kann, die von IBM MQ unterstützt wird.

Wenn eine Anwendung mehrere Verbindungen verwendet, bestimmt der Wert von SSLFIPSREQUIRED, der bei der ersten Erstellung der Verbindung durch die Anwendung verwendet wird, den Wert, der verwendet wird, wenn die Anwendung nachfolgende Verbindungen erstellt. Dies bedeutet, dass der Wert der Eigenschaft SSLFIPSREQUIRED der Verbindungsfactory, der für die Erstellung einer nachfolgenden Verbindung verwendet wird, ignoriert wird. Sie müssen die Anwendung erneut starten, wenn Sie einen anderen Wert für SSLFIPSREQUIRED verwenden möchten.

Eine Anwendung kann diese Eigenschaft festlegen, indem sie die Methode setSSLFipsRequired() eines ConnectionFactory-Objekts aufruft. Die Eigenschaft wird ignoriert, wenn keine Cipher-Suite festgelegt wurde.

Zugehörige Tasks
Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden
Zugehörige Verweise
Federal Information Processing Standards (FIPS) für AIX, Linux, and Windows

Objekteigenschaft SSLPEERNAME
Mithilfe von SSLPEERNAME können Sie ein Muster für einen definierten Namen angeben, um sicherzustellen, dass sich Ihre JMS-Anwendung mit dem richtigen Warteschlangenmanager verbindet.

Eine JMS-Anwendung kann sicherstellen, dass sie sich mit dem richtigen Warteschlangenmanager verbindet, indem ein Muster für einen definierten Namen (DN) angegeben wird. Die Verbindung ist nur erfolgreich, wenn der Warteschlangenmanager einen DN präsentiert, der mit dem Muster übereinstimmt. In den zugehörigen Abschnitten finden Sie weitere Details zum Format dieses Musters.

Der DN wird mit der Eigenschaft SSLPEERNAME eines ConnectionFactory-Objekts festgelegt. Mit dem folgenden JMSAdmin-Befehl wird beispielsweise festgelegt, dass ein ConnectionFactory-Objekt erwartet, dass ein Warteschlangenmanager sich mit einem allgemeinen Namen identifiziert der mit den Zeichen `QMGR.` beginnt, sowie mit mindestens zwei Namen von Organisationseinheiten, von denen die erste IBM und die zweite WEBSPHERE sein muss:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Bei der Prüfung wird die Groß-/Kleinschreibung nicht beachtet und anstelle von Kommas können Semikola verwendet werden. SSLPEERNAME kann auch von einer Anwendung aus festgelegt werden, indem die Methode setSSLPeerName() für ein MQConnectionFactory-Objekt verwendet wird. Wenn diese Eigenschaft nicht festgelegt wird, wird der vom Warteschlangenmanager bereitgestellte definierte Name nicht geprüft. Diese Eigenschaft wird ignoriert, wenn keine Cipher-Suite festgelegt wurde.

Objekteigenschaft SSLCERTSTORES
Mithilfe von SSLCERTSTORES können Sie eine Liste der LDAP-Server angeben, die für die Prüfung anhand einer Zertifikatswiderrufsliste (Certificate Revocation List, CRL) verwendet werden.

Häufig wird eine Zertifikatswiderrufsliste (Certificate Revocation List, CRL) verwendet, um Zertifikate anzugeben, die nicht mehr als vertrauenswürdig gelten. CRLs werden in der Regel auf LDAP-Servern gehostet. JMS ermöglicht die Angabe eines LDAP-Servers für die CRL-Prüfung unter Java 2 v1.4 oder höher. Im folgenden JMSAdmin-Beispiel wird JMS angewiesen, eine Zertifikatssperrliste zu verwenden, die auf einem LDAP-Server mit dem Namen crl1.ibm.com gehostet wird:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

Anmerkung: Damit ein CertStore erfolgreich mit einer auf einem LDAP-Server gehosteten CRL verwendet werden kann, müssen Sie sicherstellen, dass Ihr Java-Software Development Kit (SDK) mit der CRL kompatibel ist. Manche SDKs verlangen, dass die Zertifikatswiderrufsliste RFC 2587 entspricht, in dem ein Schema für LDAP v2 definiert ist. Die meisten LDAP v3-Server verwenden stattdessen RFC 2256.

Wenn Ihr LDAP-Server nicht am Standardport 389 ausgeführt wird, können Sie den Port angeben, indem Sie einen Doppelpunkt (:) und die Portnummer an den Hostnamen anhängen. Wenn das vom Warteschlangenmanager vorgelegte Zertifikat in der unter crl1.ibm.com gehosteten CRL enthalten ist, wird die Verbindung nicht ausgeführt. Zur Vermeidung eines Single Point of Failure erlaubt JMS die Bereitstellung mehrerer LDAP-Server durch die Angabe einer Liste mit LDAP-Servern, die durch ein Leerzeichen voneinander getrennt sind. Beispiel:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

Wenn mehrere LDAP-Server angegeben werden, versucht JMS der Reihe nach jeden Server, bis ein Server gefunden wird, mit dem das Zertifikat des Warteschlangenmanagers erfolgreich geprüft werden kann. Jeder Server muss identische Informationen enthalten.

Eine Zeichenfolge in diesem Format kann von einer Anwendung in der Methode MQConnectionFactory.setSSLCertStores() bereitgestellt werden. Alternativ kann die Anwendung ein oder mehrere java.security.cert.CertStore-Objekte erstellen, diese in ein geeignetes Objektgruppenobjekt stellen und dieses Objektgruppenobjekt für die Methode setSSLCertStores() bereitstellen. Auf diese Weise kann die Anwendung die CRL-Prüfung anpassen. In Ihrer JSSE-Dokumentation finden Sie Details zur Erstellung und Verwendung von CertStore-Objekten.

Das Zertifikat, das vom Warteschlangenmanager beim Aufbau der Verbindung vorgelegt wird, wird wie folgt überprüft:

1. Das erste CertStore-Objekt in der durch sslCertStores angegebenen Objektgruppe wird für die Identifizierung eines CRL-Servers verwendet.
2. Es wird versucht, den CRL-Server zu kontaktieren.
3. Wenn der Versuch erfolgreich ist, wird der Server nach einer Übereinstimmung mit dem Zertifikat durchsucht.
   a. Falls festgestellt wird, dass das Zertifikat widerrufen wurde, wird der Suchvorgang beendet und die Verbindungsanforderung schlägt mit dem Ursachencode MQRC_SSL_CERTIFICATE_REVOKED fehl.
   b. Wenn das Zertifikat nicht gefunden werden kann, wird der Suchvorgang beendet und die Verbindung kann fortgesetzt werden.
4. Wenn der Versuch der Kontaktierung des Servers nicht erfolgreich war, wird das nächste CertStore-Objekt für die Identifizierung eines CRL-Servers verwendet und der Prozess wird ab Schritt 2 wiederholt.

   Falls das Objekt der letzte CertStore in der Objektgruppe war oder die Objektgruppe keine CertStore-Objekte enthält, ist der Suchvorgang fehlgeschlagen und die Verbindungsanforderung schlägt mit dem Ursachencode MQRC_SSL_CERT_STORE_ERROR fehl.

Das Objektgruppenobjekt bestimmt die Reihenfolge, in der CertStores verwendet werden.

Wenn Ihre Anwendung mithilfe von setSSLCertStores() eine Objektgruppe mit CertStore-Objekten festlegt, kann die MQConnectionFactory nicht mehr an einen JNDI-Namensbereich gebunden werden. Ein derartiger Versuch führt zu einer Ausnahmebedingung. Wenn die Eigenschaft 'sslCertStores' nicht festgelegt wird, wird keine Widerrufsprüfung für das vom Warteschlangenmanager bereitgestellte Zertifikat ausgeführt. Diese Eigenschaft wird ignoriert, wenn keine Cipher-Suite festgelegt wurde.

SSLRESETCOUNT, Objekteigenschaft
Diese Eigenschaft stellt die Gesamtzahl der Bytes dar, die von einer Verbindung gesendet und empfangen werden, bevor der für die Verschlüsselung verwendete geheime Schlüssel erneut vereinbart wird.

Dabei ist die Anzahl der gesendeten Bytes die Anzahl vor der Verschlüsselung und die Anzahl der empfangenen Bytes die Anzahl nach der Entschlüsselung. Die Anzahl der Bytes umfasst auch Steuerinformationen, die von IBM MQ classes for JMS gesendet und empfangen werden.

Wenn Sie beispielsweise ein ConnectionFactory-Objekt konfigurieren möchten, das zum Herstellen einer Verbindung über einen TLS-fähigen MQI-Kanal mit einem geheimen Schlüssel verwendet werden kann, der nach der Übergabe von 4 MB an Daten neu vereinbart wird, dann geben Sie folgenden Befehl an JMSAdmin aus:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

Eine Anwendung kann diese Eigenschaft festlegen, indem sie die Methode setSSLResetCount() eines ConnectionFactory-Objekts aufruft.

Wenn der Wert dieser Eigenschaft null ist (Standardwert), wird der geheime Schlüssel niemals erneut vereinbart. Die Eigenschaft wird ignoriert, wenn keine Cipher-Suite festgelegt wurde.

Objekteigenschaft SSLSocketFactory
Wenn Sie weitere Aspekte der TLS-Verbindung für eine Anwendung anpassen möchten, erstellen Sie eine SSLSocketFactory und konfigurieren Sie JMS für deren Verwendung.

Möglicherweise möchten Sie weitere Aspekte der TLS-Verbindung für eine Anwendung anpassen. Sie können beispielsweise eine Verschlüsselungshardware initialisieren oder den Keystore und Truststore, die verwendet werden, ändern. Hierfür muss die Anwendung zuerst ein javax.net.ssl.SSLSocketFactory-Objekt erstellen, das entsprechend angepasst wird. Schlagen Sie in Ihrer JSSE-Dokumentation die jeweilige Vorgehensweise nach, da die anpassbaren Funktionen je nach Provider variieren. Sobald Sie ein geeignetes SSLSocketFactory-Objekt erhalten haben, verwenden Sie die Methode MQConnectionFactory.setSSLSocketFactory(), um JMS für die Verwendung des angepassten SSLSocketFactory-Objekts zu konfigurieren.

Wenn Ihre Anwendung mithilfe der Methode setSSLSocketFactory() ein angepasstes SSLSocketFactory-Objekt festlegt, kann das MQConnectionFactory-Objekt nicht mehr an einen JNDI-Namensbereich gebunden werden. Ein derartiger Versuch führt zu einer Ausnahmebedingung. Wenn diese Eigenschaft nicht festgelegt wird, wird das standardmäßige SSLSocketFactory-Objekt verwendet. In Ihrer JSSE-Dokumentation finden Sie Details zum Verhalten des standardmäßigen SSLSocketFactory-Objekts. Diese Eigenschaft wird ignoriert, wenn keine Cipher-Suite festgelegt wurde.

Wichtig: Gehen Sie nicht davon aus, dass die Verwendung der SSL-Eigenschaften für Sicherheit sorgt, wenn ein ConnectionFactory-Objekt aus einem JNDI-Namensbereich abgerufen wird, der selbst nicht sicher ist. Insbesondere müssen Sie beachten, dass die LDAP-Standardimplementierung von JNDI nicht sicher ist. Ein Angreifer kann den LDAP-Server imitieren, sodass eine JMS-Anwendung sich mit dem falschen Server verbindet, ohne dies zu bemerken. Wenn geeignete Sicherheitsmaßnahmen ergriffen werden, sind sonstige Implementierungen von JNDI (beispielsweise die fscontext-Implementierung) sicher.

Änderungen am JSSE-Schlüsselspeicher oder -Truststore vornehmen
Wenn Sie den Schlüsselspeicher oder Truststore ändern, müssen Sie bestimmte Aktionen ausführen, damit die Änderungen in Kraft treten.

Wenn Sie Änderungen am Inhalt des JSSE-Schlüsselspeichers oder -Truststores vornehmen oder die Position der Schlüsselspeicher- bzw. Truststore-Datei ändern, werden die Änderungen nicht automatisch

von den Anwendungen der IBM MQ classes for JMS übernommen, die zu diesem Zeitpunkt aktiv sind. Damit die Änderungen in Kraft treten, müssen die folgenden Aktionen ausgeführt werden:

- Die Anwendungen müssen alle ihre Verbindungen schließen und nicht verwendete Verbindungen in Verbindungspools löschen.
- Wenn Ihr JSSE-Provider Informationen aus dem Schlüsselspeicher und Truststore zwischenspeichert, müssen diese Informationen aktualisiert werden.

Nachdem diese Aktionen ausgeführt wurden, können die Anwendungen ihre Verbindungen erneut erstellen.

Abhängig vom Design Ihrer Anwendungen und je nachdem, welche Funktion von Ihrem JSSE-Provider bereitgestellt wird, können diese Aktionen unter Umständen ohne Stoppen und Neustart Ihrer Anwendungen ausgeführt werden. Das Stoppen und der Neustart der Anwendungen ist aber gegebenenfalls die einfachste Lösung.

TLS CipherSpecs und Cipher-Suites in IBM MQ classes for JMS
Ob Anwendungen der IBM MQ classes for JMS Verbindungen zu einem Warteschlangenmanager herstellen können, hängt davon ab, welche CipherSpec am Serverende des MQI-Kanals und welche Cipher-Suite am Clientende angegeben ist.

In der folgenden Tabelle sind die von IBM MQ unterstützten CipherSpecs sowie die entsprechenden Cipher-Suites aufgeführt.

Deprecated Überprüfen Sie im Abschnitt Nicht weiter unterstützte CipherSpecs, ob und ggf. ab welchem Upgrade die in der folgenden Tabelle aufgeführten CipherSpecs von IBM MQ nicht mehr unterstützt wurden.

Wichtig: Die aufgeführten Cipher-Suites werden von der mit IBM MQ bereitgestellten IBM Java Runtime Environment (JRE) unterstützt. Sie beinhalten auch die von der Oracle Java-JRE unterstützten Cipher-Suites. Weitere Informationen zur Konfiguration Ihrer Anwendung für eine Oracle Java-JRE finden Sie im Abschnitt Anwendung für IBM Java- oder Oracle Java-Cipher-Suite-Zuordnungen konfigurieren.

In der Tabelle ist auch das für die Kommunikation verwendete Protokoll angegeben sowie, ob die Cipher-Suite dem FIPS 140-2-Standard entspricht.

Anmerkung: Unter AIX, Linux, and Windowsstellt IBM MQ die Konformität mit FIPS 140-2 über das Verschlüsselungsmodul IBM Crypto for C (ICC) bereit. Das Zertifikat für dieses Modul wurde in den Langzeitstatus versetzt. Kunden sollten das IBM Crypto for C (ICC) -Zertifikat anzeigen und sich über alle Empfehlungen von NIST im Klaren sein. Ein Ersatz-FIPS 140-3-Modul ist derzeit in Bearbeitung und sein Status kann angezeigt werden, indem in der NIST-CMVP-Module in der Prozesslistenach ihm gesucht wird.

IBM MQ Operator 3.2.0 und das Container-Image des Warteschlangenmanagers ab 9.4.0.0 basieren auf UBI 9. Die Konformität mit FIPS 140-3 steht derzeit an und ihr Status kann angezeigt werden, indem Sie in der NIST CMVP-Module in der Prozesslistenach "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" suchen.

Wenn die Anwendung nicht so konfiguriert ist, dass eine Einhaltung des FIPS 140-2-Standards erzwungen wird, können Cipher-Suites, die als FIPS 140-2-konform gekennzeichnet sind, verwendet werden. Wurde die Anwendung für eine Einhaltung des FIPS 140-2-Standards konfiguriert (siehe die folgenden Hinweise zur Konfiguration), können nur die Cipher-Suites verwendet werden, die als FIPS 140-2-konform gekennzeichnet sind. Bei einem Versuch, eine andere Cipher-Suite zu verwenden, wird ein Fehler zurückgegeben.

Anmerkung: Jede JRE kann über mehrere Provider für Sicherheit über Verschlüsselung verfügen, von denen jeder eine Implementierung derselben Cipher-Suite bereitstellt. Nicht alle Sicherheitsprovider sind allerdings FIPS 140-2-zertifiziert. Wenn für eine Anwendung keine Einhaltung des FIPS 140-2-Standards erzwungen wird, besteht die Möglichkeit, dass eine nicht zertifizierte Implementierung der Cipher-Suite verwendet wird. Nicht zertifizierte Implementierungen arbeiten allerdings unter Umständen nicht FIPS 140-2-konform, auch wenn die Cipher-Suite theoretisch die von diesem Standard geforderten Mindestsicherheitsanforderungen erfüllt. Die folgenden Hinweise enthalten weitere Informationen zur Konfiguration der Durchsetzung von FIPS 140-2 in IBM MQ JMS-Anwendungen.

Weitere Informationen zur Konformität mit FIPS 140-2 und Suite-B bei CipherSpecs und Cipher-Suites finden Sie unter Specifying CipherSpecs. Unter Umständen müssen Sie auch Informationen in Zusammenhang mit den US-amerikanischen Federal Information Processing Standards beachten.

Damit damit alle Cipher-Suites verwendet werden können und FIPS 140-2- und/oder Suite-B-zertifiziert arbeiten, ist eine entsprechende JRE erforderlich.IBM Java 7 Serviceaktualisierung 4 Fixpack 2 oder eine höhere Version von IBM JRE bietet die entsprechende Unterstützung für die TLS 1.2 CipherSuites , die in Tabelle 41 auf Seite 272aufgelistet sind.

Um TLS 1.3 -Verschlüsselungen verwenden zu können, muss die JRE, in der Ihre Anwendung ausgeführt wird, TLS 1.3unterstützen.

Anmerkung: Für die Verwendung einiger Cipher-Suites müssen die uneingeschränkten Richtliniendateien in der JRE konfiguriert werden. Ausführlichere Informationen zur Konfiguration der Richtliniendateien in einem SDK oder in einer JRE finden Sie im Abschnitt *IBM SDK Policy files* der *Security Reference for IBM SDK, Java Technology Edition* für die von Ihnen verwendete Version.

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites

CipherSpec **„1" auf Seite 291**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1" auf Seite 291**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1” auf Seite 291**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1” auf Seite 291**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	nein
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	nein

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	ja

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	ja
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	nein

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	nein
TLS_RSA_WITH_3DES_EDE_CBC_SHA „2” auf Seite 291	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	nein „4” auf Seite 291

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	nein „4” auf Seite 291
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	nein „4” auf Seite 291

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	nein „4" auf Seite 291
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	nein „4" auf Seite 291

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	nein „4" auf Seite 291
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	nein „4" auf Seite 291

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	nein
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	nein

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	nein
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	ja
TLS_AES_128_GCM_SHA256 „3" auf Seite 291	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	nein

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1" auf Seite 291**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_AES_256_GCM_SHA384 „3" auf Seite 291	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	nein
TLS_CHACHA20_POLY1305_SHA256 „3" auf Seite 291	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	nein

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec **„1" auf Seite 291**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_AES_128_CCM_SHA256 „3" auf Seite 291	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	nein
TLS_AES_128_CCM_8_SHA256 „3" auf Seite 291	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	nein
Beliebig„3" auf Seite 291	*ANY	*ANY	Mehrfach	nein
ANY_TLS13 „3" auf Seite 291	*TLS13	*TLS13	TLS V13	nein

Tabelle 41. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 291	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ANY_TLS12_OR_HIGHER „3" auf Seite 291	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 und höher	nein
ANY_TLS13_OR_HIGHER „3" auf Seite 291	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 und höher	nein

Anmerkungen:

1. Dies ist der Wert, der in einem Kanal in IBM MQkonfiguriert ist, einschließlich einer CCDT (Binary oder JSON).
2. Deprecated Die CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA wird nicht weiter unterstützt. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.
3. Damit TLS- v1.3 -Verschlüsselungen verwendet werden können, muss die Java runtime environment (JRE), die Ihre Anwendung ausführt, TLS v1.3unterstützen.
4. V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0entfernt die IBM Java 8 JRE die Unterstützung für den RSA-Schlüsselaustausch, wenn sie im FIPS-Modus ausgeführt wird.

Cipher-Suites und FIPS-Konformität in einer IBM MQ classes for JMS-Anwendung konfigurieren

- Für eine Anwendung, die IBM MQ classes for JMS verwendet, gibt es die folgenden beiden Möglichkeiten, die Cipher-Suite für eine Verbindung zu konfigurieren:
  - Aufruf der setSSLCipherSuite-Methode eines ConnectionFactory-Objekts
  - Verwendung des IBM MQ JMS-Verwaltungstools zur Definition der Eigenschaft SSLCIPHERSUITE eines ConnectionFactory-Objekts.
- Eine Anwendung, die IBM MQ classes for JMS verwendet, kann die Konformität mit FIPS 140-2 auf zwei Arten durchsetzen:
  - Aufruf der setSSLFipsRequired-Methode eines ConnectionFactory-Objekts
  - Verwendung des IBM MQ JMS-Verwaltungstools zur Definition der Eigenschaft SSLFIPSREQUIRED eines ConnectionFactory-Objekts.

Anwendung für Oracle IBM Java- oder Oracle Java-Cipher-Suite-Zuordnungen konfigurieren

V 9.4.0 Ab IBM MQ 9.4.0kann ein Cipher entweder als CipherSpec oder als CipherSuite -Name definiert werden und wird von IBM MQordnungsgemäß verarbeitet.

Anmerkung: Removed Die Java Systemeigenschaft `com.ibm.mq.cfg.useIBMCipherMappings`, die steuert, welche Zuordnungen in früheren Versionen von IBM MQverwendet werden, wird nicht mehr benötigt und aus dem Produkt unter IBM MQ 9.4.0entfernt.

Einschränkungen bei der Interoperabilität

Je nach verwendetem Protokoll sind einige Cipher-Suites auch mit mehreren IBM MQ-CipherSpecs kompatibel, allerdings wird nur die Cipher-Suite/CipherSpec-Kombination unterstützt, die die in Tabelle 1 angegeben TLS-Version verwendet. Versuche, eine der nicht unterstützten Cipher-Suite/CipherSpecs-Kombinationen zu verwenden, schlagen mit einer entsprechenden Ausnahmebedingung fehl. In Installationen, in denen eine dieser Cipher-Suite/CipherSpec-Kombinationen verwendet wird, sollte zur Verwendung einer unterstützten Kombination übergegangen werden.

In der folgenden Tabelle sind die Cipher-Suites aufgeführt, für die diese Einschränkung gilt.

Tabelle 42. Cipher-Suites und die unterstützten bzw. nicht unterstützten CipherSpecs

CipherSuite	Unterstützte TLS-CipherSpec	Nicht unterstützte SSL-CipherSpec
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA „1" auf Seite 292	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Anmerkung:

1. Deprecated Die CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA wird nicht weiter unterstützt. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.

Kanalexits in Java für IBM MQ classes for JMS schreiben
Sie erstellen Kanalexits durch die Definition von Java-Klassen, die angegebene Schnittstellen implementieren.

Eine Einführung in Sicherheitsexits finden Sie im Abschnitt Kanalsicherheitsexitprogramme .

Im Paket com.ibm.mq.exits sind drei Schnittstellen definiert:

- WMQSendExit, für einen Sendeexit
- WMQReceiveExit, für einen Empfangsexit
- WMQSecurityExit, für einen Sicherheitsexit

Der folgende Beispielcode definiert eine Klasse, die alle drei Schnittstellen implementiert:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method implements the send exit interface
  public ByteBuffer channelSendExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the send exit here
  }
    // This method implements the receive exit interface
  public ByteBuffer channelReceiveExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the receive exit here
  }
    // This method implements the security exit interface
  public ByteBuffer channelSecurityExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the security exit here
  }
}
```

Jeder Exit empfängt ein MQCXP-Objekt und ein MQCD-Objekt als Parameter. Diese Objekte stellen die MQCXP- und MQCD-Strukturen dar, die in der prozedurgesteuerten Schnittstelle definiert werden.

Wird ein Sendeexit aufgerufen, enthält der Parameter 'agentBuffer' die Daten, die an den Serverwarteschlangenmanager gesendet werden sollen. Ein Längenparameter ist nicht erforderlich, da der Ausdruck agentBuffer.limit() die Länge der Daten angibt. Der Sendeexit liefert als Wert die Daten, die an den Serverwarteschlangenmanager gesendet werden sollen. Ist der Sendeexit jedoch nicht der letzte Sendeexit in einer Reihe von Sendeexits, werden die gelieferten Daten stattdessen an den nächsten Sendeexit in der Folge übergeben. Ein Sendeexit kann eine geänderte Version der Daten liefern, die er im Parameter 'agentBuffer' empfängt, oder er kann die Daten unverändert liefern. Daher lautet der einfachste mögliche Exithauptteil wie folgt:

```
{ return agentBuffer; }
```

Wird ein Empfangsexit aufgerufen, enthält der Parameter 'agentBuffer' die Daten, die vom Serverwarteschlangenmanager empfangen wurden. Der Empfangsexit liefert als Wert die Daten, die an die Anwendung durch IBM MQ classes for JMS übergeben werden sollen. Ist der Empfangsexit jedoch nicht der letzte Empfangsexit in einer Folge von Empfangsexits, werden die zurückgegebenen Daten stattdessen an den nächsten Empfangsexit in der Folge übergeben.

Wird ein Sicherheitsexit aufgerufen, enthält der Parameter 'agentBuffer' die Daten, die in einem Sicherheitsfluss vom Sicherheitsexit auf der Serverseite der Verbindung empfangen wurden. Der Sicherheitsexit liefert als Wert die Daten, die in einem Sicherheitsfluss an den Serversicherheitsexit gesendet werden sollen.

Kanalexits werden mit einem Puffer aufgerufen, der über ein Sicherungsarray verfügt. Zur Leistungsoptimierung sollte der Exit einen Puffer mit einem Sicherungsarray liefern.

Beim Aufruf eines Kanalexits können Benutzerdaten von bis zu 32 Zeichen an ihn übergeben werden. Der Exit greift durch den Aufruf der Methode getExitData() des MQCXP-Objekts auf die Benutzerdaten zu. Obwohl der Exit die Benutzerdaten durch den Aufruf der Methode setExitData() ändern kann, werden die Benutzerdaten bei jedem Aufruf des Exits aktualisiert. Daher gehen sämtliche Änderungen, die an den Benutzerdaten vorgenommen wurden, verloren. Der Exit kann jedoch Daten aus einem Aufruf an den nächsten übergeben, indem er den Exitbenutzerbereich des MQCXP-Objekts verwendet. Der Exit greift über eine Referenz auf den Exitbenutzerbereich zu, indem er die Methode getExitUserArea() aufruft.

Jede Exitklasse muss über einen Konstruktor verfügen. Der Konstruktor kann entweder wie im vorherigen Beispiel der Standardkonstruktor oder ein Konstruktor mit einem Zeichenfolgeparameter sein. Der Konstruktor wird aufgerufen, um für jeden in der Klasse definierten Exit eine Instanz der Exitklasse zu erstellen. Wenn also im vorherigen Beispiel eine Instanz der MyMQExits-Klasse für den Sendeexit erstellt wird, wird für den Empfangsexit eine weitere Instanz und für den Sicherheitsexit eine dritte Instanz erstellt. Wird ein Konstruktor mit einem Zeichenfolgeparameter aufgerufen, enthält der Parameter dieselben Benutzerdaten, die an den Kanalexit übergeben werden, für den die Instanz erstellt wird. Verfügt eine Exitklasse sowohl über einen Standardkonstruktor als auch über einen einzelnen Parameterkonstruktor, hat der einzelne Parameterkonstruktor Vorrang.

Schließen Sie die Verbindung nicht aus einem Kanalexit heraus.

Werden Daten an die Serverseite einer Verbindung gesendet, erfolgt die TLS-Verschlüsselung *nach* dem Aufruf der Kanalexits. Analog hierzu erfolgt die TLS-Entschlüsselung beim Datenempfang von der Serverseite einer Verbindung *vor* dem Aufruf der Kanalexits.

In Versionen von IBM MQ classes for JMS vor IBM WebSphere MQ 7.0 wurden Kanalexits mithilfe der Schnittstellen MQSendExit, MQReceiveExit und MQSecurityExit implementiert. Diese Schnittstellen können nach wie vor genutzt werden, die neuen Schnittstellen bieten jedoch bessere Funktionen und eine höhere Leistung.

IBM MQ classes for JMS für Verwendung von Kanalexits konfigurieren
Eine Anwendung der IBM MQ classes for JMS kann Kanalsicherheits-, Sende- und Empfangsexits in dem MQI-Kanal verwenden, der gestartet wird, wenn sich die Anwendung mit einem Warteschlangenmanager verbindet. Die Anwendung kann Exits verwenden, die in Java, C oder C++ geschrieben wurden. Darüber hinaus kann die Anwendung eine Folge von Sende- oder Empfangsexits verwenden, die nacheinander ausgeführt werden.

Mit den folgenden Eigenschaften wird ein Sendeexit oder eine Folge von Sendeexits konfiguriert, die von einer JMS-Verbindung verwendet werden:

- Die Eigenschaft **SENDEXIT** eines MQConnectionFactory-Objekts.
- Die Eigenschaft **sendexit** in einer Aktivierungsspezifikation, die vom IBM MQ-Ressourcenadapter für die eingehende Kommunikation verwendet wird.
- Die Eigenschaft **sendexit** in einem ConnectionFactory-Objekt, die vom IBM MQ-Ressourcenadapter für die abgehende Kommunikation verwendet wird.

Der Wert der Eigenschaft ist eine Zeichenfolge, die aus einem oder mehreren Elementen besteht, die durch Kommas getrennt sind. Jedes Element identifiziert einen Sendeexit auf eine der folgenden Arten:

- Der Name einer Klasse, die die Schnittstelle `WMQSendExit` für einen Sendeexit implementiert, der in Java geschrieben ist.
- Eine Zeichenfolge im Format *Bibliotheksname* (*Eingangspunktname*) für einen Sendeexit, der in C oder C++ geschrieben ist.

Auf ähnliche Weise geben die folgenden Eigenschaften den Empfangsexit oder eine Folge von Empfangsexits an, die von einer Verbindung verwendet werden:

- Die Eigenschaft **RECEXIT** eines MQConnectionFactory-Objekts.
- Die Eigenschaft **receiveexit** in einer Aktivierungsspezifikation, die vom IBM MQ-Ressourcenadapter für die eingehende Kommunikation verwendet wird.
- Die Eigenschaft **receiveexit** in einem ConnectionFactory-Objekt, die vom IBM MQ-Ressourcenadapter für die abgehende Kommunikation verwendet wird.

Die folgenden Eigenschaften geben den Sicherheitsexit an, der von einer Verbindung verwendet wird:

- Die Eigenschaft **SECEXIT** eines MQConnectionFactory-Objekts.
- Die Eigenschaft **securityexit** in einer Aktivierungsspezifikation, die vom IBM MQ-Ressourcenadapter für die eingehende Kommunikation verwendet wird.
- Die Eigenschaft **securityexit** in einem ConnectionFactory-Objekt, die vom IBM MQ-Ressourcenadapter für die abgehende Kommunikation verwendet wird.

Für MQConnectionFactories können Sie die Eigenschaften **SENDEXIT**, **RECEXIT** und **SECEXIT** mit dem IBM MQ JMS-Verwaltungstool oder dem IBM MQ Explorer festlegen. Alternativ kann eine Anwendung die Eigenschaften durch Aufrufe der Methoden `setSendExit()`, `setReceiveExit()` und `setSecurityExit()` festlegen.

Kanalexits werden durch ihr eigenes Klassenladeprogramm geladen. Um einen Kanalexit zu finden, durchsucht das Klassenladeprogramm die folgenden Positionen in der angegebenen Reihenfolge.

1. Der Klassenpfad, der durch die Eigenschaft **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** oder durch das Attribut **JavaExitsClassPath** in der Zeilengruppe 'Channels' der IBM MQ-Clientkonfigurationsdatei angegeben wird.
2. Deprecated Der Klassenpfad, der durch die Java-Systemeigenschaft **com.ibm.mq.exitClasspath** angegeben wird. Hinweis: Diese Eigenschaft ist jetzt veraltet.
3. Das IBM MQ-Exitverzeichnis wie in Tabelle 43 auf Seite 295 dargestellt. Das Klassenladeprogramm durchsucht zuerst das Verzeichnis nach Klassendateien, die nicht in Java-Archivdateien (JAR) gepackt sind. Wenn der Kanalexit nicht gefunden wird, durchsucht das Klassenladeprogramm danach die JAR-Dateien im Verzeichnis.

Tabelle 43. IBM MQ-Exitverzeichnis

Plattform	Directory
Linux AIX AIX and Linux	`/var/mqm/exits` (32-Bit-Kanalexits) `/var/mqm/exits64` (64-Bit-Kanalexits)
Windows Windows	*Installationsdatenverzeichnis* \exits Dabei steht *Installationsdatenverzeichnis* für das Verzeichnis, das Sie während der Installation für die IBM MQ-Datendateien ausgewählt haben. Das Standardverzeichnis ist C:\ProgramData \IBM \MQ.

Anmerkung: Wenn ein Kanalexit an mehr als einer Position vorhanden ist, laden die IBM MQ classes for JMS die erste Instanz, die sie finden.

Dem Klassenladeprogramm ist das Klassenladeprogramm übergeordnet, mit dem die IBM MQ classes for JMS geladen werden. Daher kann das übergeordnete Klassenladeprogramm einen Kanalexit laden, wenn er an keiner der vorherigen Positionen gefunden werden kann. Wenn Sie jedoch die IBM MQ classes for JMS in einer Umgebung wie beispielsweise einem JEE-Anwendungsserver verwenden, können Sie die Auswahl des übergeordneten Klassenladers in der Regel nicht beeinflussen. Deshalb sollte das Klassenladeprogramm konfiguriert werden, indem die Java-Systemeigenschaft **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** auf dem Anwendungsserver festgelegt wird.

Wenn Ihre Anwendung mit aktiviertem Java security manager ausgeführt wird, muss die Richtlinienkonfigurationsdatei, die von der Java-Laufzeitumgebung, in der die Anwendung ausgeführt wird, verwendet wird, über die Berechtigungen zum Laden einer Kanalexitklasse verfügen. Weitere Informationen hierzu finden Sie im Abschnitt IBM MQ Classes for JMS-Anwendungen unter Java Security Manager ausführen.

Die Schnittstellen `MQSendExit`, `MQReceiveExit` und `MQSecurityExit`, die mit Versionen vor IBM WebSphere MQ 7.0 geliefert wurden, werden weiterhin unterstützt. Bei Verwendung von Kanalexits, die diese Schnittstellen implementieren, muss `com.ibm.mq.jar` im Klassenpfad enthalten sein.

Informationen zum Schreiben von Kanalexits in der Programmiersprache C finden Sie unter „Kanalexitprogramme für Messaging-Kanäle” auf Seite 1013. Sie müssen Kanalexitprogramme, die in C oder C++ geschrieben wurden, in dem Verzeichnis speichern, das in Tabelle 43 auf Seite 295 aufgelistet wird.

Wenn Ihre Anwendung für die Verbindung zu einem Warteschlangenmanager eine Definitionstabelle für Clientkanäle (CCDT) verwendet, lesen Sie den Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden” auf Seite 296.

Bei Verwendung von IBM MQ classes for JMS die an Kanalexits zu übergebenden Benutzerdaten angeben
Beim Aufruf eines Kanalexits können Benutzerdaten von bis zu 32 Zeichen an ihn übergeben werden.

Die Eigenschaft SENDEXITINIT eines MQConnectionFactory-Objekts gibt die Benutzerdaten an, die an jeden Sendeexit übergeben werden, sobald dieser aufgerufen wird. Der Wert der Eigenschaft ist eine Zeichenfolge, die aus einem oder mehreren Elementen mit Benutzerdaten besteht, die durch Kommas getrennt sind. Die Position der einzelnen Elemente mit Benutzerdaten innerhalb der Zeichenfolge bestimmt, an welchen Sendeexit in einer Folge von Sendeexits die Benutzerdaten übergeben werden. Das erste Element mit Benutzerdaten in der Zeichenfolge wird beispielsweise an den ersten Sendeexit in einer Folge von Sendeexits übergeben.

Sie können die Eigenschaft SENDEXITINIT mit dem IBM MQ-JMS-Verwaltungstool oder mit IBM MQ Explorer festlegen. Alternativ dazu kann eine Anwendung die Eigenschaft festlegen, indem sie die Methode setSendExitInit() aufruft.

Auf ähnliche Weise gibt die Eigenschaft RECEXITINIT eines ConnectionFactory-Objekts die Benutzerdaten an, die an jeden Empfangsexit übergeben werden, während die Eigenschaft SECEXITINIT die an einen Sicherheitsexit übergebenen Benutzerdaten angibt. Sie können diese Eigenschaften mit dem IBM MQ-JMS-Verwaltungstool oder mit IBM MQ Explorer festlegen. Alternativ dazu kann eine Anwendung die Eigenschaften festlegen, indem sie die Methoden setReceiveExitInit() und setSecurityExitInit() aufruft.

Beachten Sie bei der Angabe von Benutzerdaten, die an Kanalexits übergeben werden, die folgenden Regeln:

- Wenn die Anzahl der Elemente mit Benutzerdaten in einer Zeichenfolge die Anzahl der Exits in einer Folge überschreitet, werden die überschüssigen Elemente mit Benutzerdaten ignoriert.
- Wenn die Anzahl der Elemente mit Benutzerdaten in einer Zeichenfolge die Anzahl der Exits in einer Folge unterschreitet, wird jedes nicht angegebene Element mit Benutzerdaten auf eine leere Zeichenfolge gesetzt. Zwei Kommas nacheinander innerhalb einer Zeichenfolge oder ein Komma am Anfang einer Zeichenfolge bezeichnen ebenfalls ein nicht angegebenes Element mit Benutzerdaten.

Wenn eine Anwendung eine Definitionstabelle für den Clientkanal (CCDT) verwendet, um sich mit einem Warteschlangenmanager zu verbinden, werden die in einer Definition des Clientverbindungskanals angegebenen Benutzerdaten an Kanalexits übergeben, wenn sie aufgerufen werden. Weitere Informationen zur Verwendung einer Definitionstabelle für den Clientkanal finden Sie unter „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden” auf Seite 296.

Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden
Eine Anwendung der IBM MQ classes for JMS kann Definitionen für den Clientverbindungskanal verwenden, die in einer Definitionstabelle für Clientkanal (Client Channel Definition Table, CCDT) gespeichert werden. Sie konfigurieren ein ConnectionFactory-Objekt, um die CCDT zu verwenden. Für die Verwendung gelten einige Einschränkungen.

Alternativ zur Erstellung einer Definition für den Clientverbindungskanal durch die Festlegung bestimmter Eigenschaften eines ConnectionFactory-Objekts kann eine Anwendung der IBM MQ classes for JMS die Definitionen eines Clientverbindungskanals verwenden, die in einer Definitionstabelle für den Clientkanal gespeichert sind. Diese Definitionen werden durch IBM MQ-Scriptbefehle (MQSC) oder IBM MQ-PCF-Befehle (PCF = Programmable Command Format) erstellt. Wenn die Anwendung ein Verbindungsobjekt (Connection) erstellt, durchsuchen die IBM MQ classes for JMS die Definitionstabelle für den Clientkanal nach einer geeigneten Definition eines Clientverbindungskanals und verwenden die Kanaldefinition für den Start eines MQI-Kanals. Sie finden weitere Informationen zu den Definitionstabellen für Clientkanäle und Anweisungen zu deren Erstellung im Abschnitt Definitionstabelle für Clientkanal.

Um eine Definitionstabelle für den Clientkanal zu verwenden, muss die Eigenschaft CCDTURL eines ConnectionFactory-Objekts auf ein URL-Objekt gesetzt sein. IBM MQ classes for JMS lesen die Informationen zur Definitionstabelle für Clientkanäle (CCDT) nicht aus der Konfigurationsdatei des IBM MQ MQI clients aus, auch wenn einige andere Werte daraus verwendet werden (die jeweiligen Werte werden im Abschnitt „Die Konfigurationsdatei IBM MQ classes for JMS/Jakarta Messaging” auf Seite 103 genannt). Das URL-Objekt bindet eine URL ein, die den Namen und die Position der Datei angibt, die die Definitionstabelle des Clientkanals enthält und die festlegt, wie auf die Datei zugegriffen werden kann. Sie können die Eigenschaft 'CCDTURL' mit dem IBM MQ-JMS-Verwaltungstool festlegen. Die Eigenschaft kann jedoch auch von einer Anwendung festgelegt werden, indem ein URL-Objekt erstellt und die Methode 'setCCDTURL()' des ConnectionFactory-Objekts aufgerufen wird.

Wenn die Datei 'ccdt1.tab' beispielsweise eine Definitionstabelle für den Clientkanal enthält und in dem System gespeichert ist, in dem die Anwendung ausgeführt wird, kann die Anwendung die Eigenschaft CCDTURL wie folgt festlegen:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

Ein weiteres Beispiel: Nehmen Sie an, die Datei 'ccdt2.tab' enthält eine Definitionstabelle für Clientkanäle und ist in einem System gespeichert, das nicht dem entspricht, in dem die Anwendung ausgeführt wird. Wenn mithilfe des FTP-Protokolls ein Zugriff auf die Datei erfolgen kann, kann die Anwendung die Eigenschaft CCDTURL wie folgt festlegen:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

Zusätzlich zum Festlegen der Eigenschaft CCDTURL des ConnectionFactory-Objekts muss für die Eigenschaft QMANAGER desselben Objekts einer der folgenden Werte gesetzt werden:

- Der Name eines Warteschlangenmanagers
- Ein Stern (*), gefolgt vom Namen einer Warteschlangenmanagergruppe

Dies sind die gleichen Werte, die für den Parameter **QMgrName** bei einem MQCONN-Aufruf verwendet werden können, der von einer Clientanwendung ausgegeben wird, die Message Queue Interface (MQI) nutzt. Weitere Informationen zur Bedeutung dieser Werte finden Sie im Abschnitt MQCONN. Sie können die Eigenschaft 'QMANAGER' mit dem IBM MQ-JMS-Verwaltungstool oder mit IBM MQ Explorer festlegen. Alternativ kann eine Anwendung die Eigenschaft angeben, indem sie die Methode setQueueManager() des ConnectionFactory-Objekts aufruft.

Wenn eine Anwendung anschließend ein Verbindungsobjekt (Connection) aus dem ConnectionFactory-Objekt erstellt, greifen die IBM MQ classes for JMS auf die Definitionstabelle für den Clientkanal zu, die durch die Eigenschaft CCDTURL angegeben wird. Mit der Eigenschaft QMANAGER durchsuchen sie die Definitionstabelle für den Clientkanal nach einer geeigneten Definition eines Clientverbindungskanals und verwenden die Kanaldefinition für den Start eines MQI-Kanals zum Warteschlangenmanager.

Beachten Sie, dass die Eigenschaften CCDTURL und CHANNEL eines ConnectionFactory-Objekts nicht beide festgelegt werden dürfen, wenn die Anwendung die Methode createConnection() aufruft. Wenn beide Eigenschaften festgelegt sind, löst die Methode eine Ausnahmebedingung aus. Die Eigenschaft CCDTURL oder die Eigenschaft CHANNEL sollte festgelegt werden, wenn ihr Wert nicht null, eine leere Zeichenfolge oder eine Zeichenfolge ist, die nur Leerzeichen enthält.

Wenn IBM MQ classes for JMS eine geeignete Definition eines Clientverbindungskanals Definitionstabelle für den Clientkanal finden, verwenden sie für den Start eines MQI-Kanals nur die Informationen, die aus der Tabelle extrahiert wurden. Alle kanalbezogenen Eigenschaften des ConnectionFactory-Objekts werden ignoriert.

Beachten Sie insbesondere die folgenden Punkte, wenn Sie Transport Layer Security (TLS) verwenden:

- Ein MQI-Kanal verwendet TLS nur dann, wenn die aus der Clientkanal-Definitionstabelle extrahierte Kanaldefinition den Namen einer CipherSpec angibt, die von IBM MQ classes for JMS unterstützt wird.

- Eine Definitionstabelle für Clientkanäle enthält außerdem Informationen zur Position der LDAP-Server (LDAP = Lightweight Directory Access Protocol), auf denen die Zertifikatswiderrufslisten (CRLs) gespeichert sind. IBM MQ classes for JMS verwenden für den Zugriff auf LDAP-Server, die CRLs enthalten, ausschließlich diese Informationen.
- Eine Definitionstabelle für den Clientkanal kann auch die Position eines OCSP-Responders enthalten. IBM MQ classes for JMS kann die OCSP-Informationen einer Clientkanaldefinitionstabelle nicht nutzen. Sie können OCSP jedoch wie im Abschnitt Online Certificate Status Protocol (OCSP) in Java- und JMS-Clientanwendungen beschrieben konfigurieren.

Weitere Informationen zur Verwendung von TLS mit einer Definitionstabelle für Clientkanäle finden Sie im Abschnitt Erweiterten transaktionsorientierten Client mit TLS-Kanälen verwenden.

Beachten Sie auch die folgenden Punkte, wenn Sie Kanalexits verwenden:

- Ein MQI-Kanal verwendet nur die Kanalexits und die zugehörigen Benutzerdaten, die durch die Kanaldefinition angegeben werden, die aus der Definitionstabelle für Clientkanäle extrahiert wurde.
- Eine Kanaldefinition, die aus einer Definitionstabelle für den Clientkanal extrahiert wird, kann Kanalexits angeben, die in Java geschrieben wurden. Dies bedeutet beispielsweise, dass der Parameter SCYEXIT im Befehl DEFINE CHANNEL zum Erstellen einer Clientverbindungskanaldefinition den Namen einer Klasse angeben kann, die die Schnittstelle WMQSecurityExit implementiert. Ähnlich wie im vorherigen Beispiel kann der Parameter SENDEXIT den Namen einer Klasse angeben, die die Schnittstelle 'WMQSendExit' implementiert, und der Parameter RCVEXIT kann den Namen einer Klasse angeben, die die Schnittstelle 'WMQReceiveExit' implementiert. Weitere Informationen zum Schreiben eines Kanalexits in Java finden Sie unter „Kanalexits in Java für IBM MQ classes for JMS schreiben” auf Seite 292.

  Die Verwendung von Kanalexits, die in einer anderen Sprache als Java geschrieben wurden, wird ebenfalls unterstützt. Informationen dazu, wie die Parameter SCYEXIT, SENDEXIT und RCVEXIT im Befehl DEFINE CHANNEL für Kanalexits in anderen Sprachen angegeben werden, finden Sie im Abschnitt DEFINE CHANNEL.

Automatische Wiederholung einer JMS-Clientverbindung
Konfigurieren Sie Ihren JMS-Client so, dass er die Verbindung nach einem Netz-, Warteschlangenmanager- oder Serverfehler automatisch wiederholt.

Wenn eine eigenständige IBM MQ classes for JMS-Anwendung über 'Clienttransport' mit einem Warteschlangenmanager verbunden ist und der Warteschlangenmanager nicht verfügbar ist (z. B. aufgrund eines Netzausfalls, eines Warteschlangenmanagerfehlers oder weil der Warteschlangenmanager gestoppt wurde), lösen IBM MQ classes for JMS beim nächsten Kommunikationsversuch der Anwendung mit dem Warteschlangenmanager die Ausnahmebedingung 'JMSException' aus. Die Anwendung muss die JMSException-Ausnahme abfangen und versuchen, die Verbindung mit dem Warteschlangenmanager wiederherzustellen. Sie können das Design der Anwendung durch das Aktivieren der automatischen Wiederholung einer Clientverbindung vereinfachen. Wenn der Warteschlangenmanager nicht mehr verfügbar ist, versuchen die IBM MQ classes for JMS, automatisch für die Anwendung die Verbindung mit dem Warteschlangenmanager zu wiederholen. Dies bedeutet, dass die Anwendung keine Logik für eine Verbindungswiederholung enthalten muss.

Die Verwendung dieser Implementierung der automatischen Clientverbindungswiederholung wird von Anwendungsservern in Java Platform, Enterprise Edition nicht unterstützt. Informationen zu einer alternativen Implementierung finden Sie im Abschnitt „Automatische Clientverbindungswiederholung in Java EE-Umgebungen verwenden” auf Seite 305.

Automatische JMS-Clientverbindungswiederholung verwenden
Wenn eine eigenständige Anwendung der IBM MQ classes for JMS eine Verbindungsfactory verwendet, für die die Eigenschaft CONNECTIONNAMELIST oder CCDTURL definiert ist, kann die Anwendung die automatische Clientverbindungswiederholung verwenden.

Mithilfe der automatischen Clientverbindungswiederholung können Verbindungen zu Warteschlangenmanagern wiederhergestellt werden, einschließlich Warteschlangenmanagern, die Teil einer Konfiguration für hohe Verfügbarkeit (HA-Konfiguration) sind. Zu den HA-Konfigurationen gehören Multi-Instanz-Warte-

schlangenmanager, RDQM-Warteschlangenmanager oder Warteschlangenmanager mit hoher Verfügbarkeit auf einer IBM MQ-Appliance.

Das Verhalten der Funktion für die automatische Clientverbindungswiederholung, die von den IBM MQ classes for JMS bereitgestellt wird, hängt von den nachfolgenden Eigenschaften ab:

Eigenschaft TRANSPORT der JMS-Verbindungsfactory (Kurzname TRAN)

TRANSPORT gibt an, wie Anwendungen, die die Verbindungsfactory verwenden, eine Verbindung zu einem Warteschlangenmanager herstellen. Diese Eigenschaft muss auf den Wert CLIENT gesetzt werden, damit die automatische Clientverbindungswiederholung verwendet wird. Für Anwendungen, die eine Verbindung zu einem Warteschlangenmanager herstellen, welcher eine Verbindungsfactory verwendet, deren Eigenschaft TRANSPORT auf BIND, DIRECT oder DIRECTHTTP gesetzt ist, ist die automatische Clientverbindungswiederholung nicht verfügbar.

Eigenschaft QMANAGER der JMS-Verbindungsfactory (Kurzname QMGR)

Die Eigenschaft QMANAGER gibt den Namen des Warteschlangenmanagers an, zu dem die Verbindungsfactory eine Verbindung herstellt.

Eigenschaft CONNECTIONNAMELIST der JMS-Verbindungsfactory (Kurzname CRHOSTS)

Bei der Eigenschaft CONNECTIONNAMELIST handelt es sich um eine durch Kommas getrennte Liste, in der jeder Eintrag Informationen zu dem Hostnamen und Port enthält, die bei Verwendung des CLIENT-Transportprotokolls für die Verbindung zu dem Warteschlangenmanager verwendet werden sollen, welcher über die Eigenschaft QMANAGER angegeben ist. Die Liste hat das folgende Format: Hostname(Port), Hostname(Port).

Eigenschaft CCDTURL der JMS-Verbindungsfactory (Kurzname CCDT)

Die Eigenschaft CCDTURL verweist auf die Clientkanaldefinitionstabelle, die von den IBM MQ classes for JMS zum Herstellen einer Verbindung zu einem Warteschlangenmanager mithilfe einer CCDT verwendet wird.

Eigenschaft CLIENTRECONNECTOPTIONS der JMS-Verbindungsfactory (Kurzname CROPT)

Über CLIENTRECONNECTOPTIONS wird gesteuert, ob die IBM MQ classes for JMS versuchen, automatisch für eine Anwendung eine Verbindung zu einem Warteschlangenmanager herzustellen, wenn ein Warteschlangenmanager vorübergehend nicht mehr verfügbar ist.

Attribut DefRecon in der Zeilengruppe Channels der Clientkonfigurationsdatei

Das Attribut DefRecon bietet eine Administrationsoption, mit der für alle Anwendungen die automatische Verbindungswiederholung aktiviert bzw. die automatische Verbindungswiederholung für Anwendungen, die für die automatische Verbindungswiederholung geschrieben wurden, inaktiviert werden kann.

Die automatische Clientverbindungswiederholung ist nur verfügbar, wenn eine Anwendung erfolgreich eine Verbindung zu einem Warteschlangenmanager herstellt.

Wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager herstellt, der das CLIENT-Transportprotokoll verwendet, stellen die IBM MQ classes for JMS anhand des Werts der Eigenschaft CLIENTRECONNECTOPTIONS der Verbindungsfactory fest, ob die automatische Clientverbindungswiederholung verwendet werden soll, wenn der Warteschlangenmanager, mit dem die Anwendung verbunden ist, nicht mehr verfügbar ist. In Tabelle 1 sind die möglichen Werte für die Eigenschaft CLIENTRECONNECTOPTIONS zusammen mit dem jeweiligen Verhalten der IBM MQ classes for JMS aufgeführt:

Tabelle 44. Mögliche Werte für die Eigenschaft CLIENTRECCECTOPTIONS

CLIENTRECONNECTOPTIONS	**Verhalten der IBM MQ classes for JMS**
ANY	Wenn CONNECTIONNAMELIST festgelegt ist, wird der Wert der Eigenschaft CONNECTIONNAMELIST zum Öffnen einer Verbindung zu einer Kombination aus Hostname und Port sowie zum Herstellen einer Verbindung zu einem beliebigen Warteschlangenmanager verwendet. Um diese Option für die automatische Clientverbindungswiederholung verwenden zu können, muss für die Eigenschaft QMANAGER einer der folgenden Werte definiert sein: der Standardwert oder "*". Wenn CCDTURL festgelegt ist, öffnen Sie die durch die Eigenschaft CCDTURL angegebene Clientkanaldefinitionstabelle, wählen Sie einen Eintrag in der Tabelle aus und verwenden Sie dann diesen Eintrag, um einen Clientverbindungskanal zu einem Warteschlangenmanager zu starten. Um diese Option für die automatische Clientverbindungswiederholung verwenden zu können, muss für die Eigenschaft QMANAGER einer der folgenden Werte definiert sein: • Ein Stern (*) • Ein Stern (*), gefolgt vom Namen einer Warteschlangenmanagergruppe • Eine leere Zeichenfolge bzw. eine Zeichenfolge, die nur Leerzeichen enthält
ASDEF	Verwenden Sie den Wert von DefRecon, um herauszufinden, ob die automatische Clientverbindungswiederholung verfügbar ist.
INAKTIVIERT	Es erfolgt keine automatische Clientverbindungswiederholung und es wird eine JMS-Ausnahme (JMSException) an die Anwendung zurückgegeben.

Tabelle 44. Mögliche Werte für die Eigenschaft CLIENTRECCECTOPTIONS (Forts.)	
CLIENTRECONNECTOPTIONS	**Verhalten der IBM MQ classes for JMS**
QMGR	Gibt an, dass der Client erneut eine Verbindung zu demselben Warteschlangenmanager herstellen muss. Diese Option muss für Hochverfügbarkeitslösungen verwendet werden, in denen ein erneutes Verbinden mit einer anderen Instanz desselben Warteschlangenmanagers erforderlich ist. Wenn CONNECTIONNAMELIST festgelegt ist, wird der Wert der Eigenschaft CONNECTIONNAMELIST zum Öffnen einer Verbindung zu einer Kombination aus Hostname und Port sowie zum Herstellen einer Verbindung zu dem durch die Eigenschaft QMANAGER angegebenen Warteschlangenmanager verwendet. Wenn CCDTURL festgelegt ist, öffnen Sie die durch die Eigenschaft CCDTURL angegebene Clientkanaldefinitionstabelle, suchen Sie die Einträge in der Tabelle, die mit dem durch die Eigenschaft QMANAGER angegebenen Warteschlangenmanagernamen übereinstimmen und verwenden Sie dann diese Einträge, um einen Clientverbindungskanal zu dem Warteschlangenmanager zu starten.

Wenn CONNECTIONNAMELIST festgelegt ist, stellen die IBM MQ classes for JMS bei der automatischen Clientverbindungswiederholung anhand der Informationen in der Eigenschaft CONNECTIONNAMELIST der Verbindungsfactory fest, zu welchem System die Verbindung wiederhergestellt werden soll.

Zunächst versuchen die IBM MQ classes for JMS, die Verbindung unter Verwendung des im ersten Eintrag von CONNECTIONNAMELIST angegebenen Hostnamens und Ports wiederherzustellen. Wenn eine Verbindung hergestellt wird, versuchen die IBM MQ classes for JMS anschließend eine Verbindung zu dem Warteschlangenmanager mit dem in der Eigenschaft QMANAGER angegebenen Namen. Falls eine Verbindung zu dem Warteschlangenmanager hergestellt werden kann, öffnen die IBM MQ classes for JMS alle IBM MQ-Objekte erneut, die die Anwendung vor der automatischen Clientverbindungswiederholung geöffnet hatte, und werden weiter wie zuvor ausgeführt.

Falls über den ersten Eintrag in CONNECTIONNAMELIST keine Verbindung zu dem erforderlichen Warteschlangenmanager hergestellt werden kann, probieren die IBM MQ classes for JMS den zweiten Eintrag in CONNECTIONNAMELIST aus, usw.

Wenn die IBM MQ classes for JMS alle Einträge in CONNECTIONNAMELIST ausprobiert haben, warten sie eine gewisse Zeit, bevor sie erneut versuchen, die Verbindung wiederherzustellen. Für den neuen Verbindungsversuch beginnen die IBM MQ classes for JMS mit dem ersten Eintrag in CONNECTIONNAMELIST. Anschließend probieren sie nacheinander alle Einträge in CONNECTIONNAMELIST aus, bis eine Verbindungswiederholung möglich ist oder das Ende von CONNECTIONNAMELIST erreicht wurde. In diesem Fall warten die IBM MQ classes for JMS eine gewisse Zeit, bevor sie die Verbindung erneut versuchen.

Wenn CCDTURL festgelegt ist, stellen die IBM MQ classes for JMS bei der automatischen Clientverbindungswiederholung anhand der Clientkanaldefinitionstabelle, die in der Eigenschaft CCDTURL angegeben ist, fest, zu welchem System die Verbindung wiederhergestellt werden soll.

Von den IBM MQ classes for JMS wird zunächst die Definitionstabelle für Clientkanal analysiert und ein geeigneter Eintrag gefunden, der dem Wert der Eigenschaft QMANAGER entspricht. Wenn ein Eintrag gefunden wird, versuchen die IBM MQ classes for JMS, über diesen Eintrag die Verbindung zu dem erforderlichen Warteschlangenmanager wiederherzustellen. Falls eine Verbindung zu dem Warteschlangenmanager hergestellt werden kann, öffnen die IBM MQ classes for JMS alle IBM MQ-Objekte erneut, die

die Anwendung vor der automatischen Clientverbindungswiederholung geöffnet hatte, und werden weiter wie zuvor ausgeführt.

Falls keine Verbindung zu dem erforderlichen Warteschlangenmanager hergestellt werden kann, suchen die IBM MQ classes for JMS in der Definitionstabelle für Clientkanal nach einem anderen geeigneten Eintrag und versuchen, diesen zu verwenden usw.

Wenn die IBM MQ classes for JMS alle geeigneten Einträge in der Definitionstabelle für Clientkanal ausprobiert haben, warten sie eine gewisse Zeit, bevor sie erneut versuchen, die Verbindung wiederherzustellen. Für einen erneuten Verbindungsversuch analysieren die IBM MQ classes for JMS die Definitionstabelle für Clientkanal erneut und versuchen es mit dem ersten geeigneten Eintrag. Anschließend probieren sie nacheinander alle geeigneten Einträge in der Definitionstabelle für Clientkanal aus, bis eine Verbindungswiederholung möglich ist oder der letzte geeignete Eintrag in der Tabelle erreicht wurde. In diesem Fall warten die IBM MQ classes for JMS eine gewisse Zeit, bevor sie die Verbindung erneut versuchen.

Unabhängig davon, ob CONNECTIONNAMELIST oder CCDTURL verwendet wird, wird der Prozess der automatischen Clientverbindungswiederholung fortgesetzt, bis die IBM MQ classes for JMS erfolgreich wieder mit dem in der Eigenschaft QMANAGER angegebenen Warteschlangenmanager verbunden sind.

Standardmäßig erfolgen die Verbindungswiederholungsversuche in folgenden Intervallen:

- Der erste Versuch nach einer Anfangsverzögerung von 1 Sekunde plus einem zufälligen Element von bis zu 250 Millisekunden.
- Der zweite Versuch 2 Sekunden, plus ein zufälliges Intervall von bis zu 500 Millisekunden, nachdem der erste Versuch fehlgeschlagen ist.
- Der dritte Versuch 4 Sekunden, plus ein zufälliges Intervall von bis zu 1 Sekunde, nachdem der zweite Versuch fehlgeschlagen ist.
- Der vierte Versuch 8 Sekunden, plus ein zufälliges Intervall von bis zu 2 Sekunden, nachdem der dritte Versuch fehlgeschlagen ist.
- Der fünfte Versuch 16 Sekunden, plus ein zufälliges Intervall von bis zu 4 Sekunden, nachdem der vierte Versuch fehlgeschlagen ist.
- Der sechste Versuch und alle nachfolgenden Versuche erfolgen 25 Sekunden plus ein zufälliges Intervall von bis zu 6 Sekunden und 250 Millisekunden, nachdem der vorherige Versuch fehlgeschlagen ist.

Die Verbindungswiederholungsversuche finden um teils feste, teils zufällige Intervalle verzögert statt. Auf diese Weise soll verhindert werden, dass alle Anwendungen der IBM MQ classes for JMS, die mit einem jetzt nicht mehr verfügbaren Warteschlangenmanager verbunden waren, gleichzeitig versuchen, die Verbindung wiederherzustellen.

Wenn Sie die Standardwerte erhöhen müssen, damit sie besser der Zeit entsprechen, die ein Warteschlangenmanager für die Wiederherstellung bzw. ein Standby-Warteschlangenmanager für die Aktivierung benötigt, ändern Sie das Attribut ReconDelay in der Zeilengruppe Channel der Clientkonfigurationsdatei. Weitere Informationen finden Sie im Abschnitt Zeilengruppe CHANNELS der Clientkonfigurationsdatei.

Ob eine Anwendung der IBM MQ classes for JMS nach der automatischen Verbindungswiederholung weiterhin ordnungsgemäß funktioniert, hängt von ihrer Gestaltung ab. Informieren Sie sich in den entsprechenden Abschnitten, wie Anwendungen gestaltet werden müssen, damit sie die Funktion für die automatische Verbindungswiederholung verwenden können.

Ursachencodes, die angeben, dass ein Warteschlangenmanager nicht mehr verfügbar ist
Hier wird erläutert, welche Ursachencodes bei einem automatischen Verbindungswiederholungsversuch der IBM MQ classes for JMS angeben, dass ein Warteschlangenmanager nicht mehr verfügbar ist oder nicht erreicht werden kann.

Im Abschnitt „Automatische Wiederholung einer JMS-Clientverbindung” auf Seite 298 finden Sie eine Übersicht über JMS-Ausnahmebedingungen (JMSExceptions) und erfahren, wie Ihre Anwendungen automatisch erneut gestartet werden können. Im Abschnitt „Automatische JMS-Clientverbindungswiederholung verwenden” auf Seite 298 werden die Anforderungen für eine automatische Clientverbindungswiederholung erläutert.

Im Folgenden sind die IBM MQ-Ursachencodes aufgelistet, die von Ihrer Anwendung geprüft werden sollten:

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

Die meisten JMS-Ausnahmebedingungen (JMSExceptions), die an Unternehmensanwendungen zurückgegeben werden, enthalten eine verlinkte MQException, die den Ursachencode enthält. Um die Wiederholungslogik für die Ursachencodes in der vorherigen Liste zu implementieren, sollten Ihre Unternehmensanwendungen diese verlinkte Ausnahmebedingung mithilfe von ähnlichem Code wie im folgenden Beispiel überprüfen:

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

Zugehörige Konzepte
IBM MQ-Klassen für JMS

Automatische Clientverbindungswiederholung in Java SE- und Java EE-Umgebungen verwenden
Sie können die automatische IBM MQ-Clientverbindungswiederholung verwenden, um verschiedene Lösungen für Hochverfügbarkeit (HA) und Disaster-Recovery (DR) in einer Java SE-und Java EE-Umgebung zu erleichtern.

Auf den unterschiedlichen Plattformen sind verschiedene HA- und DR-Lösungen verfügbar:

- Multi Multi-Instanz-Warteschlangenmanager sind Instanzen desselben Warteschlangenmanagers, die auf verschiedenen Servern konfiguriert sind (siehe Multi-Instanz-Warteschlangenmanager). Eine Instanz des Warteschlangenmanagers ist als aktive Instanz definiert, die andere ist eine Standby-Instanz. Wenn die aktive Instanz ausfällt, wird der Multi-Instanz-Warteschlangenmanager automatisch auf dem Standby-Server gestartet.

  Sowohl der aktive Warteschlangenmanager als auch der Standby-Warteschlangenmanager haben dieselbe Warteschlangenmanager-ID (QMID). IBM MQ -Clientanwendungen, die eine Verbindung zu einem Warteschlangenmanager mit mehreren Instanzen herstellen, können so konfiguriert werden, dass

sie mithilfe der automatischen Clientverbindungswiederholung automatisch eine Verbindung zu einer Standby-Instanz eines Warteschlangenmanagers herstellen.

- Linux RDQM (Replicated Data Queue Manager) ist eine Hochverfügbarkeitslösung, die auf Linux-Plattformen verfügbar ist (siehe RDQM-Hochverfügbarkeit). Eine RDQM-Konfiguration besteht aus drei Servern, die, jeder mit einer Instanz des Warteschlangenmanagers, in einer Hochverfügbarkeitsgruppe (HA-Gruppe) konfiguriert werden. Eine Instanz ist der aktive Warteschlangenmanager, der seine Daten synchron zu den anderen beiden Instanzen repliziert. Fällt der Server mit dem aktiven Warteschlangenmanager aus, wird eine andere Instanz des Warteschlangenmanagers gestartet, die über die aktuellen Betriebsdaten verfügt. Die drei Instanzen des Warteschlangenmanagers teilen sich eine variable IP-Adresse, sodass Clients nur mit einer einzigen IP-Adresse konfiguriert werden müssen. Clientanwendungen, die eine Verbindung zu einem RDQM-Warteschlangenmanager herstellen, können so konfiguriert werden, dass sie mithilfe der automatischen Clientverbindungswiederholung automatisch eine Verbindung zu einer Standby-Instanz eines Warteschlangenmanagers herstellen.
- MQ Appliance Eine Hochverfügbarkeitslösung kann auch durch ein Paar von IBM MQ-Appliances bereitgestellt werden (siehe Hochverfügbarkeit und Disaster-Recovery in der IBM MQ Appliance-Dokumentation). Ein Warteschlangenmanager mit hoher Verfügbarkeit wird auf einer der Appliances ausgeführt und repliziert dabei synchron Daten auf die Standby-Instanz des Warteschlangenmanagers auf der anderen Appliance. Wenn die primäre Appliance ausfällt, wird der Warteschlangenmanager automatisch gestartet und auf der anderen Appliance ausgeführt. Die zwei Instanzen des Warteschlangenmanagers können so konfiguriert werden, dass sie sich eine variable IP-Adresse teilen, sodass Clients nur mit einer einzigen IP-Adresse konfiguriert werden müssen. Clientanwendungen, die eine Verbindung zu einem Warteschlangenmanager mit hoher Verfügbarkeit auf einer IBM MQ-Appliance herstellen, können so konfiguriert werden, dass sie mithilfe der automatischen Clientverbindungswiederholung automatisch eine Verbindung zur Standby-Instanz eines Warteschlangenmanagers herstellen.

Anmerkung: In Java EE-Umgebungen (z. B. WebSphere Application Server) wird die automatische Clientverbindungswiederholung mit Aktivierungsspezifikationen, die die von IBM MQ classes for JMS bereitgestellte Funktionalität nutzen, nicht unterstützt. Der IBM MQ-Ressourcenadapter bietet einen eigenen Mechanismus zur Verbindungswiederherstellung für Aktivierungsspezifikationen, wenn der Warteschlangenmanager, zu dem die Aktivierungsspezifikation gerade eine Verbindung hergestellt hat, nicht mehr verfügbar ist. Weitere Informationen finden Sie unter „Unterstützung für automatische Clientverbindungswiederholung in Java EE-Umgebungen” auf Seite 306.

Zugehörige Konzepte

Warteschlangenmanager mit mehreren Instanzen

Automatische Clientverbindungswiederholung

Zugehörige Verweise

RDQM-Hochverfügbarkeit

Automatische Clientverbindungswiederholung in Java SE-Umgebungen verwenden

Anwendungen, die die IBM MQ classes for JMS in Java SE-Umgebungen verwenden, können die Funktionalität der automatischen Wiederherstellung der Clientverbindung über die Verbindungsfactory-Eigenschaft **CLIENTRECONNECTOPTIONS** nutzen.

Die Verbindungsfactory-Eigenschaft **CLIENTRECONNECTOPTIONS** verwendet zwei weitere Verbindungsfactory-Eigenschaften, **CONNECTIONNAMELIST** und **CCDTURL**, um festzustellen, wie die Verbindung mit dem Server hergestellt werden soll, auf dem der Warteschlangenmanager ausgeführt wird.

CONNECTIONNAMELISTEigenschaft

Die Eigenschaft **CONNECTIONNAMELIST** ist eine durch Kommas getrennte Liste mit Hostnamen und Ports, die zum Herstellen einer Verbindung zu einem Warteschlangenmanager im Clientmodus verwendet werden können. Diese Eigenschaft wird mit den Werten **QMANAGER** und **CHANNEL** verwendet. Wenn eine Anwendung die Eigenschaft **CONNECTIONNAMELIST** zur Herstellung einer Clientverbindung verwendet, versuchen die IBM MQ classes for JMS in der durch die Liste vorgegebenen Reihenfolge eine Verbindung zu einem Host herzustellen. Ist der erste Warteschlangenmanagerhost nicht verfügbar, versuchen die IBM MQ classes for JMS eine Verbindung zum nächsten Host auf der Liste herzustellen. Kann bis zum Ende der

Liste keine Verbindung hergestellt werden, so geben die IBM MQ classes for JMS eine Ausnahme mit dem Ursachencode `MQRC_QMGR_NOT_AVAILABLE` IBM MQ aus.

Schlägt der Warteschlangenmanager, mit dem die Anwendung verbunden ist, fehl, erhalten alle Anwendungen, die zur Verbindung mit diesem Warteschlangenmanager eine **CONNECTIONNAMELIST** verwendet hatten, eine Ausnahme, die darauf hinweist, dass der Warteschlangenmanager nicht verfügbar ist. Die Anwendung muss diese Ausnahme abfangen und alle Ressourcen freigeben, die sie in Verbindung mit diesem Warteschlangenmanager verwendet hatte. Zur Herstellung einer erneuten Verbindung muss die Anwendung dann die Verbindungsfactory verwenden. Die Verbindungsfactory versucht dann erneut ebenfalls in Listenreihenfolge eine Verbindung zu einem Host herzustellen, wobei der zuvor ausgefallene Warteschlangenmanager nun nicht zur Verfügung steht. Daher versucht die Verbindungsfactory eine Verbindung mit einem anderen Host der Liste herzustellen.

CCDTURLEigenschaft

Die Eigenschaft **CCDTURL** enthält einen Uniform Resource Locator (URL), der auf eine Clientkanal-Definitionstabelle (CCDT) verweist. Diese Eigenschaft wird in Verbindung mit der Eigenschaft **QMANAGER** verwendet. Die CCDT enthält eine Liste mit Clientkanälen, über die Verbindungen zu den in einem IBM MQ-System definierten Warteschlangenmanagern hergestellt werden können. Informationen zur Verwendung von CCDTs durch die IBM MQ classes for JMS finden Sie im Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden” auf Seite 296.

Eigenschaft CLIENTRECONNECTOPTIONS zur Aktivierung der automatischen Wiederherstellung der Clientverbindung innerhalb der IBM MQ classes for JMS verwenden

Mit der Eigenschaft **CLIENTRECONNECTOPTIONS** kann die Funktionalität der automatischen Wiederherstellung der Clientverbindung innerhalb der IBM MQ classes for JMS aktiviert werden. Diese Eigenschaft kann folgende Werte haben:

ASDEF
: Das Verhalten der Funktionalität der automatischen Wiederherstellung der Clientverbindung wird durch den Standardwert definiert, der in der Zeilengruppe 'channel' der Konfigurationsdatei des IBM MQ-Clients (`mqclient.ini`) angegeben ist.

Inaktiviert
: Die Funktionalität der automatischen Wiederherstellung der Clientverbindung ist inaktiviert.

QMGR
: Die IBM MQ classes for JMS versuchen eine Verbindung zu einem Warteschlangenmanager mit der gleichen Warteschlangenmanager-ID wie zuvor herzustellen. Dazu wird eine der folgenden Optionen verwendet:

  - Die Eigenschaft **CONNECTIONNAMELIST** und der in der Eigenschaft **CHANNEL** definierte Kanal.
  - Die in der Eigenschaft **CCDTURL** definierte CCDT.

Beliebig
: Die IBM MQ classes for JMS versuchen eine Verbindung zu einem Warteschlangenmanager mit dem gleichen Namen wie zuvor herzustellen. Dazu wird die Eigenschaft **CONNECTIONNAMELIST** oder die Eigenschaft **CCDTURL** verwendet.

Zugehörige Informationen

Zeilengruppe 'CHANNELS' in der Clientkonfigurationsdatei

Automatische Clientverbindungswiederholung in Java EE-Umgebungen verwenden

Der IBM MQ-Ressourcenadapter, der in Java EE-Umgebungen (Java Platform, Enterprise Edition) bereitgestellt werden kann, und der WebSphere Application Server IBM MQ-Messaging-Provider verwenden die IBM MQ classes for JMS zur Kommunikation mit IBM MQ-Warteschlangenmanagern. Der IBM MQ-Ressourcenadapter und der WebSphere Application Server-Messaging-Provider für IBM MQ stellen eine Reihe von Mechanismen bereit, die Aktivierungsspezifikationen, WebSphere Application Server-Listener-Ports und Anwendungen, die innerhalb von Client-Containern ausgeführt werden, die automatische

Verbindungswiederholung zu einem Warteschlangenmanager ermöglichen. Enterprise JavaBeans und webbasierte Anwendungen müssen ihre eigene Verbindungswiederholungslogik implementieren.

Anmerkung: Die automatische Clientverbindungswiederholung mit Aktivierungsspezifikationen unter Verwendung der von IBM MQ classes for JMS bereitgestellten Funktionalität wird nicht unterstützt (siehe „Automatische Wiederholung einer JMS-Clientverbindung" auf Seite 298). Der IBM MQ-Ressourcenadapter bietet einen eigenen Mechanismus zur Verbindungswiederherstellung für Aktivierungsspezifikationen, wenn der Warteschlangenmanager, zu dem die Aktivierungsspezifikation gerade eine Verbindung hergestellt hat, nicht mehr verfügbar ist.

Der vom Ressourcenadapter bereitgestellte Mechanismus wird wie folgt gesteuert:

- über die IBM MQ-Ressourcenadaptereigenschaft **reconnectionRetryCount**
- über die IBM MQ-Ressourcenadaptereigenschaft **reconnectionRetryInterval**
- über die Aktivierungsspezifikationseigenschaft **connectionNameList**

Weitere Informationen zu diesen Eigenschaften finden Sie im Abschnitt „Eigenschaften des ResourceAdapter-Objekts konfigurieren" auf Seite 472.

Die Verwendung der automatischen Clientverbindungswiederholung innerhalb der Methode `onMessage()` einer Message-driven-Bean-Anwendung oder einer anderen Anwendung, die in der Java Platform, Enterprise Edition-Umgebung ausgeführt wird, wird nicht unterstützt. Die Anwendung muss ihre eigene Logik für die Verbindungswiederholung implementieren, wenn der Warteschlangenmanager, zu dem sie eine Verbindung hergestellt hat, nicht mehr verfügbar ist. Weitere Informationen finden Sie unter „Verbindungswiederherstellungslogik in einer Java EE-Anwendung implementieren" auf Seite 314.

Unterstützung für automatische Clientverbindungswiederholung in Java EE-Umgebungen
In Java EE-Umgebungen, wie z. B. WebSphere Application Server, unterstützen der IBM MQ-Ressourcenadapter und der IBM MQ-Messaging-Provider von WebSphere Application Server eine Reihe von Mechanismen, mit denen Anwendungen eine Verbindung zu einem Warteschlangenmanager automatisch wiederherstellen können. In einigen Konfigurationen ist diese Unterstützung jedoch eingeschränkt.

Der IBM MQ -Ressourcenadapter, der in Java EE -Umgebungen und dem WebSphere Application Server IBM MQ -Messaging-Provider implementiert werden kann, verwendet IBM MQ classes for JMS für die Kommunikation mit den IBM MQ -Warteschlangenmanagern.

Eine Übersicht über die Unterstützung der automatischen Wiederherstellung der Clientverbindung durch den IBM MQ-Ressourcenadapter und den WebSphere Application Server IBM MQ-Messaging-Provider finden Sie in den folgenden Tabellen.

Tabelle 45. Übersicht über die Unterstützung für Optionen für die automatische Clientverbindungswiederholung in Java EE-Umgebungen

Optionen für die automatische Verbindungswiederholung	**Eigenschaft CONNECTIONNAMELIST**	**Eigenschaft CCDTURL**	**Eigenschaft CLIENTRECONNECTOPTIONS**	**Alternative Methode der automatischen Wiederherstellung der Clientverbindung**
Aktivierungsspezifikationen	Unterstützt mit Einschränkungen	Unterstützt mit Einschränkungen	Nicht unterstützt	Die Java EE-Umgebung und die Aktivierungsspezifikationen geben ihren eigenen Verbindungswiederholungsmechanismus an

Tabelle 45. Übersicht über die Unterstützung für Optionen für die automatische Clientverbindungswiederholung in Java EE-Umgebungen (Forts.)

Optionen für die automatische Verbindungswiederholung	**Eigenschaft CONNECTIONNAMELIST**	**Eigenschaft CCDTURL**	**Eigenschaft CLIENTRECONNECTOPTIONS**	**Alternative Methode der automatischen Wiederherstellung der Clientverbindung**
WebSphere Application Server-Listenerports	Unterstützt mit Einschränkungen	Unterstützt mit Einschränkungen	Nicht unterstützt	WebSphere Application Server stellt seinen eigenen Wiederherstellungsmechanismus bereit
Enterprise JavaBeans und webbasierte Anwendungen	Unterstützt mit Einschränkungen	Unterstützt mit Einschränkungen	Nicht unterstützt	Anwendung implementiert ihre eigene Wiederherstellungslogik
Innerhalb von Clientcontainern ausgeführte Anwendungen	Unterstützt	Unterstützt	Unterstützt	Nicht zutreffend

Message-driven Bean-Anwendungen in einer Java EE-Umgebung, wie zum Beispiel IBM MQ classes for JMS, können zur Verarbeitung von Nachrichten in einem IBM MQ-System Aktivierungsspezifikationen verwenden. Aktivierungsspezifikationen erkennen in einem IBM MQ-System ankommende Nachrichten und stellen diese den MDBs zur Verarbeitung zu. Message-driven Beans können auch innerhalb ihrer Methode **onMessage()** weitere Verbindungen zu IBM MQ -Systemen herstellen. Informationen zur automatischen Wiederherstellung der Clientverbindung im Kontext mit diesen Verbindungen finden Sie im Abschnitt Enterprise JavaBeans und webbasierte Anwendungen.

Aktivierungsspezifikationen
In Aktivierungsspezifikationen werden die Eigenschaften **CONNECTIONNAMELIST** und **CCDTURL** mit Einschränkungen unterstützt. Die Eigenschaft **CLIENTRECONNECTOPTIONS** wird nicht unterstützt.

Message-driven Bean-Anwendungen (MDBs) in einer Java EE-Umgebung, wie zum Beispiel WebSphere Application Server, können zur Verarbeitung von Nachrichten in einem IBM MQ-System Aktivierungsspezifikationen verwenden.

Aktivierungsspezifikationen erkennen in einem IBM MQ-System ankommende Nachrichten und stellen diese den MDBs zur Verarbeitung zu. In diesem Abschnitt wird erläutert, wie Aktivierungsspezifikationen das IBM MQ-System überwachen.

MDBs können auch innerhalb ihrer `onMessage()` -Methode zusätzliche Verbindungen zu IBM MQ -Systemen herstellen.

Details zur automatischen Wiederherstellung der Clientverbindung im Kontext mit diesen Verbindungen finden Sie im Abschnitt „Enterprise JavaBeans und webbasierte Anwendungen” auf Seite 311.

CONNECTIONNAMELISTEigenschaft

Beim Start versucht die Aktivierungsspezifikation wie folgt eine Verbindung zu einem Warteschlangenmanager herzustellen:

- Direkt mit dem in der Eigenschaft **QMANAGER** angegebenen Warteschlangenmanager
- Über den in der Eigenschaft **CHANNEL** genannten Kanal
- Über den im ersten Eintrag der **CONNECTIONNAMELIST** angegebenen Hostnamen und Port

Wenn die Aktivierungsspezifikation über den ersten Eintrag der Verbindungsliste keine Verbindung mit einem Warteschlangenmanager herstellen kann, versucht sie es über den zweiten, dann den dritten Eintrag usw., bis der Verbindungsaufbau gelingt oder das Ende der Liste erreicht ist.

Gelingt der Verbindungsaufbau über keinen der Einträge der Liste **CONNECTIONNAMELIST**, wird die Aktivierungsspezifikation beendet und muss erneut gestartet werden.

Wenn die Verbindung für die Aktivierungsspezifikation steht, erhält sie Nachrichten vom IBM MQ-System und stellt diese einem MDB zur Verarbeitung zu.

Sollte der Warteschlangenmanager während der Verarbeitung einer Nachricht ausfallen, so wird dieser Fehler von der Java EE-Umgebung erkannt, und es wird versucht, die Verbindung für die Aktivierungsspezifikation wiederherzustellen.

Bei diesem Wiederherstellungsversuch verwendet die Aktivierungsspezifikation die Informationen der Eigenschaft **CONNECTIONNAMELIST** wie zuvor bei der ursprünglichen Verbindung.

Wenn die Aktivierungsspezifikation alle Einträge in der **CONNECTIONNAMELIST** versucht und immer noch keine Verbindung zum Warteschlangenmanager herstellen kann, wartet die Aktivierungsspezifikation den in der IBM MQ Ressourcenadaptereigenschaft **reconnectionRetryInterval** angegebenen Zeitraum, bevor sie es erneut versucht.

Die IBM MQ Ressourcenadaptereigenschaft **reconnectionRetryCount** definiert die Anzahl aufeinanderfolgender Verbindungswiederholungen, die vorgenommen werden, bevor eine Aktivierungsspezifikation gestoppt wird, und erfordert einen manuellen Neustart

Gelingt die Verbindungswiederherstellung der Aktivierungsspezifikation mit einem IBM MQ-System, so führt die Java EE-Umgebung die erforderlichen transaktionsorientierten Bereinigungen durch und setzt anschließend die Nachrichtenzustellung an die MDBs zur Weiterverarbeitung fort.

Damit diese transaktionsorientierte Bereinigung ordnungsgemäß durchgeführt werden kann, muss die Java EE-Umgebung Zugriff auf die Protokolle des ausgefallenen Warteschlangenmanagers haben.

Wenn die Aktivierungsspezifikationen mit transaktionsorientierten MDBs verwendet werden, die an XA-Transaktionen mitwirken, und die Spezifikationen Verbindungen zu einem Multi-Instanz-Warteschlangenmanager herstellen, muss die **CONNECTIONNAMELIST** einen Eintrag sowohl für die aktive als auch für die Standby-Warteschlangenmanagerinstanz enthalten.

Das bedeutet, dass die Java EE-Umgebung bei einer erforderlichen Transaktionswiederherstellung unabhängig davon, mit welchem Warteschlangenmanager die Verbindung nach einem Ausfall wiederhergestellt wird, auf die Protokolle der Warteschlangenmanager zugreifen kann.

Wenn die transaktionsorientierten MDBs mit eigenständigen Warteschlangenmanagern verwendet werden, muss die Eigenschaft **CONNECTIONNAMELIST** einen einzelnen Eintrag enthalten, um sicherzustellen, dass die Aktivierungsspezifikation nach einem Ausfall immer eine Verbindung zu demselben Warteschlangenmanager herstellt, der auf demselben System ausgeführt wird.

CCDTURLEigenschaft

Beim Start versucht die Aktivierungsspezifikation, unter Verwendung des ersten Eintrags in der Definitionstabelle für Clientkanäle (CCDT) eine Verbindung zu dem in der Eigenschaft **QMANAGER** angegebenen Warteschlangenmanager herzustellen.

Wenn die Aktivierungsspezifikation über den ersten Eintrag der Tabelle keine Verbindung mit einem Warteschlangenmanager herstellen kann, versucht sie es über den zweiten, dann den dritten Eintrag usw., bis der Verbindungsaufbau gelingt oder das Ende der Tabelle erreicht ist.

Gelingt der Verbindungsaufbau über keinen der Einträge der CCDT, wird die Aktivierungsspezifikation beendet und muss erneut gestartet werden.

Wenn die Verbindung für die Aktivierungsspezifikation steht, erhält sie Nachrichten vom IBM MQ-System und stellt diese einem MDB zur Verarbeitung zu.

Sollte der Warteschlangenmanager während der Verarbeitung einer Nachricht ausfallen, so wird dieser Fehler von der Java EE-Umgebung erkannt, und es wird versucht, die Verbindung für die Aktivierungsspezifikation wiederherzustellen.

Bei diesem Wiederherstellungsversuch verwendet die Aktivierungsspezifikation die Informationen der Eigenschaft CCDT wie zuvor bei der ursprünglichen Verbindung.

Wenn die Aktivierungsspezifikation alle Einträge in der CCDT versucht und immer noch keine Verbindung zum Warteschlangenmanager herstellen kann, wartet die Aktivierungsspezifikation die in der IBM MQ Ressourcenadaptereigenschaft **reconnectionRetryInterval** angegebene Zeit, bevor sie es erneut versucht.

Die IBM MQ Ressourcenadaptereigenschaft **reconnectionRetryCount** definiert die Anzahl aufeinanderfolgender Verbindungswiederholungen, die vorgenommen werden, bevor eine Aktivierungsspezifikation gestoppt wird, und erfordert einen manuellen Neustart

Gelingt die Verbindungswiederherstellung der Aktivierungsspezifikation mit einem IBM MQ-System, so führt die Java EE-Umgebung die erforderlichen transaktionsorientierten Bereinigungen durch und setzt anschließend die Nachrichtenzustellung an die MDBs zur Weiterverarbeitung fort.

Damit diese transaktionsorientierte Bereinigung ordnungsgemäß durchgeführt werden kann, muss die Java EE-Umgebung Zugriff auf die Protokolle des ausgefallenen Warteschlangenmanagers haben.

Wenn die Aktivierungsspezifikationen mit transaktionsorientierten MDBs verwendet werden, die an XA-Transaktionen mitwirken, und die Spezifikationen Verbindungen zu einem Multi-Instanz-Warteschlangenmanager herstellen, muss die CCDT einen Eintrag sowohl für die aktive als auch für die Standby-Warteschlangenmanagerinstanz enthalten.

Das bedeutet, dass die Java EE-Umgebung bei einer erforderlichen Transaktionswiederherstellung unabhängig davon, mit welchem Warteschlangenmanager die Verbindung nach einem Ausfall wiederhergestellt wird, auf die Protokolle der Warteschlangenmanager zugreifen kann.

Wenn die transaktionsorientierten MDBs mit eigenständigen Warteschlangenmanagern verwendet werden, darf die CCDT nur einen Eintrag enthalten, um sicherzustellen, dass die Aktivierungsspezifikation die Verbindung nach einem Ausfall immer mit dem gleichen Warteschlangenmanager auf dem gleichen System wiederherstellt.

Stellen Sie für CCDTs, die mit Aktivierungsspezifikationen verwendet werden, sicher, dass die Eigenschaft **AFFINITY** auf den Standardwert *PREFERRED* gesetzt ist, damit die Verbindungen immer mit dem gleichen aktiven Warteschlangenmanager hergestellt werden.

CLIENTRECONNECTOPTIONSEigenschaft

Aktivierungsspezifikationen verfügen über eine eigene Verbindungswiederholungsfunktionalität. Durch diese Funktionalität werden die Spezifikationen bei einem Ausfall des verbundenen Warteschlangenmanagers automatisch wieder mit dem IBM MQ-System verbunden.

Aus diesem Grund wird die von IBM MQ classes for JMS bereitgestellte Funktionalität der automatischen Wiederherstellung der Clientverbindung nicht unterstützt.

Sie müssen die Eigenschaft **CLIENTRECONNECTOPTIONS** für alle Aktivierungsspezifikationen, die in Java EEverwendet werden, auf *DISABLED* setzen.

WebSphere Application Server-Listenerports
Message-driven Bean-Anwendungen (MDBs), die in WebSphere Application Server installiert sind, können zur Verarbeitung der Nachrichten in einem IBM MQ-System auch Listenerports verwenden.

Listenerports erkennen in einem IBM MQ-System ankommende Nachrichten und stellen diese den MDBs zur Verarbeitung zu. In diesem Abschnitt wird erläutert, wie Listenerports das IBM MQ-System überwachen.

MDBs können auch innerhalb ihrer onMessage() -Methode zusätzliche Verbindungen zu IBM MQ -Systemen herstellen.

Informationen zur automatischen Wiederherstellung der Clientverbindung im Kontext mit diesen Verbindungen finden Sie im Abschnitt „Enterprise JavaBeans und webbasierte Anwendungen” auf Seite 311.

Für WebSphere Application Server-Listenerports gilt Folgendes:

- **CONNECTIONNAMELIST** und **CCDTURL** werden mit Einschränkungen unterstützt
- **CLIENTRECONNECTOPTIONS** wird nicht unterstützt

CONNECTIONNAMELISTEigenschaft

Listener-Ports verwenden JMS -Verbindungspools, wenn sie eine Verbindung zu IBM MQherstellen, und unterliegen daher den Auswirkungen der Verwendung von Verbindungspools. Weitere Informationen hierzu finden Sie im Thema „Aktivierungsspezifikationen” auf Seite 307.

Wenn keine Verbindungen mehr frei sind und die maximale Anzahl an Verbindungen für diese Verbindungsfactory noch nicht erreicht ist, wird über die Eigenschaft **CONNECTIONNAMELIST** versucht, eine neue Verbindung zu IBM MQ herzustellen.

Wenn nicht auf alle IBM MQ -Systeme im **CONNECTIONNAMELIST** zugegriffen werden kann, wird der Listener-Port gestoppt.

Danach wartet der Listenerport das durch die angepasste Eigenschaft **RECOVERY.RETRY.INTERVAL** (des Nachrichten-Listener-Service) angegebene Intervall ab, und wiederholt den Verbindungsversuch dann.

Auch bei diesem erneuten Verbindungsversuch wird zunächst überprüft, ob der Verbindungspool mittlerweile freie Verbindungen enthält (für den Fall, dass zwischen den Verbindungsversuchen Verbindungen in den Pool zurückgestellt wurden). Sind wiederum keine Verbindungen verfügbar, verwendet der Listenerport wie zuvor die Liste der Eigenschaft **CONNECTIONNAMELIST**.

Gelingt die Verbindungswiederherstellung des Listenerports mit einem IBM MQ-System, so führt die Java EE-Umgebung die erforderlichen transaktionsorientierten Bereinigungen durch und setzt anschließend die Nachrichtenzustellung an die MDBs zur Weiterverarbeitung fort.

Damit diese transaktionsorientierte Bereinigung ordnungsgemäß durchgeführt werden kann, muss die Java EE-Umgebung Zugriff auf die Protokolle des ausgefallenen Warteschlangenmanagers haben.

Wenn die Listenerports mit transaktionsorientierten MDBs verwendet werden, die an XA-Transaktionen mitwirken, und die Ports Verbindungen zu einem **Multi-Instanz-Warteschlangenmanager** herstellen, muss die **CONNECTIONNAMELIST** einen Eintrag sowohl für die aktive als auch für die Standby-Warteschlangenmanagerinstanz enthalten.

Das bedeutet, dass die Java EE-Umgebung bei einer erforderlichen Transaktionswiederherstellung unabhängig davon, mit welchem Warteschlangenmanager die Verbindung nach einem Ausfall wiederhergestellt wird, auf die Protokolle der Warteschlangenmanager zugreifen kann.

Wenn die transaktionsorientierten MDBs mit eigenständigen Warteschlangenmanagern verwendet werden, muss die Eigenschaft **CONNECTIONNAMELIST** einen einzelnen Eintrag enthalten, um sicherzustellen, dass die Aktivierungsspezifikation nach einem Ausfall immer eine Verbindung zu demselben Warteschlangenmanager herstellt, der auf demselben System ausgeführt wird.

CCDTURLEigenschaft

Beim Start versucht der Listener-Port, unter Verwendung des ersten Eintrags in der CCDT eine Verbindung zu dem in der Eigenschaft **QMANAGER** angegebenen Warteschlangenmanager herzustellen.

Wenn der Listenerport über den ersten Eintrag der Tabelle keine Verbindung mit einem Warteschlangenmanager herstellen kann, versucht er es über den zweiten, dann den dritten Eintrag usw., bis der Verbindungsaufbau gelingt oder das Ende der Tabelle erreicht ist.

Gelingt der Verbindungsaufbau über keinen der Einträge der CCDT, so wird der Listenerport beendet.

Danach wartet der Listenerport das durch die angepasste Eigenschaft **RECOVERY.RETRY.INTERVAL** (des Nachrichten-Listener-Service) angegebene Intervall ab, und wiederholt den Verbindungsversuch dann.

Bei diesem Verbindungswiederholungsversuch werden die Einträge der CCDT wie zuvor der Reihe nach durchgegangen.

Sobald der Listenerport läuft, erhält er Nachrichten vom IBM MQ-System und stellt diese einem MDB zur Verarbeitung zu.

Sollte der Warteschlangenmanager während der Verarbeitung einer Nachricht ausfallen, so wird dieser Fehler von der Java EE-Umgebung erkannt, und es wird versucht, die Verbindung für den Listenerport wiederherzustellen. Bei diesem Wiederherstellungsversuch verwendet der Listenerport wie zuvor die Informationen der CCDT.

Hat der Listenerport alle Einträge der CCDT ausprobiert, ohne dass die Verbindung mit dem Warteschlangenmanager gelingt, startet der Port den nächsten Verbindungsversuch erst wieder nach der in der Eigenschaft **RECOVERY.RETRY.INTERVAL** angegebenen Zeit.

Die Eigenschaft **MAX.RECOVERY.RETRIES** des Nachrichten-Listener-Service definiert die Anzahl der aufeinanderfolgenden Verbindungswiederholungen, bevor ein Listenerport beendet wird und ein manueller Neustart erforderlich ist.

Gelingt die Verbindungswiederherstellung des Listenerports mit einem IBM MQ-System, so führt die Java EE-Umgebung die erforderlichen transaktionsorientierten Bereinigungen durch und setzt anschließend die Nachrichtenzustellung an die MDBs zur Weiterverarbeitung fort.

Damit diese transaktionsorientierte Bereinigung ordnungsgemäß durchgeführt werden kann, muss die Java EE-Umgebung Zugriff auf die Protokolle des ausgefallenen Warteschlangenmanagers haben.

Wenn die Listenerports mit transaktionsorientierten MDBs verwendet werden, die an XA-Transaktionen mitwirken, und die Spezifikationen Verbindungen zu einem Multi-Instanz-Warteschlangenmanager herstellen, muss die CCDT einen Eintrag sowohl für die aktive als auch für die Standby-Warteschlangenmanagerinstanz enthalten.

Das bedeutet, dass die Java EE-Umgebung bei einer erforderlichen Transaktionswiederherstellung unabhängig davon, mit welchem Warteschlangenmanager die Verbindung nach einem Ausfall wiederhergestellt wird, auf die Protokolle der Warteschlangenmanager zugreifen kann.

Wenn die transaktionsorientierten MDBs mit eigenständigen Warteschlangenmanagern verwendet werden, darf die CCDT nur einen Eintrag enthalten, um sicherzustellen, dass der Listenerport die Verbindung nach einem Ausfall immer mit dem gleichen Warteschlangenmanager auf dem gleichen System wiederherstellt.

Stellen Sie für CCDTs, die mit Listenerports verwendet werden, sicher, dass die Eigenschaft **AFFINITY** auf den Standardwert *PREFERRED* gesetzt ist, damit die Verbindungen immer mit dem gleichen aktiven Warteschlangenmanager hergestellt werden.

CLIENTRECONNECTOPTIONSEigenschaft

Listenerports verfügen über eine eigene Verbindungswiederholungsfunktionalität. Durch diese Funktionalität werden die Listenerports bei einem Ausfall des verbundenen Warteschlangenmanagers automatisch wieder mit dem IBM MQ-System verbunden.

Aus diesem Grund wird die von IBM MQ classes for JMS bereitgestellte Funktionalität der automatischen Wiederherstellung der Clientverbindung nicht unterstützt.

Sie müssen die Eigenschaft **CLIENTRECONNECTOPTIONS** für alle Listener-Ports, die in Java EEverwendet werden, auf *DISABLED* setzen.

Enterprise JavaBeans und webbasierte Anwendungen
EJB-Anwendungen (Enterprise JavaBean) und Anwendungen, die im Webcontainer ausgeführt werden, wie z. B. Servlets, verwenden eine JMS -Verbindungsfactory, um eine Verbindung zu einem IBM MQ -Warteschlangenmanager herzustellen.

Für EJBs und webbasierte Anwendungen gelten folgende Einschränkungen:

- **CONNECTIONNAMELIST** und **CCDTURL** werden mit Einschränkungen unterstützt

- **CLIENTRECONNECTOPTIONS** wird nicht unterstützt

CONNECTIONNAMELISTEigenschaft

Wenn die Java EE-Umgebung einen Verbindungspool für JMS-Verbindungen bereitstellt, finden Sie in „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden" auf Seite 313 Informationen darüber, wie dies das Verhalten der Eigenschaft **CONNECTIONNAMELIST** beeinflusst.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, verwendet die Anwendung die Eigenschaft **CONNECTIONNAMELIST** auf die gleiche Weise wie Java SE-Anwendungen.

Wenn die Anwendungen mit transaktionsorientierten MDBs verwendet werden, die an XA-Transaktionen mitwirken, und die Spezifikationen Verbindungen zu einem Multi-Instanz-Warteschlangenmanager herstellen, muss die **CONNECTIONNAMELIST** einen Eintrag sowohl für die aktive als auch für die Standby-Warteschlangenmanagerinstanz enthalten.

Das bedeutet, dass die Java EE-Umgebung bei einer erforderlichen Transaktionswiederherstellung unabhängig davon, mit welchem Warteschlangenmanager die Verbindung nach einem Ausfall wiederhergestellt wird, auf die Protokolle der Warteschlangenmanager zugreifen kann.

Wenn die Anwendungen mit eigenständigen Warteschlangenmanagern verwendet werden, muss die Eigenschaft **CONNECTIONNAMELIST** einen einzigen Eintrag enthalten, um sicherzustellen, dass die Anwendung nach einem Fehler immer eine Verbindung zu demselben Warteschlangenmanager herstellt, der auf demselben System ausgeführt wird.

CCDTURLEigenschaft

Wenn die Java EE-Umgebung einen Verbindungspool für JMS-Verbindungen bereitstellt, finden Sie in „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden" auf Seite 313 Informationen darüber, wie dies das Verhalten der Eigenschaft **CCDTURL** beeinflusst.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, verwendet die Anwendung die Eigenschaft **CCDTURL** auf die gleiche Weise wie Java SE-Anwendungen.

Wenn die Anwendungen mit transaktionsorientierten MDBs verwendet werden, die an XA-Transaktionen mitwirken, und die Anwendungen Verbindungen zu einem Multi-Instanz-Warteschlangenmanager herstellen, muss die CCDT einen Eintrag sowohl für die aktive als auch für die Standby-Warteschlangenmanagerinstanz enthalten.

Das bedeutet, dass die Java EE-Umgebung bei einer erforderlichen Transaktionswiederherstellung unabhängig davon, mit welchem Warteschlangenmanager die Verbindung nach einem Ausfall wiederhergestellt wird, auf die Protokolle der Warteschlangenmanager zugreifen kann.

Wenn die Anwendungen mit eigenständigen Warteschlangenmanagern verwendet werden, darf die CCDT nur einen Eintrag enthalten, um sicherzustellen, dass die Anwendung die Verbindung nach einem Ausfall immer mit dem gleichen Warteschlangenmanager auf dem gleichen System wiederherstellt.

CLIENTRECONNECTOPTIONSEigenschaft

Sie müssen die Eigenschaft **CLIENTRECONNECTOPTIONS** für alle JMS -Verbindungsfactorys, die von EJBs oder Anwendungen verwendet werden, die im Web-Container ausgeführt werden, auf *DISABLED* setzen.

Anwendungen, die beim Ausfall des verwendeten Warteschlangenmanagers eine automatische Herstellung der Verbindung mit einem neuen Warteschlangenmanager erfordern, müssen ihre eigene Verbindungswiederholungslogik implementieren. Weitere Informationen finden Sie unter „Verbindungswiederherstellungslogik in einer Java EE-Anwendung implementieren" auf Seite 314.

Szenarios: WebSphere Application Server mit IBM MQ
Szenarios: WebSphere Application Server Liberty-Profil mit IBM MQ

Innerhalb von Clientcontainern ausgeführte Anwendungen
Einige Java EE-Umgebungen, beispielsweise WebSphere Application Server, stellen einen Clientcontainer bereit, der auch für die Ausführung von Java SE-Anwendungen verwendet werden kann.

Anwendungen, die in diesen Umgebungen ausgeführt werden, verwenden eine JMS -Verbindungsfactory, um eine Verbindung zu einem IBM MQ -Warteschlangenmanager herzustellen.

Für Anwendungen innerhalb von Clientcontainern gilt Folgendes:

- **CONNECTIONNAMELIST** und **CCDTURL** werden vollständig unterstützt
- **CLIENTRECONNECTOPTIONS** wird vollständig unterstützt

CONNECTIONNAMELISTEigenschaft

Wenn die Java EE-Umgebung einen Verbindungspool für JMS-Verbindungen bereitstellt, finden Sie in „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden” auf Seite 313 Informationen darüber, wie dies das Verhalten der Eigenschaft **CONNECTIONNAMELIST** beeinflusst.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, verwendet die Anwendung die Eigenschaft **CONNECTIONNAMELIST** auf die gleiche Weise wie Java SE-Anwendungen.

CCDTURLEigenschaft

Wenn die Java EE-Umgebung einen Verbindungspool für JMS-Verbindungen bereitstellt, finden Sie in „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden” auf Seite 313 Informationen darüber, wie dies das Verhalten der Eigenschaft **CCDTURL** beeinflusst.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, verwendet die Anwendung die Eigenschaft **CCDTURL** auf die gleiche Weise wie Java SE-Anwendungen.

CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden
Einige Java EE-Umgebungen, beispielsweise WebSphere Application Server, stellen einen JMS-Verbindungspoolcontainer bereit, der auch für die Ausführung von Java SE-Anwendungen verwendet werden kann.

Anwendungen, die eine Verbindung über eine in der Java EE-Umgebung definierte Verbindungsfactory erstellen, erhalten entweder eine Verbindung aus dem Verbindungspool dieser Verbindungsfactory oder für sie wird eine neue Verbindung erstellt, wenn im Pool keine geeignete Verbindung verfügbar ist.

Wenn die Verbindungsfactory mit der Eigenschaft **CONNECTIONNAMELIST** oder **CCDTURL** konfiguriert wurde, kann dies Auswirkungen haben.

Wenn die Verbindungsfactory das erste Mal zur Erstellung einer Verbindung verwendet wird, verwendet die Java EE-Umgebung entweder die Eigenschaft **CONNECTIONNAMELIST** oder die Eigenschaft **CCDTURL** zur Erstellung der neuen Verbindung für das IBM MQ-System. Wird diese Verbindung nicht mehr benötigt, so wird Sie an den Verbindungspool zurückgegeben, wo sie zur Wiederverwendung zur Verfügung steht.

Wird daraufhin eine weitere Verbindung aus der Verbindungsfactory angefordert, so gibt die Java EE-Umgebung die Verbindung aus dem Verbindungspool zurück, anstatt sie mit der Eigenschaft **CONNECTIONNAMELIST** oder **CCDTURL** neu zu erstellen.

Wenn eine Warteschlangenmanagerinstanz während der Verwendung einer Verbindung ausfällt, wird die Verbindung verworfen. Der Inhalt des Verbindungspools wird aber vermutlich nicht verworfen. Er kann also noch Verbindungen mit einem Warteschlangenmanager enthalten, der gar nicht mehr aktiv ist.

Wird in einer solchen Situation erneut eine Verbindung aus der Verbindungsfactory angefordert, so kann es sein, dass eine Verbindung für den ausgefallenen Warteschlangenmanager zurückgegeben wird. Jeder Versuch, diese Verbindung zu verwenden, schlägt jedoch fehl, da der Warteschlangenmanager ja nicht mehr aktiv ist, und so wird diese Verbindung schließlich doch verworfen.

Nur wenn der Verbindungspool leer ist, verwendet die Java EE-Umgebung die Eigenschaft **CONNECTIONNAMELIST** oder **CCDTURL**, um eine neue Verbindung mit IBM MQ herzustellen.

Der Verbindungspool kann aufgrund der Art und Weise, wie **CONNECTIONNAMELIST** und die CCDT-Tabelle zur Erstellung von JMS-Verbindungen verwendet werden, auch Verbindungen zu verschiedenen IBM MQ-Systemen enthalten.

Stellen Sie sich zum Beispiel vor, dass die Eigenschaft **CONNECTIONNAMELIST** einer Verbindungsfactory wie folgt eingestellt ist:

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

Nehmen Sie dann an, dass der Warteschlangenmanager auf System `hostname1(port1)` beim ersten Versuch einer Anwendung, eine Verbindung mit einem eigenständigen Warteschlangenmanager aus dieser Verbindungsfactory zu erstellen, nicht zugänglich ist. Die Anwendung erhält daraufhin eine Verbindung mit dem Warteschlangenmanager auf System `hostname2(port2)`.

Nun erstellt eine andere Anwendung über die gleiche Verbindungsfactory eine JMS-Verbindung. Der Warteschlangenmanager auf System `hostname1(port1)` ist nun verfügbar, so dass eine neue JMS-Verbindung mit diesem IBM MQ-System erstellt und an die Anwendung zurückgegeben wird.

Nachdem beide Anwendungen fertig sind, schließen sie ihre JMS-Verbindungen, wodurch diese an den Verbindungspool zurückgegeben werden.

Der Verbindungspool dieser Verbindungsfactory enthält nun also zwei JMS-Verbindungen:

- Die Verbindung mit dem Warteschlangenmanager auf System `hostname1(port1)`
- Die Verbindung mit dem Warteschlangenmanager auf System `hostname2(port2)`

Dies kann hinsichtlich einer eventuell erforderlichen Transaktionswiederherstellung zu Problemen führen. Wenn das Java EE-System eine Transaktion mittels Rollback zurücksetzen muss, muss das System in der Lage sein, eine Verbindung mit einem Warteschlangenmanager herzustellen, der Zugriff auf die Transaktionsprotokolle hat.

Verbindungswiederherstellungslogik in einer Java EE-Anwendung implementieren
Enterprise JavaBeans und webbasierte Anwendungen, deren Verbindung bei Ausfall eines Warteschlangenmanagers automatisch wiederhergestellt werden soll, müssen ihre eigene Verbindungswiederholungslogik implementieren.

Nachfolgend erhalten Sie weitere Informationen, wie dies erreichbar ist.

Anwendungsfehler zulassen

Für diese Vorgehensweise sind keine Anwendungsänderungen erforderlich, jedoch eine administrative Rekonfiguration der Verbindungsfactory-Definition, um dieser die Eigenschaft **CONNECTIONNAMELIST** hinzuzufügen. Das aufrufende Programm muss jedoch in der Lage sein, einen Ausfall entsprechend zu handhaben. Beachten Sie aber, dass dies auch auf Fehler wie MQRC_Q_FULL zutrifft, die nichts mit Verbindungsfehlern zu tun haben.

Beispielcode für diesen Vorgang:

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
         try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
 catch (JMSException je) {
 // process exception
```

```
    }
  }
}
```

Im vorangegangenen Codebeispiel wird davon ausgegangen, dass für die vom Servlet verwendete Verbindungsfactory die Eigenschaft **CONNECTIONNAMELIST** definiert ist.

Wenn das Servlet zum ersten Mal verarbeitet wird, wird mithilfe der **CONNECTIONNAMELIST** -Eigenschaft eine neue Verbindung erstellt, vorausgesetzt, dass keine gepoolten Verbindungen von anderen Anwendungen verfügbar sind, die eine Verbindung zu demselben Warteschlangenmanager herstellen.

Nach der durch einen `close()`-Aufruf initiierten Verbindungsfreigabe wird diese Verbindung wieder in den Pool zurückgestellt und bei der nächsten Ausführung des Servlets - ohne Bezug auf die Eigenschaft **CONNECTIONNAMELIST** - wiederverwendet. Dies gilt bis zu einem Verbindungsfehler, in welchem Fall ein CONNECTION_ERROR_OCCURRED-Ereignis generiert wird. Durch dieses Ereignis wird der Pool aufgefordert, die fehlerhafte Verbindung zu vernichten.

Bei der nächsten Ausführung des Servlets steht nun keine gepoolte Verbindung mehr zur Verfügung. Daher wird wieder auf die Eigenschaft **CONNECTIONNAMELIST** zurückgegriffen, um eine Verbindung mit dem ersten verfügbaren Warteschlangenmanager herzustellen. Bei einem Warteschlangenmanager-Failover (wenn es sich beispielsweise nicht um einen vorübergehenden Netzausfall handelte), stellt das Servlet eine Verbindung mit der Backup-Instanz her, sofern eine solche verfügbar ist.

Sind an der Anwendung auch andere Ressourcen beteiligt, beispielsweise Datenbanken, ist eventuell die Instruktion erforderlich, dass der Anwendungsserver die Transaktion durch ein Rollback zurücksetzen muss.

Verbindungswiederherstellung innerhalb der Anwendung regeln

Wenn das aufrufende Programm einen Fehler des Servlets nicht verarbeiten kann, muss die Verbindungswiederherstellung innerhalb der Anwendung geregelt werden. Wie das folgende Beispiel zeigt, muss eine Anwendung, wenn innerhalb dieser eine Verbindung wiederhergestellt werden soll, eine neue Verbindung anfordern, so dass die aus JNDI abgerufene Verbindungsfactory im Cache zwischengespeichert und `JMSException`-Ausnahmen wie die folgende verarbeitet werden können: JMSCMQ0001: `WebSphere MQ call failed with compcode '2' ('MQCC_FAILED') reason '2009' ('MQRC_CONNECTION_BROKEN')`.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
        // drive reconnection
        setupResources();
    }
  }
```

Im folgenden Beispiel erstellt `setupResources()` die JMS-Objekte und fügt einen Sleep- und Retry-Loop für die Handhabung verzögerter Verbindungswiederherstellungen hinzu. In der Praxis können da-

durch sehr viele Verbindungswiederherstellungsversuche vermieden werden. Beachten Sie, dass in diesem Beispiel Exit-Bedingungen weggelassen wurden, um den Code so einfach wie möglich darzustellen.

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

Wenn die Verbindungswiederherstellung durch die Anwendung geregelt wird, müssen Sie auch darauf achten, dass die Anwendung alle Verbindungen mit anderen Ressourcen freigibt, sobald sie nicht mehr benötigt werden, und zwar unabhängig davon, ob es sich bei diesen Ressourcen um andere IBM MQ-Warteschlangenmanager oder um andere Back-End-Services (wie Datenbanken) handelt. Diese Verbindungen müssen erneut eingerichtet werden, wenn die Verbindungswiederherstellung mit einer neuen IBM MQ-Warteschlangenmanagerinstanz abgeschlossen ist. Andernfalls bleiben Anwendungsserverressourcen für die Dauer des Verbindungswiederherstellungsversuchs unnötig gesperrt und haben zu dem Zeitpunkt, zu dem sie schließlich verwendet werden können, eventuell bereits ihr Zeitlimit erreicht.

WorkManager verwenden

Für Anwendungen, deren Ausführung länger als nur ein paar Zehntelsekunden dauert (z. B. Anwendungen mit Stapelverarbeitung), kann der WebSphere Application Server WorkManager verwendet werden. Hierzu sehen Sie einen Codeausschnitt für WebSphere Application Server:

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
    InitialContext ctx = new InitialContext();
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

Dabei enthält die Datei `web.xml` Folgendes:

```
<resource-ref>
      <description>WorkManager</description>
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
```

```
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

Der Stapel wird nun durch die Arbeitsschnittstelle implementiert:

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
               ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

Wenn die Stapelverarbeitung zum Beispiel aufgrund großer Nachrichten, eines langsamen Netzes oder extensiven Datenbankzugriffs (insbesondere in Verbindung mit langsamem Failover) lange dauert, beginnt der Server mit der Ausgabe von Warnungen zu blockierten Threads wie der Folgenden:

WSVR0605W: Thread "`WorkManager.DefaultWorkManager` : 0" (00000035) war 694061 Millisekunden aktiv und möglicherweise blockiert. There is/are 1 thread(s) in total in the server that may be hung.

Diese Warnungen lassen sich durch Verringerung der Stapelgröße oder durch Erhöhung des Zeitlimits für blockierte Threads reduzieren. Besser wäre aber die Implementierung dieser Verarbeitung in einem Enterprise JavaBean (EJB) (zum Senden von Stapeln) oder in einem Message-driven Bean (MDB) (zum Verarbeiten oder zum Verarbeiten und Antworten).

Beachten Sie, dass eine durch die Anwendung verwaltete Verbindungswiederherstellungslösung keine generelle Lösung für die Verarbeitung von Laufzeitfehlern ist und die Anwendung nach wie vor einen Mechanismus zur Behandlung von Fehlern mit Ursachen jenseits einer funktionierenden Verbindung bereitstellen muss.

Hier seien zum Beispiel der Versuch einer Anwendung genannt, eine Nachricht in eine volle Warteschlange einzureihen (2053 MQRC_Q_FULL), oder der Versuch, eine Verbindung mit einem Warteschlangenmanager mit ungültigen Berechtigungsnachweisen herzustellen (2035 MQRC_NOT_AUTHORIZED).

Außerdem muss die Anwendung 2059 MQRC_Q_MGR_NOT_AVAILABLE-Fehler handhaben können, wenn bei aktiviertem Failover keine Instanzen sofort zur Verfügung stehen. Hierzu sollte die Anwendung die JMS-Ausnahmen sofort melden, statt eigenständig und im Hintergrund zu versuchen, die Verbindung wiederherzustellen.

Objektpooling in IBM MQ classes for JMS
Durch Verwendung einer Form von Verbindungspooling außerhalb von Java EE lässt sich die Gesamtauslastung verringern, die beispielsweise entstehen kann durch eigenständige Anwendungen, die Frameworks nutzen oder in Cloudumgebungen bereitgestellt werden, und auch durch eine größere Anzahl von

Clientverbindungen in QueueManagers, was zu einer Zunahme in der Serverkonsolidierung von Anwendungen und Warteschlangenmanagern führt.

Innerhalb des Java EE-Programmiermodells gibt es einen gut definierten Lebenszyklus der verschiedenen verwendeten Objekte. Message-driven Beans (MDBs) sind am meisten eingeschränkt, während Servlets mehr Freiheit bieten. Deshalb sind die Poolingoptionen, die in den Java EE-Servern verfügbar sind, an die verschiedenen verwendeten Programmiermodelle angepasst.

Mit Java SE (oder mit einem anderen Framework wie etwa Spring) sind die Programmiermodelle extrem flexibel. Aus diesem Grund ist eine einzige Poolingstrategie nicht für alle geeignet. Sie sollten überlegen, ob Sie zu einem Framework wechseln sollten, das für jede Form des Poolings geeignet ist, z. B. Spring.

Die zu verwendende Poolingstrategie hängt von der Umgebung ab, in der Ihre Anwendung ausgeführt wird.

Objektpooling in einer Java EE-Umgebung
Java EE -Anwendungsserver bieten Verbindungspooling-Funktionalität, die von MDB-Anwendungen, Enterprise Java Beans und Servlets verwendet werden können.

Zur Verbesserung der Leistung stellt WebSphere Application Server einen Pool für die Verbindungen zum JMS-Provider bereit. Wenn eine Anwendung eine JMS-Verbindung anfordert, ermittelt der Anwendungsserver zunächst, ob der freie Verbindungspool bereits eine Verbindung enthält. Falls ja, wird diese Verbindung an die Anwendung zurückgeben. Andernfalls wird eine neue Verbindung erstellt.

Abbildung 41 auf Seite 318 zeigt, wie Aktivierungsspezifikationen und Listenerports eine JMS-Verbindung einrichten und diese Verbindung zur Überwachung eines Ziels auf Nachrichten im Normalmodus verwenden.

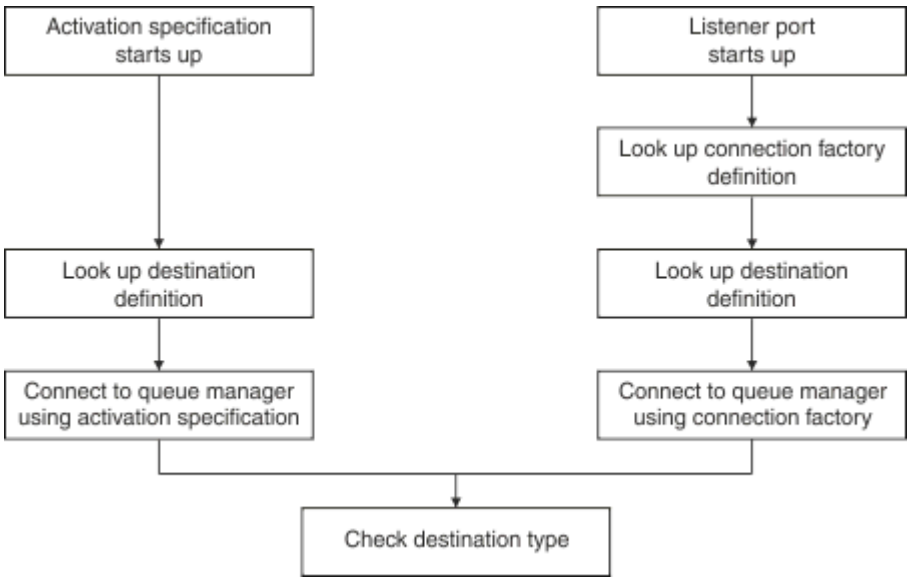


Abbildung 41. Normalmodus

Wenn Sie den IBM MQ -Messaging-Provider verwenden, können Anwendungen, die abgehendes Messaging durchführen (z. B. Enterprise Java Beans und Servlets), und die Listener-Portkomponente für Message-driven Beans diese Verbindungspools verwenden.

Die Aktivierungsspezifikationen des IBM MQ-Messaging-Providers verwenden die vom IBM MQ-Ressourcenadapter bereitgestellte Verbindungspooling-Funktionalität. Nähere Informationen hierzu finden Sie im Abschnitt Eigenschaften für den WebSphere MQ-Ressourcenadapter konfigurieren.

Im Abschnitt „Beispiele für die Verwendung des Verbindungspools” auf Seite 322 wird erläutert, wie Anwendungen, aus denen Nachrichten versendet werden, sowie Listenerports bei der Erstellung von JMS-Verbindungen den freien Pool nutzen.

Im Abschnitt „Poolwartungsthreads für Verbindungspools” auf Seite 325 wird erläutert, was mit diesen Verbindungen passiert, wenn die Anwendung bzw. der Listenerport sie nicht mehr benötigt.

Im Abschnitt „Beispiele für Poolwartungsthreads” auf Seite 327 wird erläutert, wie der freie Verbindungspool bereinigt wird, um zu verhindern, dass die JMS-Verbindungen veralten.

WebSphere Application Server begrenzt die Anzahl der Verbindungen, die aus einer Verbindungsfactory erstellt werden können, mit der Eigenschaft *maximum connections* (Maximale Anzahl an Verbindungen) der Factory. Der Standardwert dieser Eigenschaft ist 10, d. h., es können zu jeder Zeit maximal nur 10 Verbindungen bestehen, die aus der gleichen Factory erstellt wurden.

Jeder Verbindungsfactory ist ein freier Verbindungspool zugeordnet. Beim Starten des Anwendungsservers ist dieser Pool noch leer. Die maximale Anzahl an Verbindungen, die in diesem Pool vorliegen können, wird ebenfalls durch die Eigenschaft 'maximum connections' bestimmt.

Tipp: Mit JMS 2.0 können mit einer Verbindungsfactory sowohl Verbindungen also auch Kontexte erstellt werden. Daher kann ein Verbindungspool einer Verbindungsfactory zugeordnet sein, die sowohl Verbindungen als auch Kontexte enthält. Allerdings wird empfohlen, eine Verbindungsfactory nur zur Erstellung von Verbindungen oder von Kontexten zu verwenden. Dadurch wird sichergestellt, dass der Verbindungspool für diese Verbindungsfactory nur Objekte eines Typs enthält, wodurch der Pool effizienter wird.

Informationen zur Funktionsweise des Verbindungspoolings in WebSphere Application Server finden Sie unter Verbindungspooling für JMS-Verbindungen konfigurieren. Informationen zu anderen Anwendungsservern finden Sie in der entsprechenden Anwendungsserverdokumentation.

Verwendung des Verbindungspools

Jeder JMS-Verbindungsfactory ist ein Verbindungspool zugeordnet, der null oder mehr JMS-Verbindungen enthält. Jeder JMS-Verbindung ist ein JMS-Sitzungspool zugeordnet; jeder JMS-Sitzungspool enthält null oder mehr JMS-Sitzungen.

Abbildung 42 auf Seite 319 zeigt die Beziehung zwischen diesen Objekten.

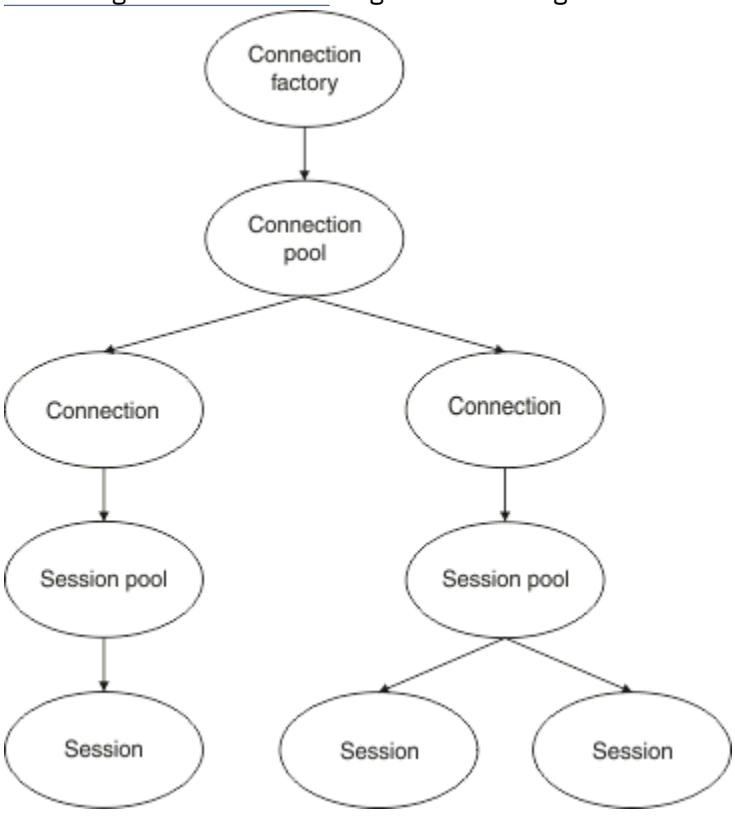


Abbildung 42. Verbindungs- und Sitzungspools

Ein Listenerport, der gestartet wird, oder eine Anwendung, die eine Nachricht versenden will und zu diesem Zweck eine Verbindung aus der Factory anfordern muss, rufen eine der folgenden Methoden auf:

- **`connectionFactory.createConnection()`**

- **ConnectionFactory.createConnection(String, String)**
- **QueueConnectionFactory.createQueueConnection()**
- **QueueConnectionFactory.createQueueConnection(String, String)**
- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

Der Verbindungsmanager von WebSphere Application Server versucht daraufhin eine Verbindung aus dem freien Pool dieser Factory abzurufen, und gibt diese, sofern eine vorhanden ist, an die Anwendung zurück.

Wenn keine Verbindungen mehr frei sind und die durch die Eigenschaft *maximum connections* festgelegte, maximal erlaubte Anzahl an Verbindungen für diese Verbindungsfactory noch nicht erreicht ist, erstellt der Verbindungsmanager eine neue Verbindung für die Anwendung bzw. den Listener.

Fordert eine Anwendung hingegen eine Verbindung an, wenn die durch die Eigenschaft *maximum connections* festgelegte, maximal erlaubte Anzahl an Verbindungen für die Factory bereits erreicht ist, so muss die Anwendung so lange warten, bis eine belegte Verbindung frei und zurück in den freien Pool gestellt wird.

Wie lange die Anwendung in diesem Fall wartet, wird durch die Eigenschaft *connection timeout* (Verbindungszeitlimit) des Verbindungspools festgelegt, deren Standardwert 180 Sekunden ist. Wird eine Verbindung innerhalb dieses Zeitlimits zurück in den freien Pool gestellt, so nimmt der Verbindungsmanager diese sofort wieder aus dem Pool und übergibt sie der Anwendung. Vergeht das Zeitlimit hingegen, ohne dass eine Verbindung frei wird, so erhält die Anwendung die Ausnahme *ConnectionWaitTimeoutException*.

Wenn eine Anwendung eine Verbindung nicht mehr benötigt und diese durch einen der folgenden Aufrufe schließt:

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

bleibt die Verbindung tatsächlich jedoch offen und wird lediglich in den freien Pool zurückgestellt, so dass sie von einer anderen Anwendung wiederverwendet werden kann. Es kann daher durchaus auch der Fall sein, dass Verbindungen zwischen WebSphere Application Server und dem JMS-Provider geöffnet sind, obwohl auf dem Anwendungsserver gar keine JMS-Anwendungen ausgeführt werden.

Erweiterte Eigenschaften für Verbindungspools
Das Verhalten der JMS-Verbindungspools kann durch eine Reihe erweiterter Eigenschaften gesteuert werden.

Überlastschutz

Eine Beschreibung der Methode `sendMessage()`, die `connectionFactory.createConnection()` beinhaltet, finden Sie im Abschnitt „Verwendung des Verbindungspools durch Anwendungen, aus denen Nachrichten versendet werden" auf Seite 324.

Stellen Sie sich ein Szenario vor, in dem 50 EJBs im Zuge ihrer `ejbCreate()`-Methode alle aus der gleichen Verbindungsfactory JMS-Verbindungen erstellen.

Wenn diese Beans alle zur selben Zeit erstellt werden und der freie Verbindungspool der Factory keine Verbindungen mehr enthält, versucht der Anwendungsserver auf einen Schlag 50 JMS-Verbindungen für den selben JMS-Provider zu erstellen. Dies bedeutet sowohl für WebSphere Application Server als auch für den JMS-Provider eine erhebliche Last.

Eine solche Situation lässt sich durch die Eigenschaften des Überlastschutzes entschärfen, indem die Anzahl der JMS-Verbindungen, die gleichzeitig aus einer Verbindungsfactory erstellt werden können, eingeschränkt wird und die Erstellung aller weiteren Verbindungen zeitlich versetzt wird.

Die Anzahl der gleichzeitig erstellten JMS-Verbindungen kann mit den folgenden beiden Eigenschaften eingegrenzt werden:

- Surge threshold (Schwellenwert für Spitzenauslastung)

- Surge creation interval (Erstellungsintervall für Spitzenauslastung)

Wenn EJB-Anwendungen versuchen, aus einer Verbindungsfactory JMS-Verbindungen zu erstellen, überprüft der Verbindungsmanager die Anzahl der erstellten Verbindungen. Wenn diese Anzahl kleiner-gleich dem Wert der Eigenschaft `surge threshold` ist, setzt der Verbindungsmanager das Öffnen neuer Verbindungen fort.

Wenn jedoch die Anzahl der erstellten Verbindungen die Eigenschaft `surge threshold` überschreitet, wartet der Verbindungsmanager die mit der Eigenschaft `surge creation interval` angegebene Zeit, bevor er eine neue Verbindung erstellt und öffnet.

Blockierte Verbindungen

Eine JMS Verbindung wird als `stuck`betrachtet, wenn eine JMS -Anwendung diese Verbindung verwendet, um eine Anforderung an den JMS -Provider zu senden, und der Provider nicht innerhalb eines bestimmten Zeitraums antwortet.

WebSphere Application Server bietet eine Möglichkeit, `stuck` JMS -Verbindungen zu erkennen. Um diese Funktion verwenden zu können, müssen Sie drei Eigenschaften festlegen:

- Stuck Time Timer (Timer für Blockierungszeit)
- Stuck Time (Blockierungszeit)
- Stuck Threshold (Schwellenwert für Blockierungen)

Die regelmäßige Ausführung des Poolwartungsthreads wird im Abschnitt „Beispiele für Poolwartungsthreads” auf Seite 327 beschrieben. Sie erfahren dort, wie der Thread den Inhalt des freien Pools einer Verbindungsfactory überprüft und nach Verbindungen sucht, die schon seit längerem nicht mehr benutzt wurden oder schon zu lange existieren.

Zur Feststellung "blockierter" Verbindungen verfügt der Anwendungsserver auch über einen Thread für blockierte Verbindungen, der den Status aller von einer Verbindungsfactory erstellten, aktiven Verbindungen überprüft, um festzustellen, ob diese noch auf eine Antwort vom JMS-Provider warten.

Der Ausführungszeitpunkt dieses Threads wird durch die Eigenschaft `Stuck time timer` (Timer für Blockierungszeit) bestimmt. Der Standardwert dieser Eigenschaft ist 0 (null), d. h., der Thread ist inaktiv und die Erkennung blockierter Verbindungen wird nicht ausgeführt.

Wenn der Thread eine Verbindung findet, die auf eine Antwort wartet, ermittelt er, wie lange die Verbindung schon wartet, und vergleicht diese Zeit mit dem Wert der Eigenschaft `Stuck time` (Blockierungszeit).

Falls die Antwortzeit des JMS-Providers die durch `Stuck time` (Blockierungszeit) festgelegte Zeit überschreitet, markiert der Anwendungsserver die JMS-Verbindung als blockiert.

Angenommen, für die Verbindungsfactory `jms/CF1` ist die Eigenschaft `Stuck time timer` auf 10 und die Eigenschaft `Stuck time` auf 15 gesetzt.

Der Thread für blockierte Verbindungen wird also alle 10 Sekunden ausgeführt und überprüft dabei, ob aus `jms/CF1` erstellte Verbindungen schon länger als 15 Sekunden auf eine Antwort von IBM MQ warten.

Angenommen, eine EJB erstellt eine JMS -Verbindung zu IBM MQ mit `jms/CF1`und versucht dann, eine JMS -Sitzung über diese Verbindung zu erstellen, indem sie `Connection.createSession()`aufruft.

Der JMS-Provider wird jedoch durch irgendeinen Umstand davon abgehalten, sofort zu antworten (vielleicht ist das System abgestürzt oder ein Prozess auf dem JMS-Provider ist hängen geblieben und verhindert so die Ausführung neuer Prozesse):

Zehn Sekunden nach dem Aufruf von `Connection.createSession()` wird der Timer für Blockierungszeit aktiv und untersucht die aktiven Verbindungen aus `jms/CF1`.

Gehen wir nun davon aus, dass es nur eine aktive Verbindung, `c1` gibt. Ein erstes EJB wartet bereits 10 Sekunden auf eine Antwort auf seine Anforderung, die es über `c1` an den Provider gesendet hat. Dies liegt jedoch noch innerhalb der Grenzen von `Stuck time` (Blockierungszeit), weshalb der Timer für Blockierungszeit diese Verbindung ignoriert und wieder inaktiv wird.

10 Sekunden später wird der Timer für Blockierungszeit wieder aktiv und untersucht die aktiven Verbindungen für `jms/CF1` erneut. Wie zuvor gehen wir davon aus, dass es nur die Verbindung `c1` gibt.

Seit dem Aufruf von `createSession()` durch das erste EJB sind nun 20 Sekunden vergangen und das EJB wartet immer noch auf Antwort. 20 Sekunden sind länger als die in der Eigenschaft `Stuck time` angegebene Zeit, sodass der Thread für blockierte Verbindungen `c1` als blockiert markiert.

Wenn IBM MQ nun 5 Sekunden später endlich antwortet und dem ersten EJB die Erstellung einer JMS-Sitzung erlaubt, wird die Markierung wieder aufgehoben und die Verbindung ist wieder verfügbar.

Der Anwendungsserver zählt die Anzahl der blockierten JMS-Verbindungen einer Verbindungsfactory. Verwendet eine Anwendung nun diese Verbindungsfactory, um eine neue JMS-Verbindung zu erstellen, und der freie Verbindungspool dieser Factory enthält keine Verbindungen mehr, so vergleicht der Verbindungsmanager die Anzahl der blockierten Verbindungen mit dem Wert der Eigenschaft `Stuck threshold` (Schwellenwert für Blockierungen).

Wenn die Anzahl blockierter Verbindungen kleiner als der für die Eigenschaft `Stuck threshold`festgelegte Wert ist, erstellt der Verbindungsmanager eine neue Verbindung und übergibt sie an die Anwendung.

Wenn die Anzahl der blockierten Verbindungen jedoch dem Wert der Eigenschaft `Stuck threshold` entspricht, erhält die Anwendung eine Ressourcenausnahme.

Poolpartitionen

WebSphere Application Server stellt zwei Eigenschaften bereit, mit denen der freie Verbindungspool einer Verbindungsfactory partitioniert werden kann:

- `Number of free pool partitions` (Anzahl der Partitionen des freien Pools) weist den Anwendungsserver an, in wie viele Partitionen der freie Verbindungspool aufgeteilt werden soll.
- `Free pool distribution table size` (Größe der Zuordnungstabelle des freien Pools) bestimmt die Indizierung der Partitionen.

Übernehmen Sie für diese Eigenschaften die Standardwerte 0 (null), es sei denn, Sie werden vom IBM Support Center angeleitet, diese Werte zu ändern.

WebSphere Application Server stellt darüber hinaus noch die erweiterte Eigenschaft `Number of shared partitions` (Anzahl gemeinsam genutzter Partitionen) bereit. Diese Eigenschaft gibt die Anzahl der Partitionen für gemeinsam genutzte Verbindungen an. Da JMS-Verbindungen nicht freigegeben werden können, trifft diese Eigenschaft jedoch nicht zu.

Beispiele für die Verwendung des Verbindungspools
Die Listenerport-Komponente für Message-driven Beans (MDBs) und Anwendungen, aus denen Nachrichten versendet werden, verwenden einen JMS-Verbindungspool.

Abbildung 43 auf Seite 323 zeigt, wie der Verbindungspool in WebSphere Application Server V7.5 und V8.0 funktioniert.

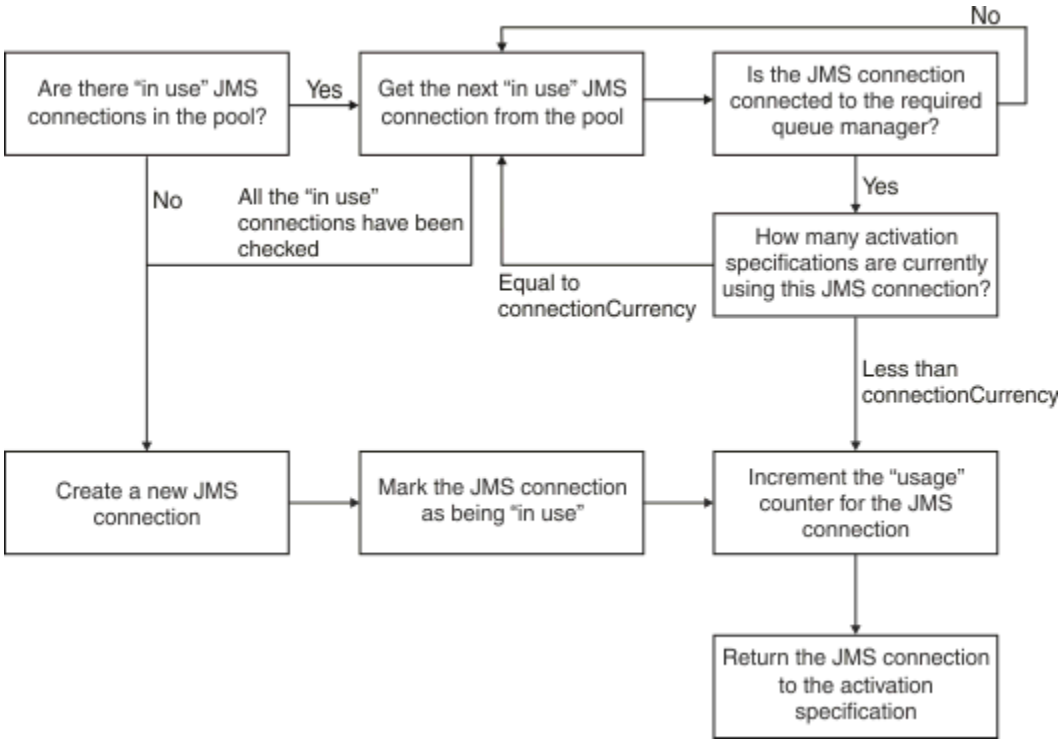


Abbildung 43. WebSphere Application Server V7.5 und V8.0 - Funktionsweise des Verbindungspools

Abbildung 44 auf Seite 323 zeigt, wie der Verbindungspool in WebSphere Application Server V8.5 funktioniert.

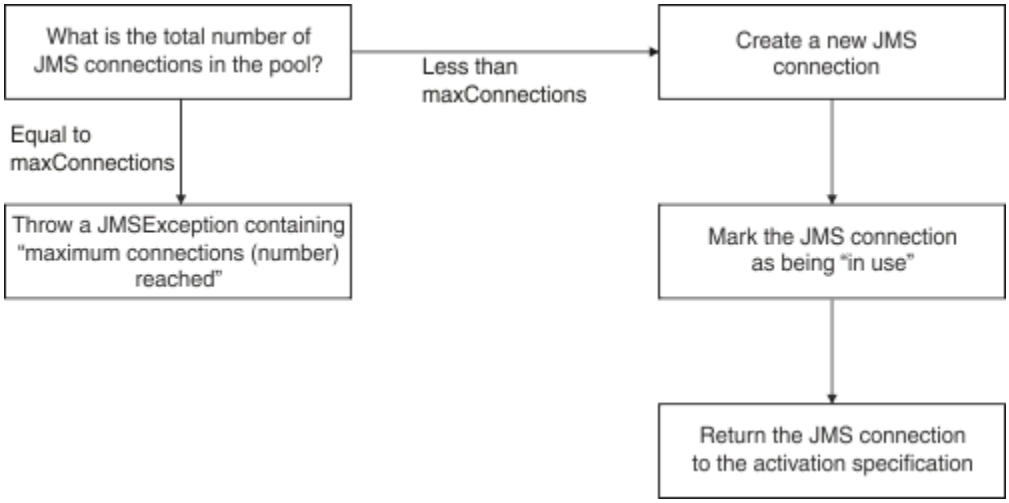


Abbildung 44. WebSphere Application Server V8.5 - Funktionsweise des Verbindungspools

Verwendung des Verbindungspools durch MDB-Listenerports

Nehmen wir an, Sie haben ein MDB in einem WebSphere Application Server Network Deployment-System implementiert, das IBM MQ als JMS-Provider verwendet. Das MDB ist an einem Listenerport implementiert, der eine Verbindungsfactory mit dem Namen `jms/CF1` verwendet. Deren Eigenschaft *maximum connections* (Maximale Anzahl an Verbindungen) ist auf 2 gesetzt, d. h., diese Factory stellt zu jedem Zeitpunkt maximal nur zwei Verbindungen bereit.

Wenn der Listener-Port gestartet wird, versucht der Port, eine Verbindung zu IBM MQüber die`jms/CF1` -Verbindungsfactory herzustellen.

Dazu fordert der Port eine Verbindung vom Verbindungsmanager an. Da die Verbindungsfactory `jms/CF1` damit zum ersten Mal verwendet wird, enthält der freie Pool von `jms/CF1` noch keine Verbindungen. Der

Verbindungsmanager erstellt also eine neue Verbindung, zum Beispiel eine mit dem Namen c1. Beachten Sie, dass diese Verbindung nun für den gesamten Lebenszyklus des Listenerports besteht.

Stellen Sie sich nun vor, dass Sie den Listenerport über die Administrationskonsole von WebSphere Application Server beenden. In diesem Fall stellt der Verbindungsmanager die Verbindung wieder zurück in den freien Pool. Die Verbindung zu IBM MQ bleibt jedoch offen.

Wird der Listenerport nun neu gestartet, fordert er vom Verbindungsmanager erneut eine Verbindung zum Warteschlangenmanager an. Da der freie Pool nun eine Verbindung (c1) enthält, nimmt der Verbindungsmanager diese Verbindung aus dem Pool und stellt sie dem Listenerport zur Verfügung.

Gehen Sie nun davon aus, dass auf dem Anwendungsserver ein zweites MDB implementiert ist, das einen anderen Listenerport verwendet.

Und nun versuchen Sie einen dritten Listenerport zu starten, der ebenfalls die Verbindungsfactory jms/CF1 verwendet. Dieser dritte Listenerport fordert eine Verbindung vom Verbindungsmanager an, der den freien Pool der Verbindungsfactory jms/CF1 nun jedoch leer vorfindet. Er überprüft daraufhin, wie viele Verbindungen bereits über die Verbindungsfactory jms/CF1 erstellt wurden.

Da die maximale Anzahl an Verbindungen für jms/CF1 auf 2 gesetzt ist und Sie bereits zwei Verbindungen aus dieser Factory erstellt haben, wartet der Verbindungsmanager nun 180 Sekunden (Standardwert für das Verbindungszeitlimit) auf eine frei werdende Verbindung.

Wird der erste Listenerport nun erneut gestoppt, so wird seine Verbindung c1 zurück in den freien Pool von jms/CF1 gestellt. Der Verbindungsmanager ruft diese Verbindung ab und übergibt sie dem dritten Listener.

Versuchen Sie nun, den ersten Listener wieder zu starten, so muss dieser warten, bis einer der beiden anderen Listenerports beendet wird. Wird keiner der anderen beiden Listenerports innerhalb von 180 Sekunden gestoppt, erhält der erste Listener einen ConnectionWaitTimeoutException-Fehler und wird beendet.

Verwendung des Verbindungspools durch Anwendungen, aus denen Nachrichten versendet werden

Zur Veranschaulichung dieser Option bitten wir Sie zunächst, sich vorzustellen, dass auf dem Anwendungsserver ein einziges EJB mit dem Namen EJB1 installiert ist. Das Bean implementiert eine Methode mit dem Namen sendMessage() und führt dazu die folgenden Schritte durch:

- Erstellen einer JMS Verbindung zu IBM MQ über eine Factory jms/CF1mit connectionFactory.createConnection().
- Es erstellt aus dieser Verbindung eine JMS-Sitzung.
- Es erstellt aus dieser Sitzung einen Nachrichtenproduzenten.
- Es sendet eine Nachricht.
- Es schließt den Produzenten.
- Es schließt die Sitzung.
- Es schließt die Verbindung durch Aufruf von connection.close().

Gehen Sie nun davon aus, dass der freie Pool der Verbindungsfactory jms/CF1 leer ist. Wenn die EJB zum ersten Mal aufgerufen wird, versucht die Bean, eine Verbindung zu IBM MQ über die Factory jms/CF1herzustellen. Da der freie Pool der Factory noch leer ist, erstellt der Verbindungsmanager eine neue Verbindung und übergibt sie dem Bean EJB1.

Vor Beendigung der Methode ruft die Methode connection.close() auf. Anstatt die Verbindung c1 zu schließen, stellt der Verbindungsmanager die Verbindung nun in den freien Pool der Factory jms/CF1.

Beim nächsten Aufruf von sendMessage() gibt die Methode connectionFactory.createConnection() c1 an die Anwendung zurück.

Stellen Sie sich nun vor, dass zur selben Zeit mit der ersten Instanz eine weitere Instanz dieses EJB ausgeführt wird. Wenn beide Instanzen `sendMessage()` aufrufen, werden aus der Verbindungsfactory `jms/CF1` zwei Verbindungen erstellt.

Stellen Sie sich nun vor, dass eine dritte Instanz des Beans erstellt wird. Auch dieses Bean startet `sendMessage()`, worauf die Methode `connectionFactory.createConnection()` aufruft, um eine Verbindung aus `jms/CF1` zu erstellen.

Es bestehen aber bereits zwei Verbindungen aus `jms/CF1`, so dass die maximale Anzahl an Verbindungen für diese Factory erreicht ist. Die Methode `createConnection()` wartet daher 180 Sekunden (Standardwert für das Verbindungszeitlimit) auf eine frei werdende Verbindung.

Ruft nun die Methode `sendMessage()` des ersten EJB die Methode `connection.close()` auf, um sich zu beenden, wird die vormals von ihm belegte Verbindung `c1` in den freien Pool zurückgestellt. Der Verbindungsmanager nimmt die Verbindung daraufhin aus dem Pool und gibt sie dem dritten EJB. Der Aufruf `connectionFactory.createConnection()` dieses Beans wird daraufhin erfolgreich beschieden und die Methode `sendMessage()` kann ausgeführt werden.

Wenn MDB-Listenerports und EJBs den gleichen Verbindungspool verwenden

In den vorangegangenen Beispielen wurde gezeigt, wie Listenerports und EJBs den Verbindungspool isoliert voneinander verwenden. Selbstverständlich ist es jedoch auch möglich, Listenerports und EJBs parallel auf dem gleichen Anwendungsserver auszuführen und für beide über die gleiche Verbindungsfactory JMS-Verbindungen zu erstellen.

Allerdings müssen Sie in diesem Fall auch die Auswirkungen dieser Situation berücksichtigen.

Denken Sie vor allem daran, dass die Verbindungsfactory zwischen Listenerport und EJB aufgeteilt wird.

Stellen Sie sich zum Beispiel vor, dass zur selben Zeit ein Listener und ein EJB ausgeführt werden. Beide verwenden die Verbindungsfactory `jms/CF1`, was bei den obigen Beispieleinstellungen bereits bedeutet, dass die maximale Anzahl an Verbindungen für diese Factory erreicht ist.

Wenn Sie in dieser Situation versuchen, einen weiteren Listenerport oder eine weitere Instanz des EJB zu starten, müssen beide warten, bis dem freien Pool der Verbindungsfactory `jms/CF1` eine Verbindung zurückgegeben wird.

Poolwartungsthreads für Verbindungspools
Jedem freien Verbindungspool ist ein Poolwartungsthread zugeordnet, der überwacht, dass die Verbindungen in diesem Pool noch gültig sind.

Wenn der Poolwartungsthread erkennt, dass eine Verbindung im freien Pool ungültig ist und daher verworfen werden muss, schließt er die JMS-Verbindung mit dem IBM MQ physisch.

Funktionsweise des Poolverwaltungsthreads

Das Verhalten des Poolverwaltungsthreads bestimmt sich aus vier Eigenschaften des Verbindungspools:

Aged timeout (Zeitlimit für veraltete Verbindungen)
Die Dauer, die eine Verbindung offen bleiben kann.

Minimum connections (Mindestanzahl an Verbindungen)
Die Mindestanzahl an Verbindungen, die der Verbindungsmanager im freien Pool einer Verbindungsfactory aufbewahrt.

Reap time (Bereinigungsintervall)
Das Ausführungsintervall des Poolverwaltungsthreads.

Unused timeout (Zeitlimit für nicht verwendete Verbindungen)
Die Dauer, die eine Verbindung im freien Pool aufbewahrt wird, bevor sie geschlossen wird.

Der Poolverwaltungsthread wird standardmäßig alle 180 Sekunden ausgeführt. Dieser Wert kann durch die Eigenschaft **`Reap time`** (Bereinigungsintervall) des Verbindungspools jedoch geändert werden.

Der Verwaltungsthread sieht sich jede Verbindung des Pools an und überprüft, wie lange sie sich schon im Pool befindet und wie viel Zeit seit ihrer Erstellung und ihrer letzten Nutzung vergangen ist.

Wenn die Verbindung länger als der Wert der Eigenschaft **Unused timeout** für den Verbindungspool nicht verwendet wurde, überprüft der Verwaltungsthread die Anzahl der Verbindungen, die sich momentan im freien Pool befinden. Wenn diese Anzahl:

- größer als der Wert von **Minimum connections** (Mindestanzahl an Verbindungen) ist, schließt der Verbindungsmanager die Verbindung.
- kleiner oder gleich dem Wert von **Minimum connections** (Mindestanzahl an Verbindungen) ist, verbleibt die Verbindung im Pool.

Der Standardwert von **Minimum connections** (Mindestanzahl an Verbindungen) ist *1*, d. h., der Verbindungsmanager versucht aus Leistungsgründen immer mindestens eine Verbindung im freien Pool zu halten.

Der Standardwert von **Unused timeout** (Zeitlimit für nicht verwendete Verbindungen) ist 1800 Sekunden. Standardmäßig wird eine Verbindung also nach 1800 Sekunden der Inaktivität im freien Pool geschlossen, sofern dadurch die Mindestanzahl der Verbindungen im Pool noch gewahrt bleibt.

Dadurch wird verhindert, dass nicht verwendete Verbindungen veraltern. Wenn Sie diese Funktion inaktivieren möchten, setzen Sie die Eigenschaft **Unused timeout** auf 0 (null).

Eine Verbindung im freien Pool, deren Erstellung länger zurück liegt als der Wert der Eigenschaft **Aged timeout** (Zeitlimit für veraltete Verbindungen) des Verbindungspools, wird unabhängig vom Zeitpunkt ihrer letzten Verwendung geschlossen.

Die Eigenschaft **Aged timeout** (Zeitlimit für veraltete Verbindungen) ist standardmäßig aber auf 0 (null) gesetzt, d. h., diese Prüfung wird vom Verwaltungsthread in der Regel nicht ausgeführt. Andernfalls werden Verbindungen, die sich bereits länger als **Aged timeout** im freien Pool befinden, unabhängig von der Anzahl der im Pool verbleibenden Verbindungen geschlossen und entfernt. Die Eigenschaft **Minimum connections** (Mindestanzahl an Verbindungen) bleibt in diesem Fall unberücksichtigt.

Poolverwaltungsthread inaktivieren

Wie Sie den vorangegangenen Erläuterungen entnehmen konnten, hat der Poolverwaltungsthread, sofern er aktiv ist, recht viel zu tun, besonders, wenn der freie Pool der Verbindungsfactory sehr viele Verbindungen enthält.

Um dies zu illustrieren, nehmen wir an, dass ein System über drei JMS-Verbindungsfactorys verfügt und die Eigenschaft **Maximum connections** (Maximale Anzahl an Verbindungen) für jede Factory auf 10 gesetzt ist. Alle 180 Sekunden werden nun drei Poolverwaltungsthreads aktiv, um die Verbindungspools dieser Verbindungsfactorys zu überprüfen. Enthalten die Pools sehr viele Verbindungen, so haben diese Verwaltungsthreads sehr viel Arbeit, was sich auch auf die Leistung auswirken kann.

Sie können den Poolverwaltungsthread eines jeden freien Verbindungspools inaktivieren, indem Sie die Eigenschaft **Reap time** (Bereinigungsintervall) des Pools auf 0 (null) setzen.

In diesem Fall werden die Verbindungen aber niemals geschlossen, selbst dann nicht, wenn das **Unused timeout** (Zeitlimit für nicht verwendete Verbindungen) verstrichen ist. Nur wenn das **Aged timeout** (Zeitlimit für veraltete Verbindungen) vergangen ist, können die Verbindungen noch geschlossen werden.

Wenn eine Anwendung eine Verbindung beendet hat, prüft der Verbindungsmanager, wie lange die Verbindung vorhanden ist. Wenn dieser Zeitraum länger ist als der Wert der Eigenschaft **Aged timeout** , schließt der Verbindungsmanager die Verbindung, anstatt sie an den freien Pool zurückzugeben.

Auswirkungen des Zeitlimits für nicht verwendete Verbindungen auf Transaktionen

Wie im vorangegangenen Abschnitt erläutert, gibt die Eigenschaft **Aged timeout** (Zeitlimit für veraltete Verbindungen) an, wie lange eine Verbindung mit dem JMS-Provider offen bleibt, bevor sie durch den Verbindungsmanager geschlossen wird.

Der Standardwert der Eigenschaft **Aged timeout** (Zeitlimit für veraltete Verbindungen) ist 0 (null), in welchem Fall die Verbindung nie aufgrund ihres Alters geschlossen wird. Sie sollten diesen Wert für die Eigenschaft **Aged timeout** beibehalten, da sich die Aktivierung von **Aged timeout** auf Transaktionen auswirken kann, wenn JMS in EJBs verwendet wird.

In JMS ist die Einheit einer Transaktion eine JMS-*Sitzung*, die aus einer JMS-*Verbindung* erstellt wird. Dabei ist es die JMS-*Sitzung* und nicht die JMS-*Verbindung*, die in die Transaktion eingetragen wird.

Aufgrund des Designs des Anwendungsservers können JMS-Verbindungen aufgrund einer Überschreitung des **Aged timeout**-Zeitlimits geschlossen werden, selbst wenn aus diesen Verbindungen erstellte JMS-Sitzungen noch an einer Transaktion beteiligt sind.

Durch das Schließen einer JMS-Verbindung werden in diesem Fall alle noch ausstehenden transaktionsorientierten Arbeiten innerhalb der zugehörigen JMS-Sitzungen zurückgesetzt, wie in der JMS-Spezifikation beschrieben. Allerdings ist sich der Anwendungsserver in diesem Fall nicht bewusst, dass die aus der Verbindung erstellten JMS-Sitzungen nicht mehr gültig sind. Versucht der Server nun, eine Transaktion über eine solche Sitzung festzuschreiben oder zurückzusetzen, wird eine `IllegalStateException`-Ausnahme zurückgegeben.

Wichtig: Möchten Sie **Aged timeout** mit JMS-Verbindungen aus EJBs verwenden, so müssen Sie daher sicherstellen, dass alle JMS-Vorgänge explizit in der JMS-Sitzung festgeschrieben werden, bevor die EJB-Methode, die die JMS-Vorgänge ausführt, beendet wird.

Beispiele für Poolwartungsthreads
Verwenden Sie das EJB-Beispiel (Enterprise JavaBean), um zu verstehen, wie der Poolverwaltungsthread funktioniert. Ebenso können Sie aber auch Message-driven Beans (MDBs) und Listenerports verwenden, da Sie ja lediglich eine Methode brauchen, um freie Verbindungen aus dem Pool abzurufen.

Weitere Informationen zur Methode `sendMessage()` finden Sie im Abschnitt „Verwendung des Verbindungspools durch Anwendungen, aus denen Nachrichten versendet werden" auf Seite 324.

Sie haben die Verbindungsfactory mit folgenden Werten konfiguriert:

- **Reap time** (Bereinigungsintervall) ist 180 Sekunden (Standardwert).
- **Aged timeout** (Zeitlimit für veraltete Verbindungen) ist 0 Sekunden (Standardwert).
- **Unused timeout** auf 300 Sekunden gesetzt

Nach dem Starten des Anwendungsservers wird die Methode`sendMessage()` aufgerufen.

Die Methode erstellt aus der Factory `jms/CF1` eine Verbindung, zum Beispiel mit dem Namen `c1`, die sie zum Senden einer Nachricht verwendet. Danach ruft die Methode `connection.close()` auf, wodurch `c1` in den freien Pool gestellt wird.

Nach 180 Sekunden startet der Poolverwaltungsthread, um den freien Verbindungspool `jms/CF1` zu überprüfen. Der Pool enthält die freie Verbindung `c1`, weshalb der Thread überprüft, wann die Verbindung zurück in den Pool gestellt wurde, und diesen Zeitpunkt mit der aktuellen Uhrzeit vergleicht.

Seit die Verbindung in den Pool zurückgestellt wurde, sind 180 Sekunden vergangen. Dies ist weniger als der Wert von **Unused timeout** der Factory `jms/CF1`. Der Verwaltungsthread lässt die Verbindung daher unberührt.

180 Sekunden später wird der Poolverwaltungsthread erneut ausgeführt. Wieder erkennt der Thread die Verbindung `c1` und stellt dieses Mal fest, dass sie sich bereits seit 360 Sekunden im Pool befindet. Dies ist mehr als der Wert von **Unused timeout**, weshalb der Verbindungsmanager die Verbindung schließt.

Wenn Sie nun die Methode `sendMessage()` erneut ausführen und die Anwendung `connectionFactory.createConnection()` aufruft, erstellt der Verbindungsmanager eine neue Verbindung mit IBM MQ, da der freie Verbindungspool der Verbindungsfactory nun ja leer ist.

Das vorherige Beispiel zeigt, wie der Verwaltungsthread die Eigenschaften **Reap time** und **Unused timeout** verwendet, um veraltete Verbindungen zu vermeiden, wenn die Eigenschaft **Aged timeout** auf null gesetzt ist.

Funktionsweise der Eigenschaft **Aged timeout**

Stellen Sie sich für das nächste Beispiel folgende Werte vor:

- **Aged timeout** (Zeitlimit für veraltete Verbindungen) ist 300 Sekunden.
- **Unused timeout** (Zeitlimit für nicht verwendete Verbindungen) ist 0 (null).

Sie rufen die Methode `sendMessage()` auf, und diese Methode versucht über die Verbindungsfactory `jms/CF1` eine Verbindung zu erstellen.

Da der freie Pool dieser Factory noch leer ist, erstellt der Verbindungsmanager die neue Verbindung `c1` und gibt sie an die Anwendung zurück. `sendMessage()` ruft nun `connection.close()` auf, und `c1` wird in den freien Verbindungspool zurückgestellt.

180 Sekunden später wird der Poolverwaltungsthread ausgeführt. Der Thread findet im freien Verbindungspool `c1` vor und überprüft, wie viel Zeit seit der Erstellung dieser Verbindung vergangen ist. Die Verbindung besteht seit 180 Sekunden, also noch nicht so lange wie das **Aged timeout** (Zeitlimit für veraltete Verbindungen). Also lässt der Poolverwaltungsthread die Verbindung in Ruhe und wird wieder inaktiv.

60 Sekunden später wird `sendMessage()` erneut aufgerufen. Beim Aufruf von `connectionFactory.createConnection()` stellt der Verbindungsmanager dieses Mal fest, dass der freie Pool der Verbindungsfactory `jms/CF1` die Verbindung `c1` enthält. Der Verbindungsmanager nimmt die Verbindung `c1` daraufhin aus dem Pool und gibt sie der Anwendung.

Beim Beenden von `sendMessage()` wird die Verbindung wieder in den freien Pool zurückgestellt. 120 Sekunden später wird der Poolverwaltungsthreads erneut aktiviert. Wieder findet er bei der Untersuchung des Verbindungspools der Factory `jms/CF1` die Verbindung `c1`.

Obwohl die Verbindung erst vor 120 Sekunden verwendet wurde, schließt der Thread die Verbindung, da sie nun bereits seit 360 Sekunden besteht und dies länger als die 300 Sekunden ist, die die Eigenschaft **Aged timeout** (Zeitlimit für veraltete Verbindungen) zulässt.

Beeinflussung des Poolverwaltungsthreads durch die Eigenschaft "Minimum connections"

Wir wollen nun erneut das Beispiel für „Verwendung des Verbindungspools durch MDB-Listenerports” auf Seite 323 verwenden. Dieses Mal aber sind auf dem Anwendungsserver zwei MDBs implementiert, jedes mit einem anderen Listenerport.

Jeder Listenerport ist mit der Verbindungsfactory `jms/CF1` konfiguriert, deren Eigenschaften folgende Werte haben:

- **Unused timeout** (Zeitlimit für nicht verwendete Verbindungen) ist 120 Sekunden.
- **Reap time** (Bereinigungsintervall) ist 180 Sekunden.
- **Minimum connections** (Mindestanzahl an Verbindungen) ist 1.

Gehen wir davon aus, dass der erste Listener gestoppt wird und seine Verbindung `c1` in den freien Pool zurückgestellt wird. 180 Sekunden später wird der Poolverwaltungsthreads aktiviert und stellt bei der Untersuchung des Verbindungspools der Verbindungsfactory `jms/CF1` fest, dass sich die Verbindung `c1` bereits länger im Pool der Factory befindet als der Wert der Eigenschaft **Unused timeout** (Zeitlimit für nicht verwendete Verbindungen) der Factory zulässt.

Vor dem Schließen von `c1` sieht der Poolverwaltungsthread jedoch nach, wie viele Verbindungen im Pool verbleiben, wenn diese Verbindung geschlossen wird. Da `c1` zur Zeit die einzige Verbindung im freien Verbindungspool ist, schließt der Verbindungsmanager sie nicht, da andernfalls weniger Verbindungen im Pool verblieben, als von **Minimum connections** (Mindestanzahl an Verbindungen) gefordert.

Nehmen Sie nun an, dass auch der zweite Listener gestoppt wird. Der freie Verbindungspool enthält nun zwei Verbindungen, `c1` und `c2`.

180 Sekunden später wird der Poolverwaltungsthread erneut ausgeführt. Dieses Mal befindet sich `c1` seit 360 Sekunden im Pool und `c2` seit 180 Sekunden.

Der Poolverwaltungsthread überprüft c1 und stellt fest, dass sich diese Verbindung weit länger im Pool befindet, als die Eigenschaft **Unused timeout** (Zeitlimit für nicht verwendete Verbindungen) zulässt.

Danach untersucht der Thread die Anzahl der Verbindungen im Pool und vergleicht diese mit dem Wert der Eigenschaft **Minimum connections** (Mindestanzahl an Verbindungen). Da der Pool zwei Verbindungen enthält und **Minimum connections** auf 1 gesetzt ist, schließt der Verbindungsmanager c1.

Nun untersucht der Poolverwaltungsthread c2. Dies war auch länger im freien Verbindungspool als der Wert der Eigenschaft **Unused timeout** . Da das Schließen von c2 jedoch dazu führen würde, dass der freie Verbindungspool weniger als die festgelegte Anzahl an minimalen Verbindungen enthält, lässt der Verbindungsmanager c2 allein.

JMS-Verbindungen und IBM MQ
Informationen zur Verwendung von IBM MQ als JMS-Provider.

BINDINGS-Transport verwenden

Wenn eine Verbindungsfactory mit BINDINGS-Transport konfiguriert wurde, richtet jede JMS-Verbindung einen Dialog (auch als **hconn** bezeichnet) mit IBM MQ ein. Dieser Dialog verwendet für die Kommunikation mit dem Warteschlangenmanager die Interprozesskommunikation (bzw. gemeinsam genutzten Speicher).

CLIENT-Transport verwenden

Wenn die Verbindungsfactory eines IBM MQ-Messaging-Providers mit CLIENT-Transport konfiguriert wurde, richtet jede aus der Factory erstellte Verbindung einen neuen Dialog (auch als **hconn** bezeichnet) mit IBM MQ ein.

Stellt eine Verbindungsfactory hingegen eine Verbindung zum Warteschlangenmanager im Normalmodus des IBM MQ-Messaging-Providers her, kann es passieren, dass sich mehrere aus der Verbindungsfactory erstellte JMS-Verbindungen eine TCP/IP-Verbindung mit IBM MQ teilen. Weitere Informationen finden Sie unter „TCP/IP-Verbindung in IBM MQ classes for JMS gemeinsam nutzen" auf Seite 332.

Zur Bestimmung der maximalen Anzahl an Clientkanälen, die zum selben Zeitpunkt von JMS-Verbindungen genutzt werden können, addieren Sie die Werte der Eigenschaft *Maximum connections* (Maximale Anzahl an Verbindungen) aller Verbindungsfactorys, die auf den gleichen Warteschlangenmanager verweisen.

Beispiel: Sie haben die beiden Verbindungsfactorys jms/CF1 und jms/CF2, in deren Konfigurationen festgelegt ist, dass sie über den gleichen IBM MQ-Kanal Verbindungen zum gleichen IBM MQ-Warteschlangenmanager herstellen.

Diese Factorys verwenden die Standard-Verbindungspooleigenschaften, d. h. *Maximum connections* (Maximale Anzahl an Verbindungen) ist auf 10 gesetzt. Wenn sowohl von jms/CF1 als auch von jms/CF2 zur selben Zeit alle erlaubten Verbindungen verwendet werden, sind zwischen dem Anwendungsserver und IBM MQ 20 Dialoge eingerichtet.

Wenn die Verbindungsfactory Verbindungen zum Warteschlangenmanager im Normalmodus des IBM MQ-Messaging-Providers herstellt, berechnet sich die maximal erlaubte Anzahl an TCP/IP-Verbindungen zwischen dem Anwendungsserver und dem Warteschlangenmanager für diese Verbindungsfactorys wie folgt:

```
20/the value of SHARECNV for the IBM MQ channel
```

Wenn die Verbindungsfactory Verbindungen im Migrationsmodus des IBM MQ-Messaging-Providers herstellt, sind für diese Verbindungsfactorys zwischen dem Anwendungsserver und IBM MQ maximal 20 TCP/IP-Verbindungen erlaubt (eine für jede JMS-Verbindung in den Verbindungspools der beiden Factorys).

Zugehörige Konzepte
„IBM MQ classes for JMS/Jakarta Messaging verwenden" auf Seite 85

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Java -Messaging-Provider, die mit IBM MQbereitgestellt werden. Neben der Implementierung der in den Spezifikationen JMS und Jakarta Messaging definierten Schnittstellen fügen diese Messaging-Provider zwei Gruppen von Erweiterungen zur Java -Messaging-API hinzu.

Objektpooling in einer Java SE-Umgebung
Mit Java SE (oder mit einem anderen Framework wie etwa Spring) sind die Programmiermodelle extrem flexibel. Aus diesem Grund ist eine einzige Poolingstrategie nicht für alle geeignet. Sie sollten überlegen, ob ein Framework vorhanden ist, das für jede Form des Poolings geeignet ist, z. B. Spring.

Andernfalls könnte die Anwendungslogik diese Aufgabe übernehmen. Stellen Sie sich die Frage, wie komplex die Anwendung selbst ist? Dies ist die beste Methode, um die Anwendung und die Anforderungen, die sie an die Verbindung mit dem Messaging-System stellt, zu verstehen. Anwendungen werden oft auch innerhalb ihres eigenen Wrapper-Codes um die Basis-JMS-API geschrieben.

Dies kann zwar ein sehr vernünftiger Ansatz sein und die Komplexität verbergen, aber es sollte auch bedacht werden, dass es Probleme mit sich bringen kann. Beispielsweise sollte eine generische `getMessage()`-Methode, die häufig aufgerufen wird, Konsumenten nicht nur öffnen und schließen.

Punkte, die Sie beachten sollten:

- Wie lange wird die Anwendung Zugriff auf IBM MQ benötigen? Die ganze Zeit über oder nur gelegentlich.
- Wie oft werden Nachrichten gesendet? Bei wenigen Nachrichten genügt vielleicht eine einzige Verbindung zu IBM MQ, die gemeinsam genutzt wird.
- Eine Ausnahmebedingung wegen Verbindungsunterbrechung bedeutet in der Regel, dass eine gepoolte Verbindung neu erstellt werden muss. Was ist mit:
  - Sicherheitsausnahmebedingungen oder Host nicht verfügbar
  - Ausnahmebedingungen wegen voller Warteschlange
- Was sollte bei einer Ausnahmebedingung wegen Verbindungsunterbrechung mit den anderen freien Verbindungen im Pool passieren? Sollen sie geschlossen und neu erstellt werden?
- Wenn beispielsweise TLS verwendet wird: Wie lange sollte eine einzelne Verbindung offen bleiben?
- Wie wird sich eine Verbindung im Pool selbst identifizieren, sodass der Administrator eines Warteschlangenmanagers die Verbindung erkennen und zurückverfolgen kann.

Sie sollten alle JMS-Objekte für das Pooling in Betracht ziehen und die betreffenden Objekte, wann immer möglich, in Pools zusammenfassen. Zu den Objekten gehören:

- JMS-Verbindungen
- Sitzung
- Kontexte
- Produzenten und Konsumenten aller möglichen Typen

Bei Verwendung des Clienttransports nutzen JMS-Verbindungen, Sitzungen und Kontexte Sockets, wenn Sie mit dem IBM MQ-Warteschlangenmanager kommunizieren. Ein Pooling dieser Objekte führt dazu, dass die Anzahl eingehender IBM MQ-Verbindungen (`hConns`) beim Warteschlangenmanager abnimmt und die Anzahl der Kanalinstanzen sinkt.

Bei Verwendung des Bindungstransports zum Warteschlangenmanager entfällt die Netzbetriebsschicht völlig. Viele Anwendungen verwenden jedoch den Clienttransport, um eine Konfiguration mit höherer Verfügbarkeit und gleichmäßiger Auslastung bereitzustellen.

JMS-Produzenten und -Konsumenten öffnen Ziele auf dem Warteschlangenmanager. Wenn weniger Warteschlangen oder Themen geöffnet werden und mehrere Teile der Anwendung diese Objekte verwenden, kann ein Pooling dieser Objekte nützlich sein.

Aus IBM MQ-Sicht werden durch diesen Prozess zahlreiche MQOPEN- und MQCLOSE-Operation eingespart.

Verbindungen, Sitzungen und Kontexte

Alle diese Objekte beinhalten IBM MQ-Verbindungskennungen zum Warteschlangenmanager und werden aus einer `ConnectionFactory` generiert. Sie können Logik zu einer Anwendung hinzufügen, um die Anzahl der Verbindungen zu begrenzen und andere Objekte, die aus einer einzelnen Verbindungsfactory erstellt werden, auf eine bestimmte Anzahl zu beschränken.

Die Verbindungen, die erstellt werden, können in einer einfachen Datenstruktur in der Anwendung enthalten sein. Der Anwendungscode, der eine dieser Datenstrukturen verwenden muss, kann ein Objekt zur Verwendung *auschecken*.

Beachten Sie folgende Faktoren:

- Wann sollten Verbindungen aus dem Pool entfernt werden? Erstellen Sie generell einen Listener für Ausnahmebedingungen für die Verbindung. Wenn der Listener aufgerufen wird, um eine Ausnahmebedingung zu verarbeiten, sollten Sie die Verbindung neu erstellen ebenso wie alle Sitzungen, die aus dieser Verbindung entstanden sind.
- Wenn eine CCDT für den Lastausgleich verwendet wird, können die Verbindungen zu verschiedenen Warteschlangenmanagern führen. Dies kann für die Poolinganforderungen wichtig sein.

Denken Sie daran, dass es gemäß der JMS-Spezifikation ein Programmierfehler ist, wenn mehrere Threads gleichzeitig auf eine Sitzung oder einen Kontext zugreifen. Der IBM MQ JMS-Code geht bei der Handhabung von Threads nicht unbedingt rigoros vor. Sie sollten jedoch Logik zur Anwendung hinzufügen, um sicherzustellen, dass ein Sitzungs- oder Kontextobjekt immer nur von einem Thread verwendet wird.

Produzenten und Konsumenten

Jeder Produzent oder Konsument, der erstellt wird, öffnet ein Ziel auf dem Warteschlangenmanager. Wenn dasselbe Ziel für eine Vielzahl von Tasks verwendet werden soll, ist es sinnvoll, die Konsumenten- oder Produzentenobjekte offen zu halten. Schließen Sie das Objekt nur, wenn die gesamte Arbeit abgeschlossen ist.

Auch wenn das Öffnen und Schließen eines Ziels kurze Operationen sind, können sie, wenn sie häufig ausgeführt werden, auf Dauer doch eine erhebliche Zeit in Anspruch nehmen.

Der Geltungsbereich dieser Objekte befindet sich in der Sitzung oder dem Kontext, aus dem sie erstellt werden. Deshalb müssen sie in ihrem jeweiligen Bereich gehalten werden. Im Allgemeinen werden Anwendungen so geschrieben, dass dies relativ einfach durchzuführen ist.

Überwachung

Wie werden die Anwendungen ihre Objektpools überwachen? Die Antwort darauf wird im Wesentlichen durch die Komplexität der implementierten Poolinglösung bestimmt.

Wenn Sie eine Implementierung des Java EE-Poolings in Betracht ziehen, gibt es eine Vielzahl von Optionen, einschließlich:

- Aktuelle Größe der Pools
- Zeit, die Objekte darin verbracht haben
- Bereinigung der Pools
- Aktualisierung der Verbindungen

Sie sollten auch bedenken, wie eine einzelne, wiederverwendete Sitzung auf dem Warteschlangenmanager in Erscheinung tritt. Es gibt Verbindungsfactory-Eigenschaften zur Identifizierung der Anwendung (z. B. `appName`), die nützlich sein können.

„IBM MQ classes for JMS/Jakarta Messaging verwenden” auf Seite 85
IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Java -Messaging-Provider, die mit IBM MQbereitgestellt werden. Neben der Implementierung der in den Spezifikationen JMS und Jakarta Messaging definierten Schnittstellen fügen diese Messaging-Provider zwei Gruppen von Erweiterungen zur Java -Messaging-API hinzu.

TCP/IP-Verbindung in IBM MQ classes for JMS gemeinsam nutzen
Mehrere Instanzen eines MQI-Kanals können so festgelegt werden, dass sie eine einzelne TCP/IP-Verbindung gemeinsam nutzen.

Anwendungen, die in derselben Java -Laufzeitumgebung ausgeführt werden und den IBM MQ classes for JMS -oder IBM MQ -Ressourcenadapter verwenden, um über den CLIENT-Transport eine Verbindung zu einem Warteschlangenmanager herzustellen, können für die gemeinsame Nutzung einer Kanalinstanz eingerichtet werden.

Wenn ein Kanal definiert ist und der Parameter **SHARECNV** auf einen Wert größer als 1gesetzt ist, kann diese Anzahl von Dialogen eine Kanalinstanz gemeinsam nutzen. Damit eine Verbindungsfactory oder Aktivierungsspezifikation diese Funktion verwenden kann, setzen Sie die Eigenschaft **SHARECONVALLOWED** auf JA.

Jede von einer JMS-Anwendung erstellte JMS-Verbindung und JMS-Sitzung erstellt einen eigenen Dialog mit dem Warteschlangenmanager.

Wenn eine Aktivierungsspezifikation gestartet wird, startet der IBM MQ-Ressourcenadapter für die zu verwendende Aktivierungsspezifikation einen Dialog mit dem Warteschlangenmanager. Jede Serversitzung in dem Serversitzungspool, die der Aktivierungsspezifikation zugeordnet ist, startet einen Dialog mit dem Warteschlangenmanager.

Das Attribut **SHARECNV** ist eine Best-Effort-Methode für die gemeinsame Nutzung von Verbindungen. Wenn daher ein **SHARECNV** -Wert größer als 0 mit dem IBM MQ classes for JMSverwendet wird, ist nicht garantiert, dass eine neue Verbindungsanforderung immer eine bereits eingerichtete Verbindung gemeinsam nutzt.

Gemeinsame Nutzung der TCP/IP-Verbindungen

Es gibt zwei Strategien für die gemeinsame Nutzung von TCP/IP-Verbindungen:

Die GLOBAL-Strategie
Diese Strategie ist die Standardstrategie für die gemeinsame Nutzung von TCP/IP-Verbindungen. Jede JMS -Verbindung oder -Sitzung kann einen Dialog über jede geeignete TCP/IP-Verbindung verwenden. Die Eignung wird durch Faktoren wie Hostadresse, Portnummer, Benutzer-ID und Kennwort sowie TLS/SSL-Parameter bestimmt.

Dieser Ansatz für die gemeinsame Nutzung von TCP/IP-Verbindungen minimiert die Anzahl der Kanalinstanzen, die im Gebrauch sind, jedoch auf Kosten der Konkurrenzsituation für den Zugriff auf einen globalen Pool von TCP/IP-Verbindungen.

Die CONNECTION-Strategie
Bei dieser Strategie werden die Kanalinstanzen nur von zugehörigen JMS -Objekten gemeinsam genutzt. Wenn eine JMS -Verbindung erstellt wird, wird eine Kanalinstanz für sie erstellt und zusätzliche Dialoge in dieser Kanalinstanz sind nur für JMS -Sitzungen verfügbar, die von dieser JMS -Verbindung erstellt wurden.

Wenn mehr Dialoge erstellt werden als das Attribut SHARECNV angibt, wird eine neue Kanalinstanz erstellt, die nur von JMS -Sitzungen verwendet werden kann, die von der ursprünglichen JMS -Verbindung erstellt wurden.

Dieser Ansatz für die gemeinsame Nutzung von Kanalinstanzen verringert die Anzahl der Konkurrenzsituationen für Dialoge auf Kosten von potenziell deutlich mehr Kanalinstanzen.

Strategie für gemeinsame Nutzung einer Kanalinstanz explizit angeben

V 9.4.0

Standardmäßig wird die Strategie GLOBAL verwendet, wenn keine Verbindung zu Anwendungen wiederhergestellt werden kann. Wiederverbindbare Anwendungen verwenden immer die Strategie CONNECTION.

Bei Anwendungen, die IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingverwenden, kann die CONNECTION-Strategie für nicht wiederverbindbare Anwendungen auf anwendungsweiter Basis

aktiviert werden. Sie können die CONNECTION-Strategie aktivieren, indem Sie die Systemeigenschaft `com.ibm.mq.jms.channel.sharing` auf den Wert `CONNECTION`setzen. Bei diesem Wert muss die Groß-/Kleinschreibung nicht beachtet werden und jeder andere Wert als `CONNECTION` wird ignoriert.

Sie können die Systemeigenschaft `com.ibm.mq.jms.channel.sharing` auf eine der folgenden Arten festlegen:

- Legen Sie die Eigenschaft als Teil der JVM-Initialisierung fest, indem Sie die Befehlszeilenoption "-D" verwenden:

  ```
  -Dcom.ibm.mq.jms.channel.sharing=CONNECTION
  ```

- Legen Sie die Eigenschaft fest, bevor Sie IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging mit `System.setProperty()` verwenden.

Anzahl der Kanalinstanzen für die GLOBAL-Strategie für gemeinsame Nutzung berechnen

Verwenden Sie die folgenden Formeln, um die maximale Anzahl der Kanalinstanzen zu bestimmen, die von einer Anwendung erstellt werden:

Aktivierungsspezifikationen

Anzahl der Kanalinstanzen = (*maxPoolDepth_value* + 1) / *SHARECNV_value*

Dabei ist *maxPoolDepth_wert* der Wert der Eigenschaft **maxPoolDepth** und *SHARECNV_wert* der Wert der Eigenschaft **SHARECNV** in dem Kanal, der von der Aktivierungsspezifikation verwendet wird.

Sonstige JMS-Anwendungen

Anzahl der Kanalinstanzen = (*jms_connections* + *jms_sessions*) / *SHARECNV_value*

Dabei ist *jms_connections* die Anzahl der Verbindungen, die von der Anwendung erstellt werden, *jms_sessions* ist die Anzahl der JMS -Sitzungen, die von der Anwendung erstellt werden, und *SHARECNV_value* ist der Wert der Eigenschaft **SHARECNV** in dem Kanal, der von der Aktivierungsspezifikation verwendet wird.

Anzahl der Kanalinstanzen für die CONNECTION-Strategie zur gemeinsamen Nutzung berechnen

Die Anzahl der Kanalinstanzen hängt von der Verteilung der JMS -Sitzungen auf die JMS -Verbindungen in der Anwendung ab.

Lassen Sie einen Dialog für die JMS -Verbindung und einen Dialog für jede JMS -Sitzung unter dieser JMS -Verbindung zu. Dividieren Sie dann durch den Wert **SHARECNV** und runden Sie den Wert ab. Diese Berechnung gibt die Kanalinstanzen an, die für diese JMS -Verbindung erforderlich sind.

Dasselbe Prinzip kann auf Aktivierungsspezifikationen angewendet werden. Betrachten Sie die Aktivierungsspezifikation als JMS -Verbindung und die Eigenschaft **maxPoolDepth** als Anzahl der JMS -Sitzungen.

Beispiele

Die folgenden Beispiele zeigen, wie Sie mit den Formeln die Anzahl der Kanalinstanzen berechnen können, die auf einem Warteschlangenmanager von Anwendungen entweder mithilfe der IBM MQ classes for JMS oder mit dem IBM MQ-Ressourcenadapter erstellt werden.

Beispiel für eine JMS-Anwendung

Eine JMS-Anwendungsverbindung verbindet sich mithilfe des CLIENT-Transports mit einem Warteschlangenmanager und erstellt eine JMS-Verbindung sowie drei JMS-Sitzungen. Für den Kanal, über den die Anwendung eine Verbindung zum Warteschlangenmanager herstellt, ist die Eigenschaft **SHARECNV** auf den Wert 10 gesetzt. Bei Ausführung der Anwendung bestehen vier Dialoge zwischen der

Anwendung und dem Warteschlangenmanager und eine Kanalinstanz. Die vier Dialoge nutzen alle die Kanalinstanz gemeinsam.

Beispiel für eine Aktivierungsspezifikation

Eine Aktivierungsspezifikation verbindet sich mithilfe des CLIENT-Transports mit einem Warteschlangenmanager. In der Konfiguration der Aktivierungsspezifikation ist die Eigenschaft **maxPoolDepth** auf 10 gesetzt. Für den Kanal, für dessen Verwendung die Aktivierungsspezifikation konfiguriert ist, ist die Eigenschaft **SHARECNV** auf 10 gesetzt. Wenn die Aktivierungsspezifikation ausgeführt wird und 10 Nachrichten gleichzeitig verarbeitet, entspricht die Anzahl der Dialoge zwischen der Aktivierungsspezifikation und dem Warteschlangenmanager 11 (zehn Dialoge für die Serversitzungen und ein Dialog für die Aktivierungsspezifikation). Die Anzahl der Kanalinstanzen, die von der Aktivierungsspezifikation verwendet wird, entspricht 2.

Beispiel für eine Aktivierungsspezifikation

Eine Aktivierungsspezifikation verbindet sich mithilfe des CLIENT-Transports mit einem Warteschlangenmanager. In der Konfiguration der Aktivierungsspezifikation ist die Eigenschaft **maxPoolDepth** auf 5 gesetzt. Für den Kanal, für dessen Verwendung die Aktivierungsspezifikation konfiguriert ist, ist die Eigenschaft **SHARECNV** auf 0 gesetzt. Wenn die Aktivierungsspezifikation ausgeführt wird und 5 Nachrichten gleichzeitig verarbeitet, entspricht die Anzahl der Dialoge zwischen der Aktivierungsspezifikation und dem Warteschlangenmanager 6 (fünf Dialoge für die Serversitzungen und ein Dialog für die Aktivierungsspezifikation). Die Anzahl der Kanalinstanzen, die von der Aktivierungsspezifikation verwendet wird, ist 6, da die Eigenschaft **SHARECNV** im Kanal auf 0 gesetzt ist. Jeder Dialog verwendet eine eigene Kanalinstanz.

Zugehörige Tasks

„Anzahl der zwischen WebSphere Application Server und IBM MQ erstellten TCP/IP-Verbindungen bestimmen" auf Seite 529

Dank der Funktion für gemeinsame Dialognutzung können mehrere Dialoge die gleiche MQI-Kanalinstanz (auch als TCP/IP-Verbindung bezeichnet) gemeinsam nutzen.

Portbereich für Clientverbindungen in IBM MQ classes for JMS angeben

Mit der Eigenschaft LOCALADDRESS können Sie einen Bereich der Ports angeben, an die sich Ihre Anwendung binden kann.

Wenn eine Anwendung der IBM MQ classes for JMS versucht, im Clientmodus eine Verbindung zu einem IBM MQ-Warteschlangenmanager herzustellen, erlaubt die Firewall möglicherweise nur Verbindungen, die aus angegebenen Ports oder Portbereichen stammen. In dieser Situation können Sie mit der Eigenschaft LOCALADDRESS eines ConnectionFactory-, QueueConnectionFactory- oder TopicConnectionFactory-Objekts einen Port oder auch einen Bereich mehrerer Ports angeben, an die die Anwendung gebunden werden kann.

Sie können die Eigenschaft 'LOCALADDRESS' mit dem IBM MQ-JMS-Verwaltungstool oder durch Aufruf der Methode 'setLocalAddress()' in einer JMS-Anwendung festlegen. Es folgt ein Beispiel für die Festlegung der Eigenschaft innerhalb einer Anwendung:

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

Wenn sich die Anwendung später mit einem Warteschlangenmanager verbindet, bindet sich die Anwendung an eine lokale IP-Adresse und Portnummer im Bereich von 192.0.2.0(2000) bis 192.0.2.0(3000).

In einem System mit mehr als einer Netzschnittstelle können Sie mit der Eigenschaft LOCALADDRESS auch angeben, welche Netzschnittstelle für eine Verbindung verwendet werden muss.

Für eine Echtzeitverbindung mit einem Broker ist die Eigenschaft LOCALADDRESS nur relevant, wenn Multicasting verwendet wird. In diesem Fall können Sie mit der Eigenschaft angeben, welche lokale Netzschnittstelle für eine Verbindung verwendet werden muss. Der Wert der Eigenschaft darf jedoch keine Portnummer oder einen Bereich von Portnummern enthalten.

Wenn Sie den Bereich der Ports beschränken, können Verbindungsfehler auftreten. Falls ein Fehler auftritt, wird eine JMS-Ausnahmebedingung (JMSException) mit einer eingebetteten MQ-Ausnahmebedingung (MQException) ausgelöst, die den IBM MQ-Ursachencode MQRC_Q_MGR_NOT_AVAILABLE und folgende Nachricht enthält:

```
Socketverbindungsversuch aufgrund von LOCAL_ADDRESS_PROPERTY-Einschränkungen abgelehnt
```

Ein Fehler kann auftreten, wenn alle Ports im angegebenen Bereich belegt sind oder wenn die Angaben der IP-Adresse, des Hostnamens oder der Portnummer nicht gültig sind (da beispielsweise eine negative Portnummer angegeben wurde).

Da IBM MQ classes for JMS möglicherweise andere Verbindungen erstellen als diejenigen, die von einer Anwendung benötigt werden, sollten Sie immer einen Portbereich angeben. Im Allgemeinen benötigt jede Sitzung, die von einer Anwendung erstellt wird, einen Port und die IBM MQ classes for JMS benötigen möglicherweise drei oder vier zusätzliche Ports. Wenn ein Verbindungsfehler auftritt, erhöhen Sie den Portbereich.

Das in IBM MQ classes for JMS standardmäßig verwendete Verbindungspooling kann sich auf die Geschwindigkeit auswirken, in der Ports wiederverwendet werden können. Dadurch kann ein Verbindungsfehler auftreten, während Ports freigegeben werden.

Kanalkomprimierung in IBM MQ classes for JMS
Eine Anwendung der IBM MQ classes for JMS kann IBM MQ-Funktionen verwenden, um einen Nachrichtenheader oder Daten zu komprimieren.

Durch die Komprimierung der durch einen IBM MQ-Kanal fließenden Daten können Sie die Leistung des Kanals verbessern und den Netzverkehr verringern. Mithilfe der in IBM MQ bereitgestellten Funktion können Sie die Daten komprimieren, die durch Nachrichtenkanäle und MQI-Kanäle fließen. Bei beiden Kanaltypen können Sie Headerdaten und Nachrichtendaten unabhängig voneinander komprimieren. Standardmäßig werden keine Daten in einem Kanal komprimiert.

Eine Anwendung der IBM MQ classes for JMS gibt durch die Erstellung eines java.util.Collection-Objekts die Techniken an, die für die Komprimierung von Header- oder Nachrichtendaten in einer Verbindung verwendet werden können. Jede Komprimierungstechnik ist ein Ganzzahlobjekt in der Objektgruppe. Die Reihenfolge, in der die Anwendung die Komprimierungstechniken zur Objektgruppe hinzufügt, bestimmt die Reihenfolge, in der die Komprimierungstechniken vereinbart werden, wenn die Anwendung die Verbindung erstellt. Die Anwendung kann dann die Objektgruppe an ein ConnectionFactory-Objekt übergeben, indem sie für Headerdaten die Methode setHdrCompList() oder für Nachrichtendaten die Methode setMsgCompList() aufruft. Sobald die Anwendung bereit ist, kann sie die Verbindung erstellen.

Die folgenden Codefragmente veranschaulichen die beschriebene Methode. Das erste Codefragment zeigt Ihnen, wie Sie eine Komprimierung der Headerdaten implementieren können:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

Das zweite Codefragment zeigt Ihnen, wie Sie eine Komprimierung der Nachrichtendaten implementieren können:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

Im zweiten Beispiel werden die Komprimierungstechniken in der Reihenfolge RLE und dann ZLIBHIGH vereinbart, wenn die Verbindung erstellt wird. Die ausgewählte Komprimierungstechnik kann während

der Lebensdauer des Connection-Objekts nicht geändert werden. Wenn Sie die Komprimierung für eine Verbindung verwenden möchten, müssen die Methoden setHdrCompList() und setMsgCompList() vor Erstellung des Verbindungsobjekts (Connection) aufgerufen werden.

Nachrichten asynchron in IBM MQ classes for JMS einreihen
Wenn eine Anwendung Nachrichten an ein Ziel sendet, muss sie normalerweise warten, bis der Warteschlangenmanager bestätigt, dass er die Anforderung verarbeitet hat. Sie können die Messaging-Leistung unter gewissen Umständen verbessern, indem Sie stattdessen wählen, dass Nachrichten asynchron eingereiht werden sollen. Wenn eine Anwendung eine Nachricht asynchron einreiht, meldet der Warteschlangenmanager nicht für jeden einzelnen Aufruf einen Erfolg oder Fehler, Sie können jedoch stattdessen laufend prüfen, ob Fehler aufgetreten sind.

Ob ein Ziel die Steuerung an die Anwendung zurückgibt, ohne zu ermitteln, ob der Warteschlangenmanager die Nachricht fehlerfrei erhalten hat, hängt von den folgenden Eigenschaften ab:

JMS-Zieleigenschaft PUTASYNCALLOWED (Kurzname PAALD).
PUTASYNCALLOWED steuert, ob JMS-Anwendungen Nachrichten asynchron einreihen können, wenn diese Option von der zugrunde liegenden Warteschlange oder dem zugrunde liegenden Thema, die bzw. das vom JMS-Ziel dargestellt wird, zugelassen wird.

IBM MQ-Warteschlangen- oder -Themeneigenschaft DEFPRESP (Standardantworttyp für Einreihung).
DEFPRESP gibt an, ob Anwendungen, die Nachrichten in die Warteschlange einreihen oder Nachricht im Thema veröffentlichen, die Funktionalität der asynchronen Put-Operationen nutzen können.

Die folgende Tabelle enthält die möglichen Werte für die Eigenschaften PUTASYNCALLOWED und DEFPRESP und die Wertekombinationen, die für eine Aktivierung der Funktionalität der asynchronen Put-Operationen verwendet werden:

Tabelle 46. Kombination der Eigenschaften PUTASYNCALLOWED und DEFPRESP zur Festlegung, ob Nachrichten asynchron in ein Ziel eingereiht werden

IBM MQ-Warteschlangeneigenschaft	PUTASYNCALLOWED = NO	PUTASYNCALLOWED = YES	Funktionalität der asynchronen Put-Operation aktiviert
DEFPRESP=SYNC	Funktionalität der asynchronen Put-Operation nicht aktiviert	Funktionalität der asynchronen Put-Operation aktiviert	PUTASYNCALLOWED = AS_DEST oder AS_Q_DEF oder AS_T_DEF
DEFPRESP=ASYNC	Funktionalität der asynchronen Put-Operation nicht aktiviert	Funktionalität der asynchronen Put-Operation aktiviert	PUTASYNCALLOWED = AS_DEST oder AS_Q_DEF oder AS_T_DEF

Sie können das Verhalten ändern, indem Sie die Zieleigenschaft IBM MQ-JMS wie in der Tabelle gezeigt mit "NO" oder "YES" angeben. Es kann jedoch auch für die gesamte Java Virtual Machine mit der JVM **SystemProperty** und dem folgenden Wert überschrieben werden:

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

Bei Nachrichten, die in einer transaktionsbasierten Sitzung gesendet wurden, ermittelt die Anwendung am Ende, ob der Warteschlangenmanager die Nachrichten fehlerfrei erhalten hat, wenn sie `commit()` aufruft.

Wenn eine Anwendung persistente Nachrichten innerhalb einer transaktionsbasierten Sitzung sendet und eine oder mehrere der Nachrichten nicht fehlerfrei erhalten wurde, kann die Transaktion nicht festgeschrieben werden und erstellt eine Ausnahmebedingung. Wenn eine Anwendung jedoch nicht persistente Nachrichten innerhalb einer transaktionsbasierten Sitzung sendet und eine oder mehrere der Nachrichten nicht fehlerfrei erhalten wurden, kann die Transaktion erfolgreich eine Festschreibung durchführen. Die Anwendung erhält keine Rückmeldung darüber, dass die nicht persistenten Nachrichten nicht fehlerfrei angekommen sind.

Bei nicht persistenten Nachrichten, die in einer nicht transaktionsbasierten Sitzung gesendet wurden, gibt die Eigenschaft SENDCHECKCOUNT des *ConnectionFactory*-Objekts an, wie viele Nachrichten gesendet werden sollen, bevor IBM MQ classes for JMS prüfen, ob der Warteschlangenmanager die Nachrichten fehlerfrei erhalten hat.

Wenn bei einer Prüfung festgestellt wird, dass eine oder mehrere Nachrichten nicht fehlerfrei erhalten wurden und die Anwendung einen Listener für Ausnahmebedingungen bei der Verbindung registriert hat, rufen IBM MQ classes for JMS die Methode `onException()` des Listeners für Ausnahmebedingungen auf, um eine JMS-Ausnahmebedingung an die Anwendung zu übergeben.

Die JMS-Ausnahmebedingung hat den Fehlercode JMSWMQ0028, der die folgende Nachricht anzeigt:

```
At least one asynchronous put message failed or gave a warning.
```

Die JMS-Ausnahmebedingung ist zudem mit einer Ausnahmebedingung verlinkt, die weitere Einzelangaben bereitstellt. Der Standardwert der Eigenschaft SENDCHECKCOUNT ist null, was bedeutet, dass keine derartigen Prüfungen durchgeführt werden.

Diese Optimierung ist am nützlichsten für eine Anwendung, die sich im Clientmodus mit einem Warteschlangenmanager verbindet und schnell nacheinander eine Folge von Nachrichten senden muss, jedoch nicht für jede gesendete Nachricht eine sofortige Rückmeldung vom Warteschlangenmanager benötigt. Eine Anwendung kann diese Optimierung aber auch dann nutzen, wenn sie sich im Bindungsmodus mit einem Warteschlangenmanager verbindet. Allerdings ist der erwartete Leistungsvorteil in diesem Fall geringer.

Anmerkung: Wenn Sie eine nicht identifizierte **`MessageProducer`** zum Senden einer Nachricht unter einer Transaktion verwenden, werden die Nachrichten standardmäßig mit dem asynchronen Einreihungsmechanismus in die Warteschlange eingereiht.

Dies kann auftreten, weil die JMS -API die Erstellung von **`MessageProducer`** ohne Angabe eines Ziels mit der folgenden Syntax ermöglicht:

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
timeToLive);
```

In diesem Szenario wird das JMS -Ziel bereitgestellt, wenn die Nachricht gesendet wird, und nicht im Voraus, wenn **`MessageProducer`** erstellt wird. In Bezug auf die IBM MQ -API führt dies dazu, dass ein MQPUT1 ausgegeben wird, um die Nachricht in die Warteschlange einzureihen.

Wenn Sie dies unter einem IBM MQ -Synchronisationspunkt tun, bedeutet dies (in der JMS -Terminologie), dass die Nachricht unter einer Transaktion eingereiht wird, entweder mithilfe einer JMS -Sitzung mit Transaktionsunterstützung oder durch die Verwendung einer XA`Session` -API IBM MQ classes for JMS zur Verwendung der asynchronen Put-Operation.

Vorauslesen zusammen mit IBM MQ classes for JMS verwenden
Die Vorauslesefunktion, die von IBM MQ bereitgestellt wird, ermöglicht es, dass nicht persistente Nachrichten, die außerhalb einer Transaktion empfangen werden, an die IBM MQ classes for JMS gesendet werden, bevor sie von einer Anwendung angefordert werden. Die IBM MQ classes for JMS speichern die Nachrichten in einem internen Puffer und übergeben sie an die Anwendung, sobald sie von ihr angefordert werden.

Anwendungen der IBM MQ classes for JMS, die `MessageConsumers` oder `MessageListeners` beim Empfang von Nachrichten aus einem Ziel außerhalb einer Transaktion verwenden, können die Vorauslesefunktion verwenden. Wenn das Vorauslesen verwendet wird, können Anwendungen, die diese Objekte nutzen, beim Empfang von Nachrichten von einer verbesserten Leistung profitieren.

Ob eine Anwendung, die `MessageConsumers` oder `MessageListeners` verwendet, das Vorauslesen verwenden kann, hängt von den folgenden Eigenschaften ab:

JMS-Zieleigenschaft READAHEADALLOWED (Kurzname RAALD)
READAHEADALLOWED steuert, ob JMS-Anwendungen das Vorauslesen verwenden können, wenn sie Nachrichten außerhalb einer Transaktion abrufen oder durchsuchen, falls die zugrunde liegende Warteschlange oder das zugrunde liegende Thema, die bzw. das vom JMS-Ziel dargestellt wird, diese Option erlaubt.

IBM MQ-Warteschlangen- oder -Themeneigenschaft DEFREADA (standardmäßiges Vorauslesen)
DEFREADA gibt an, ob Anwendungen, die nicht persistente Nachrichten außerhalb einer Transaktion abrufen oder durchsuchen, das Vorauslesen verwenden können.

Die folgende Tabelle enthält die möglichen Werte für die Eigenschaften READAHEADALLOWED und DEFREADA und die Wertekombinationen, die für eine Aktivierung der Vorauslesefunktion verwendet werden:

Tabelle 47. Wie die Eigenschaften READAHEADALLOWED und DEFREADA kombiniert werden, um festzulegen, ob das Vorauslesen beim Abrufen oder Durchsuchen von nicht persistenten Nachrichten außerhalb einer Transaktion verwendet wird

IBM MQ-Warteschlangeneigenschaft	READAHEADALLOWED= YES	READAHEADALLOWED= NO	AS_DEST oder AS_Q_DEF or AS_T_DEF
DEFREADA = NO	Vorauslesefunktion aktiviert	Vorauslesefunktion nicht aktiviert	Vorauslesefunktion nicht aktiviert
DEFREADA = YES	Vorauslesefunktion aktiviert	Vorauslesefunktion nicht aktiviert	Vorauslesefunktion aktiviert
DEFREADA = DISABLED	Vorauslesefunktion nicht aktiviert	Vorauslesefunktion nicht aktiviert	Vorauslesefunktion nicht aktiviert

Wenn die Vorauslesefunktion aktiviert ist und von einer Anwendung ein `MessageConsumer` oder `MessageListener` erstellt wird, erstellen die IBM MQ classes for JMS einen internen Puffer für das Ziel, das vom `MessageConsumer` oder `MessageListener` überwacht wird. Für jeden `MessageConsumer` oder `MessageListener` gibt es einen internen Puffer. Der Warteschlangenmanager startet das Senden von nicht persistenten Nachrichten an die IBM MQ classes for JMS, sobald die Anwendung eine der folgenden Methoden aufruft:

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

Die IBM MQ classes for JMS geben automatisch die erste Nachricht an die Anwendung zurück, und zwar auf Basis des von der Anwendung gestellten Methodenaufrufs. Die übrigen nicht persistenten Nachrichten werden von den IBM MQ classes for JMS in dem internen Puffer gespeichert, der für das Ziel erstellt wurde. Wenn die Anwendung die Verarbeitung der nächsten Nachricht anfordert, geben die IBM MQ classes for JMS die nächste Nachricht im internen Puffer zurück.

Die IBM MQ classes for JMS fordern weitere nicht persistente Nachrichten vom Warteschlangenmanager an, wenn der interne Puffer leer ist.

Der von den IBM MQ classes for JMS verwendete interne Puffer wird gelöscht, wenn eine Anwendung einen `MessageConsumer` oder die JMS-Sitzung schließt, die einem `MessageListener` zugeordnet ist.

Bei `MessageConsumers` gehen sämtliche nicht verarbeitete Nachrichten im internen Puffer verloren.

Falls `MessageListeners` verwendet werden, hängt die Aktion für die Nachrichten im internen Puffer von der JMS-Zieleigenschaft READAHEADCLOSEPOLICY (Kurzname - RACP) ab. Der Standardwert der Eigenschaft ist DELIVER_ALL. Dies bedeutet, dass die JMS-Sitzung, mit der der `MessageListener` erstellt wurde, erst dann geschlossen wird, wenn alle Nachrichten im internen Puffer an die Anwendung zugestellt wurden. Wenn die Eigenschaft auf DELIVER_CURRENT gesetzt ist, wird die JMS-Sitzung geschlossen, sobald die aktuelle Nachricht von der Anwendung verarbeitet wurde. Alle übrigen Nachrichten im internen Puffer werden gelöscht.

Ständige Veröffentlichungen in IBM MQ classes for JMS
Ein Client der IBM MQ classes for JMS kann für die Verwendung von ständigen Veröffentlichungen konfiguriert werden.

Ein Publisher kann festlegen, dass eine Kopie der Veröffentlichung aufbewahrt werden muss, damit diese an künftige Subskribenten gesendet werden kann, die ein Interesse an dem Thema anmelden. Hierfür wird in IBM MQ classes for JMS die ganzzahlige Eigenschaft JMS_IBM_RETAIN auf den Wert 1 gesetzt. Für diese Werte wurden in der Schnittstelle com.ibm.msg.client.jms.JmsConstants Konstanten definiert. Wenn Sie beispielsweise die Nachricht *msg* erstellt haben, müssen Sie folgenden Code verwenden, um sie als ständige Veröffentlichung festzulegen:

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

Sie können die Nachricht jetzt wie eine normale Nachricht senden. JMS_IBM_RETAIN kann auch in einer empfangenen Nachricht abgefragt werden. Daher kann abgefragt werden, ob eine empfangene Nachricht eine ständige Veröffentlichung ist.

XA-Unterstützung in IBM MQ classes for JMS
In den Bindungs- und Clientmodi unterstützt JMS XA-kompatible Transaktionen mit einem unterstützten Transaktionsmanager innerhalb eines JEE-Containers.

Wenn Sie die XA-Funktionalität in einer Anwendungsserverumgebung benötigen, müssen Sie Ihre Anwendung entsprechend konfigurieren. In der Dokumentation zu Ihrem Anwendungsserver finden Sie Informationen darüber, wie Sie Anwendungen für die Verwendung von verteilten Transaktionen konfigurieren können.

Ein IBM MQ-Warteschlangenmanager kann nicht als Transaktionsmanager für JMS agieren.

Zustellungsverzögerung für JMS -Nachrichten
Für JMS 2.0 oder höher können Sie beim Senden einer Nachricht eine Zustellungsverzögerung festlegen. Der Warteschlangenmanager stellt die Nachricht dann erst nach Ablauf der angegebenen Zustellungsverzögerung zu.

Eine Anwendung kann beim Senden einer Nachricht über `MessageProducer.setDeliveryDelay(long deliveryDelay)` oder `JMSProducer.setDeliveryDelay(long deliveryDelay)`eine Zustellungsverzögerung in Millisekunden angeben. Dieser Wert wird zu dem Zeitpunkt, zu dem die Nachricht gesendet wurde, hinzugefügt und gibt den frühesten Zeitpunkt an, zu dem die betreffende Nachricht von anderen Anwendungen abgerufen werden kann.

Die Zustellungsverzögerung wird über eine einzelne interne Staging-Warteschlange implementiert. Nachrichten mit einer Zustellungsverzögerung ungleich null werden in diese Warteschlange mit einer Kopfzeile gestellt, in der die Zustellungsverzögerung sowie Informationen zur Zielwarteschlange angegeben sind. Eine Komponente des Warteschlangenmanagers, der sogenannte Zustellungsverzögerungsprozessor, überwacht die Nachrichten in der Staging-Warteschlange. Wenn die Zustellungsverzögerung für eine Nachricht abläuft, wird die Nachricht aus der Staging-Warteschlange in die Zielwarteschlange gestellt.

Messaging-Clients

Die IBM MQ-Implementierung der Zustellungsverzögerung kann nur verwendet werden, wenn Sie den JMS-Client verwenden. Bei Verwendung der Zustellungsverzögerung mit IBM MQ gelten folgende Einschränkungen. Diese Einschränkungen gelten gleichermaßen für `MessageProducers` und `JMSProducers`, aber `JMSRuntimeExceptions` werden bei `JMSProducers`ausgelöst.

- Jeder Versuch, `MessageProducer.setDeliveryDelay` mit einem Wert ungleich null aufzurufen, wenn eine Verbindung zu einem Warteschlangenmanager vor IBM MQ 8.0 besteht, führt zu einer `JMSException`-Ausnahme mit der Nachricht MQRC_FUNCTION_NOT_SUPPORTED.
- Für Ziele, die in Gruppen zusammengefasst sind und bei denen für **DEFBIND** ein anderer Wert als MQBND_BIND_NOT_FIXED angegeben ist, wird die Zustellungsverzögerung nicht unterstützt. Wenn für einen `MessageProducer` eine Zustellungsverzögerung ungleich null festgelegt ist und versucht wird,

eine Nachricht an ein Ziel zu senden, das diese Anforderung nicht erfüllt, führt dieser Aufruf zu einer `JMSException`-Ausnahme mit der Nachricht MQRC_OPTIONS_ERROR.

- Jeder Versuch, einen Wert für die Lebensdauer festzulegen, der niedriger als eine zuvor angegebene Zustellungsverzögerung (ungleich null) ist, oder umgekehrt, führt zu einer `JMSException`-Ausnahme mit der Nachricht MQRC_EXPIRY_ERROR. Diese Prüfung erfolgt beim Aufruf der `setTimeToLive`-, `setDeliveryDelay`- oder `send`-Methoden, je nachdem, welche genaue Arbeitsganggruppe ausgewählt wurde.
- Die Verwendung von ständigen Veröffentlichungen und Zustellungsverzögerung wird nicht unterstützt. Wenn versucht wird, eine Nachricht mit einer Zustellungsverzögerung zu veröffentlichen, die mit `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` als ständige Veröffentlichung festgelegt wurde, kommt es zu einer `JMSException`-Ausnahme mit der Nachricht MQRC_OPTIONS_ERROR.
- Die Zustellungsverzögerung und die Nachrichtengruppierung werden nicht unterstützt und jeder Versuch, diese Kombination zu verwenden, führt zu einer `JMSException` mit der Nachricht MQRC_OPTIONS_ERROR.

Wenn eine Nachricht nicht mit Zustellungsverzögerung gesendet werden kann, löst der Client eine `JMSException`-Ausnahme mit der entsprechenden Fehlernachricht aus (beispielsweise mit der Angabe, dass die Warteschlange voll ist). In manchen Situationen kann sich die Fehlernachricht auf das Ziel und/oder die Staging-Warteschlange beziehen.

Anmerkung: Bei IBM MQ können Anwendungen, die eine Nachricht in eine Arbeitseinheit stellen, dieselbe Nachricht auch dann erneut abrufen, wenn die Arbeitseinheit nicht festgeschrieben wurde. Dieses Verfahren funktioniert nicht in Verbindung mit Zustellungsverzögerung, da die Nachricht erst in die Staging-Warteschlange gestellt wird, wenn die Arbeitseinheit festgeschrieben wurde, und somit nicht an das Ziel gesendet wird.

Autorisierung

IBM MQ führt Berechtigungsprüfungen auf dem ursprünglichen Ziel durch, wenn die Anwendung eine Nachricht mit einer Zustellungsverzögerung ungleich null sendet. Ist die Anwendung nicht berechtigt, schlägt der Sendevorgang fehl. Wenn der Warteschlangenmanager feststellt, dass die Zustellungsverzögerung für eine Nachricht abgeschlossen ist, öffnet er die Zielwarteschlange. An diesem Punkt werden keine Berechtigungsprüfungen durchgeführt.

SYSTEM.DDELAY.LOCAL.QUEUE

Zur Implementierung der Zustellungsverzögerung wird eine neue Systemwarteschlange, SYSTEM.DDELAY.LOCAL.QUEUE, verwendet.

- Multi Unter Multiplatforms: SYSTEM.DDELAY.LOCAL.QUEUE ist standardmäßig vorhanden. Die Systemwarteschlange muss so geändert werden, dass die Werte der Attribute MAXMSGL und MAXDEPTH für die erwartete Auslastung ausreichen.
- z/OS Unter IBM MQ for z/OS: SYSTEM.DDELAY.LOCAL.QUEUE wird als Staging-Warteschlange für Nachrichten verwendet, die mit Zustellungsverzögerung an lokale und gemeinsam genutzte Warteschlangen gesendet werden. Unter z/OS muss die Warteschlange erstellt und so definiert werden, dass die Werte für die Attribute MAXMSGL und MAXDEPTH für die erwartete Auslastung ausreichen.

Wenn diese Warteschlange erstellt wird, muss sie so gesichert werden, dass möglichst wenige Benutzer Zugriff darauf haben. Der Zugriff auf die Warteschlange darf nur zu Wartungs- und Überwachungszwecken möglich sein.

Wenn von einer JMS-Anwendung eine Nachricht mit einer Zustellungsverzögerung ungleich null gesendet wird, wird diese Nachricht mit einer neuen Nachrichten-ID in diese Warteschlange eingereiht. Die ursprüngliche Nachrichten-ID wird in die Korrelations-ID der Nachricht eingefügt. Mithilfe dieser Korrelations-ID kann eine Anwendung eine Nachricht bei Bedarf aus der Staging-Warteschlange abrufen, beispielsweise wenn versehentlich eine zu große Zustellungsverzögerung eingestellt wurde.

Hinweise zu z/OS

z/OS

Wenn Ihr System unter z/OS ausgeführt wird, sind zusätzliche Punkte zu berücksichtigen, wenn Zustellungsverzögerung verwendet werden soll.

Für die Verwendung der Zustellungsverzögerung muss die Systemwarteschlange SYSTEM.DDELAY.LOCAL.QUEUE definiert werden. Die Warteschlange muss mit einer ausreichenden Speicherklasse für die erwartete Auslastung definiert werden, außerdem müssen INDXTYPE(NONE) und MSGDLVSQ(FIFO) angegeben sein. In der CSQ4INSG-JCL ist eine auf Kommentar gesetzte Beispieldefinition der Systemwarteschlange angegeben.

Gemeinsam genutzte Warteschlangen

Die Zustellungsverzögerung wird für das Senden von Nachrichten an gemeinsam genutzte Warteschlangen unterstützt. Es gibt allerdings nur eine einzelne nicht öffentliche Staging-Warteschlange, die unabhängig davon verwendet wird, ob die Zielwarteschlange gemeinsam genutzt wird oder nicht. Der Warteschlangenmanager, der Eigner dieser nicht öffentlichen Warteschlange ist, muss aktiv sein, damit die verzögerte Nachricht nach Ablauf der Verzögerung an ihre gemeinsam genutzte Zielwarteschlange gesendet wird.

Anmerkung: Wenn eine nicht persistente Nachricht mit einer Zustellungsverzögerung in eine gemeinsam genutzte Warteschlange eingereiht wird und der Warteschlangenmanager, der Eigner der Staging-Warteschlange ist, abgeschaltet wird, geht die ursprüngliche Nachricht verloren. Folglich besteht bei nicht persistenten Nachrichten, die mit Zustellungsverzögerung an eine gemeinsam genutzte Warteschlange gesendet werden, eher die Gefahr, dass sie verloren gehen, als bei nicht persistenten Nachrichten, die ohne Zustellungsverzögerung an eine gemeinsam genutzte Warteschlange gesendet werden.

Zielauflösung

Wenn die Nachricht an eine Warteschlange gesendet wird, wird zweimal eine Auflösung veranlasst, einmal von der JMS-Anwendung und einmal von dem Warteschlangenmanager, wenn dieser die Nachricht aus der Staging-Warteschlange an die Zielwarteschlange sendet.

Der Abgleich von Zielsubskriptionen für Veröffentlichungen erfolgt, wenn die JMS-Anwendung die Sendemethode aufruft.

Wenn eine Nachricht mit der Persistenz oder Priorität gemäß der Warteschlangendefinition gesendet wird, wird der Wert bei der ersten und nicht bei der zweiten Auflösung definiert.

Ablaufintervall

Die Zustellungsverzögerung behält das Verhalten der Ablaufeigenschaft **MQMD.Expiry**bei. Wenn beispielsweise eine Nachricht mit einem Ablaufintervall von 20.000 ms und einer Zustellungsverzögerung von 5.000 ms aus einer JMS-Anwendung eingereiht und nach Ablauf von 10.000 ms abgerufen wurde, könnte im Feld 'MQMD.expirey' ungefähr ein Wert von 50 Zehntelsekunden angegeben sein. Dieser Wert gibt an, dass von dem Zeitpunkt, an dem die Nachricht eingereiht wurde, bis zu dem Moment, an dem sie abgerufen wurde, 15 Sekunden vergangen sind.

Wenn eine Nachricht abläuft, solange sie sich in der Staging-Warteschlange befindet, und eine der Optionen MQRO_EXPIRATION_* definiert ist, bezieht sich der generierte Bericht auf die ursprüngliche von der Anwendung gesendete Nachricht, der Header mit den Informationen zur Zustellungsverzögerung ist entfernt.

Zustellungsverzögerungsprozessor stoppen und starten

z/OS Unter z/OS ist der Zustellungsverzögerungsprozessor in den Warteschlangenmanager-MSTR-Adressraum integriert. Beim Start des Warteschlangenmanagers wird auch der Zustellungsverzögerungsprozessor gestartet. Wenn die Staging-Warteschlange verfügbar ist, öffnet er sie und wartet auf eingehende zu verarbeitende Nachrichten. Falls die Staging-Warteschlange nicht definiert wurde oder

für Abrufe inaktiviert ist oder falls ein anderer Fehler auftritt, wird der Zustellungsverzögerungsprozessor abgeschaltet. Wenn die Staging-Warteschlange später definiert oder für Abrufe aktiviert wird, wird der Zustellungsverzögerungsprozessor erneut gestartet. Wenn der Zustellungsverzögerungsprozessor aus einem anderen Grund beendet wird, kann er neu gestartet werden, indem das Attribut **PUT** der Staging-Warteschlange von ENABLED in DISABLED und wieder zurück in ENABLED geändert wird. Sollten Sie den Zustellungsverzögerungsprozessor aus irgendeinem Grund stoppen müssen, setzen Sie das PUT-Attribut der Staging-Warteschlange auf DISABLED.

Auf Multiplatforms wird der Zustellungsverzögerungsprozessor mit dem Warteschlangenmanager gestartet und im Fall eines behebbaren Fehlers automatisch erneut gestartet.

Fehler beim Einreihen in die Zielwarteschlange

Wenn eine verzögerte Nachricht nach Ablauf der Verzögerung nicht in die Zielwarteschlange eingereiht werden kann, wird mit der Nachricht wie in den entsprechenden Berichtsoptionen angegeben verfahren: die Nachricht wird entweder gelöscht oder an die Warteschlange für nicht zustellbare Nachrichten gesendet. Kommt es dabei zu einem Fehler, wird versucht, die Nachricht später einzureihen. Ist die Maßnahme erfolgreich, wird ein Ausnahmebericht generiert und an die angegebene Warteschlange gesendet, wenn der Bericht angefordert wird. Falls die Berichtsnachricht nicht gesendet werden konnte, wird sie an die Warteschlange für nicht zustellbare Nachrichten gesendet. Wenn es beim Senden des Berichts an die Warteschlange für nicht zustellbare Nachrichten zu einem Fehler kommt und es sich um eine persistente Nachricht handelt, werden alle Änderungen verworfen und die ursprüngliche Nachricht wird wiederhergestellt und später erneut übermittelt. Ist die Nachricht nicht persistent, wird die Berichtsnachricht gelöscht, andere Änderungen werden jedoch festgeschrieben. Falls eine verzögerte Veröffentlichung nicht übermittelt werden kann, da ein Subskribent seine Subskription aufgehoben hat oder es sich um einen nicht permanenten Subskribenten handelt, der die Verbindung getrennt hat, wird die Nachricht im Hintergrund gelöscht. Es werden nach wie vor Berichtsnachrichten wie oben beschrieben generiert.

Wenn eine verzögerte Veröffentlichung nicht an einen Subskribenten übermittelt werden kann und stattdessen in die Warteschlange für nicht zustellbare Nachrichten eingereiht wird und es beim Einreihen in diese Warteschlange zu einem Fehler kommt, wird die Nachricht gelöscht.

Um die Gefahr zu verringern, dass es nach Ablauf der Zustellungsverzögerung beim Einreihen in die Zielwarteschlange zu einem Fehler kommt, führt der Warteschlangenmanager einige grundlegende Prüfungen durch, wenn der JMS-Client eine Nachricht mit einer Zustellungsverzögerung ungleich null sendet. Unter anderem wird geprüft, ob die Warteschlange für Einreihungen deaktiviert ist, ob die Nachricht größer als die zulässige maximale Nachrichtenlänge ist und ob die Warteschlange voll ist.

Publish/Subscribe

Der Abgleich einer Veröffentlichung mit den verfügbaren Subskriptionen erfolgt, wenn die JMS-Anwendung eine Nachricht mit einer Zustellungsverzögerung ungleich null sendet. Für jeden übereinstimmenden Subskribenten wird eine Nachricht in die Warteschlange SYSTEM.DDELAY.LOCAL.QUEUE eingereiht und bleibt dort bis zum Ablauf der Zustellungsverzögerung. Wenn es sich bei einem der Subskribenten um eine Proxy-Subskription für einen anderen Warteschlangenmanager handelt, erfolgt die Ausgabefächerung auf diesem Warteschlangenmanager nach Ablauf der Zustellungsverzögerung. Folglich könnte es passieren, dass Subskribenten auf dem anderen Warteschlangenmanager Veröffentlichungen erhalten, die ursprünglich vor ihrer Subskription veröffentlicht wurden. Dies ist eine Abweichung von der Spezifikation JMS 2.0 oder höher.

Die Zustellungsverzögerung mit Publish/Subscribe wird nur unterstützt, wenn das Zielthema mit (N)PMSGDLV = ALLAVAIL konfiguriert ist. Wenn versucht wird, andere Werte zu verwenden, kommt es zu einem Fehler MQRC_PUBLICATION_FAILURE. Falls der Zustellungsverzögerungsprozessor fehlschlägt, während er die Nachricht in die Zielwarteschlange einreiht, führt dies zu der im Abschnitt "Fehler beim Einreihen in die Zielwarteschlange" beschriebenen Situation.

Berichtsnachrichten

Vom Übermittlungsprozessor werden alle Berichtsoptionen unterstützt und verarbeitet, mit Ausnahme der folgenden Optionen, die ignoriert, aber in der Nachricht weitergeleitet werden, wenn diese an die Zielwarteschlange gesendet wird:

- MQRO_COA*
- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY

Klone und gemeinsam genutzte Subskriptionen

Es gibt zwei Methoden, mehreren Konsumenten Zugriff auf dieselbe Subskription zu erteilen. die Verwendung geklonter Subskriptionen oder die Verwendung gemeinsam genutzter Subskriptionen.

Geklonte Subskriptionen

Eine geklonte Subskription ist eine IBM MQ-Erweiterung. Mithilfe geklonter Subskriptionen können mehrere Konsumenten auf verschiedenen Java Virtual Machines (JVMs) gleichzeitig auf die Subskription zugreifen. Dieses Verhalten kann verwendet werden, wenn die Eigenschaft **CLONESUPP** in einem connectionFactory-Objekt auf `Enabled` gesetzt wird. Standardmäßig ist für **CLONESUPP** `Disabled` festgelegt. Geklonte Subskriptionen können nur auf permanenten Subskriptionen aktiviert werden. Wenn **CLONESUPP** aktiviert ist, wird jede nachfolgende Verbindung, die über diese Verbindungsfactory hergestellt wird, geklont.

Eine permanente Subskription kann als geklont angesehen werden, wenn ein oder mehrere Konsumenten erstellt wurden, die Nachrichten von dieser Subskription erhalten sollen, d. h., wenn beim Erstellen der Konsumenten derselbe Subskriptionsname angegeben wurde. Dies ist nur möglich, wenn für die Verbindung, unter der die Konsumenten erstellt wurden, der Parameter **CLONESUPP** im MQConnectionFactory-Objekt auf `Enabled` gesetzt ist. Wenn eine Nachricht zu dem Thema der Subskription veröffentlicht wird, wird eine Kopie dieser Nachricht an die Subskription gesendet. Die Nachricht ist für alle Konsumenten verfügbar, aber nur einer erhält sie.

Anmerkung: Die Aktivierung geklonter Subskriptionen erweitert die JMS-Spezifikation.

Gemeinsam genutzte Subskriptionen

Gemeinsam genutzte Subskriptionen, die durch die Spezifikation JMS 2.0 eingeführt wurden, ermöglichen die gemeinsame Nutzung von Nachrichten aus einer Topicsubskription durch mehrere Konsumenten. Jede Nachricht von der Subskription wird nur einem der Konsumenten für die betreffende Subskription zugestellt. Gemeinsam genutzte Subskriptionen werden durch den relevanten Aufruf der API JMS 2.0 oder höher aktiviert.

Die APIs können auf eine der folgenden Arten aufgerufen werden:

- Aus einer Java SE-Anwendung (oder Java EE Client Container).
- Aus einem Servlet oder der Implementierung einer MDB.

Die Spezifikation JMS 2.0 oder höher definiert keine Standardmethode für die Ansteuerung einer MDB aus einer gemeinsam genutzten Subskription. Daher stellt IBM MQ zu diesem Zweck die Aktivierungsspezifikationseigenschaft **sharedSubscription** bereit. Weitere Informationen zu dieser Eigenschaft finden Sie in den Abschnitten „Ressourcenadapter für eingehende Kommunikation konfigurieren” auf Seite 475 und „Beispiele für Definition der Eigenschaft sharedSubscription” auf Seite 493.

Wenn eine gemeinsam genutzte Subskription aktiviert ist, kann die gemeinsame Nutzung nicht aufgehoben werden.

Gemeinsam genutzte Subskriptionen können als permanente oder nicht permanente Subskriptionen erstellt werden. Es ist nicht erforderlich, Objekte auf der Warteschlangenmanagerseite über die normale JMS -Konfiguration hinaus separat zu erstellen. Alle erforderlichen Objekte werden dynamisch erstellt.

Entscheidung zwischen gemeinsam genutzten oder geklonten Subskriptionen

Wenn Sie entscheiden, ob gemeinsam genutzte oder geklonte Subskriptionen verwendet werden sollen, sollten Sie die Vorteile beider Optionen berücksichtigen. Verwenden Sie nach Möglichkeit gemeinsam genutzte Subskriptionen, da es sich um ein spezifikationsdefiniertes Verhalten handelt, und nicht um eine IBM MQ -spezifische Erweiterung.

In der folgenden Tabelle sind einige Punkte aufgeführt, die bei der Entscheidung für gemeinsam genutzte oder geklonte Subskriptionen zu beachten sind:

Tabelle 48. Vergleich der Aspekte für gemeinsam genutzte und geklonte Subskriptionen

Gemeinsam genutzte Subskriptionen	**Geklonte Subskriptionen**
Gemeinsam genutzte Subskriptionen sind ein Standardteil der Spezifikation JMS 2.0 oder höher.	Bei geklonten Subskriptionen handelt es sich um eine IBM MQ-spezifische Erweiterung.
Gemeinsam genutzte Subskriptionen werden mithilfe expliziter API-Methodenaufrufe erstellt.	Geklonte Subskriptionen werden verwaltungstechnisch auf Verbindungsfactoryebene gesteuert.
Gemeinsam genutzte Subskriptionen können permanent oder nicht permanent sein.	Geklonte Subskriptionen können nur permanent sein.
Gemeinsam genutzte Subskriptionen werden explizit auf einer einzelnen Subskriptionsbasis erstellt.	Geklonte Subskriptionen werden für permanente Subskriptionen unter einer Verbindung verwendet, für welche die Funktion aktiviert ist.
Wenn eine Subskription als gemeinsam genutzt erstellt wird, kann die gemeinsame Nutzung später nicht aufgehoben werden. Dasselbe gilt auch umgekehrt.	Eine Subskription kann bei jedem erneuten Öffnen von geklont in nicht geklont geändert werden, wenn sich die Eigenschaft **CLONESUPP** der Eignerverbindung geändert hat.

Zugehörige Konzepte

Subskribenten und Subskriptionen

Subskriptionspermanenz

Zugehörige Tasks

Gemeinsam genutzte JMS 2.0-Subskriptionen verwenden

Zugehörige Verweise

„Beispiele für Definition der Eigenschaft sharedSubscription” auf Seite 493

Sie können die Eigenschaft 'sharedSubscription' einer Aktivierungsspezifikation in einer Datei `server.xml` von WebSphere Liberty definieren. Alternativ können Sie die Eigenschaft in einer Message-driven Bean (MDB) mithilfe von Annotationen definieren.

CLONESUPP

Eigenschaft SupportMQExtensions

Die Spezifikation JMS 2.0 hat Änderungen an der Funktionsweise bestimmter Verhaltensweisen eingeführt. IBM MQ 8.0 und höher enthält die Eigenschaft **`com.ibm.mq.jms.SupportMQExtensions`**, die auf TRUE gesetzt werden kann, um diese geänderten Verhaltensweisen auf frühere Implementierungen zurückzusetzen.

JM 3.0 Die Eigenschaft **`com.ibm.mq.jakarta.jms.SupportMQExtensions`** (Jakarta Messaging 3.0) wird von IBM MQ classes for Jakarta Messagingunterstützt, die in `com.ibm.mq.jakarta.client.jar`verfügbar sind.

JMS 2.0 Die Eigenschaft **`com.ibm.mq.jms.SupportMQExtensions`** (JMS 2.0) wird von den IBM MQ classes for JMSunterstützt, die in `com.ibm.mq.allclient.jar` oder `com.ibm.mqjms.jar`verfügbar sind.

Drei Funktionalitätsbereiche werden zurückgesetzt, wenn **`SupportMQExtensions`** auf `True` gesetzt wird:

Nachrichtenpriorität
Nachrichten kann eine Priorität von 0 bis 9 zugewiesen werden. Vor JMS 2.0könnten Nachrichten auch den Wert -1 verwenden, der angibt, dass die Standardpriorität einer Warteschlange verwendet wird. In JMS 2.0 und höher kann die Nachrichtenpriorität -1 nicht festgelegt werden. Wenn Sie **SupportMQExtensions** aktivieren, kann der Wert -1 verwendet werden.

Client-ID
Die Spezifikation JMS 2.0 oder höher erfordert, dass Client-IDs ungleich null auf Eindeutigkeit überprüft werden, wenn sie eine Verbindung herstellen. Die Aktivierung von **SupportMQExtensions** bedeutet, dass diese Anforderung ignoriert wird und eine Client-ID wiederverwendet werden kann.

NoLocal
Die Spezifikation JMS 2.0 oder höher erfordert, dass ein Konsument keine Nachrichten empfangen kann, die von derselben Client-ID veröffentlicht werden, wenn diese Konstante aktiviert ist. Vor JMS 2.0 wurde dieses Attribut für einen Subskribenten definiert, um zu verhindern, dass er Nachrichten empfängt, die von seiner eigenen Verbindung veröffentlicht werden. Durch Aktivierung von **SupportMQExtensions** wird diese Verhaltensweise wieder auf die vorherige Implementierung zurückgesetzt.

Diese Eigenschaft kann wie folgt festgelegt werden:

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

Diese Eigenschaft kann entweder als Standard-JVM-Systemeigenschaft im **java**-Befehl festgelegt werden oder in der IBM MQ classes for JMS-Konfigurationsdatei enthalten sein.

Zugehörige Konzepte
„Die Konfigurationsdatei IBM MQ classes for JMS/Jakarta Messaging” auf Seite 103
Eine IBM MQ classes for JMS -und IBM MQ classes for Jakarta Messaging -Konfigurationsdatei gibt Eigenschaften an, die für die Konfiguration von IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverwendet werden.

Zugehörige Verweise
„Zur Konfiguration des JMS-Clientverhaltens verwendete Eigenschaften” auf Seite 110
Mit diesen Eigenschaften können Sie das Verhalten des JMS-Clients konfigurieren.

Gemeinsam genutzte Subskriptionen in JMS -Anwendungen verwenden

Bei gemeinsam genutzten Subskriptionen wird eine einzelne Subskription von mehreren Konsumenten gemeinsam genutzt, wobei jeweils nur einer der Konsumenten eine Veröffentlichung empfängt.

Gemeinsam genutzte Subskriptionen sind ab JMS 2.0 verfügbar. Wenn Sie eine JMS -Anwendung für IBM MQ 8.0 oder höher entwickeln, müssen Sie möglicherweise die Auswirkungen dieser Funktionalität auf Ihren Warteschlangenmanager berücksichtigen.

Hinter gemeinsam genutzten Subskriptionen steht die Idee, die Last auf mehrere Konsumenten zu verteilen. Auch eine permanente Subskription kann unter mehreren Konsumenten aufgeteilt werden.

Ein Beispiel:

- Subskription SUB subskribiert ein Thema FIFA2014/UPDATES, um aktuelle Informationen zu Fußballspielen zu erhalten. Die Subskription wird von drei Konsumenten, C1, C2 und C3, gemeinsam genutzt.
- Produzent P1 veröffentlicht im Thema FIFA2014/UPDATES.

Wenn in FIFA2014/UPDATES eine Veröffentlichung erfolgt, wird diese nur von einem der drei Konsumenten (C1, C2 oder C3), nicht von allen erhalten.

Das folgende Beispiel veranschaulicht die Verwendung gemeinsam genutzter Subskriptionen sowie die Verwendung der zusätzlichen API in JMS 2.0, `Message.receiveBody()`, um nur den Nachrichtenhauptteil abzurufen.

In dem Beispiel werden drei Subskribententhreads, die eine gemeinsam genutzte Subskription für das Thema FIFA2014/UPDATES erstellen, sowie ein Publisher-Thread erstellt.

JM 3.0

```
package mqv91Samples;

import jakarta.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

JMS 2.0

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;
```

```
import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
```

```
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
```

```
   }
}
```

```
/*
* Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new SharedNonDurableSubscriberAndPublis
her( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new SharedNonDurableSubscriberAndPublis
her( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new SharedNonDurableSubscriberAndPublis
her( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new SharedNonDurableSubscriberAndPublis
her( "PUBLISHER");
        publisher.start();
    }
}
```

Zugehörige Konzepte

IBM MQ Java-Sprachenschnittstellen

V 9.4.0 Modulare Anwendung für die Verwendung von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging konfigurieren

V 9.4.0 Sie können IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging auf modulare Weise verwenden, indem Sie das entsprechende Modul in Ihrer Anwendung anfordern und das entsprechende Verzeichnis im Modulpfad einschließen.

Die modulare Verpackung

Die einheitlichen JAR-Dateien für IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging stellen automatische Modulnamen bereit, die die Standardnamen ersetzen, die von den JAR-Dateinamen abgeleitet werden.

- IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) werden mit dem Modulnamen `com.ibm.mq.javax`bereitgestellt.
- IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) werden mit dem Modulnamen `com.ibm.mq.jakarta`bereitgestellt.

Das Standardverzeichnis `MQ_HOME/java/lib` ist für die modulare Verwendung nicht geeignet, da Module nicht dasselbe Paket enthalten können und das Standardverzeichnis dieselben Pakete in mehreren JAR-Dateien enthält. Daher sind neue Verzeichnisse für verfügbar, die nur die erforderlichen JAR-Dateien enthalten, ohne dass die Pakete zwischen den JAR-Dateien dupliziert werden müssen. Diese Verzeichnisse sind für die Aufnahme in eine `module-path`geeignet.

Anmerkung: Wenn Sie Anwendungen haben, die die verfügbaren JAR-Dateien in einem modularen Kontext verwenden, indem Sie sich auf die Standardmodulnamen verlassen, müssen Sie Ihre Anwendungen aktualisieren, damit die neuen Modulnamen erforderlich sind. Die Standardmodulnamen werden aus den JAR-Dateinamen abgeleitet.

Modulare Anwendung für die Verwendung von IBM MQ classes for JMS konfigurieren

Sie können Ihre modulare Anwendung für die Verwendung von IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) konfigurieren, indem Sie die folgenden Schritte ausführen:

- Konfigurieren Sie die Anwendung so, dass das Modul `com.ibm.mq.javax` erforderlich ist.
- Konfigurieren Sie die Anwendung so, dass das Verzeichnis `MQ_HOME/java/lib/modules/javax` im Modulpfad enthalten ist.

Modulare Anwendung für die Verwendung von IBM MQ classes for Jakarta Messaging konfigurieren

Sie können Ihre modulare Anwendung für die Verwendung von IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) konfigurieren, indem Sie die folgenden Schritte ausführen:

- Konfigurieren Sie die Anwendung so, dass das Modul `com.ibm.mq.jakarta` erforderlich ist.
- Konfigurieren Sie die Anwendung so, dass das Verzeichnis `MQ_HOME/java/lib/modules/jakarta` im Modulpfad enthalten ist.

Modulare Anwendung für die Verwendung von IBM MQ classes for Java konfigurieren

Um IBM MQ classes for Java aus einer modularen Anwendung zu verwenden, können Sie entweder die Konfiguration für IBM MQ classes for JMS oder die Konfiguration für IBM MQ classes for Jakarta Messagingverwenden, da beide Client-JAR-Dateien IBM MQ classes for Javaunterstützen. Ihre Anwendung darf jedoch nur eine dieser Konfigurationen verwenden, nicht beide.

Anwendungsserverfunktionen der IBM MQ classes for JMS

In diesem Abschnitt wird beschrieben, wie IBM MQ classes for JMS die ConnectionConsumer-Klasse und die erweiterte Funktionalität in der Sitzungsklasse implementiert. Darüber hinaus enthält der Abschnitt eine Zusammenfassung der Funktion eines Serversitzungspools.

Wichtig: Diese Informationen werden nur zu Referenzzwecken bereitgestellt. Eine Anwendung darf nicht für die Verwendung dieser Schnittstelle geschrieben werden: sie wird innerhalb des IBM MQ-Ressourcenadapters für die Verbindung mit Java EE-Servern verwendet. Im Abschnitt „IBM MQ-Ressourcenadapter verwenden” auf Seite 457 finden Sie praxisbezogene Verbindungsinformationen.

IBM MQ classes for JMS unterstützt Application Server Facilities (ASF), die in der *Java Message Service -Spezifikation* angegeben sind (siehe Oracle Technology Network for Java Developers). Diese Spezifikation gibt drei Rollen innerhalb dieses Programmiermodells an:

- **Der JMS-Provider** stellt die ConnectionConsumer-Funktionalität und eine erweiterte Sitzungsfunktionalität bereit.
- **Der Anwendungsserver** stellt die ServerSessionPool- und ServerSession-Funktionalität bereit.
- **Die Clientanwendung** verwendet die vom JMS-Provider und Anwendungsserver bereitgestellte Funktionalität.

Die Informationen in diesem Abschnitt gelten nicht, wenn eine Anwendung eine Echtzeitverbindung zu einem Broker nutzt.

JMS-ConnectionConsumer

Die ConnectionConsumer-Schnittstelle stellt eine leistungsfähige Methode für die gleichzeitige Übermittlung von Nachrichten an einen Thread-Pool zur Verfügung.

Die Spezifikation JMS ermöglicht einem Anwendungsserver die enge Integration mit einer JMS -Implementierung über die Schnittstelle `ConnectionConsumer` . Diese Funktion stellt die gleichzeitig ablaufende Verarbeitung von Nachrichten zur Verfügung. In der Regel erstellt ein Anwendungsserver einen Thread-Pool und die JMS-Implementierung stellt diesen Threads Nachrichten zur Verfügung. Ein JMS-

fähiger Anwendungsserver (beispielsweise WebSphere Application Server) kann mit dieser Funktion eine übergeordnete Messaging-Funktionalität wie nachrichtengesteuerte Beans (Message-driven Beans, MDBs) bereitstellen.

Normale Anwendungen verwenden den ConnectionConsumer nicht, spezielle JMS-Clients nutzen diese Schnittstelle möglicherweise jedoch. Für diese Clients stellt die ConnectionConsumer-Schnittstelle eine leistungsfähige Methode für die gleichzeitige Übermittlung von Nachrichten an einen Thread-Pool zur Verfügung. Wenn eine Nachricht bei einer Warteschlange oder einem Thema eintrifft, wählt JMS einen Thread aus dem Pool aus und übermittelt einen Nachrichtenstapel an ihn. Zu diesem Zweck führt JMS die Methode `onMessage()` eines zugehörigen MessageListeners aus.

Sie erzielen denselben Effekt, wenn Sie mehrere Session- und MessageConsumer-Objekte erstellen, die alle einen registrierten MessageListener aufweisen. Der ConnectionConsumer bietet jedoch eine bessere Leistung, belegt weniger Ressourcen und ist flexibler. Es werden vor allem weniger Session-Objekte benötigt.

Anwendung mit ASF planen

In diesem Abschnitt erfahren Sie, wie Sie eine Anwendung planen. Dies umfasst auch folgende Punkte:

- „Allgemeine Grundsätze für das Punkt-zu-Punkt-Messaging unter Verwendung von ASF” auf Seite 351
- „Allgemeine Grundsätze für das Publish/Subscribe-Messaging unter Verwendung von ASF” auf Seite 352
- „Nachrichten aus der Warteschlange in ASF entfernen” auf Seite 353
- Behandlung von nicht verarbeitbaren Nachrichten in ASF. Siehe „Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln” auf Seite 244.

Allgemeine Grundsätze für das Punkt-zu-Punkt-Messaging unter Verwendung von ASF
In diesem Abschnitt finden Sie allgemeine Informationen zum Punkt-zu-Punkt-Messaging mit ASF.

Wenn eine Anwendung einen ConnectionConsumer aus einem QueueConnection-Objekt erstellt, gibt sie ein JMS-Warteschlangenobjekt und eine Selektorzeichenfolge an. Anschließend beginnt der ConnectionConsumer mit der Bereitstellung von Nachrichten für Sitzungen im zugeordneten ServerSessionPool. Die Nachrichten treffen in der Warteschlange ein und falls sie dem Selektor entsprechen, werden sie an die Sitzungen im zugeordneten ServerSessionPool übermittelt.

In IBM MQ bezieht sich der Begriff 'Warteschlangenobjekt' entweder auf eine QLOCAL oder auf ein QALIAS im lokalen Warteschlangenmanager. Falls es sich um ein QALIAS handelt, muss sich QALIAS auf eine QLOCAL beziehen. Die vollständig aufgelöste IBM MQ-QLOCAL wird als *zugrunde liegende QLOCAL* bezeichnet. Ein ConnectionConsumer gilt als *aktiv*, wenn er nicht geschlossen ist und seine übergeordnete QueueConnection gestartet wurde.

Für dieselbe zugrunde liegende QLOCAL können mehrere ConnectionConsumers ausgeführt werden, die jeweils unterschiedliche Selektoren haben. Zur Erzielung einer konstanten Leistung dürfen sich in der Warteschlange keine nicht erwünschten Nachrichten anhäufen. Nicht erwünschte Nachrichten sind Nachrichten, für die kein aktiver ConnectionConsumer einen passenden Selektor hat. Sie können die QueueConnectionFactory so festlegen, dass diese nicht erwünschten Nachrichten aus der Warteschlange entfernt werden (Details hierzu finden Sie unter „Nachrichten aus der Warteschlange in ASF entfernen” auf Seite 353). Sie können dieses Verhalten auf eine von zwei Arten festlegen:

- Verwenden Sie das JMS-Verwaltungstool, um die QueueConnectionFactory auf MRET(NO) zu setzen.
- Verwenden Sie in Ihrem Programm Folgendes:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

Wenn Sie diese Einstellung nicht ändern, verbleiben nicht erwünschte Nachrichten standardmäßig in der Warteschlange.

Berücksichtigen Sie bei der Einrichtung des IBM MQ-Warteschlangenmanagers die folgenden Punkte:

- Die zugrunde liegende Warteschlange QLOCAL muss für die gemeinsame Eingabe aktiviert sein. Verwenden Sie hierzu den folgenden MQSC-Befehl:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- Ihr Warteschlangenmanager muss über eine aktivierte Warteschlange für nicht zustellbare Nachrichten verfügen. Falls ein ConnectionConsumer beim Einreihen einer Nachricht in die Warteschlange für nicht zustellbare Nachrichten ein Problem feststellt, wird die Nachrichtenübermittlung aus der zugrunde liegenden Warteschlange QLOCAL gstoppt. Verwenden Sie für die Definition einer Warteschlange für nicht zustellbare Nachrichten folgenden Befehl:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- Der Benutzer, der den ConnectionConsumer ausführt, muss für die Ausführung von MQOPEN mit MQOO_SAVE_ALL_CONTEXT und MQOO_PASS_ALL_CONTEXT berechtigt sein. Sie finden Details hierzu in der IBM MQ-Dokumentation für Ihre jeweilige Plattform.
- Falls nicht erwünschte Nachrichten in der Warteschlange verbleiben, vermindern sie die Systemleistung. Planen Sie daher Ihre Nachrichtenselektoren so, dass ConnectionConsumers zwischen diesen sämtliche Nachrichten aus der Warteschlange entfernen.

Weitere Informationen zu MQSC-Befehlen finden Sie im Abschnitt MQSC-Befehle.

Allgemeine Grundsätze für das Publish/Subscribe-Messaging unter Verwendung von ASF
ConnectionConsumers erhalten Nachrichten zu einem angegebenen Thema (Topic). Ein ConnectionConsumer kann permanent oder nicht permanent sein. Sie müssen angeben, welche Warteschlange oder Warteschlangen das ConnectionConsumer verwendet.

Wenn eine Anwendung einen ConnectionConsumer aus einem TopicConnection-Objekt erstellt, gibt sie ein Topic-Objekt und eine Selektorzeichenfolge an. Der ConnectionConsumer beginnt dann mit dem Empfang von Nachrichten, die mit dem Selektor für das betreffende Thema übereinstimmen. Dazu gehören auch ständige Veröffentlichungen zu dem subkribierten Thema.

Alternativ dazu kann eine Anwendung einen permanenten ConnectionConsumer erstellen, der einem bestimmten Namen zugeordnet ist. Dieser ConnectionConsumer empfängt Nachrichten, die im Thema veröffentlicht wurden, seit der permanente ConnectionConsumer zuletzt aktiv war. Er empfängt alle derartigen Nachrichten, die dem Selektor für das Thema entsprechen. Wenn der ConnectionConsumer jedoch das Vorauslesen verwendet, kann er nicht persistente Nachrichten verlieren, die sich beim Schließen im Clientpuffer befinden.

Wenn sich IBM MQ classes for JMS im Migrationsmodus des IBM MQ-Messaging-Providers befinden, wird für nicht permanente ConnectionConsumer-Subskriptionen eine separate Warteschlange verwendet.Die konfigurierbare CCSUB-Option in der TopicConnectionFactory gibt die zu verwendende Warteschlange an. Normalerweise gibt CCSUB eine einzelne Warteschlange an, die von allen ConnectionConsumers genutzt werden kann, die dieselbe TopicConnectionFactory verwenden. Jeder ConnectionConsumer kann jedoch eine temporäre Warteschlange generieren, indem ein Präfix für den Warteschlangennamen gefolgt von einem Stern (*) angegeben wird.

Wenn sich IBM MQ classes for JMS im Migrationsmodus des IBM MQ-Messaging-Providers befinden, gibt die CCDSUB-Eigenschaft des Themas die Warteschlange an, die für permanente Subskriptionen verwendet werden soll. Auch hier kann es sich um eine bereits vorhandene Warteschlange oder um ein Warteschlangenpräfix gefolgt von einem Stern (*) handeln. Wenn Sie eine bereits vorhandene Warteschlange angeben, verwenden alle permanenten ConnectionConsumers, die das Thema subskribiert haben, diese Warteschlange. Wenn Sie ein von einem Stern (*) gefolgtes Warteschlangennamenspräfix angeben, wird bei der ersten Erstellung eines permanenten ConnectionConsumer mit einem bestimmten Namen eine Warteschlange generiert. Diese Warteschlange wird später wiederverwendet, wenn ein permanenter ConnectionConsumer mit demselben Namen erstellt wird.

Berücksichtigen Sie bei der Einrichtung des IBM MQ-Warteschlangenmanagers die folgenden Punkte:

- Ihr Warteschlangenmanager muss über eine aktivierte Warteschlange für nicht zustellbare Nachrichten verfügen. Falls ein ConnectionConsumer beim Einreihen einer Nachricht in die Warteschlange für nicht

zustellbare Nachrichten ein Problem feststellt, wird die Nachrichtenübermittlung aus der zugrunde liegenden Warteschlange QLOCAL gstoppt. Verwenden Sie für die Definition einer Warteschlange für nicht zustellbare Nachrichten folgenden Befehl:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- Der Benutzer, der den ConnectionConsumer ausführt, muss für die Ausführung von MQOPEN mit MQOO_SAVE_ALL_CONTEXT und MQOO_PASS_ALL_CONTEXT berechtigt sein. Sie finden Details hierzu in der IBM MQ-Dokumentation für Ihre jeweilige Plattform.
- Sie können die Leistung für einen einzelnen ConnectionConsumer optimieren, indem Sie eine separate dedizierte Warteschlange für ihn erstellen. Hierfür werden jedoch zusätzliche Ressourcen belegt.

Nachrichten aus der Warteschlange in ASF entfernen
Wenn eine Anwendung ConnectionConsumers verwendet, muss JMS in bestimmten Situationen möglicherweise Nachrichten aus der Warteschlange entfernen.

Dabei handelt es sich um folgende Situationen:

Fehlerhaft formatierte Nachricht
Es kann eine Nachricht eintreffen, die nicht von JMS analysiert werden kann.

Nicht verarbeitbare Nachricht
Möglicherweise erreicht eine Nachricht den Rücksetzschwellenwert, der ConnectionConsumer kann sie jedoch nicht neu in die Rücksetzwarteschlange einreihen.

Kein interessierter ConnectionConsumer
Wenn die QueueConnectionFactory beim Punkt-zu-Punkt-Messaging so festgelegt ist, dass sie nicht erwünschte Nachrichten nicht behält, trifft eine Nachricht ein, die von keinem der ConnectionConsumers gewünscht wird.

In diesen Situationen versucht der ConnectionConsumer, die Nachricht aus der Warteschlange zu entfernen. Die Dispositionsoptionen im Berichtsfeld des MQMD der Nachricht bestimmen das genaue Verhalten. Diese Optionen sind:

MQRO_DEAD_LETTER_Q
Die Nachricht wird neu in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers eingereiht. Dies ist die Standardeinstellung.

MQRO_DISCARD_MSG
Die Nachricht wird gelöscht.

Der ConnectionConsumer generiert auch eine Berichtsnachricht. Dieses Verhalten hängt ebenfalls vom Berichtsfeld des MQMD der Nachricht ab. Diese Nachricht wird an die Antwortwarteschlange (ReplyToQ) im ReplyToQmgr gesendet. Falls beim Senden der Berichtsnachricht ein Fehler auftritt, wird die Nachricht stattdessen an die Warteschlange für nicht zustellbare Nachrichten gesendet. Die Ausnahmeberichtsoptionen im Berichtsfeld des MQMD der Nachricht legen Details der Berichtsnachricht fest. Diese Optionen sind:

MQRO_EXCEPTION
Eine Berichtsnachricht wird generiert, die den MQMD der ursprünglichen Nachricht enthält. Sie enthält keinen Nachrichteninhalt.

MQRO_EXCEPTION_WITH_DATA
Eine Berichtsnachricht wird generiert, die den MQMD, alle MQ-Header und 100 Byte der Nachrichtendaten enthält.

MQRO_EXCEPTION_WITH_FULL_DATA
Eine Berichtsnachricht wird generiert, die alle Daten aus der ursprünglichen Nachricht enthält.

Standard
Es wird keine Berichtsnachricht generiert.

Bei der Generierung von Berichtsnachrichten werden die folgenden Optionen berücksichtigt:

- MQRO_NEW_MSG_ID

- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Wenn eine nicht verarbeitbare Nachricht nicht neu eingereiht werden kann, da möglicherweise die Warteschlange für nicht zustellbare Nachrichten voll oder die Berechtigung falsch angegeben ist, hängt die jeweilige Aktion von der Persistenz der Nachricht ab. Wenn die Nachricht nicht persistent ist, wird die Nachricht gelöscht und es wird keine Berichtsnachricht generiert. Wenn die Nachricht persistent ist, wird die Nachrichtenübermittlung an alle Verbindungskonsumenten gestoppt, die an dem betreffenden Ziel empfangsbereit sind. Diese Verbindungskonsumenten müssen geschlossen werden. Das Problem muss gelöst werden, bevor sie wieder erstellt werden können und ein Neustart der Nachrichtenübermittlung möglich ist.

Sie müssen unbedingt eine Warteschlange für nicht zustellbare Nachrichten definieren und diese regelmäßig prüfen, um sicherzustellen, dass keine Probleme auftreten. Vor allem müssen Sie darauf achten, dass die Warteschlange für nicht zustellbare Nachrichten ihre maximale Tiefe nicht erreicht. Außerdem muss ihre maximale Nachrichtengröße für alle Nachrichten ausreichen.

Wenn eine Nachricht neu in die Warteschlange für nicht zustellbare Nachrichten eingereiht wird, wird ihr ein IBM MQ-Header einer nicht zustellbaren Nachricht (MQDLH) vorangestellt. Im Abschnitt MQDLH - Header einer nicht zustellbaren Nachricht finden Sie Details zum Format des MQDLH. Sie können mithilfe der folgenden Felder Nachrichten identifizieren, die von einem ConnectionConsumer in die Warteschlange für nicht zustellbare Nachrichten eingereiht wurden, oder Berichtsnachrichten, die von einem ConnectionConsumer generiert wurden:

- PutApplType ist `MQAT_JAVA (0x1C)`
- PutApplName ist "`MQ JMS ConnectionConsumer`"

Diese Felder befinden sich im MQDLH der Nachrichten in der Warteschlange für nicht zustellbare Nachrichten sowie im MQMD von Berichtsnachrichten. Das Feedback-Feld des MQMD und das Ursachenfeld des MQDLH enthalten einen Code, der den Fehler beschreibt. Details zu diesen Codes finden Sie in „Ursachen- und Feedbackcodes in ASF" auf Seite 355. Die übrigen Felder entsprechen der Beschreibung im Abschnitt MQDLH - Header einer nicht zustellbaren Nachricht.

Behandlung nicht verarbeitbarer Nachrichten in ASF
Innerhalb der Anwendungsserverfunktionen (Application Server Facilities, ASF) weicht die Behandlung nicht verarbeitbarer Nachrichten geringfügig von der sonstigen Vorgehensweise in IBM MQ classes for JMS ab.

Sie finden Informationen zur Behandlung nicht verarbeitbarer Nachrichten in IBM MQ classes for JMS im Abschnitt „Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln" auf Seite 244.

Wenn Sie Anwendungsserverfunktionen (Application Server Facilities, ASF) verwenden, werden nicht verarbeitbare Nachrichten nicht vom MessageConsumer, sondern vom ConnectionConsumer verarbeitet. Der ConnectionConsumer reiht Nachrichten in Übereinstimmung mit den Eigenschaften 'BackoutThreshold' und 'BackoutRequeueQName' der Warteschlange neu ein.

Wenn eine Anwendung ConnectionConsumers verwendet, hängen die Umstände, unter denen eine Nachricht zurückgesetzt wird, von der Sitzung ab, die der Anwendungsserver bereitstellt:

- Wenn die Sitzung nicht transaktionsbasiert und AUTO_ACKNOWLEDGE oder DUPS_OK_ACKNOWLEDGE festgelegt ist, wird eine Nachricht nur nach einem Systemfehler oder einer unerwarteten Beendigung der Anwendung zurückgesetzt.
- Wenn die Sitzung transaktionsbasiert und CLIENT_ACKNOWLEDGE festgelegt ist, können unbestätigte Nachrichten vom Anwendungsserver mit dem Aufruf von Session.recover() zurückgesetzt werden.

  Normalerweise ruft die Clientimplementierung von MessageListener oder der Anwendungsserver Message.acknowledge() auf. Message.acknowledge() bestätigt alle Nachrichten, die bisher in der Sitzung übermittelt wurden.
- Wenn die Sitzung transaktionsbasiert ist, können unbestätigte Nachrichten vom Anwendungsserver mit dem Aufruf von Session.rollback() zurückgesetzt werden.

- Wenn der Anwendungsserver eine XASession bereitstellt, werden Nachrichten abhängig von einer verteilten Transaktion festgeschrieben oder zurückgesetzt. Der Anwendungsserver ist für den Abschluss der Transaktion zuständig.

Zugehörige Konzepte

„Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln" auf Seite 244
Eine nicht verarbeitbare Nachricht ist eine Nachricht, die von einer empfangenden Anwendung nicht verarbeitet werden kann. Wenn eine nicht verarbeitbare Nachricht an eine Anwendung übermittelt und eine bestimmte Anzahl Male zurückgesetzt wird, können die IBM MQ classes for JMS die Nachricht in eine Rücksetzwarteschlange verschieben.

Fehlerbehandlung

In diesem Abschnitt werden verschiedene Aspekte der Fehlerbehandlung beschrieben, darunter„Wiederherstellung nach Fehlerbedingungen in ASF" auf Seite 355 und „Ursachen- und Feedbackcodes in ASF" auf Seite 355.

Wiederherstellung nach Fehlerbedingungen in ASF
Wenn ein ConnectionConsumer einen schwerwiegenden Fehler feststellt, wird die Nachrichtenübermittlung an alle ConnectionConsumers, die an derselben QLOCAL interessiert sind, gestoppt. In diesem Fall wird jeder ExceptionListener, der bei der betroffenen Verbindung registriert ist, benachrichtigt. Es gibt zwei Möglichkeiten für die Wiederherstellung einer Anwendung nach diesen Fehlerbedingungen.

In der Regel tritt ein schwerwiegender Fehler dieses Typs auf, wenn der ConnectionConsumer eine Nachricht nicht in die Warteschlange für nicht zustellbare Nachrichten neu einreihen kann oder wenn er beim Lesen von Nachrichten aus der QLOCAL einen Fehler feststellt.

Da jeder ExceptionListener, der bei der betroffenen Verbindung registriert ist, benachrichtigt wird, können Sie mit seiner Hilfe die Ursache des Problems ermitteln. In manchen Fällen muss der Systemadministrator eingreifen, um das Problem zu beheben.

Verwenden Sie eines der folgenden Verfahren, um eine Wiederherstellung nach diesen Fehlerbedingungen zu erreichen:

- Rufen Sie `close()` für alle betroffenen ConnectionConsumers auf. Die Anwendung kann neue ConnectionConsumers erst dann erstellen, wenn alle betroffenen ConnectionConsumers geschlossen und vorhandene Systemfehler behoben wurden.
- Rufen Sie `stop()` für alle betroffenen Verbindungen (Connections) auf. Sobald alle Verbindungen gestoppt und vorhandene Systemfehler behoben wurden, kann die Anwendung ihre Verbindungen mit `start()` erfolgreich starten.

Ursachen- und Feedbackcodes in ASF
Mithilfe der Ursachen- und Feedbackcodes können Sie die Ursache eines Fehlers feststellen. An dieser Stelle werden gängige Ursachencodes genannt, die vom ConnectionConsumer generiert werden.

Mithilfe der folgenden Informationen können Sie die Ursache eines Fehlers feststellen:

- Feedbackcode in vorhandenen Berichtsnachrichten
- Ursachencode im MQDLH von Nachrichten in der Warteschlange für nicht zustellbare Nachrichten

ConnectionConsumers generieren die folgenden Ursachencodes.

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Ursache
Die Nachricht hat den Rücksetzschwellenwert erreicht, der in der QLOCAL definiert ist, es ist jedoch keine Rücksetzwarteschlange definiert.

Die Nachricht hat auf Plattformen, auf denen Sie die Rücksetzwarteschlange nicht definieren können, den JMS-definierten Rücksetzschwellenwert 20 erreicht.

Action
Falls dies nicht gewünscht wird, definieren Sie die Rücksetzwarteschlange für die relevante QLOCAL. Suchen Sie außerdem nach der Ursache für die mehrfachen Rücksetzungen.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Ursache
Beim Punkt-zu-Punkt-Messaging gibt es eine Nachricht, die keinem der Selektoren für die ConnectionConsumers entspricht, die die Warteschlange überwachen. Aus Leistungsgründen wird die Nachricht neu in die Warteschlange für nicht zustellbare Nachrichten eingereiht.

Action
Stellen Sie zur Vermeidung dieser Situation sicher, dass ConnectionConsumers, die die Warteschlange verwenden, eine Gruppe von Selektoren bereitstellen, die alle Nachrichten handhaben, oder legen Sie für die QueueConnectionFactory fest, dass Nachrichten behalten werden.

Untersuchen Sie alternativ dazu die Nachrichtenquelle.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Ursache
JMS kann die Nachricht in der Warteschlange nicht interpretieren.

Action
Überprüfen Sie die Herkunft der Nachricht. JMS übermittelt Nachrichten in einem nicht erwarteten Format normalerweise als BytesMessage oder TextMessage. Wenn die Nachricht sehr fehlerhaft formatiert ist, schlägt dies gelegentlich fehl.

Sonstige Codes, die in diesem Feldern angezeigt werden, sind auf einen fehlgeschlagenen Versuch der Neueinreihung der Nachricht in eine Rücksetzwarteschlange zurückzuführen. In dieser Situation beschreibt der Code den Grund der fehlgeschlagenen Neueinreihung. Informationen zur Diagnose der Fehlerursache finden Sie unter API > API-Beendigungs-und Ursachencodes.

Wenn die Berichtsnachricht nicht in die ReplyToQ eingereiht werden kann, wird sie in die Warteschlange für nicht zustellbare Nachrichten eingereiht. In dieser Situation wird das Feedback-Feld des MQMD wie in diesem Abschnitt beschrieben ausgefüllt. Im Ursachenfeld des MQDLH wird erläutert, weshalb die Berichtsnachricht nicht in die ReplyToQ eingereiht werden konnte.

Funktion eines Serversitzungspools in AFS

Dieser Abschnitt enthält eine Zusammenfassung der Funktion eines Serversitzungspools.

In Abbildung 45 auf Seite 357 werden die Grundsätze der ServerSessionPool- und ServerSession-Funktionalität zusammengefasst.

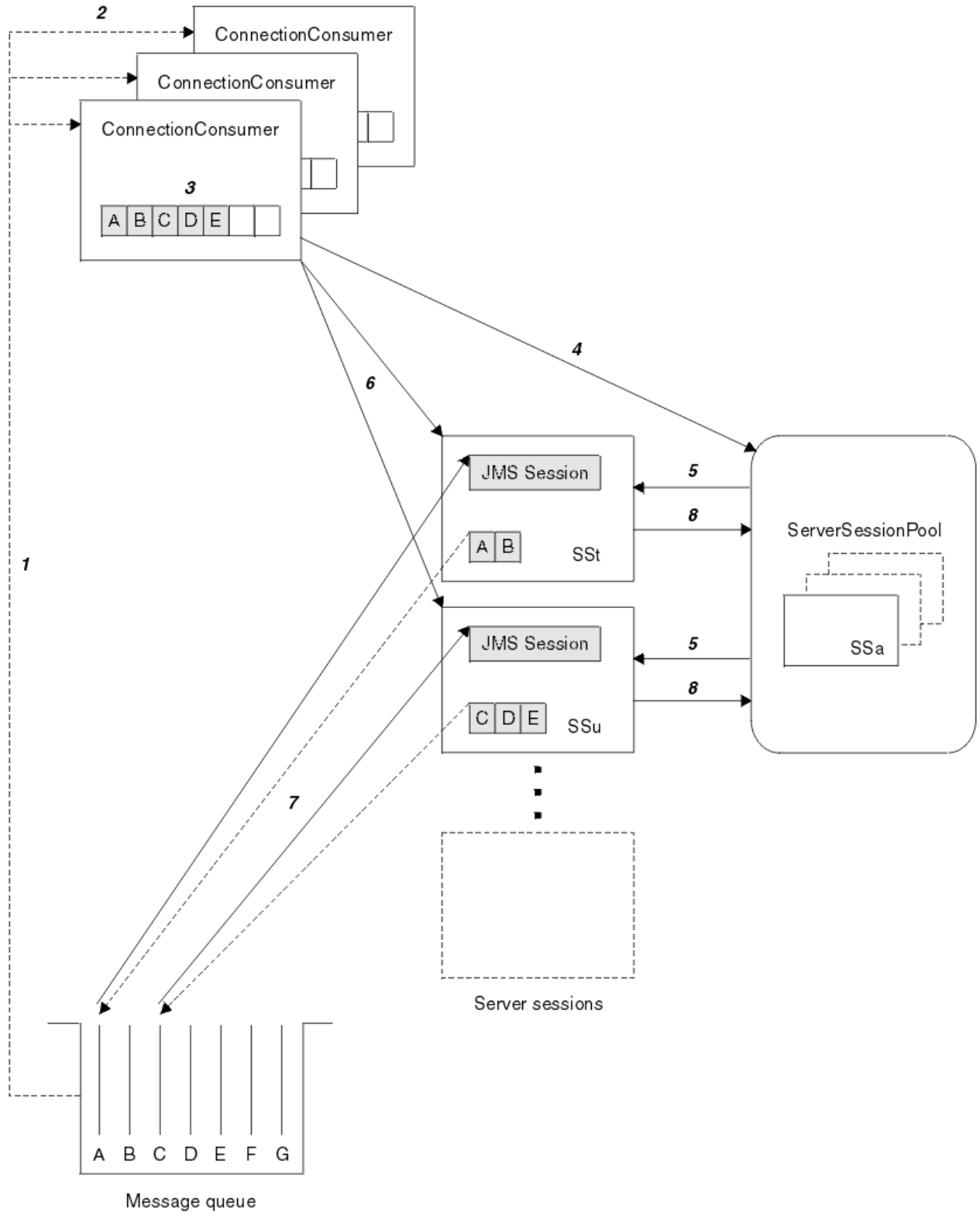


Abbildung 45. ServerSessionPool- und ServerSession-Funktionalität

1. Die ConnectionConsumers rufen Nachrichtenreferenzen aus der Warteschlange ab.
2. Jeder ConnectionConsumer wählt bestimmte Nachrichtenreferenzen aus.
3. Der ConnectionConsumer-Puffer enthält die ausgewählten Nachrichtenreferenzen.
4. Der ConnectionConsumer fordert eine oder mehrere ServerSessions aus dem ServerSessionPool an.

5. ServerSessions werden vom ServerSessionPool zugeordnet.
6. Der ConnectionConsumer weist den ServerSessions Nachrichtenreferenzen zu und startet die aktiven ServerSession-Threads.
7. Jede ServerSession ruft ihre referenzierten Nachrichten aus der Warteschlange ab. Sie übergibt sie über den MessageListener , der der JMS -Sitzung zugeordnet ist, an die Methode onMessage .
8. Nach Abschluss der zugehörigen Verarbeitung wird die ServerSession an den Pool zurückgegeben.

Ein Anwendungsserver stellt normalerweise die ServerSessionPool- und ServerSession-Funktionalität bereit.

z/OS Using IBM MQ classes for JMS in a CICS Liberty JVM server

Java programs running in a CICS Liberty JVM server can use the IBM MQ classes for JMS to access IBM MQ.

You must be using an IBM MQ 9.1.0 or later version of the IBM MQ resource adapter. You can obtain the resource adapter from Fix Central (see “Ressourcenadapter in Liberty installieren” on page 467).

There are two flavors of Liberty Profile JVMs available in CICS 5.3 and later, the types of connection possible to IBM MQ are restricted as follows:

CICS Liberty Standard

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in CLIENT mode
- The IBM MQ resource adapter can connect to any in-service version of IBM MQ for z/OS in BINDINGS mode when there is no CICS connection (active CICS MQCONN resource definition) to the same queue manager from the same CICS region.

CICS Liberty Integrated

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in CLIENT mode.
- BINDINGS mode connection is not supported.

For details on setting up and configuring your system, see Using IBM MQ classes for JMS in a Liberty JVM server in the CICS documentation.

IBM MQ classes for JMS/ Jakarta Messaging in IMS verwenden

Standardbasierte Messaging-Unterstützung in einer IMS -Umgebung wird durch die Verwendung von IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingbereitgestellt.

Überprüfen Sie die Systemvoraussetzungen für das IMS-System, das Ihr Unternehmen verwendet. Weitere Informationen finden Sie unter IMS 15.2.

In diesen Abschnitten wird beschrieben, wie die IBM MQ classes for JMS in einer IMS-Umgebung eingerichtet werden und welche API-Einschränkungen bei Verwendung der klassischen (JMS 1.1) und vereinfachten (JMS 2.0) Schnittstellen gelten. Eine Liste der API-spezifischen Informationen finden Sie im Abschnitt „JMS-API-Einschränkungen” auf Seite 363.

Anmerkung: Ähnliche Einschränkungen gelten für die traditionellen (JMS 1.0.2) domänenspezifischen Schnittstellen, die hier jedoch nicht speziell beschrieben werden.

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.

Unterstützte IMS-abhängige Regionen

Folgende abhängige Regionstypen werden unterstützt:

- MPR

- BMP
- IFP
- JMP 31 und 64-Bit-JVMs (Java Virtual Machines)
- JBP 31 und 64-Bit-JVMs

Sofern in den folgenden Abschnitten nicht ausdrücklich erwähnt, verhalten sich IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging in allen Regionstypen gleich.

Unterstützte Java Virtual Machines

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging erfordern IBM Runtime Environment, Java Technology Edition 8. IBM Semeru Runtime Certified Edition for z/OS Version 11 wird nicht unterstützt.

Sonstige Einschränkungen

Bei Verwendung der IBM MQ classes for JMS in einer IMS-Umgebung gelten folgende Einschränkungen:

- Verbindungen im Clientmodus werden nicht unterstützt.
- Verbindungen zu IBM MQ 8.0-Warteschlangenmanagern werden nur im Modus IBM MQ Messaging-Provider `Normal` unterstützt.

  Das Attribut **PROVIDERVERSION** in der Verbindungsfactory darf nicht angegeben werden oder einen Wert größer-gleich sieben haben.
- Der Gebrauch jedweder XA Verbindungsfactories, zum Beispiel `com.ibm.mq.jms.MQXAConnectionFactory`, wird nicht unterstützt.

Zugehörige Tasks

IBM MQ für IMS definieren

IMS -Adapter für die Verwendung mit IBM MQ classes for JMS/Jakarta Messaging einrichten

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging verwenden denselben IBM MQ-IMS -Adapter wie andere Programmiersprachen. Dieser Adapter verwendet die IMS External Subsystem Attach Facility (ESAF).

Vorbereitende Schritte

Bevor Sie die folgende Prozedur ausführen, müssen Sie den IMS-Adapter für die relevanten Warteschlangenmanager sowie die IMS-Steuerung und die abhängigen Regionen konfigurieren, so wie im Abschnitt IMS-Adapter einrichten beschrieben.

Achtung: Sie müssen den Schritt, in dem die Erstellung eines dynamischen Stubs beschrieben wird, nicht ausführen, es sei denn, Sie benötigen den dynamischen Stub für andere Zwecke.

Führen Sie nach der Konfiguration des IMS -Adapters die folgende Prozedur durch.

Vorgehensweise

1. Aktualisieren Sie die Variable LIBPATH im Member Ihres IMS PROCLIB, auf die der Parameter ENVIRON in der JCL der abhängigen Region verweist (z. B. DFSJVMEV), sodass sie die nativen Bibliotheken der IBM MQ classes for JMS enthält.

   Dies ist das zFS-Verzeichnis, das `libmqjims.so` enthält. Beispiel: DFSJVMEV könnte wie folgt aussehen, wobei die letzte Zeile das Verzeichnis ist, das die nativen Bibliotheken IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging enthält:

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
```

```
/ims/latest/dbdc/imsjava/classic/lib:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. Fügen Sie IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging zum Klassenpfad der JVM hinzu, die von Ihrer abhängigen IMS -Region verwendet wird, indem Sie die Option `java.class.path` aktualisieren.

   Folgen Sie dazu den Anweisungen im Abschnitt DFSJVMMS member of the IMS PROCLIB data set.

   Sie können beispielsweise Folgendes verwenden, wobei die Zeile in Fettdruck die Aktualisierung angibt:

JM 3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

   Anmerkung: Obwohl in dem Verzeichnis, das IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingenthält, viele verschiedene JAR-Dateien verfügbar sind, benötigen Sie nur `com.ibm.mq.allclient.jar` (JMS 2.0) oder `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0).

3. Stoppen Sie alle abhängigen IMS -Regionen, die IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messagingverwenden, und starten Sie sie erneut.

Nächste Schritte

Erstellen und konfigurieren Sie Verbindungsfactorys und Ziele.

Es gibt drei mögliche Methoden, die IBM MQ-Implementierungen von Verbindungsfactorys und Zielen zu instanziieren. Details finden Sie im Abschnitt „Verbindungsfactorys und Ziele erstellen und konfigurieren” auf Seite 214.

Beachten Sie, dass diese drei Methoden alle in einer IMS-Umgebung gültig sind.

Zugehörige Konzepte

IMS-Adapter einrichten

IBM MQ für IMS definieren

Transaktionsverhalten

Nachrichten, die von IBM MQ classes for JMS in einer IMS -Umgebung gesendet und empfangen werden, sind immer der IMS -Arbeitseinheit zugeordnet, die in der aktuellen Aufgabe aktiv ist.

Diese UOW kann nur durch einen Commit- oder Rollback-Aufruf an einer Instanz des `com.ibm.ims.dli.tm.Transaction`-Objekts oder durch eine normale Beendigung der IMS-Task abgeschlossen werden, in welchem Fall die UOW implizit durch ein Commit festgeschrieben wird. Bei einer abnormalen Beendigung der IMS-Task wird die UOW durch ein Rollback zurückgesetzt.

Folglich werden die Werte der Argumente **`transacted`** und **`acknowledgeMode`** ignoriert, wenn die Methoden `Connection.createSession` oder `ConnectionFactory.createContext` aufgerufen werden. Die folgenden Methoden werden zudem nicht unterstützt. Bei Aufruf einer der folgenden Methoden kommt es bei einer Sitzung zu `IllegalStateException`:

- `javax.jms.Session.commit()`
- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

sowie `IllegalStateRuntimeSession` bei JMS 'Context':

- `javax.jms.JMSContext.commit()`

- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`

Es gibt eine Ausnahme von diesem Verhalten. Wenn eine Sitzung oder ein JMS-Kontext über einen der folgenden Mechanismen erstellt wird:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

sieht das Verhalten dieser Sitzung bzw. dieses JMS-Kontexts wie folgt aus:

- Sämtliche gesendeten Nachrichten werden außerhalb der IMS-UOW übertragen. Sie sind also sofort oder nach Ablauf der angegebenen Zustellungsverzögerung auf dem Ziel verfügbar.
- Alle nicht persistenten Nachrichten werden außerhalb der IMS -Arbeitseinheit empfangen, sofern die Eigenschaft SYNCPOINTALLGETS nicht in der Verbindungsfactory angegeben wurde, die die Sitzung oder den JMS -Kontext erstellt hat.
- Persistente Nachrichten werden immer innerhalb der IMS-UOW empfangen.

Dies kann zum Beispiel nützlich sein, wenn Sie trotz eines Rollbacks der UOW eine Prüfnachricht in eine Warteschlange schreiben wollen.

Auswirkungen von IMS-Synchronisationspunkten

Die IBM MQ classes for JMS bauen auf der vorhandenen IBM MQ-Adapterunterstützung auf, die ESAF verwendet. Dies bedeutet, dass das dokumentierte Verhalten gilt, einschließlich aller offenen Handles, die vom IMS-Adapter geschlossen werden, wenn ein Synchronisationspunkt auftritt.

z/OS Weitere Informationen finden Sie unter „Syncpoints in IMS applications” auf Seite 74.

Zur Veranschaulichung dieses Punkts soll der folgende Code dienen, der in einer JMP-Umgebung ausgeführt wird. Der zweite Aufruf von `mp.send()` führt zu einer `JMSException`, da der Code `messageQueue.getUnique(inputMessage)` zur Folge hat, dass alle offenen Verbindungs- und Objekthandles von IBM MQ geschlossen werden.

Ein ähnliches Verhalten ist zu beobachten, wenn der `getUnique()`-Aufruf durch `Transaction.commit()` ersetzt wurde, aber nicht, wenn `Transaction.rollback()` verwendet wurde.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

Der richtige Code für dieses Szenario folgt unten. In diesem Fall wird die Verbindung zu IBM MQ geschlossen, bevor `getUnique()` aufgerufen wird. Verbindung und Sitzung werden dann neu aufgebaut, um eine weitere Nachricht zu senden.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

Hinweise zur Verwendung des IMS-Adapters

Die folgenden Einschränkungen sind zu beachten. Es kann für jeden Warteschlangenmanager nur ein Verbindungshandle geben. Es hat Auswirkungen auf die Interaktion mit IBM MQ, wenn sowohl JMS- als auch nativer Code verwendet werden. Es gibt Einschränkungen bei der Verbindungsauthentifizierung und -autorisierung.

Nur ein Verbindungshandle für jeden Warteschlangenmanager

In IMS-abhängigen Regionen ist immer nur ein Verbindungshandle zu einem bestimmten Warteschlangenmanager zulässig. Bei allen nachfolgenden Versuchen, eine Verbindung zu demselben Warteschlangenmanager herzustellen, wird das vorhandene Handle verwendet.

Dieses Verhalten sollte zwar in einer Anwendung, die nur die IBM MQ classes for JMS verwendet, keine Probleme verursachen, kann dies jedoch in Anwendungen tun, die mit IBM MQ interagieren, wenn sowohl die IBM MQ classes for JMS als auch MQI in einem nativen Code, der in Sprachen wie COBOL oder C geschrieben ist, verwendet werden.

Auswirkungen auf Interaktion mit IBM MQ bei gleichzeitiger Verwendung von JMS und nativem Code

Probleme können auftreten, wenn Java -Code und nativen Code, die beide die IBM MQ -Funktionalität verwenden, verzahnt werden und die Verbindung zu IBM MQ nicht geschlossen wird, bevor entweder der native Code oder der Java-Code verlassen wird.

Im folgenden Pseudocode wird beispielsweise eine Verbindungskennung zu einem Warteschlangenmanager ursprünglich im Java -Code mithilfe von IBM MQ classes for JMSeingerichtet. Das Verbindungshandle wird im COBOL-Code wiederverwendet und durch einen Aufruf von MQDISC ungültig gemacht.

Wenn die IBM MQ classes for JMS das Verbindungshandle das nächste Mal verwenden, führt dies zu einer JMSException mit Ursachencode MQRC_HCONN_ERROR.

```
COBOL code running in message processing region
    Use the Java Native Interface (JNI) to call Java code
       Create MQ connection and session - this creates an MQ connection handle
       Send message to MQ queue
       Store connection and session in static variable
       Return to COBOL code

    MQCONN - picks up MQ connection handle established in Java code
    MQDISC - invalidates connection handle

    Use the Java Native Interface (JNI) to call Java code
       Get session from static variable
       Create a message consumer - fails as connection handle invalidated
```

Es gibt andere ähnliche Verwendungsmuster, die zu MQRC_HCONN_ERROR führen können.

Es ist zwar möglich, IBM MQ -Verbindungskennungen zwischen nativem Code und Java -Code gemeinsam zu nutzen (das vorherige Beispiel würde beispielsweise funktionieren, wenn kein MQDISC-Aufruf vorhanden wäre), aber es hat sich bewährt, alle Verbindungskennungen zu schließen, bevor von Java auf nativen Code oder umgekehrt gewechselt wird.

Verbindungsauthentifizierung und -autorisierung

Gemäß der JMS-Spezifikation ist es möglich, beim Erstellen einer Verbindung oder eines JMS-Kontextobjekts zur Authentifizierung und Autorisierung einen Benutzernamen und ein Kennwort anzugeben.

Dies wird in einer IMS-Umgebung nicht unterstützt. Der Versuch, eine Verbindung zu erstellen, während ein Benutzername und ein Kennwort angegeben werden, führt dazu, dass ein JMS Exception ausgelöst wird. Beim Versuch, einen JMS-Kontext zu erstellen und dabei einen Benutzernamen und ein Kennwort anzugeben, wird eine Laufzeitausnahme (JMSRuntimeException) ausgelöst.

Stattdessen müssen bei der Herstellung einer Verbindung mit IBM MQ aus einer IMS-Umgebung bestehende Verfahren zur Authentifizierung und Autorisierung verwendet werden.

Weitere Informationen finden Sie im Abschnitt Sicherheit unter z/OS einrichten. Besonders wichtig ist der Abschnitt Benutzer-IDs für Sicherheitsprüfung, in dem die verwendbaren Benutzer-IDs beschrieben werden.

Zugehörige Tasks

Sicherheit unter z/OS einrichten

JMS-API-Einschränkungen

Aus der Perspektive der JMS -Spezifikation gilt IBM MQ classes for JMS treat IMS als Java EE oder Jakarta EE -konformer Anwendungsserver, für den immer eine JTA-Transaktion in Bearbeitung ist.

So können Sie beispielsweise in IMS nie javax.jms.Session.commit() aufrufen, weil es in der JMS-Spezifikation heißt, dass ein Aufruf in einer JEE-EJB oder einem Web-Container nicht möglich ist, solange eine JTA-Transaktion in Bearbeitung ist.

Daraus ergeben sich für die JMS-API neben den im Abschnitt „Transaktionsverhalten” auf Seite 360 beschriebenen Beschränkungen folgende Einschränkungen.

Einschränkungen der klassischen API

- javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int) löst immer eine JMSExceptionaus.
- javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) löst immer eine JMSExceptionaus.
- Alle drei Varianten von javax.jms.Connection.createSession lösen immer eine JMSException aus, wenn bereits eine vorhandene Sitzung für die Verbindung aktiv ist.

- `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` löst immer eine `JMSException`aus.
- `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` löst immer eine `JMSException`aus.
- `javax.jms.Connection.setClientID()` löst immer eine `JMSException`aus.
- `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` löst immer eine `JMSException`aus.
- `javax.jms.Connection.stop()` löst immer eine `JMSException`aus.
- `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` löst immer eine `JMSException`aus.
- `javax.jms.MessageConsumer.getMessageListener()` löst immer eine `JMSException`aus.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` löst immer eine `JMSException`aus.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` löst immer eine `JMSException`aus.
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` löst immer eine `JMSException`aus.
- `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener)` löst immer eine `JMSException`aus.
- `javax.jms.Session.run()` löst immer eine `JMSRuntimeException`aus.
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` löst immer eine `JMSException`aus.
- `javax.jms.Session.getMessageListener()` löst immer eine `JMSException`aus.

Einschränkungen der vereinfachten API

- `javax.jms.JMSContext.createContext(int)` löst immer eine `JMSRuntimeException`aus.
- `javax.jms.JMSContext.setClientID(String)` löst immer eine `JMSRuntimeException`aus.
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` löst immer eine `JMSRuntimeException`aus.
- `javax.jms.JMSContext.stop()` löst immer eine `JMSRuntimeException`aus.
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` löst immer eine `JMSRuntimeException`aus.

IBM MQ classes for Java verwenden

Verwenden Sie 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:

Stabilized IBM wird keine weiteren Erweiterungen für die IBM MQ classes for Java durchführen; sie werden auf der in IBM MQ 8.0 ausgelieferten Stufe stabilisiert. Bestehende Anwendungen, die die IBM MQ classes for Java verwenden, werden weiterhin vollständig unterstützt, aber neue Funktionen werden nicht mehr hinzugefügt und Anforderungen für Erweiterungen abgelehnt. Vollständig unterstützt bedeutet, dass Fehler behoben und daraus erforderliche Änderungen an den IBM MQ-Systemvoraussetzungen vorgenommen werden.

Die IBM MQ classes for Java werden in IMS nicht unterstützt.

Die IBM MQ classes for Java werden in WebSphere Liberty nicht unterstützt. Sie dürfen weder mit der Messaging-Funktion von IBM MQ Liberty noch mit der generischen JCA-Unterstützung verwendet werden. Weitere Informationen finden Sie unter Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

IBM MQ classes for Java ist eine von drei alternativen APIs, mit denen Java -Anwendungen auf IBM MQ -Ressourcen zugreifen können. Die anderen APIs sind:

- JM 3.0 IBM MQ classes for Jakarta Messaging
- JMS 2.0 IBM MQ classes for JMS

Weitere Informationen finden Sie unter „Zugreifen auf IBM MQ von Java -Auswahl der API" auf Seite 90.

Ab IBM MQ 9.3 werden die IBM MQ classes for Java mit Java 8 entwickelt. Die Laufzeitumgebung von Java 8 unterstützt die Ausführung früherer Klassendateiversionen.

IBM MQ classes for Java binden die Message Queue Interface (MQI), die native IBM MQ-API und verwenden ein ähnliches Objektmodell für die C++- und .NET-Schnittstellen zu IBM MQ.

Programmierbare Optionen ermöglichen IBM MQ classes for Java das Herstellen einer Verbindung zu IBM MQ auf eine der folgenden Arten:

- Im Clientmodus als IBM MQ MQI client über Transmission Control Protocol/Internet Protocol (TCP/IP)
- Im Bindungsmodus durch Herstellen einer direkten Verbindung mit IBM MQ über Java Native Interface (JNI)

Anmerkung: Die automatische Clientverbindungswiederholung wird von IBM MQ classes for Java nicht unterstützt.

Clientmodusverbindung

Eine IBM MQ classes for Java-Anwendung kann im Clientmodus eine Verbindung zu jedem unterstützten Warteschlangenmanager herstellen.

Um sich im Clientmodus mit einem Warteschlangenmanager zu verbinden, kann eine Anwendung der IBM MQ classes for Java auf demselben System wie der Warteschlangenmanager ausgeführt werden, es ist aber auch die Ausführung auf einem anderen System möglich. In jedem Fall verbinden sich IBM MQ classes for Java über TCP/IP mit dem Warteschlangenmanager.

Weitere Informationen zum Erstellen von Anwendungen, die Verbindungen im Clientmodus verwenden, finden Sie im Abschnitt „IBM MQ classes for Java-Verbindungsmodi" auf Seite 391.

Bindungsmodusverbindung

Im Bindungsmodus verwendet IBM MQ classes for Java die Java Native Interface (JNI), um die vorhandene Warteschlangen-API direkt aufzurufen, anstatt über ein Netzwerk zu kommunizieren. In den meisten Umgebungen bietet das Herstellen einer Verbindung im Bindungsmodus eine bessere Performance für IBM MQ classes for Java-Anwendungen als das Verbinden im Clientmodus durch die Vermeidung der Kosten der TCP/IP-Kommunikation.

Anwendungen, die IBM MQ classes for Java für die Verbindungsherstellung im Bindungsmodus verwenden, müssen auf dem gleichen System wie der Warteschlangenmanager ausgeführt werden, zu dem sie eine Verbindung herstellen.

Die Java Runtime Environment, die für das Ausführen der IBM MQ classes for Java-Anwendung verwendet wird, muss für das Laden der IBM MQ classes for Java-Bibliotheken konfiguriert werden; unter „IBM MQ classes for Java-Bibliotheken" auf Seite 375 erhalten Sie weitere Informationen.

Weitere Informationen zum Erstellen von Anwendungen, die Verbindungen im Bindungsmodus verwenden, finden Sie im Abschnitt „IBM MQ classes for Java-Verbindungsmodi" auf Seite 391.

Zugehörige Konzepte

IBM MQ Java-Sprachenschnittstellen

„IBM MQ classes for JMS/Jakarta Messaging verwenden" auf Seite 85

IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Java -Messaging-Provider, die mit IBM MQbereitgestellt werden. Neben der Implementierung der in den Spezifikationen JMS und

Jakarta Messaging definierten Schnittstellen fügen diese Messaging-Provider zwei Gruppen von Erweiterungen zur Java -Messaging-API hinzu.

Zugehörige Tasks

Tracing von IBM MQ classes for Java-Anwendungen

Fehlerbehebung bei Problemen mit Java und JMS

Gründe für die Verwendung von IBM MQ classes for Java

Eine Java-Anwendung kann entweder IBM MQ classes for Java oder IBM MQ classes for JMS für den Zugriff auf IBM MQ-Ressourcen verwenden.

Anmerkung: Obwohl vorhandene Anwendungen, die IBM MQ classes for Java verwenden, weiterhin vollständig unterstützt werden, sollten neue Anwendungen das IBM MQ classes for Jakarta Messagingverwenden. Funktionen, die kürzlich zu IBM MQhinzugefügt wurden, wie z. B. die asynchrone Verarbeitung und die automatische Verbindungswiederholung, sind in IBM MQ classes for Javanicht verfügbar, aber in IBM MQ classes for JMS und IBM MQ classes for Jakarta Messagingverfügbar. Weitere Informationen hierzu finden Sie in den Abschnitten „Gründe für die Verwendung von IBM MQ classes for JMS" auf Seite 88 und „Gründe für die Verwendung von IBM MQ classes for Jakarta Messaging" auf Seite 87.

Anmerkung:

Stabilized IBM wird keine weiteren Erweiterungen für die IBM MQ classes for Java durchführen; sie werden auf der in IBM MQ 8.0 ausgelieferten Stufe stabilisiert. Bestehende Anwendungen, die die IBM MQ classes for Java verwenden, werden weiterhin vollständig unterstützt, aber neue Funktionen werden nicht mehr hinzugefügt und Anforderungen für Erweiterungen abgelehnt. Vollständig unterstützt bedeutet, dass Fehler behoben und daraus erforderliche Änderungen an den IBM MQ-Systemvoraussetzungen vorgenommen werden.

Die IBM MQ classes for Java werden in IMS nicht unterstützt.

Die IBM MQ classes for Java werden in WebSphere Liberty nicht unterstützt. Sie dürfen weder mit der Messaging-Funktion von IBM MQ Liberty noch mit der generischen JCA-Unterstützung verwendet werden. Weitere Informationen finden Sie unter Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

Zugehörige Konzepte

„Zugreifen auf IBM MQ von Java -Auswahl der API" auf Seite 90
IBM MQ stellt drei Java -Sprachschnittstellen bereit.

Voraussetzungen für IBM MQ classes for Java

Um IBM MQ classes for Java zu verwenden, benötigen Sie bestimmte weitere Softwareprodukte.

Informationen zu den Voraussetzungen für IBM MQ classes for Java finden Sie auf der Webseite mit den Systemvoraussetzungen für IBM MQ.

Um IBM MQ classes for Java-Anwendungen zu entwickeln, benötigen Sie ein Java Development Kit (JDK). Details zu den von Ihrem Betriebssystem unterstützten JDKs finden Sie in den Systemvoraussetzungen für IBM MQ.

Für die Ausführung von Anwendungen der IBM MQ classes for Java benötigen Sie die folgenden Softwarekomponenten:

- Einen IBM MQ-Warteschlangenmanager für Anwendungen, die eine Verbindung zu einem Warteschlangenmanager herstellen
- Eine Java Runtime Environment (JRE) für jedes System, auf dem Sie Anwendungen ausführen. Eine geeignete JRE wird mit IBM MQ bereitgestellt.
- IBM i Für IBM i, QShell - dies ist Option 30 des Betriebssystems
- z/OS Für z/OS, z/OS UNIX System Services (z/OS UNIX)

Wenn TLS-Verbindungen Verschlüsselungsmodule verwenden sollen, die FIPS 140-2-zertifiziert sind, benötigen Sie den IBM Java JSSE FIPS-Provider (IBMJSSEFIPS). Jedes IBM JDK und jede JRE ab Version 1.4.2 enthält IBMJSSEFIPS.

Sie können Adressen von Internet Protocol Version 6 (IPv6) in Ihren IBM MQ classes for Java -Anwendungen verwenden, wenn IPv6 von Ihrer Java Virtual Machine (JVM) und der TCP/IP-Implementierung auf Ihrem Betriebssystem unterstützt wird.

IBM MQ classes for Java-Anwendungen in Java EE ausführen

Es bestehen gewisse Einschränkungen und Designaspekte, die es vor der Verwendung von IBM MQ classes for Java in Java EE zu berücksichtigen gilt.

Für IBM MQ classes for Java gelten innerhalb einer Umgebung mit Java Platform, Enterprise Edition (Java EE) einige Einschränkungen. Für das Design, die Implementierung und die Verwaltung von IBM MQ classes for Java-Anwendungen, die in einer Java EE-Umgebung ausgeführt werden sollen, müssen darüber hinaus einige zusätzliche Aspekte berücksichtigt werden. Diese Einschränkungen und Aspekte werden in den folgenden Abschnitten kurz erläutert.

Einschränkungen bei JTA-Transaktionen

Für Anwendungen mit IBM MQ classes for Java wird als Transaktionsmanager nur IBM MQ selbst unterstützt. Eine Anwendung unter JTA-Steuerung kann zwar IBM MQ classes for Java nutzen, jegliche Bearbeitung, die durch diese Klassen durchgeführt wird, wird jedoch nicht durch die JTA-Arbeitseinheiten gesteuert. Sie bilden stattdessen lokale Arbeitseinheiten, die von denjenigen getrennt sind, die vom Anwendungsserver über die JTA-Schnittstellen verwaltet werden. Insbesondere führt ein Rollback der JTA-Transaktion nicht zu einem Rollback gesendeter oder empfangener Nachrichten. Diese Einschränkung gilt sowohl für Transaktionen, die durch Anwendungen, Beans oder den Container verwaltet werden, als auch für alle Java EE-Container. Wenn eine Messaging-Bearbeitung direkt mit IBM MQ innerhalb von serverkoordinierten Transaktionen der Anwendung durchgeführt werden soll, müssen stattdessen IBM MQ classes for JMS verwendet werden.

Threaderstellung

IBM MQ classes for Java erstellen intern Threads für verschiedene Operationen. Bei einer Ausführung im BINDINGS-Modus zur direkten Ausgabe eines Aufrufs in einem lokalen Warteschlangenmanager erfolgen die Aufrufe beispielsweise in einem 'worker'-Thread, der intern von den IBM MQ classes for Java erstellt wird. Darüber hinaus können weitere Threads intern erstellt werden, beispielsweise um nicht verwendete Verbindungen aus einem Verbindungspool zu löschen oder um Subskriptionen für beendete Publish/Subscribe-Anwendungen zu entfernen.

Einige Java EE-Anwendungen (beispielsweise Anwendungen, die in EJB- und Web-Containern ausgeführt werden) dürfen keine neuen Threads erstellen. Stattdessen muss die gesamte Bearbeitung in den Hauptanwendungsthreads stattfinden, die vom Anwendungsserver verwaltet werden. Wenn Anwendungen IBM MQ classes for Java verwenden, ist der Anwendungsserver möglicherweise nicht in der Lage, zwischen dem Anwendungscode und dem Code der IBM MQ classes for Java zu unterscheiden. Daher führen die zuvor beschriebenen Threads dazu, dass die Anwendung nicht mit der Containerspezifikation konform ist. Da IBM MQ classes for JMS diese Java EE-Spezifikationen nicht verletzen, können sie stattdessen verwendet werden.

Sicherheitseinschränkungen

Die von einem Anwendungsserver implementierten Sicherheitsrichtlinien verhindern möglicherweise bestimmte Operationen, die von der API der IBM MQ classes for Java ausgeführt werden. Beispiele hierfür sind die Erstellung und Ausführung neuer Steuerungsthreads (wie in den vorhergehenden Abschnitten beschrieben).

Anwendungsserver werden beispielsweise in der Regel standardmäßig mit inaktivierter Java-Sicherheit ausgeführt. Sie kann über eine anwendungsserverspezifische Konfiguration aktiviert werden (manche Anwendungsserver ermöglichen eine detailliertere Konfiguration der in der Java-Sicherheit verwendeten

Richtlinien). Wenn die Java-Sicherheit aktiviert ist, können die IBM MQ classes for Java die für den Anwendungsserver definierten Threading-Richtlinienregeln der Java-Sicherheit verletzen und die API kann unter Umständen nicht alle Threads erstellen, die sie für eine ordnungsgemäße Funktionsweise benötigt. Zur Vermeidung von Problemen beim Thread-Management wird die Verwendung von IBM MQ classes for Java in Umgebungen mit aktivierter Java-Sicherheit nicht unterstützt.

Hinweise zur Anwendungsisolierung

Ein beabsichtigter Vorteil der Ausführung von Anwendungen in einer Java EE-Umgebung ist die Isolierung der Anwendungen. Das Design und die Implementierung von IBM MQ classes for Java vor der Java EE -Umgebung. IBM MQ classes for Java können so verwendet werden, dass das Konzept der Anwendungsisolierung nicht unterstützt wird. In diesem Bereich müssen beispielsweise die folgenden speziellen Aspekte berücksichtigt werden:

- Die Verwendung von statischen Einstellungen (diese gelten für den gesamten JVM-Prozess) innerhalb der MQEnvironment-Klasse, beispielsweise:
  - Die Benutzer-ID und das Kennwort, die für die Verbindungsidentifikation und Authentifizierung verwendet werden sollen
  - Der Hostname, der Port und der Kanal, die für Clientverbindung verwendet werden
  - Die TLS-Konfiguration für sichere Clientverbindungen

  Wenn eine der MQEnvironment-Eigenschaften zugunsten einer Anwendung geändert wird, wirkt sich dies auch auf andere Anwendungen aus, die dieselben Eigenschaften verwenden. Bei der Ausführung in einer Umgebung mit mehreren Anwendungen wie in Java EE muss jede Anwendung ihre eigene spezielle Konfiguration verwenden. Dies wird durch die Erstellung von MQQueueManager-Objekten mit einem jeweils eigenem Eigenschaftensatz erreicht, anstatt die Standardeigenschaften der prozessweiten MQEnvironment-Klasse zu übernehmen.
- Die MQEnvironment-Klasse führt verschiedene statische Methoden ein, die global Aktionen für alle Anwendungen mit IBM MQ classes for Java innerhalb desselben JVM-Prozesses ausführen. Dieses Verhalten kann grundsätzlich nicht für bestimmte Anwendungen außer Kraft gesetzt werden. Beispiele hierfür sind:
  - die Konfiguration von TLS-Eigenschaften (beispielsweise die Position des Keystore)
  - die Konfiguration von Clientkanalexits
  - die Aktivierung oder Inaktivierung des Diagnosetracings
  - die Verwaltung des Standardverbindungspools, der für die Optimierung der Nutzung von Verbindungen mit Warteschlangenmanagern verwendet wird

  Der Aufruf solcher Methoden wirkt sich auf alle Anwendungen aus, die in derselben Java EE-Umgebung ausgeführt werden.
- Das Verbindungspooling wird aktiviert, um den Prozess der Herstellung mehrerer Verbindungen zu demselben Warteschlangenmanager zu optimieren. Der Standardmanager für den Verbindungspool agiert prozessweit und wird von mehreren Anwendungen gemeinsam genutzt. Änderungen an der Konfiguration des Verbindungspools, beispielsweise die Ersetzung des Standardverbindungsmanagers für eine Anwendung mit der Methode MQEnvironment.setDefaultConnectionManager(), wirken sich daher auch auf andere Anwendungen aus, die auf demselben Java EE-Anwendungsserver ausgeführt werden.
- TLS wird für Anwendungen, die IBM MQ classes for Java verwenden, mit der MQEnvironment-Klasse und den MQQueueManager-Objekteigenschaften konfiguriert. Es wird nicht in die verwaltete Sicherheitskonfiguration des Anwendungsservers selbst integriert. Sie müssen sicherstellen, dass Sie IBM MQ classes for Java entsprechend konfigurieren, damit Sie Ihre erforderliche Sicherheitsstufe erhalten. Verwenden Sie also nicht die Konfiguration des Anwendungsservers.

Einschränkungen beim Bindungsmodus

IBM MQ und WebSphere Application Server können auf derselben Maschine installiert werden, sodass die Hauptversionen des Warteschlangenmanagers und des IBM MQ -Ressourcenadapters (RA), die im Lieferumfang von WebSphere Application Server enthalten sind, unterschiedlich sind.

Wenn sich die Hauptversionen des Warteschlangenmanagers und Ressourcenadapters unterscheiden, können keine Bindungsverbindungen verwendet werden. Bei allen Verbindungen, die vom WebSphere Application Server zu dem Warteschlangenmanager hergestellt werden, der den Ressourcenadapter verwendet, müssen Verbindungen des Clienttyps verwendet werden. Wenn die Versionen identisch sind, können Bindungsverbindungen verwendet werden.

Zeichenfolgekonvertierungen in IBM MQ classes for Java

Die IBM MQ classes for Java verwenden CharsetEncoders und CharsetDecoders direkt für die Zeichenfolgenkonvertierung. Das Standardverhalten für die Zeichenfolgenkonvertierung kann mit zwei Systemeigenschaften konfiguriert werden. Die Verarbeitung von Nachrichten, die Zeichen enthalten, die nicht zugeordnet werden können, kann mithilfe von `com.ibm.mq.MQMD` konfiguriert werden.

Vor IBM MQ 8.0 erfolgten Zeichenfolgekonvertierungen in IBM MQ classes for Java durch Aufrufe der Methoden `java.nio.charset.Charset.decode(ByteBuffer)` und `Charset.encode(CharBuffer)`.

Bei Verwendung einer dieser Methoden werden fehlerhafte oder nicht übersetzbare Daten standardmäßig ersetzt (REPLACE). Durch dieses Verhalten können Fehler in Anwendungen unkenntlich gemacht werden und es kann zu nicht erwarteten Zeichen in übersetzten Daten (z. B. ?) kommen.

Ab IBM MQ 8.0 werden solche Probleme früher und effektiver erkannt, da die IBM MQ classes for Java-Klassen für Java CharsetEncoders und CharsetDecoders direkt verwenden und die Behandlung von fehlerhaften und nicht umsetzbaren Daten explizit konfigurieren. Das Standardverhalten besteht darin, dass solche Probleme gemeldet werden (REPORT), indem eine entsprechende MQ-Ausnahme (`MQException`) ausgelöst wird.

konfigurieren

Die Umsetzung von UTF-16 (der in Java verwendeten Zeichendarstellung) in einen nativen Zeichensatz wie beispielsweise UTF-8 wird als Codierung (*encoding*) und die Umsetzung in die andere Richtung als Decodierung (*decoding*) bezeichnet.

Derzeit gilt bei der Decodierung das Standardverhalten für `CharsetDecoders`, wobei Fehler durch Auslösen einer Ausnahme gemeldet werden.

Eine Einstellung wird zur Angabe von `java.nio.charset.CodingErrorAction` verwendet, um die Fehlerbehandlung bei der Codierung und Decodierung zu steuern. Eine andere Einstellung wird zur Steuerung des bzw. der Ersetzungsbytes bei der Codierung verwendet. Bei Decodierungsvorgängen wird die standardmäßige Java-Ersetzungszeichenfolge verwendet.

Konfiguration der Verarbeitung nicht übersetzbarer Daten in IBM MQ classes for Java

Ab IBM MQ 8.0 umfasst `com.ibm.mq.MQMD` die folgenden beiden Felder:

byte[] unMappableReplacement
Die Bytefolge, die in eine codierte Zeichenfolge geschrieben wird, wenn ein eingegebenes Zeichen nicht übersetzt werden kann und Sie REPLACE angegeben haben.

Standardwert: "?".getBytes()
Bei Decodierungsvorgängen wird die standardmäßige Java-Ersetzungszeichenfolge verwendet.

java.nio.charset.CodingErrorAction unmappableAction
Gibt an, welche Aktion für nicht übersetzbare Daten bei der Codierung und Decodierung durchgeführt werden soll:

Standardwert: CodingErrorAction.REPORT;

Systemeigenschaften zum Festlegen von Systemstandardwerten

Ab IBM MQ 8.0 sind die folgenden zwei Java-Systemeigenschaften verfügbar, um das Standardverhalten für die Zeichenfolgekonvertierung zu konfigurieren.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Gibt an, welche Aktion für nicht übersetzbare Daten bei der Codierung und Decodierung durchgeführt werden soll. Der Wert kann REPORT, REPLACE oder IGNORE lauten.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Legt die Ersetzungsbytes fest, die angewandt werden sollen, wenn in einem Codierungsvorgang ein Zeichen nicht zugeordnet werden kann, bzw. ruft diese ab. Bei Decodierungsvorgängen wird die standardmäßige Java-Ersetzungszeichenfolge verwendet.

Damit es nicht zu Unklarheiten zwischen den Darstellungen in Form von Java-Zeichen und nativen Bytes kommt, sollten Sie für `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` eine Dezimalzahl angeben, die das Ersetzungsbyte im nativen Zeichensatz darstellt.

Der Dezimalwert von ? als natives Byte lautet beispielsweise 63, wenn der native Zeichensatz auf ASCII basiert (z. B. ISO-8859-1) und 111 bei dem nativen Zeichensatz EBCDIC.

Anmerkung: Falls bei einem MQMD- oder MQMessage-Objekt das Feld **unmappableAction** oder **unMappableReplacement** festgelegt ist, haben die Werte dieser Felder Vorrang vor den Java-Systemeigenschaften. Auf diese Weise können die Werte der Java-Systemeigenschaften bei Bedarf für jede Nachricht überschrieben werden.

Zugehörige Konzepte
„Zeichenfolgekonvertierungen in IBM MQ classes for JMS" auf Seite 145
Die IBM MQ classes for JMS verwenden CharsetEncoders und CharsetDecoders direkt für die Zeichenfolgenkonvertierung. Das Standardverhalten für die Zeichenfolgenkonvertierung kann mit zwei Systemeigenschaften konfiguriert werden. Die Verarbeitung von Nachrichten, die Zeichen enthalten, die nicht zugeordnet werden können, kann mithilfe Nachrichteneigenschaften zum Festlegen von 'UnmappableCharacterAction' und der Ersetzungsbyte konfiguriert werden.

IBM MQ classes for Java installieren und konfigurieren

In diesem Abschnitt werden die Verzeichnisse und Dateien beschrieben, die erstellt werden, wenn Sie IBM MQ classes for Java installieren, und es wird gezeigt, wie IBM MQ classes for Java nach der Installation konfiguriert wird.

Dies wird für IBM MQ classes for Java installiert

Es wird die neueste Version von IBM MQ classes for Java mit IBM MQ installiert. Es ist möglicherweise erforderlich, hierfür Standardinstallationsoptionen aufzuheben.

Weitere Informationen zur Installation von IBM MQ finden Sie hier:

- Multi IBM MQ installieren
- z/OS IBM MQ for z/OS-Produkt installieren

IBM MQ classes for Java sind in den Java-Archivdateien (JAR-Dateien) `com.ibm.mq.jar` und `com.ibm.mq.jmqi.jar` enthalten.

Die Unterstützung für Standardnachrichtenheader, z. B. PCF (Programmable Command Format), ist in der JAR-Datei `com.ibm.mq.headers.jar` enthalten.

Die Unterstützung für Programmable Command Format (PCF) ist in der JAR-Datei `com.ibm.mq.pcf.jar` enthalten.

Anmerkung: Es wird nicht empfohlen, die IBM MQ classes for Java innerhalb eines Anwendungsservers zu verwenden. Informationen zu den Einschränkungen, die bei einer Ausführung in dieser Umgebung gelten, finden Sie im Abschnitt „IBM MQ classes for Java-Anwendungen in Java EE ausführen" auf Seite 367. Weitere Informationen finden Sie unter Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

Wichtig: Mit Ausnahme der verschiebbaren JAR-Dateien, die im Abschnitt „Verschiebbare JAR-Dateien von IBM MQ classes for Java" auf Seite 371 beschrieben werden, wird das Kopieren der JAR-Dateien von IBM MQ classes for Java oder der nativen Bibliotheken auf andere Maschinen oder an eine andere Position auf einer Maschine, auf der die IBM MQ classes for Java installiert wurden, nicht unterstützt.

Verschiebbare JAR-Dateien von IBM MQ classes for Java
Die verschiebbaren JAR-Dateien können auf Systeme verschoben werden, auf denen die IBM MQ classes for Java ausgeführt werden müssen.

Wichtig:

- Mit Ausnahme der verschiebbaren JAR-Dateien, die im Abschnitt Verschiebbare JAR-Dateien beschrieben werden, wird das Kopieren der JAR-Dateien von IBM MQ classes for Java oder der nativen Bibliotheken auf andere Maschinen oder an eine andere Position auf einer Maschine, auf der die IBM MQ classes for Java installiert wurden, nicht unterstützt.
- Um Klassenladeprogrammkonflikte zu vermeiden, sollten die verschiebbaren JAR-Dateien nicht in mehreren Anwendungen innerhalb derselben Java-Runtime gebündelt werden. Machen Sie in diesem Szenario die verschiebbaren JAR-Dateien von IBM MQ gegebenenfalls im Klassenpfad der Java-Runtime verfügbar.
- Schließen Sie die verschiebbaren JAR-Dateien nicht in Anwendungen ein, die auf Java EE-Anwendungsservern, z. B. WebSphere Application Server, bereitgestellt werden. In diesen Umgebungen sollte stattdessen der IBM MQ-Ressourcenadapter bereitgestellt und verwendet werden, da dieser die IBM MQ classes for Java enthält. Beachten Sie, dass WebSphere Application Server den IBM MQ-Ressourcenadapter einbettet, sodass er nicht manuell in dieser Umgebung bereitgestellt werden muss. Darüber hinaus werden die IBM MQ classes for Java in WebSphere Liberty nicht unterstützt. Weitere Informationen finden Sie im Abschnitt „Liberty und der IBM MQ-Ressourcenadapter" auf Seite 463.
- Wenn Sie die verschiebbaren JAR-Dateien in Ihren Anwendungen bündeln, stellen Sie sicher, dass alle vorausgesetzten JAR-Dateien eingeschlossen sind (siehe Beschreibung in Verschiebbare JAR-Dateien). Sie sollten auch sicherstellen, dass Sie über geeignete Prozeduren zum Aktualisieren der gebündelten JAR-Dateien im Rahmen der Anwendungspflege verfügen, um zu gewährleisten, dass die IBM MQ classes for Java aktuell bleiben und bekannte Probleme behoben werden.

Verschiebbare JAR-Dateien

Innerhalb eines Unternehmens können die folgenden Dateien zu Systemen verschoben werden, die IBM MQ classes for Java-Anwendungen ausführen müssen:

- JMS 2.0 `com.ibm.mq.allclient.jar` „1" auf Seite 371
- JM 3.0 `com.ibm.mq.jakarta.client.jar` „2" auf Seite 371
- V 9.4.0 `bcpkix-jdk18on.jar` „3" auf Seite 371
- `bcpkix-jdk15to18.jar` „4" auf Seite 371
- V 9.4.0 `bcprov-jdk18on.jar` „3" auf Seite 371
- `bcprov-jdk15to18.jar` „4" auf Seite 371
- V 9.4.0 `bcutil-jdk18on.jar` „3" auf Seite 371
- `bcutil-jdk15to18.jar` „4" auf Seite 371
- `org.json.jar`

Anmerkungen:

1. JMS 2.0 und JMS 1.1
2. Jakarta Messaging 3.0
3. Von IBM MQ 9.4.0
4. VorherIBM MQ 9.4.0

Bouncy Castle-Sicherheitsprovider und JAR-Dateien für CMS -Unterstützung

Der Sicherheitsprovider Bouncy Castle und die JAR-Dateien für CMS-Unterstützung sind erforderlich. Weitere Informationen enthält der Abschnitt Unterstützung für JREs anderer Anbieter als IBM bei AMS.

V 9.4.0 Die folgenden JAR-Dateien sind erforderlich:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

org.json.jar

Die Datei `org.json.jar` ist erforderlich, wenn Ihre IBM MQ classes for Java-Anwendung eine Clientkanaldefinitionstabelle (Client Canal Definition Table, CCDT) im JSON-Format verwendet.

com.ibm.mq.allclient.jar und com.ibm.mq.jakarta.client.jar

Die Dateien `com.ibm.mq.allclient.jar` und `com.ibm.mq.jakarta.client.jar` enthalten IBM MQ classes for JMS, IBM MQ classes for Javasowie die PCF-und Headerklassen. Wenn Sie diese Dateien an eine neue Position verschieben, stellen Sie sicher, dass Sie Schritte ausführen, um diese neue Position mit neuen IBM MQ -Fixpacks beizubehalten. Stellen Sie außerdem sicher, dass die Verwendung der Dateien dem IBM Support bekannt ist, wenn Sie einen vorläufigen Fix erhalten.

Verwenden Sie den folgenden Befehl, um die Version der Datei `com.ibm.mq.allclient.jar` oder `com.ibm.mq.jakarta.client.jar` zu ermitteln:

JM 3.0
```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0
```
java -jar com.ibm.mq.allclient.jar
```

Im folgenden Beispiel sehen Sie einen Beispielauszug der Ausgabe dieses Befehls:

```
C:\Programme\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Programme/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Installationsverzeichnisse für IBM MQ classes for Java
IBM MQ classes for Java-Dateien und Beispiele sind je nach Plattform in unterschiedlichen Standorten installiert. Der Standort der Java Runtime Environment (JRE), die mit IBM MQ installiert wird, hängt ebenfalls von der Plattform ab.

Installationsverzeichnis für IBM MQ classes for Java-Dateien

In Tabelle 49 auf Seite 373 werden die Installationsverzeichnisse der IBM MQ classes for Java-Dateien aufgeführt.

Tabelle 49. Installationsverzeichnisse von IBM MQ classes for Java	
Plattform	**Directory**
AIX	*MQ_INSTALLATION_PATH*/java/lib
	/QIBM/ProdData/mqm/java/lib
Linux	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Installationsverzeichnisse für Beispiele

Es werden einige Beispielanwendungen, wie z. B. die Installationsprüfprogramme, mit IBM MQ bereitgestellt. In Tabelle 50 auf Seite 373 werden die Installationsverzeichnisse der Beispielanwendungen aufgeführt. Die IBM MQ classes for Java-Beispiele befinden sich in einem Unterverzeichnis namens `wmqjava`. Die PCF-Beispiele sind in einem Unterverzeichnis namens `pcf`.

Tabelle 50. Beispielverzeichnisse	
Plattform	**Directory**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i	/QIBM/ProdData/mqm/java/samples
Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Installationsverzeichnisse für JRE

Die IBM MQ classes for JMS erfordern eine Java 7 Java Runtime Environment (JRE) (oder eine höhere Version). Eine geeignete JRE wird mit IBM MQ installiert. In Tabelle 51 auf Seite 373 werden die Installationsverzeichnisse dieser JRE aufgeführt. Um Java-Programme wie die bereitgestellten Beispiele unter Verwendung dieser JRE auszuführen, rufen Sie explizit *JRE_LOCATION*`/bin/java` auf oder fügen Sie *JRE_LOCATION*`/bin` zur PATH-Umgebung (oder funktionalen Entsprechung) für Ihre Plattform hinzu, wobei *JRE_LOCATION* das in Tabelle 51 auf Seite 373 aufgeführte Verzeichnis ist.

Tabelle 51. JRE-Verzeichnisse	
Plattform	**Directory**
AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i	/QIBM/ProdData/mqm/java/jre

Tabelle 51. JRE-Verzeichnisse (Forts.)	
Plattform	**Directory**
Linux	*MQ_INSTALLATION_PATH*/java/jre
Windows	*MQ_INSTALLATION_PATH*\java\jre
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Umgebungsvariablen, die für IBM MQ classes for Java relevant sind
Wenn Sie IBM MQ classes for Java-Anwendungen ausführen möchten, muss deren Klassenpfad die IBM MQ classes for Java- und Beispielverzeichnisse einschließen.

Damit IBM MQ classes for Java-Anwendungen ausgeführt werden können, muss ihr Klassenpfad das geeignete Verzeichnis der IBM MQ classes for Java enthalten. Um die Beispielanwendungen auszuführen, muss der Klassenpfad ebenfalls die entsprechenden Beispielverzeichnisse aufweisen. Diese Informationen können im Java -Aufrufbefehl oder in der Umgebungsvariablen **CLASSPATH** angegeben werden.

Wichtig: Das Festlegen der Java-Option `-Xbootclasspath`, um die IBM MQ classes for Java einzuschließen, wird nicht unterstützt.

Tabelle 52 auf Seite 374 zeigt die entsprechende Einstellung für **CLASSPATH** , die auf jeder Plattform zum Ausführen von IBM MQ classes for Java -Anwendungen verwendet werden kann, einschließlich der Beispielanwendungen.

*Tabelle 52. **CLASSPATH** -Einstellung zum Ausführen von IBM MQ classes for Java -Anwendungen, einschließlich der IBM MQ classes for Java -Beispielanwendungen*	
Plattform	**CLASSPATH festlegen**
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/samples/pcf

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Wenn Sie mithilfe der Option `-Xlint` kompilieren, wird möglicherweise ein Warnhinweis angezeigt, der Sie darüber informiert, dass `com.ibm.mq.ese.jar` nicht vorhanden ist. Sie können diesen Warnhinweis ignorieren. Diese Datei ist nur vorhanden, wenn Sie Advanced Message Security installiert haben.

Die zusammen mit IBM MQ classes for JMS bereitgestellten Scripts verwenden die folgenden Umgebungsvariablen:

MQ_JAVA_DATA_PATH
Diese Umgebungsvariable gibt das Verzeichnis für die Protokoll- und Traceausgabe an.

MQ_JAVA_INSTALL_PATH
Diese Umgebungsvariable gibt das Verzeichnis an, in dem IBM MQ classes for Java installiert sind (s. IBM MQ classes for Java-Installationsverzeichnisse).

MQ_JAVA_LIB_PATH
Diese Umgebungsvariable gibt das Verzeichnis an, in dem IBM MQ classes for Java-Bibliotheken gespeichert sind (s. Der Standort der IBM MQ classes for Java-Bibliotheken für jede Plattform). Einige Scripts, die mit IBM MQ classes for Java bereitgestellt werden, wie z. B. IVTRun, verwenden diese Umgebungsvariable.

Windows Unter Windows werden alle Umgebungsvariablen automatisch während der Installation festgelegt.

Linux AIX Unter AIX and Linux können die Umgebungsvariablen mit dem Script `setjmsenv` (bei Verwendung einer 32-Bit-JVM) bzw. `setjmsenv64` (bei Verwendung einer 64-Bit-JVM) festlegen. Diese Scripts befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*/java/bin.

IBM i Unter IBM imuss die Umgebungsvariable **QIBM_MULTI_THREADED** auf Ygesetzt werden. Sie können dann Multithread-Anwendungen genau wie bei der Ausführung von Einzelthread-Anwendungen ausführen. Weitere Informationen finden Sie unter IBM MQ mit Java und JMS einrichten.

IBM MQ classes for Java erfordern eine Java 7 Java Runtime Environment (JRE). Informationen zum Standort einer geeigneten JRE, die mit IBM MQ installiert wird, finden Sie unter „Installationsverzeichnisse für IBM MQ classes for Java" auf Seite 372.

IBM MQ classes for Java-Bibliotheken
Der Standort der IBM MQ classes for Java-Bibliotheken hängt von der Plattform ab. Geben Sie diesen Standort an, wenn Sie eine Anwendung starten.

Um den Standort der JNI-Bibliotheken (Java Native Interface) anzugeben, starten Sie Ihre Anwendung mithilfe eines **java**-Befehls im folgenden Format:

```
java -Djava.library.path= library_path application_name
```

Dabei steht *Bibliothekspfad* für den Pfad zu IBM MQ classes for Java, die die JNI-Bibliotheken einschließen. Tabelle 53 auf Seite 375 zeigt den Standort der IBM MQ classes for Java-Bibliotheken für jede Plattform. In dieser Tabelle steht *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Tabelle 53. Position der Bibliotheken mit IBM MQ classes for Java bei den einzelnen Plattformen

Plattform	**Verzeichnis mit den IBM MQ classes for Java-Bibliotheken**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
Linux Linux (x86-Plattform)	*MQ_INSTALLATION_PATH*/java/lib
Linux Linux (Plattformen POWER, x86-64 und zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)

Tabelle 53. Position der Bibliotheken mit IBM MQ classes for Java bei den einzelnen Plattformen (Forts.)	
Plattform	**Verzeichnis mit den IBM MQ classes for Java-Bibliotheken**
Windows Windows	*MQ_INSTALLATION_PATH*\Java\lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*\Java\lib64 (64-Bit-Bibliotheken)
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (32-Bit- und 64-Bit-Bibliotheken)

Anmerkung:

1. Linux AIX Verwenden Sie unter AIX oder Linux (Power-Plattform) entweder die 32-Bit-Bibliotheken oder die 64-Bit-Bibliotheken. Verwenden Sie die 64-Bit-Bibliotheken nur, wenn Sie Ihre Anwendung in einer 64-Bit-JVM (Java Virtual Machine) auf einer 64-Bit-Plattform ausführen. Nutzen Sie ansonsten die 32-Bit-Bibliotheken.
2. Windows Unter Windows können Sie die PATH-Umgebungsvariable zum Angeben des Standortes der IBM MQ classes for Java-Bibliothek verwenden, anstatt ihren Standort beim **java**-Befehl anzugeben.
3. IBM i Um IBM MQ classes for Java im Bindungsmodus auf IBM i zu verwenden, stellen Sie sicher, dass sich die Bibliothek QMQMJAVA in Ihrer Bibliotheksliste befindet.
4. z/OS Unter z/OS können Sie entweder eine 32-Bit- oder eine 64-Bit-JVM (Java Virtual Machine) verwenden. Sie müssen nicht angeben, welche Bibliotheken verwendet werden sollen; IBM MQ classes for Java kann selbst bestimmen, welche JNI-Bibliotheken geladen werden.

Zugehörige Konzepte

IBM MQ classes for Java verwenden

Nach der Installation von IBM MQ classes for Java können Sie die Installation für die Ausführung eigener Anwendungen konfigurieren.

Unterstützung für OSGi mit IBM MQ classes for Java

OSGi stellt ein Framework bereit, das die Implementierung von Anwendungen als Bundles unterstützt. Es werden drei OSGi-Bundles als Teil von IBM MQ classes for Java bereitgestellt.

OSGi stellt ein allgemeines, sicheres und verwaltetes Java-Framework zur Verfügung, das die Implementierung von Anwendungen unterstützt, die in Form von Bundles erhalten werden. OSGi-konforme Einheiten können Bundles herunterladen und installieren und sie anschließend entfernen, wenn sie nicht mehr benötigt werden. Das Framework verwaltet die Installation und Aktualisierung von Bundles auf eine dynamische und skalierbare Weise.

Die IBM MQ classes for Java enthalten die folgenden OSGi-Bundles.

com.ibm.mq.osgi.java_*version_number*.jar
Die JAR-Dateien, um Anwendungen die Verwendung von IBM MQ classes for Java zu ermöglichen.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 Für Jakarta Messaging 3.0ermöglicht diese JAR-Datei Anwendungen die Verwendung von IBM MQ classes for JMS und IBM MQ classes for Javaund enthält auch den Code für die Verarbeitung von PCF-Nachrichten.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Für JMS 2.0ermöglicht diese JAR-Datei Anwendungen die Verwendung von IBM MQ classes for JMS und IBM MQ classes for Javasowie den Code für die Verarbeitung von PCF-Nachrichten.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar

JM 3.0 Für Jakarta Messaging 3.0enthält diese JAR-Datei die Voraussetzungen für `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar

JMS 2.0 Für JMS 2.0enthält diese JAR-Datei die Voraussetzungen für `com.ibm.mq.osgi.allclient_version_number.jar`.

Dabei ist *version_number* die installierte Versionsnummer von IBM MQ .

Die Bundles werden im Unterverzeichnis `java/lib/OSGi` Ihrer IBM MQ-Installation oder im Ordner `java\lib\OSGi` unter Windows installiert.

Verwenden Sie ab IBM MQ 8.0 für allen neuen Anwendungen die Bundles in `com.ibm.mq.osgi.allclient_8.0.0.0.jar` und `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar`. Durch die Verwendung dieser Bundles wird die Einschränkung aufgehoben, nicht IBM MQ classes for JMS und IBM MQ classes for Java innerhalb der gleichen OSGi-Framework ausführen zu können. Alle anderen Einschränkungen gelten jedoch weiterhin. Für IBM MQ-Versionen vor IBM MQ 8.0 gilt weiterhin die Einschränkung, entweder IBM MQ classes for JMS oder IBM MQ classes for Java zu verwenden.

Neun weitere Bundles werden ebenfalls im Unterverzeichnis `java/lib/OSGi` Ihrer IBM MQ-Installation oder im Ordner `java\lib\OSGi` unter Windows installiert. Diese Bundles sind Teil der IBM MQ classes for JMS und dürfen nicht in eine OSGi-Laufzeitumgebung geladen werden, in der das IBM MQ classes for Java-Bundle geladen wurde. Wenn das IBM MQ classes for Java OSGi-Bundle in eine OSGi-Laufzeitumgebung geladen wird, in die ebenfalls IBM MQ classes for JMS-Bundles geladen wurden, treten Fehler wie im unten gezeigten Beispiel auf, wenn Anwendungen ausgeführt werden, die entweder das IBM MQ classes for Java-Bundle oder IBM MQ classes for JMS-Bundles verwenden:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

Das OSGi-Bundle für IBM MQ classes for Java wurde in die Spezifikation von OSGi Release 4 geschrieben; es funktioniert nicht in einer Umgebung von OSGi Release 3.

Sie müssen Ihren Systempfad oder Bibliothekspfad ordnungsgemäß festlegen, damit die OSGi-Laufzeitumgebung alle erforderlichen DLL-Dateien oder gemeinsam genutzten Bibliotheken finden kann.

Wenn Sie das OSGi-Bundle für die IBM MQ classes for Java verwenden, werden Kanalexit-Klassen, die in Java geschrieben sind, nicht unterstützt, da ein Problem mit dem Laden von Klassen in einer Mehrfach-Klassenladeprogramm-Umgebung, wie z. B. OSGi, vorliegt. Ein Benutzer-Bundle kann über das IBM MQ classes for Java-Bundle Bescheid wissen, aber das IBM MQ classes for Java-Bundle weiß über kein anderes Benutzer-Bundle Bescheid. Folglich kann das in einem Bundle der IBM MQ classes for Java verwendete Klassenladeprogramm keine Kanalexitklasse laden, die sich in einem Benutzerbundle befindet.

Weitere Informationen über OSGi finden Sie auf der Website der Open Service Gateway-Initiative.

z/OS *Installation of IBM MQ classes for Java on z/OS*

On z/OS, the STEPLIB used at runtime must contain the IBM MQ SCSQAUTH and SCSQANLE libraries.

From z/OS UNIX System Services, you can add these libraries by using a line in your `.profile` as shown in the following example, replacing `thlqual` with the high level data set qualifier that you chose when installing IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In other environments, you typically need to edit the startup JCL to include SCSQAUTH on the STEPLIB concatenation:

```
 STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
         DD DSN=thlqual.SCSQANLE,DISP=SHR
```

Die Konfigurationsdatei für die IBM MQ classes for Java
In einer Konfigurationsdatei für die IBM MQ classes for Java sind Eigenschaften angegeben, die zur Konfiguration der IBM MQ classes for Java verwendet werden.

Die Konfigurationsdatei für die IBM MQ classes for Java hat das Format einer standardmäßigen Java-Eigenschaftendatei.

Im Unterverzeichnis `bin` des Installationsverzeichnisses des IBM MQ classes for Java-Installationsverzeichnisses befindet sich eine Beispielkonfigurationsdatei namens `mqjava.config`. In dieser Datei sind alle unterstützten Eigenschaften und ihre Standardwerte dokumentiert.

Anmerkung: Die Beispielkonfigurationsdatei wird überschrieben, wenn für die IBM MQ-Installation ein Upgrade auf ein künftiges Fixpack durchgeführt wird. Deshalb wird empfohlen, eine Kopie der Beispielkonfigurationsdatei für die Verwendung mit Ihren Anwendungen zu erstellen.

Sie können den Namen und die Position einer Konfigurationsdatei für IBM MQ classes for Java wählen. Verwenden Sie beim Start Ihrer Anwendung einen **java**-Befehl in folgendem Format:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

In dem Befehl ist *URL_der_Konfigurationsdatei* ein Uniform Resource Locator (URL), der den Namen und die Position der Konfigurationsdatei für die IBM MQ classes for Java angibt. Unterstützt werden URLs der folgenden Typen: `http`, `file`, `ftp` und `jar`.

Hier ein Beispiel für einen **java**-Befehl:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

In diesem Befehl ist als Konfigurationsdatei für die IBM MQ classes for Java die Datei `D:\mydir\mqjava.config` im lokalen Windows-System angegeben.

Wenn eine Anwendung gestartet wird, lesen die IBM MQ classes for Java die Inhalte der Konfigurationsdatei und speichern die angegebenen Eigenschaften in einem internen Eigenschaftenspeicher. Wenn in dem **java**-Befehl keine Konfigurationsdatei angegeben ist oder die Konfigurationsdatei nicht gefunden werden kann, verwenden die IBM MQ classes for Java die Standardwerte für alle Eigenschaften. Falls erforderlich, können Sie jede Eigenschaft in der Konfigurationsdatei überschreiben, indem Sie sie im **java**-Befehl als Systemeigenschaft angeben.

Eine Konfigurationsdatei für IBM MQ classes for Java kann mit allen unterstützten Transportprotokollen zwischen einer Anwendung und einem Warteschlangenmanager oder Broker verwendet werden.

In einer IBM MQ MQI client-Konfigurationsdatei angegebene Eigenschaften überschreiben

In einer Konfigurationsdatei für den IBM MQ MQI client können ebenfalls Eigenschaften angegeben werden, die für die Konfiguration der IBM MQ classes for Java verwendet werden. Eigenschaften, die in einer IBM MQ MQI client-Konfigurationsdatei angegebenen sind, gelten allerdings nur, wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt.

Falls erforderlich, können Sie jedes Attribut in einer Konfigurationsdatei für den IBM MQ MQI client überschreiben, indem Sie es als Eigenschaft in einer Konfigurationsdatei für IBM MQ classes for Java angeben. Wenn Sie ein Attribut in einer Konfigurationsdatei für den IBM MQ MQI client überschreiben möchten, verwenden Sie in der Konfigurationsdatei für IBM MQ classes for Java einen Eintrag im folgenden Format:

```
com.ibm.mq.cfg.stanza.propName=propValue
```

Die Variablen in dem Eintrag haben folgende Bedeutungen:

Zeilengruppe
Der Name der Zeilengruppe in der Konfigurationsdatei für den IBM MQ MQI client, die das Attribut enthält.

propName
Der Name des Attributs, so wie in der Konfigurationsdatei für den IBM MQ MQI client angegeben.

propValue
Der Wert der Eigenschaft, der den Wert des Attributs überschreibt, das in der Konfigurationsdatei für den IBM MQ MQI client angegeben ist.

Alternativ können Sie ein Attribut in einer IBM MQ MQI client-Konfigurationsdatei überschreiben, indem Sie die Eigenschaft als Systemeigenschaft im Befehl **java** angeben. Verwenden Sie das vorherige Format zur Angabe der Eigenschaft als Systemeigenschaft.

Nur die folgenden Attribute in einer IBM MQ MQI client-Konfigurationsdatei sind für IBM MQ classes for Java relevant. Wenn Sie andere Attribute angeben oder außer Kraft setzen, hat dies keine Wirkung. Es ist insbesondere zu beachten, dass `ChannelDefinitionFile` und `ChannelDefinitionDirectory` in der Zeilengruppe CHANNELS der Clientkonfigurationsdatei nicht verwendet werden. Im Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for Java verwenden” auf Seite 395 finden Sie Details zur Verwendung der CCDT bei den IBM MQ classes for Java.

Tabelle 54. Welche Zeilengruppe der Clientkonfigurationsdatei enthält welches Attribut

Zeilengruppe	**Attribut**
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	Put1DefaultAlwaysSync
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	PasswordProtection
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	Standardpfad für Exits
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	ExitsDefaultPath64
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	JavaExitsClasspath
Zeilengruppe JMQI der Clientkonfigurationsdatei	useMQCSPauthentication
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	MaximumSize
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	PurgeTime
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	UpdatePercentage
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntRcvBuffSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntSndBuffSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	Connect_Timeout
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	KeepAlive

Weitere Informationen zur IBM MQ MQI client -Konfiguration finden Sie in der IBM MQ MQI client -Konfigurationsdatei `mqclient.ini`.

Zugehörige Tasks
Tracefunktion für WebSphere MQ-Klassen für Java-Anwendungen

Java Standard Environment Trace zum Konfigurieren des Java -Trace verwenden
Über die Zeilengruppe für die Java-Standardumgebungs-Traceeinstellungen können Sie die Tracefunktion der IBM MQ classes for Java konfigurieren.

com.ibm.msg.client.commonservices.trace.outputName = *NameTraceausgabe*
traceOutputName ist das Verzeichnis und der Dateiname, an die die Traceausgabe gesendet wird.

Standardmäßig werden Traceinformationen in eine Tracedatei im aktuellen Arbeitsverzeichnis der Anwendung geschrieben. Der Name der Tracedatei ist von der Umgebung abhängig, in der die Anwendung ausgeführt wird:

- Wenn die Anwendung die IBM MQ classes for Java aus der verschiebbaren JAR-Datei `com.ibm.mq.allclient.jar`geladen hat, wird der Trace in eine Datei mit dem Namen `mqjavaclient_%PID%.cl%u.trc`geschrieben.
- Wenn die Anwendung IBM MQ classes for Java aus der JAR-Datei `com.ibm.mq.jar`geladen hat, wird der Trace in eine Datei mit dem Namen `mqjava_%PID%.cl%u.trc`geschrieben.

Dabei steht *%PID%* für die Prozess-ID der Anwendung, für die ein Trace durchgeführt wird, *%u* ist eine eindeutige Zahl zur Unterscheidung der Dateien zwischen Threads, die den Trace unter verschiedenen Java-Klassenladeprogrammen ausführen.

Wenn eine Prozess-ID nicht verfügbar ist, wird eine Zufallszahl generiert und mit dem Buchstaben `f` als Präfix versehen. Wenn Sie die Prozess-ID in einen von Ihnen festgelegten Dateinamen einschließen möchten, verwenden Sie die Zeichenfolge `%PID%`.

Wenn Sie ein anderes Verzeichnis angeben, muss dieses bereits vorhanden sein und Sie müssen über Schreibzugriff auf dieses Verzeichnis verfügen. Wenn Sie nicht über Schreibzugriff verfügen, wird die Traceausgabe in die Datei `System.err` geschrieben.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList ist eine Liste von Paketen und Klassen, für die ein Trace erstellt wird, oder die Sonderwerte `ALL` oder `NONE`.

Die Paket- oder Klassennamen müssen durch ein Semikolon (`;`) getrennt werden. *includeList* nimmt standardmäßig den Wert `ALL`an und verfolgt alle Pakete und Klassen in IBM MQ classes for Java.

Anmerkung: Es ist möglich, ein Paket einzubeziehen, anschließend aber Unterpakete dieses Pakets auszuschließen. Wenn Sie beispielsweise das Paket `a.b` einbeziehen und das Paket `a.b.x` ausschließen, umfasst der Trace alles in `a.b.y` und `a.b.z`, nicht jedoch `a.b.x` oder `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList ist eine Liste von Paketen und Klassen, für die kein Trace erstellt wird, oder die Sonderwerte `ALL` oder `NONE`.

Die Paket- oder Klassennamen müssen durch ein Semikolon (`;`) getrennt werden. *excludeList* nimmt standardmäßig den Wert `NONE`an und schließt daher keine Pakete und Klassen in IBM MQ classes for JMS von der Traceerstellung aus.

Anmerkung: Sie haben die Möglichkeit, ein Paket auszuschließen, anschließend aber Unterpakete des Pakets einzubeziehen. Wenn Sie beispielsweise das Paket `a.b` ausschließen und das Paket `a.b.x` einbeziehen, umfasst der Trace alles in `a.b.x` und `a.b.x.1`, nicht jedoch `a.b.y` oder `a.b.z`.

Alle Pakete oder Klassen, die auf derselben Ebene sowohl als einbezogen als auch als ausgeschlossen angegeben sind, werden einbezogen.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxBytesFeldgruppe*
maxArrayBytes ist die maximale Anzahl Byte, für die ein Trace von einem beliebigen Byte-Array erstellt wird.

Wenn *maxArrayBytes* auf eine positive ganze Zahl gesetzt wird, begrenzt dies die Anzahl Byte in einem Byte-Array, die in die Tracedatei geschrieben werden. Die Bytefeldgruppe wird abgeschnitten, nachdem *maxArrayBytes* ausgegeben wurde. Die Einstellung *maxArrayBytes* verringert die Größe der resultierenden Tracedatei und die Auswirkungen der Traceerstellung auf die Leistung der Anwendung.

Wenn für diese Eigenschaft der Wert `0` angegeben ist, werden keine Inhalte von Bytefeldgruppen an die Tracedatei gesendet.

Der Standardwert lautet `-1`. Bei dieser Einstellung wird die Anzahl der Bytes in einer Bytefeldgruppe, die an die Tracedatei gesendet werden, nicht begrenzt.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes ist die maximale Anzahl von Bytes, die in eine Traceausgabedatei geschrieben werden.

maxTraceBytes funktioniert mit *traceCycles*. Nähert sich die Anzahl der Bytes des geschriebenen Trace dem Grenzwert, wird die Datei geschlossen und es wird eine neue Traceausgabedatei begonnen.

Bei Angabe des Werts `0` haben Traceausgabedateien die Länge Null. Der Standardwert lautet `-1`. In diesem Fall ist das Datenvolumen, das in eine Traceausgabedatei geschrieben werden kann, nicht begrenzt.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles ist die Anzahl der Traceausgabedateien, die durchlaufen werden sollen.

Wenn die aktuelle Traceausgabedatei den Grenzwert erreicht, der über *maxTraceBytes* angegeben wurde, wird die Datei geschlossen. Die weitere Traceausgabe wird in die nächste Traceausgabedatei in der Folge geschrieben. Jede Traceausgabedatei ist durch ein eigenes numerisches Suffix am Ende des Dateinamens gekennzeichnet. Die aktuelle oder jüngste Traceausgabedatei lautet `mqjms.trc.0`, die unmittelbar vorherige Traceausgabedatei `mqjms.trc.1`. Bis zum Erreichen des Höchstwerts wird dieses Nummerierungsmuster auch auf die älteren Traceausgabedateien angewandt.

Der Standardwert von *traceCycles* ist 1. Wenn *traceCycles* 1 ist und die aktuelle Traceausgabedatei ihre maximale Größe erreicht, wird die Datei geschlossen und gelöscht. Anschließend wird eine neue Traceausgabedatei mit demselben Namen begonnen. Somit ist immer nur eine Traceausgabedatei vorhanden.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters steuert, ob Methodenparameter und Rückgabewerte in den Trace eingeschlossen werden.

traceParameters ist standardmäßig TRUE. Wird *traceParameters* auf FALSE gesetzt, werden nur Methodensignaturen aufgezeichnet.

com.ibm.msg.client.commonservices.trace.startup = *Start*
In der Initialisierungsphase der IBM MQ classes for Java werden Ressourcen zugeordnet. Während der Phase der Ressourcenzuordnung wird die Haupttracefunktion initialisiert.

Wenn *startup* auf TRUEgesetzt ist, wird der Starttrace verwendet. Es werden unverzüglich Traceinformationen erstellt, die die Einrichtung aller Komponenten, einschließlich der Tracefunktion selbst, umfassen. Mithilfe der Starttraceinformationen lassen sich Konfigurationsprobleme diagnostizieren. Die Starttraceinformationen werden immer in die Datei `System.err` geschrieben.

startup ist standardmäßig FALSE.

startup wird vor Abschluss der Initialisierung geprüft. Geben Sie die Eigenschaft daher nur in der Befehlszeile als Java-Systemeigenschaft an. Geben Sie sie nicht in der Konfigurationsdatei der IBM MQ classes for Java an.

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Setzen Sie *compressedTrace* auf TRUE, um die Traceausgabe zu komprimieren.

Der Standardwert von *compressedTrace* ist FALSE.

Wenn *compressedTrace* auf TRUEgesetzt ist, wird die Traceausgabe komprimiert. Die standardmäßige Traceausgabedatei hat die Erweiterung `.trz`. Wenn für die Komprimierung der Standardwert FALSE festgelegt ist, hat die Datei die Erweiterung `.trc`. Damit wird angegeben, dass die Datei nicht komprimiert ist. Wenn jedoch der Dateiname für die Traceausgabe in *traceOutputName* angegeben wurde, wird stattdessen dieser Name verwendet; auf die Datei wird kein Suffix angewendet.

Die komprimierte Traceausgabe ist kleiner als die nicht komprimierte Ausgabe. Da weniger Ein-/Ausgaben enthalten sind, ist eine schnellere Ausgabe möglich als beim nicht komprimierten Trace. Die

komprimierte Traceerstellung wirkt sich weniger stark auf die Leistung der IBM MQ classes for Java aus als die nicht komprimierte Traceerstellung.

Wenn *maxTraceBytes* und *traceCycles* festgelegt sind, werden mehrere komprimierte Tracedateien anstelle mehrerer Flachdateien erstellt.

Falls die IBM MQ classes for Java unkontrolliert beendet werden, kann es sein, dass eine komprimierte Tracedatei nicht gültig ist. Aus diesem Grund darf die Tracekomprimierung nur verwendet werden, wenn die IBM MQ classes for Java kontrolliert geschlossen werden. Verwenden Sie die Tracekomprimierung nur, wenn die untersuchten Probleme nicht zu einer unerwarteten Beendigung der JVM selbst führen. Die Tracekomprimierung darf nicht genutzt werden, wenn Probleme diagnostiziert werden, die zu Beendigungen des Typs `System.Halt()` oder zu nicht normalen, unkontrollierten JVM-Beendigungen führen können.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*

traceLevel gibt eine Filterstufe für den Trace an Die definierten Tracestufen lauten wie folgt:

- `TRACE_NONE: 0`
- `TRACE_EXCEPTION: 1`
- `TRACE_WARNING: 3`
- `TRACE_INFO: 6`
- `TRACE_ENTRYEXIT: 8`
- `TRACE_DATA: 9`
- `TRACE_ALL: Integer.MAX_VALUE`

Jede Tracestufe umfasst alle niedrigeren Stufen. Ist die Tracestufe beispielsweise auf `TRACE_INFO` gesetzt, werden alle Tracepunkte mit der definierten Stufe `TRACE_EXCEPTION`, `TRACE_WARNING` oder `TRACE_INFO` in den Trace geschrieben. Alle anderen Tracepunkte sind ausgeschlossen.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

Über *standaloneTrace* wird gesteuert, ob der Clienttraceservice der IBM MQ classes for Java in einer WebSphere Application Server-Umgebung verwendet wird.

Ist *standaloneTrace* auf TRUE gesetzt, wird die Tracekonfiguration anhand der Clienttraceeigenschaften der IBM MQ classes for Java bestimmt.

Wenn *standaloneTrace* auf FALSE gesetzt ist und der Client der IBM MQ classes for Java in einem WebSphere Application Server-Container ausgeführt wird, wird der WebSphere Application Server-Trace-Service verwendet. Welche Traceinformationen generiert werden, hängt von den Traceeinstellungen des Anwendungsservers ab.

Der Standardwert von *standaloneTrace* ist FALSE.

IBM MQ classes for Java und Softwareverwaltungstools

Mit den IBM MQ classes for Java können Softwareverwaltungstools wie Apache Maven verwendet werden.

Viele große Entwicklungsunternehmen verwenden diese Tools, um Repositorys von Bibliotheken anderer Anbieter zentral zu verwalten.

Die IBM MQ classes for Java setzen sich aus einer Reihe von JAR-Dateien zusammen. Wenn Sie über diese API Java-Sprachanwendungen entwickeln, muss auf dem System, auf dem die Anwendung entwickelt wird, eine Installation eines IBM MQ-Servers, IBM MQ-Clients oder IBM MQ-Client-SupportPacs vorhanden sein.

Wenn Sie ein Softwareverwaltungstool verwenden und die JAR-Dateien, die die IBM MQ classes for Java bilden, zu einem zentral verwalteten Repository hinzufügen möchten, müssen folgende Punkte beachtet werden:

- Ein Repository oder Container muss nur für die Entwickler in Ihrem Unternehmen verfügbar gemacht werden. Jegliche Verteilung außerhalb des Unternehmens ist nicht zulässig.

- Das Repository muss eine vollständige und durchgängige Gruppe von JAR-Dateien aus einem einzelnen IBM MQ-Release oder -Fixpack enthalten.
- Es ist Ihre Aufgabe, das Repository mit allen vom IBM Support zur Verfügung gestellten Wartungsreleases zu aktualisieren.

Ab IBM MQ 8.0 muss die JAR-Datei `com.ibm.mq.allclient.jar` im Repository installiert sein.

Ab IBM MQ 9.0 sind der Sicherheitsprovider Bouncy Castle und die JAR-Dateien für CMS-Unterstützung erforderlich. Weitere Informationen finden Sie in den Abschnitten „Verschiebbare JAR-Dateien von IBM MQ classes for Java" auf Seite 371 und Unterstützung für JREs anderer Anbieter.

Konfiguration für Anwendungen der IBM MQ classes for Java nach der Installation

Nach der Installation von IBM MQ classes for Java können Sie die Installation für die Ausführung eigener Anwendungen konfigurieren.

Vergessen Sie nicht, die Readme-Datei des IBM MQ-Produktes für die neuesten Informationen oder für detailliertere Angaben zu Ihrer Umgebung zu erhalten. Die neueste Version der Produkt-Readme-Datei ist auf der Produkt-Readmes für IBM MQ, WebSphere MQund MQSeries-Webseite verfügbar.

Bevor Sie versuchen, eine IBM MQ classes for Java-Anwendung im Bindungsmodus auszuführen, stellen Sie sicher, dass Sie IBM MQ wie unter Konfigurieren beschrieben konfiguriert haben.

Warteschlangenmanager für das Akzeptieren von Clientverbindungen von IBM MQ classes for Java konfigurieren

Um Ihren Warteschlangenmanager für das Akzeptieren von eingehenden Verbindungen von Clients zu konfigurieren, definieren und erlauben Sie die Verwendung eines Serververbindungskanals und starten Sie ein Empfangsprogramm.

Weitere Informationen finden Sie im Artikel „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms" auf Seite 1123.

IBM MQ classes for Java-Anwendungen unter dem Java security manager ausführen

IBM MQ classes for Java können mit aktiviertem Java security manager ausgeführt werden. Um Anwendungen erfolgreich mit aktiviertem Java security manager ausführen zu können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtliniendefinitionsdatei konfigurieren.

Die einfachste Möglichkeit, eine geeignete Richtliniendefinitionsdatei zu erstellen, besteht darin, die mit der Java runtime environment (JRE) bereitgestellte Richtliniendatei zu ändern. Auf den meisten Systemen ist diese Datei im Pfad `path lib/security/java.policy` (relativ zum JRE-Verzeichnis) gespeichert. Sie können Richtliniendateien entweder mit Ihrem bevorzugten Editor oder mit dem Programm policytool, das mit Ihrer JRE bereitgestellt wurde, bearbeiten.

Sie müssen der Datei `com.ibm.mq.jmqi.jar` Berechtigungen erteilen, damit sie Folgendes kann:

- Sockets erstellen (im Clientmodus)
- Native Bibliothek laden (im Bindungsmodus)
- Verschiedene Eigenschaften aus der Umgebung lesen

Die Systemeigenschaft **`os.name`** muss für die IBM MQ classes for Java bei einer Ausführung unter dem Java security manager verfügbar sein.

Wenn Ihre Java-Anwendung den Java security manager verwendet, müssen Sie folgende Berechtigung zur von der Anwendung genutzten Datei `java.security.policy` hinzufügen, da ansonsten Ausnahmen für die Anwendung ausgelöst werden:

```
permission java.lang.RuntimePermission "modifyThread";
```

Diese RuntimePermission wird vom Client im Rahmen der Verwaltung der Zuordnung und des Abschlusses von Multiplexdialogen über TCP/IP-Verbindungen zu Warteschlangenmanagern benötigt.

Beispiel eines Richtliniendateieintrags

Unten finden Sie ein Beispiel eines Richtliniendateieintrags, der eine erfolgreiche Ausführung von IBM MQ classes for Java unter dem Standardsicherheitsmanager ermöglicht. Ersetzen Sie die Zeichenfolge *MQ_INSTALLATION_PATH* in diesem Beispiel mit dem Standort, unter dem IBM MQ classes for Java auf Ihrem System installiert sind.

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";
```

```
// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

Dieses Beispiel einer Richtliniendatei ermöglicht den IBM MQ classes for Java die ordnungsgemäße Funktion unter dem Sicherheitsmanager, aber Sie müssen möglicherweise immer noch Ihren eigenen Code für eine ordnungsgemäße Ausführung bearbeiten, bevor Ihre Anwendungen funktionieren.

Der mit IBM MQ classes for Java bereitgestellte Beispielcode ist nicht speziell für die Funktion mit dem Sicherheitsmanager geeignet; allerdings funktionieren die IVT-Tests mit dieser Richtliniendatei und dem Standardsicherheitsmanager.

Wichtig:

Für die Tracefunktion von IBM MQ classes for Java sind weitere Berechtigungen erforderlich, da sie zusätzliche Abfragen von Systemeigenschaften und auch weitere Dateisystemoperationen ausführt.

Eine geeignete Vorlagendatei für Sicherheitsrichtlinien zur Ausführung unter einem Sicherheitsmanager mit aktivierter Tracefunktion wird im Verzeichnis `samples/wmqjava` der IBM MQ-Installation unter dem Namen `example.security.policy` bereitgestellt.

Bei einer Standardinstallation befindet sich die Datei `example.security.policy` an folgender Position:

Windows
`C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy`

Linux
`/opt/mqm/samp/wmqjava/samples/example.security.policy`

Solaris
`/opt/mqm/samp/wmqjava/samples/example.security.policy`

AIX
`/usr/mqm/samp/wmqjava/samples/example.security.policy`

Running IBM MQ classes for Java applications under CICS Transaction Server
An IBM MQ classes for Java application can be run as a transaction under CICS Transaction Server.

To run an IBM MQ classes for Java application as a transaction under CICS Transaction Server for z/OS, perform the following steps:

1. Define the application and transaction to CICS by using the supplied CEDA transaction.
2. Ensure that the IBM MQ CICS adapter is installed in your CICS system. z/OS (See Using IBM MQ with CICS for details.)
3. Ensure that the JVM environment specified in CICS includes the appropriate CLASSPATH and LIBPATH entries.
4. Initiate the transaction by using any of your normal processes.

For more information on running CICS Java transactions, refer to your CICS system documentation.

IBM MQ classes for Java-Installation überprüfen

Es wird ein Installationsprüfprogramm (MQIVP) mit IBM MQ classes for Java bereitgestellt. Mithilfe dieses Programms können Sie sämtliche Verbindungsmodi von IBM MQ classes for Java testen.

Das Programm fragt nach einer Reihe von zur Auswahl stehenden Angaben und anderen Daten, um zu bestimmen, welchen Verbindungsmodus Sie bestätigen möchten. Gehen Sie wie folgt vor, um Ihre Installation zu bestätigen:

1. Wenn Sie das Programm im Clientmodus ausführen möchten, konfigurieren Sie Ihren Warteschlangenmanager wie in „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123beschrieben. Die zu verwendende Warteschlange ist SYSTEM.DEFAULT.LOCAL.QUEUE
2. Wenn Sie das Programm im Clientmodus ausführen werden, finden Sie weitere Informationen unter „IBM MQ classes for Java verwenden” auf Seite 364.

   Führen Sie die übrigen Schritte dieser Prozedur auf dem System aus, auf dem Sie das Programm ausführen werden.
3. Vergewissern Sie sich, dass Sie die CLASSPATH-Umgebungsvariable entsprechend den Anweisungen in „Umgebungsvariablen, die für IBM MQ classes for Java relevant sind” auf Seite 374 aktualisiert haben.
4. Ändern Sie das Verzeichnis in *MQ_INSTALLATION_PATH*/mqm/samp/wmqjava/samples, wobei *MQ_INSTALLATION_PATH* der Pfad zu Ihrer IBM MQ-Installation ist. Geben Sie dann bei der Eingabeaufforderung Folgendes ein:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   Hierbei steht *Bibliothekspfad* für den Pfad zu den IBM MQ classes for Java-Bibliotheken (weitere Informationen erhalten Sie unter „IBM MQ classes for Java-Bibliotheken” auf Seite 375).

   Gehen Sie bei der Eingabeaufforderung mit der Markierung (1) folgendermaßen vor:

   - Um eine TCP/IP-Verbindung zu verwenden, geben Sie einen IBM MQ-Server-Hostnamen ein.
   - Um eine native Verbindung (Bindungsmodus) zu verwenden, lassen Sie das Feld leer (geben Sie keinen Namen ein).

   Das Programm versucht Folgendes:

   - 1. Herstellen einer Verbindung zum Warteschlangenmanager
   - 2. Öffnen der Warteschlange SYSTEM.DEFAULT.LOCAL.QUEUE, einreihen einer Nachricht in die Warteschlange, abrufen einer Nachricht von der Warteschlange und danach schließen der Warteschlange
   - 3. Trennen vom Warteschlangenmanager
   - 4. Zurückgeben einer Nachricht, wenn die Operationen erfolgreich waren

Nachfolgend sehen Sie ein Beispiel zu den Eingabeaufforderungen und Antworten, die möglicherweise angezeigt werden. Die tatsächlichen Eingabeaufforderungen und Ihre Antworten hängen von Ihrem IBM MQ-Netzwerk ab.

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

Anmerkung:

1. z/OS Lassen Sie unter z/OS das Feld bei der Eingabeaufforderung mit der Markierung (1) leer.
2. Wenn Sie die Serververbindung wählen, sehen Sie keine Eingabeaufforderungen mit der Markierung (2).

3. IBM i Unter IBM i können Sie nur den Befehl `java MQIVP` von QShell ausgeben. Alternativ können Sie die Anwendung mit dem CL-Befehl `RUNJVA CLASS(MQIVP)` ausführen.

IBM MQ classes for Java-Beispielanwendungen verwenden

Die Beispielanwendungen der IBM MQ classes for Java bieten einen Überblick über die allgemeinen Funktionen der API für IBM MQ classes for Java. Mithilfe der Beispielanwendungen können Sie Ihre Installation und Messaging-Server-Konfiguration überprüfen sowie Ihre eigenen Anwendungen erstellen.

Informationen zu diesem Vorgang

Wenn Sie beim Erstellen Ihrer eigenen Anwendungen Unterstützung benötigen, können Sie die Beispielanwendungen als Ausgangspunkt dafür verwenden. Für jede Anwendung wird eine Quellenversion und ein kompilierte Version bereitgestellt. Prüfen Sie den Beispielquellcode und ermitteln Sie die wichtigsten Schritte zum Erstellen jedes erforderlichen Objekts für Ihre Anwendung (MQQueueManager, MQConstants, MQMessage, MQPutMessageOptions und MQDestination) und zur Festlegung aller spezifischen Eigenschaften, die zur Angabe der gewünschten Funktionsweise Ihre Anwendung erforderlich sind. Weitere Informationen finden Sie unter „Anwendungen der IBM MQ classes for Java schreiben" auf Seite 390. Die Beispiele könnten sich in zukünftigen Releases von IBM MQ Java ändern.

In Tabelle 55 auf Seite 387 sehen Sie, wo die Beispielanwendungen der IBM MQ classes for Java auf den einzelnen Plattformen installiert werden:

Tabelle 55. Installationsverzeichnisse für die Beispielanwendungen der IBM MQ classes for Java

Plattform	Directory
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/wmqjava/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\wmqjava\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/wmqjava/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/wmqjava`

Tabelle 56 auf Seite 387 zeigt die Gruppen von Beispielanwendungen, die mit IBM MQ classes for Java bereitgestellt werden.

Tabelle 56. Beispielanwendungen von IBM MQ classes for Java

Name der Beispielanwendung	Beschreibung
IMSBridgeSample.java	Einfaches Programm zur Demonstration der Verwendung der IMS-Bridge mit den IBM MQ classes for Java.
MQIVP.java	IBM MQ Java-Installationsprüfprogramm.
MQMessagePropertiesSample.java	Veranschaulicht die Verwendung der API für Nachrichteneigenschaften.
MQPubSubApiSample.java	Veranschaulicht die Verwendung der Publish/Subscribe-API.
MQSample.java	Einfaches Programm zur Demonstration des Einreihens von Nachrichten in eine Warteschlange und des Abrufens aus einer Warteschlange.
MQSampleMessageManager.java	Dienstprogrammklasse zur Nachrichtenbehandlung in den IBM MQ Java-Basisbeispielen.

Tabelle 56. Beispielanwendungen von IBM MQ classes for Java (Forts.)	
Name der Beispielanwendung	**Beschreibung**
mqjcivp.properties	Dieses Ressourcenpaket enthält die Nachrichten, die vom Installationsprüfprogramm der IBM MQ classes for Java (`MQIVP.java`) verwendet werden.

Die IBM MQ classes for Java stellen ein Script namens `runjms` bereit, das zur Ausführung der Beispielanwendungen verwendet werden kann. Dieses Script konfiguriert die IBM MQ-Umgebung so, dass Sie die Beispielanwendungen der IBM MQ classes for Java ausführen können.

Tabelle 57 auf Seite 388 zeigt den Speicherort des Scripts auf den einzelnen Plattformen:

Tabelle 57. Speicherort des Scripts `runjms`	
Plattform	**Directory**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	oder
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`java/bin/runjms`

Führen Sie die folgenden Schritte aus, um eine Beispielanwendung mit dem Script `runjms` aufzurufen:

Vorgehensweise

1. Rufen Sie eine Eingabeaufforderung auf und navigieren Sie zu dem Verzeichnis, das die auszuführende Beispielanwendung enthält.
2. Geben Sie den folgenden Befehl ein:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   Die Beispielanwendung zeigt eine Liste der Parameter an, die sie benötigt.
3. Geben Sie den folgenden Befehl ein, um das Beispiel mit diesen Parametern auszuführen:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Beispiel

Linux Geben Sie beispielsweise die folgenden Befehle ein, um das MQIVP-Beispiel unter Linuxauszuführen:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

Zugehörige Konzepte

„Dies wird für IBM MQ classes for JMS installiert" auf Seite 93
Bei der Installation von IBM MQ classes for JMS werden verschiedene Dateien und Verzeichnisse installiert. Unter Windows erfolgt ein Teil der Konfiguration während der Installation durch die automatische Festlegung von Umgebungsvariablen. Auf anderen Plattformen und in bestimmten Windows-Umgebun-

gen müssen Sie im Vorfeld Umgebungsvariablen festlegen, damit Sie Anwendungen der IBM MQ classes for JMS ausführen können.

IBM MQ classes for Java-Probleme lösen

Führen Sie zuerst das Installationsprüfprogramm aus. Sie müssen möglicherweise ebenfalls die Tracefunktion verwenden.

Wenn eine Anwendung nicht erfolgreich vollständig ausgeführt wird, führen Sie das Installationsprüfprogramm aus und befolgen Sie die Hinweise in den Diagnosenachrichten. Das Installationsprüfprogramm wird im Abschnitt „IBM MQ classes for Java-Installation überprüfen” auf Seite 385 erläutert.

Wenn die Probleme weiterhin bestehen und Sie das Service-Team von IBM kontaktieren müssen, werden Sie möglicherweise gebeten, die Tracefunktion zu aktivieren. Gehen Sie hierzu wie im folgenden Beispiel gezeigt vor.

So verfolgen Sie das `MQIVP`-Programm:

- Erstellen Sie eine Eigenschaftendatei `com.ibm.mq.commonservices` (siehe Eigenschaftendatei 'com.ibm.mq.commonservices' verwenden.
- Geben Sie den folgenden Befehl ein:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

Dabei gilt:

- *commonservices_Eigenschaftendatei* ist der Pfad (einschließlich des Dateinamens) zur Eigenschaftendatei `com.ibm.mq.commonservices` .
- *Bibliothekspfad* steht für den Pfad zu den IBM MQ classes for Java-Bibliotheken (weitere Informationen erhalten Sie unter „IBM MQ classes for Java-Bibliotheken” auf Seite 375).

Weitere Informationen zur Verwendung der Tracefunktion finden Sie unter Trace für Anwendungen erstellen, die die IBM MQ classes for Java verwenden.

z/OS MQ Adv. VUE Java client connectivity to batch applications running on z/OS

Under certain conditions, an IBM MQ classes for Java application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

By using a client connection, an IBM MQ classes for Java application can connect to a remote z/OS queue manager if the following conditions apply:

- The application is running in a batch environment.
- Der Warteschlangenmanager, zu dem eine Verbindung hergestellt wird, wird mit IBM MQ Advanced for z/OS Value Unit Edition -Berechtigung ausgeführt und der Parameter **ADVCAP** ist daher auf ENABLED gesetzt.

Weitere Informationen zu IBM MQ Advanced for z/OS Value Unit Edition finden Sie unter IBM MQ -Produkt-IDs und -Exportinformationen.

See ANZEIGEN QMGR for more information on **ADVCAP** and QMGR STARTEN for more information on **QMGRPROD**.

An IBM MQ classes for Java application on z/OS cannot use a client mode connection to connect to a queue manager that is not running on z/OS

If an IBM MQ classes for Java application on z/OS attempts to connect using client mode, and is not allowed to do so, MQRC_ENVIRONMENT_ERROR is returned.

Advanced Message Security (AMS) support

IBM MQ classes for Java client applications can use AMS when connecting to remote z/OS queue managers, subject to the conditions previously described in this topic.

Um AMS auf diese Weise zu verwenden, müssen die Clientanwendungen den Keystore-Typ `jceracfks` in `keystore.conf`verwenden. Dabei gilt Folgendes:

- Das Eigenschaftsnamenspräfix ist `jceracfks`. Dieses Namenspräfix ist von der Groß-/Kleinschreibung unabhängig.
- Der Keystore ist ein RACF-Schlüsselring.
- Kennwörter sind nicht erforderlich und werden ignoriert. Der Grund dafür ist, dass RACF-Schlüsselringe keine Kennwörter verwenden.
- Wenn Sie den Provider angeben, muss der Provider `IBMJCE` sein.

Wenn Sie `jceracfks` mit AMSverwenden, muss der Keystore das Format `safkeyring://`*`user`*`/`*`keyring`*haben. Dabei gilt Folgendes:

- `safkeyring` ist ein Literal. Dieser Name ist von der Groß-/Kleinschreibung unabhängig.
- *`user`* ist die RACF -Benutzer-ID, die Eigner des Schlüsselrings ist.
- *`keyring`* ist der Name des RACF -Schlüsselrings, bei dem die Groß-/Kleinschreibung beachtet werden muss.

Im folgenden Beispiel wird der Standardschlüsselring von AMS für Benutzer `JOHNDOE` verwendet:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Related concepts

"JMS/Jakarta Messaging client connectivity to batch applications running on z/OS" on page 132
Under certain conditions, an IBM MQ classes for JMS/Jakarta Messaging application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

Anwendungen der IBM MQ classes for Java schreiben

Die vorliegende Themensammlung bietet Informationen als Unterstützung beim Schreiben von Java-Anwendungen, um mit IBM MQ-Systemen zu interagieren.

Um mit IBM MQ classes for Java auf IBM MQ-Warteschlangen zuzugreifen, schreiben Sie Java-Anwendungen, die Aufrufe enthalten, die Nachrichten in IBM MQ-Warteschlangen einreihen und von ihnen empfangen. Weitere Informationen zu einzelnen Klassen finden Sie unter IBM MQ classes for Java.

Anmerkung: Die automatische Clientverbindungswiederholung wird von IBM MQ classes for Java nicht unterstützt.

Die IBM MQ classes for Java-Schnittstelle

Die prozedurale IBM MQ-Anwendungsprogrammierschnittstelle verwendet Verben, die auf Objekte einwirken. Die Java-Programmierschnittstelle verwendet Objekte, auf die Sie durch Aufrufen von Methoden einwirken.

Die prozedurale IBM MQ-Anwendungsprogrammierschnittstelle basiert auf Verben, wie z. B.:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Alle diese Verben verwenden als Parameter ein Handle für das IBM MQ-Objekt, das sie bearbeiten sollen. Ihr Programm besteht aus einer Reihe von IBM MQ-Objekten, die Sie durch Aufrufen von Methoden für diese Objekte bearbeiten können.

Wenn Sie die prozedurale Schnittstelle verwenden, können Sie mit dem Aufruf MQDISC(Hconn, CompCode, Reason) eine Verbindung zu einem Warteschlangenmanager trennen, wobei *Hconn* ein Handle für den Warteschlangenmanager ist.

In der Java-Schnittstelle wird der Warteschlangenmanager von einem Objekt der Klasse MQQueueManager dargestellt. Sie können die Verbindung zum Warteschlangenmanager trennen, indem Sie die Methode 'disconnect()' zu dieser Klasse aufrufen.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

IBM MQ classes for Java-Verbindungsmodi

Die Art und Weise, auf die Sie für IBM MQ classes for Java programmieren, hat einige Abhängigkeiten auf die Verbindungsmodi, die Sie verwenden möchten.

Wenn Sie Clientverbindungen verwenden, gibt es eine Reihe von Unterschieden zu IBM MQ MQI client, aber vom Konzept her ist eine Ähnlichkeit vorhanden. Wenn Sie den Bindungsmodus verwenden, können Sie Fastpath-Bindungen verwenden und den MQBEGIN-Befehl ausgeben. Geben Sie an, welcher Modus verwendet werden soll, indem Sie Variablen in der MQEnvironment-Klasse festlegen.

IBM MQ classes for Java-Clientverbindungen

Wenn IBM MQ classes for Java als Client verwendet wird, ist das wie beim IBM MQ MQI client, allerdings mit einer Reihe von Unterschieden.

Wenn Sie für *IBM MQ classes for Java* für die Verwendung als Client programmieren, sollten Sie folgende Unterschiede beachten:

- Es wird nur TCP/IP unterstützt.
- Es werden keine IBM MQ-Umgebungsvariablen beim Start gelesen.
- Informationen, die sonst in einer Kanaldefinition und in Umgebungsvariablen gespeichert werden würden, können in einer Klasse mit der Bezeichnung 'Umgebung' gespeichert werden. Alternativ können diese Informationen als Parameter übergeben werden, wenn die Verbindung hergestellt wird.
- Fehler und Ausnahmebedingungen werden in ein in der `MQException`-Klasse angegebenes Protokoll geschrieben. Das standardmäßige Ziel für Fehler ist die Java-Konsole.
- Es sind nur die folgenden Attribute in einer IBM MQ-Clientkonfigurationsdatei für IBM MQ classes for Java relevant. Wenn Sie andere Attribute angeben, sind diese unwirksam.

Zeilengruppe	**Attribut**
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	Standardpfad für Exits
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	ExitsDefaultPath64
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	JavaExitsClasspath
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	MaximumSize
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	PurgeTime
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	UpdatePercentage

Zeilengruppe	Attribut
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntRcvBuffSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntSndBuffSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	Connect_Timeout
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	KeepAlive

- Beim Herstellen einer Verbindung zu einem Warteschlangenmanager, der eine Konvertierung von Zeichendaten erfordert, ist der V7 Java-Client nun in der Lage, die Umwandlung durchzuführen, wenn dem Warteschlangenmanager dies nicht möglich ist. Die Client-JVM muss die Konvertierung zwischen dem CCSID des Client und dem des Warteschlangenmanagers unterstützen.
- Eine automatische Wiederherstellung einer Client-Verbindung wird von IBM MQ classes for Java nicht unterstützt.

Bei der Verwendung im Clientmodus unterstützt *IBM MQ classes for Java* nicht den MQBEGIN-Aufruf.

IBM MQ classes for Java-Bindungsmodus
Der Bindungsmodus von IBM MQ classes for Java unterscheidet sich hauptsächlich auf drei Arten vom Clientmodus.

Im Bindungsmodus verwendet IBM MQ classes for Java die Java Native Interface (JNI), um die vorhandene Warteschlangen-API direkt aufzurufen, anstatt über ein Netzwerk zu kommunizieren.

Standardmäßig stellen Anwendungen, die IBM MQ classes for Java im Bindungsmodus verwenden, eine Verbindung zu einem Warteschlangenmanager mithilfe der *ConnectOption*, MQCNO_STANDARD_BINDINGS her.

Die IBM MQ classes for Java unterstützen die folgenden *ConnectOptions*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

Weitere Informationen zu *ConnectOptions* erhalten Sie unter „Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen” auf Seite 775.

Der Bindungsmodus unterstützt auf allen Plattformen außer IBM MQ for IBM i und IBM MQ for z/OS den MQBEGIN-Aufruf, um globale Arbeitseinheiten zu initialisieren, die vom Warteschlangenmanager koordiniert werden.

Die meisten Parameter, die von der `MQEnvironment`-Klasse bereitgestellt werden, sind für den Bindungsmodus nicht relevant und werden ignoriert.

Definieren der zu verwendenden IBM MQ classes for Java-Verbindung
Der zu verwendende Verbindungstyp wird von der Einstellung von Variablen in der MQEnvironment-Klasse bestimmt.

Es werden zwei Variablen verwendet:

MQEnvironment.properties
Der Verbindungstyp wird vom Wert bestimmt, der zum Schlüsselnamen `CMQC.TRANSPORT_PROPERTY` gehört. Folgende Werte sind möglich:

CMQC.TRANSPORT_MQSERIES_BINDINGS
Verbindung im Bindungsmodus herstellen

CMQC.TRANSPORT_MQSERIES_CLIENT
Verbindung im Clientmodus herstellen

CMQC.TRANSPORT_MQSERIES
Der Verbindungsmodus wird vom Wert der Eigenschaft *hostname* bestimmt.

MQEnvironment.hostname
Legen Sie den Wert dieser Variablen wie folgt fest:

- Stellen Sie für Clientverbindungen den Wert dieser Variablen auf den Hostnamen des IBM MQ-Servers ein, zu dem Sie eine Verbindung herstellen möchten.
- Legen Sie für den Bindungsmodus diese Variable nicht fest, oder setzen Sie sie auf Null.

Operationen bei Warteschlangenmanagern

In dieser Themensammlung wird beschrieben, wie Sie mit IBM MQ classes for Java eine Verbindung zu einem Warteschlangenmanager herstellen oder trennen.

IBM MQ-Umgebung für IBM MQ classes for Java einrichten
Damit eine Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt, muss die Anwendung den Kanalnamen, den Hostnamen und die Portnummer angeben.

Anmerkung: Die Informationen in diesem Abschnitt sind nur relevant, wenn Ihre Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt. Bei einer Verbindung im Bindungsmodus sind sie nicht relevant. Weitere Informationen erhalten Sie unter: „Verbindungsmodi für IBM MQ classes for JMS" auf Seite 114

Sie können den Kanalnamen, den Hostnamen und die Portnummer auf eine der folgenden zwei Arten festlegen: entweder als Felder in der MQEnvironment-Klasse oder als Eigenschaften des MQQueueManager-Objekts.

Wenn Sie Felder in der MQEnvironment-Klasse festlegen, gelten sie für Ihre gesamte Anwendung, außer sie werden von einer Eigenschaften-Hashtabelle außer Kraft gesetzt. Um den Kanalnamen und Hostnamen in MQEnvironment anzugeben, verwenden Sie folgenden Code:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

Dies entspricht der Einstellung einer **MQSERVER**-Umgebungsvariablen:

```
"java.client.channel/TCP/host.domain.com".
```

Standardmäßig versuchen die Java -Clients, eine Verbindung zu einem IBM MQ -Listener an Port 1414herzustellen. Um einen anderen Port anzugeben, verwenden Sie folgenden Code:

```
MQEnvironment.port = nnnn;
```

Dabei steht nnnn für die erforderliche Portnummer.

Wenn Sie Eigenschaften an ein Warteschlangenmanagerobjekt bei seiner Erstellung übergeben, gelten sie ausschließlich für diesen Warteschlangenmanager. Erstellen Sie Einträge in einem Hashtabellenobjekt mit Schlüsseln von **hostname**, **channel** und optional **port** sowie mit entsprechenden Werten. Um den Standardport (1414) zu verwenden, können Sie den Eintrag **port** übergehen. Erstellen Sie das MQQueueManager-Objekt, indem Sie einen Konstruktor verwenden, der die Eigenschaften-Hashtabelle akzeptiert.

Verbindung zum Warteschlangenmanager durch Einstellung eines Anwendungsnamens ermitteln

Eine Anwendung kann einen Namen festlegen, der die Verbindung zum Warteschlangenmanager angibt. Dieser Anwendungsname wird durch den Befehl **DISPLAY CONN MQSC/PCF** (wobei das Feld die Bezeichnung **APPLTAG** trägt) oder in IBM MQ Explorer **Application Connections** angezeigt (wobei das Feld die Bezeichnung **App name** (Anwendungsname) trägt).

Anwendungsnamen sind auf 28 Zeichen beschränkt; längere Namen werden abgeschnitten. Wenn kein Anwendungsname angegeben wird, wird ein Standardwert bereitgestellt. Der Standardname basiert auf

der aufrufenden (Haupt-)Klasse, wenn diese Information jedoch nicht verfügbar ist, wird der Text `IBM MQ Client for Java` verwendet.

Wenn der Name der aufrufenden Klasse verwendet wird, dann wird er erforderlichenfalls durch Entfernen von vorangestellten Paketnamen entsprechend angepasst. Wenn zum Beispiel `com.example.MainApp` die aufrufende Klasse ist, wird der vollständige Name verwendet, wenn aber `com.example.dictionaryAndThesaurus.multilingual.mainApp` die aufrufende Klasse ist, wird der Name `multilingual.mainApp` verwendet, da dies die längste Kombination aus Klassenname und, von rechts gesehen, dem Paketnamen ist, mit dem die Längenvorgabe erfüllt ist.

Wenn der Klassenname selbst mehr als 28 Zeichen lang ist, wird er entsprechend abgeschnitten. Zum Beispiel wird statt `com.example.mainApplicationForSecondTestCase` die Zeichenfolge `mainApplicationForSecondTest` verwendet.

Um einen Anwendungsnamen in der MQEnvironment-Klasse festzulegen, fügen Sie den Namen der Hashtabelle 'MQEnvironment.properties' mit einem Schlüssel von **`MQConstants.APPNAME_PROPERTY`** hinzu. Verwenden Sie hierfür den folgenden Code:

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

Um einen Anwendungsnamen in der Eigenschaften-Hashtabelle festzulegen, die an den MQQueueManager-Konstruktor übergeben wird, fügen Sie den Namen der Eigenschaften-Hashtabelle mit einem Schlüssel von **`MQConstants.APPNAME_PROPERTY`** hinzu.

Eigenschaften außer Kraft setzen, die in einer IBM MQ-Clientkonfigurationsdatei angegeben wurden

Eine IBM MQ-Clientkonfigurationsdatei kann ebenfalls Eigenschaften angeben, mit denen IBM MQ classes for Java konfiguriert wird. Die in einer Konfigurationsdatei für den IBM MQ MQI client angegebenen Eigenschaften finden jedoch nur dann Anwendung, wenn sich eine Anwendung im Clientmodus mit einem Warteschlangenmanager verbindet.

Wenn erforderlich, können Sie ein Attribut in einer IBM MQ-Konfigurationsdatei auf eine der folgenden Weisen außer Kraft setzen. Die Optionen werden geordnet nach Ausführungspriorität gezeigt.

- Legen Sie eine Java-Systemeigenschaft für die Konfigurationseigenschaft fest.
- Legen Sie die Eigenschaft in der MQEnvironment.properties-Map fest.
- Unter Java5 und später können Sie eine Systemumgebungsvariable festlegen.

Es sind nur die folgenden Attribute in einer IBM MQ-Clientkonfigurationsdatei für IBM MQ classes for Java relevant. Wenn Sie andere Attribute angeben oder außer Kraft setzen, hat dies keine Wirkung.

Zeilengruppe	**Attribut**
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	Standardpfad für Exits
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	ExitsDefaultPath64
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	JavaExitsClasspath
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	MaximumSize
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	PurgeTime
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	UpdatePercentage

Zeilengruppe	Attribut
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntRcvBufSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntSndBufSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	Connect_Timeout
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	KeepAlive

Verbindung zu einem Warteschlangenmanager in IBM MQ classes for Java herstellen
Stellen Sie eine Verbindung zu einem Warteschlangenmanager her, indem Sie eine neue Instanz der MQQueueManager-Klasse erstellen. Sie können die Verbindung zum Warteschlangenmanager trennen, indem Sie die Methode 'disconnect()' aufrufen.

Sie können nun eine Verbindung zu einem Warteschlangenmanager herstellen, indem Sie eine neue Instanz der MQQueueManager-Klasse erstellen:

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Sie können die Verbindung zu einem Warteschlangenmanager trennen, indem Sie die Methode 'disconnect()' beim Warteschlangenmanager aufrufen.

```
queueManager.disconnect();
```

Wenn Sie die Trennungsmethode aufrufen, werden alle offenen Warteschlangen und Prozesse geschlossen, auf die Sie über diesen Warteschlangenmanager zugegriffen haben. Es wird jedoch aus programmiertechnischen Gründen empfohlen, die Ressourcen explizit zu schließen, wenn Sie mit ihrer Verwendung fertig sind. Verwenden Sie hierfür die Methode 'close()' bei den relevanten Objekten.

Die Methoden 'commit()' und 'backout()' bei einem Warteschlangenmanager entsprechen den Aufrufen MQCMIT bzw. MQBACK, die bei der prozeduralen Schnittstelle verwendet werden.

Definitionstabelle für den Clientkanal mit IBM MQ classes for Java verwenden
Eine IBM MQ classes for Java-Clientanwendung kann Clientverbindungskanaldefinitionen verwenden, die in einer Clientkanal-Definitionstabelle (CCDT) gespeichert sind.

Als Alternative zur Erstellung einer Clientverbindungskanaldefinition durch das Festlegen von bestimmten Feldern und Umgebungseigenschaften in der `MQEnvironment`-Klasse oder der Übergabe an `MQQueueManager` in einer Eigenschaften-Hashtabelle kann eine IBM MQ classes for Java-Clientanwendung Clientverbindungskanaldefinitionen nutzen, die in einer Clientkanal-Definitionstabelle gespeichert sind. Diese Definitionen werden von MQSC-Befehlen (IBM MQ) oder PCF-Befehlen (IBM MQ) erstellt oder mit dem IBM MQ Explorer.

Wenn die Anwendung ein `MQQueueManager`-Objekt erstellt, durchsucht der IBM MQ classes for Java-Client die Clientkanal-Definitionstabelle nach einer geeigneten Clientverbindungskanaldefinition und verwendet diese für den Start eines MQI-Kanals. Sie finden weitere Informationen zu den Definitionstabellen für Clientkanäle und Anweisungen zu deren Erstellung im Abschnitt Definitionstabelle für Clientkanal.

Um eine Clientkanal-Definitionstabelle zu verwenden, muss eine Anwendung zuerst ein URL-Objekt erstellen. Das URL-Objekt bindet eine URL ein, die den Namen und die Position der Datei angibt, die die Definitionstabelle des Clientkanals enthält und die festlegt, wie auf die Datei zugegriffen werden kann.

Beispiel: Wenn die Datei `ccdt1.tab` eine Clientkanaldefinitionstabelle enthält und sich auf dem gleichen System befindet, auf dem die Anwendung ausgeführt wird, kann die Anwendung wie folgt ein URL-Objekt erstellen:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

Weiteres Beispiel: Angenommen, die Datei `ccdt2.tab` enthält eine Clientkanaldefinitionstabelle und befindet sich auf einem anderen System als dem, auf dem die Anwendung ausgeführt wird. Wenn auf die

Datei mit dem FTP-Protokoll zugegriffen werden kann, dann kann die Anwendung folgendermaßen ein URL-Objekt erstellen:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

Nachdem die Anwendung ein URL-Objekt erstellt hat, kann die Anwendung ein `MQQueueManager` -Objekt mit einem der Konstruktoren erstellen, die ein URL-Objekt als Parameter verwenden. Beispiel:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

Diese Anweisung lässt den IBM MQ classes for Java-Client auf die Clientkanal-Definitionstabelle zugreifen, die durch das URL-Objekt 'chanTab2' angegeben wird, die Tabelle nach einer geeigneten Clientverbindungskanaldefinition durchsuchen und anschließend die Kanaldefinition zum Starten eines MQI-Kanals zum Warteschlangenmanager namens MARS verwenden.

Beachten Sie die folgenden Punkte, die gelten, wenn eine Anwendung eine Clientkanal-Definitionstabelle verwendet:

- Wenn die Anwendung ein `MQQueueManager`-Objekt anhand eines Konstruktors erstellt, der ein URL-Objekt als Parameter nimmt, muss kein Kanalname in der `MQEnvironment`-Klasse als Feld oder Umgebungseigenschaft festgelegt werden. Beim Festlegen eines Kanalnamens löst der IBM MQ classes for Java-Client eine `MQException` aus. Das Feld oder die Umgebungseigenschaft, die den Kanalnamen angibt, wird als festgelegt angesehen, wenn der Wert ungleich null, keine leere Zeichenfolge und keine Zeichenfolge, die ausschließlich aus Leerzeichen besteht, ist.
- Der Parameter **queueManagerName** beim Konstruktor `MQQueueManager` kann einen der folgenden Werte aufweisen:
  - Der Name eines Warteschlangenmanagers
  - Ein Stern (*), gefolgt vom Namen einer Warteschlangenmanagergruppe
  - Ein Stern (*)
  - Null, eine leere Zeichenfolge oder eine Zeichenfolge, die ausschließlich aus Leerzeichen besteht

  Dies sind die gleichen Werte, die für den Parameter **QMgrName** bei einem MQCONN-Aufruf verwendet werden können, der von einer Clientanwendung ausgegeben wird, die Message Queue Interface (MQI) nutzt. Weitere Informationen zur Bedeutung dieser Werte erhalten Sie unter „Message Queue Interface (MQI) - Übersicht” auf Seite 759.

  Wenn Ihre Anwendung Verbindungspooling nutzt, erhalten Sie weitere Informationen unter „Standardverbindungspool in IBM MQ classes for Java steuern” auf Seite 417.
- Wenn der IBM MQ classes for Java-Client eine geeignete Clientverbindungskanaldefinition in der Clientkanal-Definitionstabelle findet, verwendet er nur die Informationen, die aus dieser Kanaldefinition extrahiert wurden, um einen MQI-Kanal zu starten. Jegliche kanalbezogenen Felder oder Umgebungseigenschaften, die die Anwendung möglicherweise in der `MQEnvironment`-Klasse festgelegt hat, werden ignoriert.

  Beachten Sie insbesondere die folgenden Punkte, wenn Sie Transport Layer Security (TLS) verwenden:
  - Ein MQI-Kanal verwendet TLS nur dann, wenn die aus der Clientkanal-Definitionstabelle extrahierte Kanaldefinition den Namen einer CipherSpec angibt, die vom Client für IBM MQ classes for Java unterstützt wird.
  - Eine Definitionstabelle für Clientkanäle enthält außerdem Informationen zur Position der LDAP-Server (LDAP = Lightweight Directory Access Protocol), auf denen die Zertifikatswiderrufslisten (CRLs) gespeichert sind. Der IBM MQ classes for Java-Client verwendet nur diese Informationen, um auf LDAP-Server zuzugreifen, die CRLs enthalten.
  - Eine Definitionstabelle für den Clientkanal kann auch die Position eines OCSP-Responders enthalten. IBM MQ classes for Java kann die OCSP-Informationen einer Clientkanaldefinitionstabelle nicht nutzen. Allerdings können Sie OCSP so konfigurieren, wie im Abschnitt Online Certificate Protocol verwenden beschrieben ist.

Weitere Informationen zur Verwendung von TLS mit einer Definitionstabelle für Clientkanäle finden Sie im Abschnitt Angeben, dass ein MQI-Kanal TLS verwendet.

Beachten Sie auch die folgenden Punkte, wenn Sie Kanalexits verwenden:

- Ein MQI-Kanal verwendet die Kanalexits und zugehörigen Benutzerdaten, die von der Kanaldefinition festgelegt wurden, die aus der Clientkanal-Definitionstabelle extrahiert wurde, anstatt Kanalexits und Daten zu verwenden, die mit anderen Methoden angegeben wurden.
- Eine Kanaldefinition, die aus einer Clientkanaldefinitionstabelle extrahiert wurde, kann Kanalexits angeben, die in Java, C oder C++ geschrieben sind. Weitere Informationen zum Schreiben eines Kanalexits in Java finden Sie unter „Kanalexit in IBM MQ classes for Java erstellen" auf Seite 410. Weitere Informationen zum Schreiben eines Kanalexits in anderen Sprachen finden Sie unter „Kanalexits, die nicht in Java geschrieben wurden, mit IBM MQ classes for Java verwenden" auf Seite 414.

Portbereich für Clientverbindungen mit IBM MQ classes for Java angeben
Sie können einen Port oder auch einen Bereich mehrerer Ports angeben, an die eine Anwendung gebunden werden kann. Hierfür gibt es zwei Möglichkeiten.

Wenn eine Anwendung der IBM MQ classes for Java versucht, im Clientmodus eine Verbindung zu einem IBM MQ-Warteschlangenmanager herzustellen, erlaubt die Firewall möglicherweise nur Verbindungen, die aus angegebenen Ports oder Portbereichen stammen. In dieser Situation können Sie einen Port oder auch einen Bereich mehrerer Ports angeben, an die die Anwendung gebunden werden kann. Für die Angabe von Ports haben Sie folgende Möglichkeiten:

- Sie können das Feld 'localAddressSetting' in der MQEnvironment-Klasse festlegen. Beispiel:

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- Sie können die Umgebungseigenschaft CMQC.LOCAL_ADDRESS_PROPERTY festlegen. Beispiel:

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- Wenn Sie das MQQueueManager-Objekt erstellen können, können Sie eine Hashtabelle mit Eigenschaften übergeben, die eine Eigenschaft des Typs LOCAL_ADDRESS_PROPERTY mit dem Wert "192.0.2.0(2000,3000)" enthält.

Bei allen dieser Beispiele gilt Folgendes: Wenn sich die Anwendung später mit einem Warteschlangenmanager verbindet, bindet sich die Anwendung an eine lokale IP-Adresse und Portnummer im Bereich von 192.0.2.0(2000) bis 192.0.2.0(3000).

In einem System mit mehr als einer Netzschnittstelle können Sie auch das Feld 'localAddressSetting' oder die Umgebungseigenschaft CMQC.LOCAL_ADDRESS_PROPERTY verwenden, um anzugeben, welche Netzschnittstelle für eine Verbindung verwendet werden muss.

Wenn Sie den Bereich der Ports beschränken, können Verbindungsfehler auftreten. Falls ein Fehler auftritt, wird eine MQ-Ausnahmebedingung (MQException) ausgelöst, die den IBM MQ-Ursachencode MQRC_Q_MGR_NOT_AVAILABLE und die folgende Nachricht enthält:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

Ein Fehler kann auftreten, wenn alle Ports im angegebenen Bereich belegt sind oder wenn die Angaben der IP-Adresse, des Hostnamens oder der Portnummer nicht gültig sind (da beispielsweise eine negative Portnummer angegeben wurde).

Auf Warteschlange, Themen und Prozesse in IBM MQ classes for Java zugreifen

Verwenden Sie die Methoden der MQQueueManager-Klasse, um auf Warteschlangen, Themen und Prozesse zuzugreifen. Die Objektdeskriptorstruktur MQOD wird in den Parametern dieser Methoden komprimiert.

Warteschlangen

Um eine Warteschlange zu öffnen, können Sie die accessQueue-Methode der MQQueueManager-Klasse verwenden. Verwenden Sie beispielsweise auf einem Warteschlangenmanager mit dem Namen 'queueManager' folgenden Code:

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

Die AccessQueue-Methode gibt ein neues Objekt der MQQueue-Klasse zurück.

Wenn Sie die Warteschlange nicht mehr benötigen, schließen Sie sie wie im folgenden Beispiel mit der Methode close():

```
queue.close();
```

Sie können eine Warteschlange auch mit dem MQQueue-Konstruktor erstellen. Die Parameter sind mit denjenigen der accessQueue-Methode identisch, allerdings gibt es zusätzlich noch einen Parameter für den Warteschlangenmanager. For example:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

Bei der Erstellung von Warteschlangen können Sie mehrere Optionen angeben. Sie finden Details zu diesen Optionen unter Class.com.ibm.mq.MQQueue. Wenn Sie ein Warteschlangenobjekt auf diese Weise erstellen, können Sie Ihre eigenen Unterklassen von MQQueue schreiben.

Themen

Auf ähnliche Weise können Sie mit der accessTopic-Methode der MQQueueManager-Klasse auch ein Thema öffnen. Verwenden Sie beispielsweise auf einem Warteschlangenmanager mit dem Namen 'queueManager' folgenden Code, um einen Subskribenten und einen Publisher zu erstellen:

```
MQTopic subscriber =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

Wenn Sie das Thema nicht mehr benötigen, schließen Sie es mit der Methode close().

Sie können ein Thema auch mit dem MQTopic-Konstruktor erstellen. Die Parameter sind mit denjenigen der accessTopic-Methode identisch, allerdings gibt es zusätzlich noch einen Parameter für den Warteschlangenmanager. For example:

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

Bei der Erstellung von Themen können Sie mehrere Optionen angeben. Sie finden Details zu diesen Optionen unter Klasse com.ibm.mq.MQTopic. Wenn Sie ein Topic-Objekt (Themenobjekt) auf diese Weise erstellen, können Sie Ihre eigenen Unterklassen von MQTopic schreiben.

Ein Thema muss entweder für die Veröffentlichung oder für die Subskription geöffnet werden. Die MQQueueManager-Klasse verfügt über acht accessTopic-Methoden und die Topic-Klasse hat acht Kon-

struktoren. In jedem Fall weisen vier dieser Methoden einen **destination**-Parameter auf und vier Methoden verfügen über einen **subscriptionName**-Parameter (darunter haben zwei Methoden jeweils beide Parameter). Diese können nur verwendet werden, um das Thema für Subskriptionen zu öffnen. Die beiden verbleibenden Methoden haben einen **openAs**-Parameter und das Thema kann entweder für die Veröffentlichung oder für die Subskription geöffnet werden, je nachdem, welcher Wert dem Parameter **openAs** zugewiesen ist.

Wenn Sie ein Thema als permanenter Subskribent erstellen möchten, verwenden Sie entweder eine accessTopic-Methode der MQQueueManager-Klasse oder einen MQTopic-Konstruktor, der einen Subskriptionsnamen akzeptiert. In beiden Fällen müssen Sie die Option CMQC.MQSO_DURABLE festlegen.

Prozesse

Um auf einen Prozess zuzugreifen, verwenden Sie die accessProcess-Methode von MQQueueManager. Verwenden Sie beispielsweise auf einem Warteschlangenmanager mit dem Namen 'queueManager' folgenden Code, um ein MQProcess-Objekt zu erstellen:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

Um auf einen Prozess zuzugreifen, verwenden Sie die accessProcess-Methode von MQQueueManager.

Die accessProcess-Methode gibt ein neues Objekt der MQProcess-Klasse zurück.

Wenn Sie das Prozessobjekt nicht mehr benötigen, schließen Sie es wie im folgenden Beispiel mit der Methode close():

```
process.close();
```

Sie können einen Prozess auch mit dem MQProcess-Konstruktor erstellen. Die Parameter sind mit denjenigen der accessProcess-Methode identisch, allerdings gibt es zusätzlich noch einen Parameter für den Warteschlangenmanager. For example:

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

Wenn Sie ein Prozessobjekt auf diese Weise erstellen, können Sie Ihre eigenen Unterklassen von MQProcess schreiben.

Behandlung von Nachrichten in IBM MQ classes for Java

Nachrichten werden durch die MQMessage-Klasse dargestellt. Nachrichten werden mithilfe der Methoden der MQDestination-Klasse eingereiht und abgerufen. Diese Klasse verfügt über die Unterklassen MQQueue und MQTopic.

Mit der Methode put() der MQDestination-Klasse können Sie Nachrichten in Warteschlangen oder Themen einreihen. Mit der Methode get() der MQDestination-Klasse können Sie Nachrichten aus Warteschlangen oder Themen abrufen. Im Gegensatz zur prozedurgesteuerten Schnittstelle, bei der mit MQPUT und MQGET Byte-Arrays eingereiht und abgerufen werden, reiht die Java-Programmiersprache Instanzen der MQMessage-Klasse ein und ruft diese ab. Die MQMessage-Klasse umfasst den Datenpuffer, der die eigentlichen Nachrichtendaten sowie alle MQMD-Parameter (MQMD = Nachrichtendeskriptor) sowie Nachrichteneigenschaften enthält, die diese Nachricht beschreiben.

Wenn Sie eine neue Nachricht erstellen möchten, erstellen Sie eine neue Instanz der MQMessage-Klasse und verwenden Sie die writeXXX-Methoden, um Daten in den Nachrichtenpuffer zu schreiben.

Bei der Erstellung der neuen Nachrichteninstanz werden alle MQMD-Parameter automatisch auf ihre Standardwerte gesetzt, wie in Anfangswerte und Sprachdeklarationen für MQMD definiert. Die put()-Methode von MQDestination hat als weiteren Parameter eine Instanz der MQPutMessageOptions-Klasse.

Diese Klasse stellt die MQPMO-Struktur dar. Im folgenden Beispiel wird eine Nachricht erstellt und in eine Warteschlange eingereiht:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.put(myMessage,pmo);
```

Die get()-Methode von MQDestination gibt eine neue Instanz von MQMessage zurück, die für die Nachricht steht, die gerade aus der Warteschlange abgerufen wurde. Sie hat als weiteren Parameter eine Instanz der MQGetMessageOptions-Klasse. Diese Klasse stellt die MQGMO-Struktur dar.

Sie müssen keine maximale Nachrichtenlänge angeben, da die get()-Methode die Größe des internen Puffers automatisch an die ankommende Nachricht anpasst. Verwenden Sie die readXXX-Methoden der MQMessage-Klasse, um auf die Daten in der Antwortnachricht zuzugreifen.

Das folgende Beispiel zeigt, wie eine Nachricht aus einer Warteschlange abgerufen wird:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo); // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

Sie können das von den Lese- und Schreibmethoden (read und write) verwendete Zahlenformat ändern, indem Sie die Elementvariable *encoding* entsprechend festlegen.

Sie können den Zeichensatz ändern, der für Lese- und Schreibzeichenfolgen verwendet wird, indem Sie die Elementvariable *characterSet* entsprechend festlegen.

Weitere Informationen finden Sie unter MQMessage class.

Anmerkung: Die writeUTF()-Methode von MQMessage verschlüsselt automatisch die Länge der Zeichenfolge und die enthaltenen Unicode-Bytes. Wenn Ihre Nachricht von einem anderen Java-Programm gelesen wird (unter Verwendung von readUTF()), ist dies die einfachste Art, Zeichenfolgeinformationen zu senden.

Leistung von nicht persistenten Nachrichten in IBM MQ classes for Java verbessern
Zur Verbesserung der Leistung beim Durchsuchen von Nachrichten oder Verarbeiten von nicht persistenten Nachrichten aus einer Clientanwendung können Sie die Funktion *Vorauslesen* verwenden. Clientanwendungen, die MQGET oder eine asynchrone Verarbeitung verwenden, profitieren von den Leistungsverbesserungen, wenn sie Nachrichten durchsuchen oder nicht persistente Nachrichten verarbeiten.

Allgemeine Informationen zur Funktion 'Vorauslesen' finden Sie im zugehörigen Abschnitt.

In IBM MQ classes for Java bestimmen Sie mithilfe der Eigenschaften CMQC.MQSO_READ_AHEAD und CMQC.MQSO_NO_READ_AHEAD eines MQQueue- oder MQTopic-Objekts, ob Nachrichtenkonsumenten und Warteschlangenbrowser das Vorauslesen für dieses Objekt verwenden dürfen.

Nachrichten asynchron unter Verwendung von IBM MQ classes for Java einreihen
Wenn Sie eine Nachricht asynchron einreihen möchten, legen Sie MQPMO_ASYNC_RESPONSE fest.

Mit der Methode put() der MQDestination-Klasse können Sie Nachrichten in Warteschlangen oder Themen einreihen. Wenn Sie eine Nachricht asynchron einreihen möchten (wenn die Operation also abgeschlossen werden kann, ohne dass auf eine Antwort vom Warteschlangenmanager gewartet werden muss), können Sie MQPMO_ASYNC_RESPONSE im Optionsfeld von MQPutMessageOptions festlegen. Mit dem Aufruf MQQueueManager.getAsyncStatus können Sie den Erfolg oder das Fehlschlagen von asynchronen Einreihungen überprüfen.

Publish/Subscribe in IBM MQ classes for Java

In IBM MQ classes for Java wird das Thema durch die MQTopic-Klasse dargestellt. Mit den Methoden MQTopic.put() können Sie Veröffentlichungen im Thema vornehmen.

Allgemeine Informationen zur IBM MQ-Funktionalität für Publish/Subscribe finden Sie unter Publish/Subscribe-Messaging.

Behandlung von IBM MQ-Nachrichtenheadern mit IBM MQ classes for Java

Es werden Java-Klassen zur Verfügung gestellt, die verschiedene Arten von Nachrichtenheadern darstellen. Darüber hinaus stehen zwei Helper-Klassen zur Verfügung.

MQHeader-Schnittstelle

Headerobjekte werden durch die MQHeader-Schnittstelle beschrieben, die vielseitig einsetzbare Methoden für den Zugriff auf Headerfelder und für das Lesen und Schreiben von Nachrichteninhalten bereitstellt. Jeder Headertyp hat eine eigene Klasse, die die Schnittstelle 'MQHeader' implementiert und Getter- und Setter-Methoden für einzelne Felder hinzufügt. Zum Beispiel wird der MQRFH2-Headertyp durch die MQRFH2-Klasse dargestellt, der MQDLH-Headertyp durch die MQDLH-Klasse usw. Die Headerklassen führen automatisch die erforderliche Datenkonvertierung durch und können Daten in allen angegebenen numerischen Codierungen und Zeichensätzen (CCSID) lesen oder schreiben.

Wichtig: Die MQRFH2-Header-Klassen behandeln die Nachricht wie eine Datei mit wahlfreiem Zugriff, d. h. der Cursor muss am Anfang der Nachricht positioniert werden. Stellen Sie vor der Verwendung einer internen Nachrichtenheaderklasse wie MQRFH, MQRFH2, MQCIH, MQDEAD, MQIIH oder MQXMIT sicher, dass Sie die Cursorposition der Nachricht an die richtige Position aktualisieren, bevor Sie die Nachricht an die Klasse übergeben.

Helper-Klassen

Zwei Helper-Klassen, MQHeaderIterator und MQHeaderList, unterstützen Sie beim Lesen und Decodieren (Parsen) des Headerinhalts in Nachrichten:

- Die MQHeaderIterator-Klasse funktioniert wie ein java.util.Iterator. Solange weitere Header in der Nachricht enthalten sind, gibt die Methode next() den Wert 'true' zurück, während die Methode nextHeader() oder next() das nächste Headerobjekt zurückgibt.
- MQHeaderList funktioniert wie java.util.List. Wie auch beim MQHeaderIterator wird der Headerinhalt geparst, Sie können aber auch nach bestimmten Headern suchen, neue Header hinzufügen, bereits vorhandene Header entfernen, Headerfelder aktualisieren und den Headerinhalt anschließend wieder in eine Nachricht schreiben. Alternativ können Sie eine leere Klasse 'MQHeaderList' erstellen, dann mit Headerinstanzen füllen und die Klasse 'MQHeaderList' danach einmal oder mehrfach in eine Nachricht schreiben.

Die MQHeaderIterator- und MQHeaderList-Klassen ermitteln mithilfe der Informationen in der MQHeaderRegistry, welche IBM MQ-Headerklassen bestimmten Nachrichtentypen und -formaten zugeordnet sind. Die MQHeaderRegistry wird mit allen bekannten aktuellen IBM MQ-Formaten und -Headertypen sowie mit deren Implementierungsklassen konfiguriert. Sie können zudem Ihre eigenen Headertypen registrieren.

Die folgenden gängigen IBM MQ-Header werden unterstützt:

- MQRFH - Header für Regeln und Formatierung

- MQRFH2 - Wie MQRFH; damit werden Nachrichten an einen Nachrichtenbroker und von einem Nachrichtenbroker übergeben, der zu IBM Integration Bus gehört. Wird auch für das Speichern von Nachrichteneigenschaften verwendet.
- MQCIH- CICS -Bridge
- MQDLH - Header einer nicht zustellbaren Nachricht
- MQIIH - IMS-Informationsheader
- MQRMH - Referenznachrichtenheader
- MQSAPH - SAP-Header
- MQWIH - Auslastungs-Header
- MQXQH - Header der Übertragungswarteschlange
- MQDH - Verteilerheader
- MQEPH - Eingebundener PCF-Header

Sie können auch Klassen definieren, die Ihre eigenen Header darstellen.

Wenn Sie einen MQHeaderIterator für den Abruf eines RFH2-Headers verwenden möchten, legen Sie entweder MQGMO_PROPERTIES_FORCE_MQRFH2 in GetMessageOptions fest oder setzen Sie die Warteschlangeneigenschaft PROPCTL auf FORCE.

Alle Header in einer Nachricht unter Verwendung von IBM MQ classes for Java ausgeben
In diesem Beispiel parst eine Instanz von MQHeaderIterator die Header in einer MQMessage, die von einer Warteschlange empfangen wurde. Die von der nextHeader()-Methode zurückgegebenen MQHeader-Objekte zeigen ihre Struktur und Inhalte an, wenn ihre toString-Methode aufgerufen wird.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();

    System.out.println ("Header type " + header.type () + ": " + header);
}
```

Header in einer Nachricht unter Verwendung von IBM MQ classes for Java überspringen
In diesem Beispiel setzt die Methode skipHeaders() von MQHeaderIterator den Nachrichtenlesecursor direkt hinter den letzten Header.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```

Ursachencode in einer nicht zustellbaren Nachricht unter Verwendung von IBM MQ classes for Java suchen
In diesem Beispiel füllt die Lesemethode das MQDLH-Objekt mit den in der Nachricht gelesenen Werten. Nach der Leseoperation wird der Nachrichtenlesecursor direkt hinter dem MQDLH-Headerinhalt positioniert.

Die Nachrichten in der Warteschlange für nicht zustellbare Nachrichten, die dem Warteschlangenmanager zugeordnet ist, erhalten als Präfix einen Header einer nicht zustellbaren Nachricht (MQDLH). Um entscheiden zu können, wie mit diesen Nachrichten verfahren werden soll (ob sie beispielsweise erneut versucht

oder gelöscht werden sollen), muss eine Anwendung zur Behandlung von nicht zustellbaren Nachrichten den im MQDLH enthaltenen Ursachencode überprüfen.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

Sämtliche Headerklassen stellen zudem einen praktischen Konstruktor zur Verfügung, mit dem sie in nur einem Schritt direkt aus der Nachricht heraus initialisiert werden können. Der Code in diesem Beispiel könnte also wie folgt vereinfacht werden:

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);

System.out.println ("Reason: " + dlh.getReason ());
```

Header einer nicht zustellbaren Nachricht mithilfe von IBM MQ classes for Java lesen und entfernen
In diesem Beispiel wird MQDLH verwendet, um den Header aus einer nicht zustellbaren Nachricht zu entfernen.

Eine Anwendung zur Behandlung von nicht zustellbaren Nachrichten schickt in der Regel abgelehnte Nachrichten erneut ab, wenn ihr Ursachencode auf einen temporären Fehler hinweist. Vor dem erneuten Abschicken der Nachricht muss die Anwendung den MQDLH-Header entfernen.

Im vorliegenden Beispiel werden die folgenden Schritte ausgeführt (lesen Sie die Kommentare im Beispielcode):

1. MQHeaderList liest die gesamte Nachricht und jeder Header, der in der Nachricht gefunden wird, wird als Eintrag in der Liste aufgeführt.
2. Da nicht zustellbare Nachrichten als ersten Header einen MQDLH enthalten, befindet sich dieser in der Headerliste an erster Stelle. Der MQDLH wurde bei der Erstellung der MQHeaderList bereits mit Werten aus der Nachricht gefüllt, die zugehörige Lesemethode muss also nicht aufgerufen werden.
3. Der Ursachencode wird mithilfe der Methode getReason() extrahiert, die von der MQDLH-Klasse bereitgestellt wird.
4. Der Ursachencode wurde überprüft und er weist darauf hin, dass die Nachricht erneut abgeschickt werden kann. Der MQDLH wird unter Verwendung der MQHeaderList-Methode remove() entfernt.
5. Die MQHeaderList schreibt den zugehörigen verbleibenden Inhalt in ein neues Nachrichtenobjekt. Die neue Nachricht enthält jetzt alle Inhalte der ursprünglichen Nachricht außer dem MQDLH und kann in eine Warteschlange geschrieben werden. Das an den Konstruktor und die Schreibmethode übergebene Argument **true** gibt an, dass der Nachrichtenhauptteil in der MQHeaderList gespeichert und erneut als Ausgabe geschrieben werden soll.
6. Das Formatfeld im Nachrichtendeskriptor der neuen Nachricht enthält jetzt den Wert, der zuvor im MQDLH-Formatfeld enthalten war. Die Nachrichtendaten stimmen mit der numerischen Codierung und der CCSID-Einstellung im Nachrichtendeskriptor überein.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
```

```
...
list.remove (dlh); // Step 4.

MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

Inhalt einer Nachricht unter Verwendung von IBM MQ classes for Java ausgeben
In diesem Beispiel wird mithilfe von MQHeaderList der Inhalt einer Nachricht einschließlich der zugehörigen Header ausgegeben.

Die Ausgabe enthält eine Ansicht des gesamten Headerinhalts sowie des Nachrichtentexts. Die MQHeaderList-Klasse decodiert alle Header in einem Schritt, während der MQHeaderIterator die Header unter Anwendungssteuerung schrittweise einzeln durchgeht. Sie könnten dieses Verfahren verwenden, um ein einfaches Debugging-Tool bereitzustellen, wenn Sie WebSphere MQ-Anwendungen schreiben.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

In diesem Beispiel werden darüber hinaus die Nachrichtendeskriptorfelder unter Verwendung der MQMD-Klasse ausgegeben. Die Methode copyFrom() der Klasse com.ibm.mq.headers.MQMD füllt das Headerobjekt auf Basis der Nachrichtendeskriptorfelder der MQMessage, statt den Nachrichtenhauptteil zu lesen.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

Bestimmten Headertyp in einer Nachricht unter Verwendung von IBM MQ classes for Java suchen
In diesem Beispiel wird die Methode indexOf(String) von MQHeaderList verwendet, um einen eventuell vorhandenen MQRFH2-Header in einer Nachricht zu suchen.

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;
...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
        ...
}
```

MQRFH2-Header unter Verwendung von IBM MQ classes for Java analysieren
In diesem Beispiel wird der Zugriff auf einen bekannten Feldwert in einem namentlich genannten Ordner unter Verwendung der MQRFH2-Klasse veranschaulicht.

Die MQRFH2-Klasse bietet verschiedene Möglichkeiten, mit denen nicht nur auf die Felder im festen Teil der Struktur, sondern auch auf die Inhalte des XML-codierten Ordners zugegriffen werden kann. Sie werden im Feld NameValueData ausgeführt. Dieses Beispiel zeigt, wie auf einen bekannten Feldwert in einem namentlich genannten Ordner (in diesem Fall auf das Rto-Feld im Ordner 'jms') zugegriffen wird, der den Namen der Antwortwarteschlange in einer MQ-JMS-Nachricht darstellt.

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

Für die Erkennung des Inhalts eines MQRFH2 (im Gegensatz zur direkten Anforderung bestimmter Feld) können Sie die getFolders-Methode verwenden, mit der eine MQRFH2.Element-Liste zurückgegeben wird. Diese stellt die Struktur eines Ordners dar, die Felder und sonstige Ordner enthalten kann. Wenn ein Feld oder ein Ordner auf null gesetzt ist, wird das Feld bzw. der Ordner aus dem MQRFH2 entfernt. Wenn Sie den NameValueData-Ordner auf diese Weise bearbeiten, wird das StrucLength-Feld automatisch entsprechend aktualisiert.

Andere Byteströme als MQMessage-Objekte unter Verwendung von IBM MQ classes for Java lesen und schreiben
In diesen Beispielen werden die Headerklassen zum Parsen und Bearbeiten des IBM MQ-Headerinhalts verwendet, wenn es sich bei der Datenquelle nicht um ein MQMessage-Objekt handelt.

Sie können die Headerklassen selbst dann für das Parsen und Bearbeiten von IBM MQ-Headerinhalten verwenden, wenn die Datenquelle kein MQMessage-Objekt ist. Die von jeder Headerklasse implementierte MQHeader-Schnittstelle stellt die Methoden`int read (java.io.DataInput message, int encoding, int characterSet)` und `int write (java.io.DataOutput message, int encoding, int characterSet)` zur Verfügung. Die Klasse com.ibm.mq.MQMessage class implementiert die Schnittstellen java.io.DataInput und java.io.DataOutput. Dies bedeutet, dass Sie den MQMessage-Inhalt mithilfe der beiden MQHeader-Methoden lesen und schreiben können, wobei die im Nachrichtendeskriptor angegebenen Werte für Codierung und CCSID überschrieben werden. Dies ist bei Nachrichten hilfreich, die eine Kette von Headern in verschiedenen Codierungen enthalten.

Sie können auch DataInput- und DataOutput-Objekte aus anderen Datenströmen anfordern, zum Beispiel aus Datei- oder Socketdatenströmen oder Byte-Arrays, die in JMS-Nachrichten ausgeführt werden. Die java.io.DataInputStream-Klassen implementieren DataInput, während die java.io.DataOutputStream-Klassen DataOutput implementieren. In diesem Beispiel wird der IBM MQ-Headerinhalt aus einem Byte-Array gelesen:

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
  CMQC.MQCCSI_DEFAULT);
```

Die Zeile, die mit MQHeaderIterator beginnt, kann durch

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

In diesem Beispiel werden Daten unter Verwendung eines Datenausgabestroms (DataOutputStream) in ein Byte-Array geschrieben:

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

Wenn Sie auf diese Weise mit Datenströmen arbeiten, müssen Sie darauf achten, dass Sie die richtigen Werte für die Argumente 'encoding' und 'characterSet' (Codierung und Zeichensatz) verwenden. Geben Sie beim Lesen von Headern die Codierung und die ID des codierten Zeichensatzes (CCSID) an, in denen der Byteinhalt ursprünglich geschrieben wurde. Geben Sie beim Schreiben von Headern die Codierung und die CCSID an, die Sie erstellen möchten. Die Datenkonvertierung wird automatisch von den Headerklassen durchgeführt.

Klassen für neue Headertypen unter Verwendung von IBM MQ classes for Java erstellen
Sie können Java-Klassen für Headertypen erstellen, die nicht mit IBM MQ classes for Java bereitgestellt werden.

Wenn Sie eine Java-Klasse hinzufügen möchten, die einen neuen Headertyp darstellt und die Sie genau wie eine mit den IBM MQ classes for Java bereitgestellte Headerklasse verwenden können, müssen Sie eine Klasse erstellen, die die MQHeader-Schnittstelle implementiert. Dies geht am einfachsten, indem Sie die Klasse com.ibm.mq.headers.impl.Header erweitern. In diesem Beispiel wird eine voll funktionsfähige Klasse erstellt, die die MQTM-Headerstruktur darstellt. Sie müssen zwar nicht für jedes Feld einzelne Getter- und Setter-Methoden hinzufügen, dies ist für die Benutzer der Headerklasse jedoch hilfreich. Die generischen getValue- und setValue-Methoden, die eine Zeichenfolge für den Feldnamen übernehmen, funktionieren für alle Felder, die im Headertyp definiert sind. Die übernommenen Methoden für Lese- und Schreibvorgänge sowie die Methode für die Größe ermöglichen es, dass Instanzen des neuen Headertyps gelesen und geschrieben werden können. Darüber hinaus kann die Headergröße damit auf Basis der zugehörigen Felddefinition ordnungsgemäß berechnet werden. Die Typdefinition wird lediglich einmal erstellt, allerdings werden viele Instanzen dieser Headerklasse erstellt. Damit die neue Headerdefinition für die Decodierung unter Verwendung der MQHeaderIterator- oder MQHeaderList-Klassen verfügbar ist, müssen Sie sie mithilfe von MQHeaderRegistry registrieren. Hierbei müssen Sie allerdings beachten, dass die MQTM-Headerklasse tatsächlich bereits in diesem Paket bereitgestellt und in der Standardregistry registriert wird.

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
      CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
      CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);

    protected MQTM (HeaderType type){
        super (type);
    }
    public String getStrucId () {
        return getStringValue (StrucId);
    }
    public int getVersion () {
        return getIntValue (Version);
    }
    public String getQName () {
        return getStringValue (QName);
    }
    public void setQName (String value) {
        setStringValue (QName, value);
    }
    // ...Add convenience getters and setters for remaining fields in the same way.
}
```

Behandlung von PCF-Nachrichten mit IBM MQ classes for Java
Java-Klassen werden für die Erstellung und das Parsing von PCF-strukturierten Nachrichten zur Verfügung gestellt. Darüber hinaus erleichtern sie das Versenden von PCF-Anforderungen und das Sammeln von PCF-Antworten.

Die Klassen PCFMessage & MQCFGR stellen Arrays von PCF-Parameterstrukturen dar. Sie bieten Methoden, mit denen PCF-Parameter einfacher hinzugefügt und abgerufen werden können.

Die PCF-Parameterstrukturen werden durch die Klassen MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL und MQCFGR dargestellt. Diese nutzen gemeinsam grundlegende operative Schnittstellen:

- Methoden für das Lesen und Schreiben von Nachrichteninhalten: read (), write () und size ()
- Methoden für das Bearbeiten von Parametern: getValue (), setValue (), getParameter () und weitere Methoden
- Die Aufzählungsausdruckmethode .nextParameter (), die PCF-Inhalte in einer MQMessage parst

Der PCF-Filterparameter wird in Abfragebefehlen als Filterfunktion verwendet. Er wird in folgende Klassen eingebunden:

- MQCFIF - Ganzzahlfilter
- MQCFSF - Zeichenfolgefilter
- MQCFBF - Bytefilter

Die beiden Agentenklassen PCFAgent und PCFMessageAgent werden für die Verwaltung der Verbindung mit einem Warteschlangenmanager, der Befehlsserverwarteschlange und einer zugehörigen Antwortwarteschlange bereitgestellt. PCFMessageAgent ist eine Erweiterung von PCFAgent und sollte normalerweise bevorzugt verwendet werden. Die PCFMessageAgent-Klasse konvertiert die empfangenen MQ-Nachrichten (MQMessages) und übergibt sie als PCFMessage-Array an das aufrufende Modul zurück. PCFAgent gibt ein Array von MQMessages zurück, die Sie vor der Verwendung parsen müssen.

Handhabung von Nachrichteneigenschaften in IBM MQ classes for Java

Für Funktionsaufrufe zur Verarbeitung von Nachrichtenhandles gibt es in den IBM MQ classes for Java keine funktionale Entsprechung. Verwenden Sie die Methoden der MQMessage-Klasse, um die Eigenschaften von Nachrichtenhandles festzulegen, zurückzugeben oder zu löschen.

Allgemeine Informationen zu Nachrichteneigenschaften finden Sie unter „Eigenschaftsnamen” auf Seite 29.

In IBM MQ classes for Java erfolgt der Zugriff auf Nachrichten über die MQMessage-Klasse. Nachrichtenhandles werden daher nicht in der Java-Umgebung bereitgestellt und es gibt keine funktionale Entsprechung für die IBM MQ-Funktionsaufrufe MQCRTMH, MQDLTMH, MQMHBUF und MQBUFMH.

Zur Festlegung der Eigenschaften von Nachrichtenhandles in der prozedurgesteuerten Schnittstelle verwenden Sie den Aufruf MQSETMP. In IBM MQ classes for Java müssen Sie die entsprechende Methode der MQMessage-Klasse verwenden:

- setBooleanProperty
- setByteProperty
- setBytesProperty
- setShortProperty
- setIntProperty
- setInt2Property
- setInt4Property
- setInt8Property
- setLongProperty
- setFloatProperty
- setDoubleProperty
- setStringProperty
- setObjectProperty

Diese werden gelegentlich auch gesammelt als *set*property*-Methoden bezeichnet.

Zur Rückgabe des Werts der Eigenschaften von Nachrichtenhandles in der prozedurgesteuerten Schnittstelle verwenden Sie den Aufruf MQINQMP. In IBM MQ classes for Java müssen Sie die entsprechende Methode der MQMessage-Klasse verwenden:

- getBooleanProperty
- getByteProperty

- getBytesProperty
- getShortProperty
- getIntProperty
- getInt2Property
- getInt4Property
- getInt8Property
- getLongProperty
- getFloatProperty
- getDoubleProperty
- getStringProperty
- getObjectProperty

Diese werden gelegentlich auch gesammelt als *get*property*-Methoden bezeichnet.

Zum Löschen des Werts der Eigenschaften von Nachrichtenhandles in der prozedurgesteuerten Schnittstelle verwenden Sie den Aufruf MQDLTMP. In IBM MQ classes for Java müssen Sie die deleteProperty-Methode der MQMessage-Klasse verwenden.

Fehlerbehandlung in IBM MQ classes for Java

Fehler, die aufgrund von IBM MQ classes for Java auftreten, werden mit den Blöcken Java `try` und `catch` behandelt.

Die Methoden in der Java-Schnittstelle geben keinen Beendigungs- und Ursachencode zurück. Stattdessen lösen sie eine Ausnahmebedingung aus, wenn der Beendigungs- und Ursachencode eines IBM MQ-Aufrufs nicht beide null sind. Dadurch wird die Programmlogik vereinfacht, sodass Sie nicht nach jedem Aufruf an IBM MQ die Rückgabecodes prüfen müssen. Sie können entscheiden, an welchen Punkten Sie in Ihrem Programm die Möglichkeit von Fehlern zulassen möchten. An diesen Punkten können Sie Ihren Code in `try` -und `catch` -Blöcke einschließen, wie im folgenden Beispiel gezeigt:

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
                       "CC = " + ex.completionCode +
                       "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

Die IBM MQ -Aufrufursachencodes, die in Java Ausnahmebedingungen für z/OS zurückgemeldet werden, sind in API-Beendigungs-und Ursachencodesdokumentiert.

Ausnahmebedingungen, die während der Ausführung einer Anwendung der IBM MQ classes for Java ausgelöst werden, werden ebenfalls in das Protokoll geschrieben. Eine Anwendung kann jedoch die Methode MQException.logExclude() aufrufen, um die Protokollierung von Ausnahmebedingungen zu verhindern, die einem bestimmten Ursachencode zugeordnet sind. Dies kann in Situationen sinnvoll sein, in denen Sie bereits erwarten, dass viele Ausnahmebedingungen im Zusammenhang mit einem bestimmten Ursachencode ausgelöst werden. So können Sie vermeiden, dass das Protokoll mit diesen Ausnahmebedingungen überfrachtet wird. Wenn Ihre Anwendung beispielsweise versucht, bei jeder Iteration in einer Schleife eine Nachricht aus einer Warteschlange abzurufen und Sie bei den meisten dieser Versuche nicht davon ausgehen, dass die Warteschlange eine geeignete Nachricht enthält, können Sie verhindern, dass Ausnahmebedingungen im Zusammenhang mit dem Ursachencode MQRC_NO_MSG_AVAILABLE protokolliert werden. Falls eine Anwendung zuvor die Protokollierung von Ausnahmebedingungen, die einem bestimmten Ursachencode zugeordnet sind, verhindert hatte, kann sie die Protokollierung dieser Ausnahmebedingungen durch den Aufruf der Methode MQException.logInclude() erneut zulassen.

Manchmal enthält der Ursachencode nicht alle Details im Zusammenhang mit dem Fehler. Bei jeder Ausnahmebedingung, die ausgelöst wird, sollte eine Anwendung die verlinkte Ausnahmebedingung prüfen. Die verlinkte Ausnahmebedingung kann wiederum ihrerseits eine weitere verlinkte Ausnahmebedingung haben, so dass die verlinkten Ausnahmebedingungen eine Kette bilden, die zu dem zugrunde liegenden Problem zurückführt. Eine verlinkte Ausnahmebedingung wird über den Verkettungsmechanismus für Ausnahmebedingungen der Klasse java.lang.Throwable implementiert. Eine Anwendung erhält eine verlinkte Ausnahmebedingung durch Aufrufen der Methode Throwable.getCause(). Aus einer Ausnahmebedingung, die eine Instanz von MQException ist, ruft MQException.getCause() die zugrunde liegende Instanz von com.ibm.mq.jmqi.JmqiException ab und getCause aus dieser Ausnahmebedingung ruft wiederum die zugrunde liegende Ausnahmebedingung (java.lang.Exception) ab, die den Fehler verursacht hat.

Attributwerte in IBM MQ classes for Java abrufen und festlegen

Die Methoden getXXX() und setXXX() werden für viele gängige Attribute zur Verfügung gestellt. Andere Attribute sind über die generischen Methoden inquire() und set() zugänglich.

Die Klassen MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess und MQQueueManager enthalten für viele der gängigen Attribute die Methoden getXXX() und setXXX(). Diese Methoden ermöglichen es Ihnen, ihre Attributwerte abzurufen und festzulegen. Beachten Sie, dass die Methoden für MQDestination, MQQueue und MQTopic nur dann funktionieren, wenn Sie beim Öffnen des Objekts die entsprechenden Inquire- und Set-Flags angeben.

Bei weniger gängigen Attributen übernehmen die Klassen MQQueueManager, MQDestination, MQQueue, MQTopic und MQProcess alle die Werte aus einer Klasse mit dem Namen 'MQManagedObject'. Diese Klasse definiert die inquire()- und set()-Schnittstellen.

Wenn Sie ein neues Warteschlangenmanagerobjekt mithilfe des Operators *new* erstellen, wird es automatisch für die Abfrage (inquire) geöffnet. Falls Sie mit der Methode accessProcess() auf ein Prozessobjekt zugreifen, wird dieses Objekt automatisch für die Abfrage (inquire) geöffnet. Wenn Sie die Methode accessQueue() verwenden, um auf ein Warteschlangenobjekt zuzugreifen, wird dieses Objekt nicht automatisch für eine Inquire- oder eine Set-Operation geöffnet. Der Grund hierfür ist, dass beim automatischen Hinzufügen dieser Optionen Probleme mit einigen Arten ferner Warteschlangen auftreten können. Wenn Sie die Methoden inquire, set, getXXX und setXXX für eine Warteschlange verwenden möchten, müssen Sie die entsprechenden Inquire- und Set-Flags im Parameter openOptions der Methode accessQueue() angeben. Dies gilt auch für Ziel- und Topic-Objekte.

Die Inquire- und Set-Methoden enthalten drei Parameter:

- selectors-Array
- intAttrs-Array
- charAttrs-Array

Sie benötigen die in MQINQ enthaltenen Parameter SelectorCount, IntAttrCount und CharAttrLength nicht, da die Länge eines Arrays in Java immer bekannt ist. Das folgende Beispiel zeigt, wie in einer Warteschlange eine Abfrage durchgeführt wird:

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Multithread-Programme in Java

Die Java-Laufzeitumgebung wird grundsätzlich als Multithread-Prozess ausgeführt. IBM MQ classes for Java ermöglichen die gemeinsame Nutzung eines Warteschlangenmanagerobjekts durch mehrere Threads. Dabei wird aber gleichzeitig sichergestellt, dass der gesamte Zugriff auf den Zielwarteschlangenmanager synchronisiert ist.

Multithread-Programme sind in Java nahezu unvermeidbar. Stellen Sie sich ein einfaches Programm vor, das eine Verbindung zu einem Warteschlangenmanager herstellt und beim Systemstart eine Warteschlange öffnet. Das Programm zeigt eine einzige Schaltfläche auf dem Bildschirm an. Wenn ein Benutzer auf diese Schaltfläche klickt, ruft das Programm eine Nachricht aus der Warteschlange ab.

Die Java-Laufzeitumgebung wird grundsätzlich als Multithread-Prozess ausgeführt. Daher läuft die Initialisierung Ihrer Anwendung in einem Thread ab, während der Code, der infolge des Anklickens der Schaltfläche ausgeführt wird, in einem anderen Thread (Thread der Benutzerschnittstelle) abläuft.

Bei dem auf der Programmiersprache C basierenden IBM MQ MQI client wäre dies ein Problem, da die gemeinsame Nutzung von Handles durch mehrere Threads beschränkt wird. IBM MQ classes for Java lockern diese Beschränkung. Dort kann ein Warteschlangenmanagerobjekt (und seine zugehörige Warteschlange sowie seine Topic- und Prozessobjekte) gemeinsam von mehreren Threads genutzt werden.

Die Implementierung der IBM MQ classes for Java sorgt dafür, dass bei einer bestimmten Verbindung (MQQueueManager-Objektinstanz) der gesamte Zugriff auf den IBM MQ-Zielwarteschlangenmanager synchronisiert ist. Ein Thread, der einen Aufruf an einen Warteschlangenmanager ausgeben möchte, wird blockiert, bis alle anderen Aufrufe, die derzeit für diese Verbindung ausgeführt werden, abgeschlossen sind. Wenn Sie simultanen Zugriff auf denselben Warteschlangenmanager von mehreren Threads innerhalb Ihres Programms benötigen, erstellen Sie ein neues MQQueueManager-Objekt für jeden Thread, der gleichzeitigen Zugriff anfordert. (Dies ist mit der Ausgabe eines eigenen MQCONN-Aufrufs für jeden einzelnen Thread gleichzusetzen.)

Anmerkung: Instanzen der Klasse `com.ibm.mq.MQGetMessageOptions` dürfen nicht von mehreren Threads gemeinsam genutzt werden, die gleichzeitig Nachrichten anfordern. Die Instanzen dieser Klasse werden während der entsprechenden MQGET-Anforderung mit Daten aktualisiert. Dies kann unerwartete Folgen haben, wenn mehrere Threads gleichzeitig für dieselbe Instanz des Objekts in Betrieb sind.

Kanalexits in IBM MQ classes for Java verwenden

Hier finden Sie eine Übersicht über die Verwendung von Kanalexits in einer Anwendung mit den IBM MQ classes for Java.

In den folgenden Abschnitten wird beschrieben, wie Sie einen Kanalexit in Java schreiben und diesen zuweisen können. Außerdem erfahren Sie, wie Daten an ihn übergeben werden. Zudem erhalten Sie Informationen zur Verwendung von Kanalexits, die in der Programmiersprache C geschrieben wurden, sowie zur Verwendung einer Folge von Kanalexits.

Ihre Anwendung muss die richtige Sicherheitsberechtigung haben, um die Kanalexitklasse laden zu können.

Kanalexit in IBM MQ classes for Java erstellen

Sie können Ihre eigenen Kanalexits bereitstellen, indem Sie eine Java-Klasse definieren, die eine entsprechende Schnittstelle implementiert.

Zur Implementierung eines Exits definieren Sie eine neue Java-Klasse, die die entsprechende Schnittstelle implementiert. Im Paket com.ibm.mq.exits sind drei Exitschnittstellen definiert:

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

Anmerkung: Kanalexits werden nur für Clientverbindungen unterstützt. Für Bindungsverbindungen werden sie nicht unterstützt. Sie können einen Java-Kanalexit nicht außerhalb der IBM MQ classes for Java verwenden, wenn Sie beispielsweise eine Clientanwendung verwenden, die in der Programmiersprache C geschrieben wurde.

Jede TLS-Verschlüsselung, die für eine Verbindung definiert ist, wird *nach* dem Aufrufen von Sende- und Sicherheitsexits ausgeführt. Ähnlich wird die Entschlüsselung *vor* dem Aufrufen von Empfangs- und Sicherheitsexits ausgeführt.

Das folgende Beispiel definiert eine Klasse, die alle drei Schnittstellen implementiert:

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit interface
  public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                       MQCD channelDefinition,
                                       ByteBuffer agentBuffer)
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                        MQCD channelDefinition,
                                        ByteBuffer agentBuffer)
  {
    // Fill in the body of the security exit here
  }
}
```

An jeden Exit werden ein MQCXP-Objekt und ein MQCD-Objekt übergeben. Diese Objekte stellen die MQCXP- und MQCD-Strukturen dar, die in der prozedurgesteuerten Schnittstelle definiert werden.

Jede Exitklasse, die Sie schreiben, muss über einen Konstruktor verfügen. Dies kann der Standardkonstruktor oder ein Konstruktor sein, der ein Zeichenfolgeargument annimmt. Wenn der Konstruktor eine Zeichenfolge annimmt, werden die Benutzerdaten an die Exitklasse übergeben, wenn diese erstellt wird. Verfügt eine Exitklasse sowohl über einen Standardkonstruktor als auch über einen Einzelargumentkonstruktor, hat der Einzelargumentkonstruktor Priorität.

Für die Sende- und Sicherheitsexits muss Ihr Exitcode die Daten zurückgeben, die Sie an den Server senden möchten. Für einen Empfangsexit muss Ihr Exitcode die geänderten Daten zurückgeben, die von IBM MQ interpretiert werden sollen.

Der einfachste Exithauptteil, der möglich ist, lautet wie folgt:

```
{ return agentBuffer; }
```

Schließen Sie den Warteschlangenmanager nicht aus einem Kanalexit heraus.

Bereits vorhandene Kanalexitklassen verwenden

In Versionen vor IBM MQ 7.0 müssen Sie diese Exits wie im folgenden Beispiel mit den Schnittstellen MQSendExit, MQReceiveExit und MQSecurityExit implementieren. Diese Methode bleibt weiterhin gültig. Die neue Methode wird jedoch aufgrund der verbesserten Funktionalität und Leistung bevorzugt.

```
public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
```

```
                          byte agentBuffer[])
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
    // Fill in the body of the security exit here
  }
}
```

Kanalexit in IBM MQ classes for Java zuweisen
Sie können einen Kanalexit unter Verwendung von IBM MQ classes for Java zuweisen.

Es gibt keine direkte Entsprechung für den IBM MQ-Kanal in IBM MQ classes for Java. Kanalexits werden einem MQQueueManager zugewiesen. Nachdem sie zum Beispiel eine Klasse definiert hat, die die WMQSecurityExit-Schnittstelle implementiert, kann eine Anwendung den Sicherheitsexit auf eine von vier Arten verwenden:

- Durch die Zuweisung einer Instanz der Klasse zum Feld MQEnvironment.channelSecurityExit, bevor ein MQQueueManager-Objekt erstellt wird
- Durch das Setzen des Feldes MQEnvironment.channelSecurityExit auf eine Zeichenfolge, die die Sicherheitsexitklasse darstellt, bevor ein MQQueueManager-Objekt erstellt wird
- Durch die Erstellung eines Schlüssel/Wert-Paars mit dem Schlüssel CMQC.SECURITY_EXIT_PROPERTY in der Hashtabelle mit Eigenschaften, die an MQQueueManager übergeben wird
- Durch die Verwendung einer Definitionstabelle für den Clientkanal (CCDT)

Jeder Exit, der durch das Setzen des Feldes MQEnvironment.channelSecurityExit auf eine Zeichenfolge, durch die Erstellung eines Schlüssel/Wert-Paars in der Hashtabelle mit Eigenschaften oder mithilfe einer CCDT zugewiesen wird, muss mit einem Standardkonstruktor geschrieben werden. Ein Exit, der als Instanz einer Klasse zugewiesen wird, benötigt keinen Standardkonstruktor (je nach Anwendung).

Eine Anwendung kann auf ähnliche Weise einen Sende- oder Empfangsexit verwenden. Im folgenden Codefragment sehen Sie beispielsweise, wie die Sicherheits-, Sende- und Empfangsexits, die in der Klasse MyMQExits implementiert sind, mit MQEnvironment verwendet werden:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

Falls mehrere Methoden für die Zuweisung eines Kanalexits verwendet werden, gilt folgende Ausführungspriorität:

1. Wenn die URL einer CCDT an den MQQueueManager übergeben wird, bestimmen die Inhalte der CCDT die Kanalexits, die verwendet werden sollen. Alle vorhandenen Exitdefinitionen in MQEnvironment oder in der Hashtabelle mit Eigenschaften werden ignoriert.
2. Wenn keine CCDT-URL übergeben wird, werden die Exitdefinitionen aus MQEnvironment und der Hashtabelle zusammengeführt.
   - Wenn der gleiche Exittyp sowohl in MQEnvironment als auch in der Hashtabelle definiert ist, wird die Definition in der Hashtabelle verwendet.
   - Falls funktional entsprechende alte und neue Exittypen angegeben werden (beispielsweise das sendExit-Feld, das nur in Versionen vor IBM WebSphere MQ 7.0 als Exittyp verwendet werden kann,

und das channelSendExit-Feld, das für jeden Sendeexit verwendet werden kann, wird der neue Exit (channelSendExit) gegenüber dem alten Exit bevorzugt verwendet.

Wenn Sie einen Kanalexit als Zeichenfolge deklariert haben, müssen Sie IBM MQ die Suche nach dem Kanalexitprogramm ermöglichen. Hierfür haben Sie verschiedene Möglichkeiten. Diese hängen von der Umgebung ab, in der die Anwendung ausgeführt wird, sowie davon, wie die Kanalexitprogramme gepackt sind.

- Bei einer Anwendung, die in einem Anwendungsserver ausgeführt wird, müssen Sie die Dateien in dem in Tabelle 58 auf Seite 413 genannten Verzeichnis speichern oder diese in JAR-Dateien packen, die von **exitClasspath** referenziert werden.
- Bei einer Anwendung, die nicht in einem Anwendungsserver ausgeführt wird, gelten die folgenden Regeln:
  - Wenn Ihre Kanalexitklassen in separate JAR-Dateien gepackt werden, müssen diese JAR-Dateien in den **exitClasspath** eingeschlossen werden.
  - Wenn Ihre Kanalexitklassen nicht in JAR-Dateien gepackt sind, können die Klassendateien in dem in Tabelle 58 auf Seite 413 genannten Verzeichnis oder in einem beliebigen Verzeichnis im Klassenpfad des JVM-Systems oder in **exitClasspath** gespeichert werden.

Die Eigenschaft **exitClasspath** kann auf vier Arten angegeben werden. Nach Priorität geordnet haben Sie folgende Möglichkeiten:

1. Mit der Systemeigenschaft com.ibm.mq.exitClasspath (diese wird in der Befehlszeile mit der Option `-D` definiert)
2. Mit der Zeilengruppe 'exitPath' der Datei `mqclient.ini`
3. Über einen Hashtabelleneintrag mit dem Schlüssel CMQC.EXIT_CLASSPATH_PROPERTY
4. Mit der MQEnvironment-Variablen **exitClasspath**

Mehrere Pfade müssen mithilfe des java.io.File.pathSeparator-Zeichens voneinander getrennt werden.

Tabelle 58. Verzeichnis für Kanalexitprogramme

Plattform	Directory
AIX AIX Linux Linux	`/var/mqm/exits` (32-Bit-Kanalexitprogramme) `/var/mqm/exits64` (64-Bit-Kanalexitprogramme)
Windows	*Installationsdatenverzeichnis* \exits

Anmerkung: *Installationsdatenverzeichnis* steht für das Verzeichnis, das Sie während der Installation für die IBM MQ-Datendateien gewählt haben. Das Standardverzeichnis ist `C:\ProgramData \IBM \MQ`.

Daten an Kanalexits in IBM MQ classes for Java übergeben
Sie können Daten an Kanalexits übergeben und Daten aus Kanalexits an Ihre Anwendung zurückgeben.

Parameter 'agentBuffer'

Bei einem Sendeexit enthält der Parameter *agentBuffer* die Daten, die in Kürze gesendet werden. Bei einem Empfangsexit oder einem Sicherheitsexit enthält der Parameter *agentBuffer* die soeben empfangenen Daten. Sie benötigen keinen Längenparameter, da der Ausdruck agentBuffer.limit() die Länge des Arrays angibt.

Für die Sende- und Sicherheitsexits muss Ihr Exitcode die Daten zurückgeben, die Sie an den Server senden möchten. Für einen Empfangsexit muss Ihr Exitcode die geänderten Daten zurückgeben, die von IBM MQ interpretiert werden sollen.

Der einfachste Exithauptteil, der möglich ist, lautet wie folgt:

```
{ return agentBuffer; }
```

Kanalexits werden mit einem Puffer aufgerufen, der über ein Sicherungsarray verfügt. Zur Leistungsoptimierung sollte der Exit einen Puffer mit einem Sicherungsarray liefern.

Benutzerdaten

Wenn eine Anwendung durch die Festlegung von channelSecurityExit, channelSendExit oder channelReceiveExit eine Verbindung zu einem Warteschlangenmanager herstellt, können 32 Bytes der Benutzerdaten beim Aufruf unter Verwendung der Felder channelSecurityExitUserData, channelSendExitUserData oder channelReceiveExitUserData an die entsprechende Kanalexitklasse übergeben werden. Diese Benutzerdaten sind für die Kanalexitklasse verfügbar, werden jedoch bei jedem Aufruf des Exits aktualisiert. Daher gehen sämtliche Änderungen, die an den Benutzerdaten im Kanalexit vorgenommen wurden, verloren. Wenn Sie permanente Änderungen an Daten in einem Kanalexit vornehmen möchten, verwenden Sie den MQCXP-Parameter exitUserArea. Die Daten in diesem Feld bleiben auch bei mehrmaligen Aufrufen des Exits erhalten.

Wenn die Anwendung securityExit, sendExit oder receiveExit festlegt, können keine Benutzerdaten an diese Kanalexitklassen übergeben werden.

Wenn eine Anwendung eine Definitionstabelle für den Clientkanal (CCDT) verwendet, um sich mit einem Warteschlangenmanager zu verbinden, werden die in einer Definition des Clientverbindungskanals angegebenen Benutzerdaten an Kanalexitklassen übergeben, wenn sie aufgerufen werden. Weitere Informationen zur Verwendung einer Definitionstabelle für den Clientkanal finden Sie unter „Definitionstabelle für den Clientkanal mit IBM MQ classes for Java verwenden" auf Seite 395.

Kanalexits, die nicht in Java geschrieben wurden, mit IBM MQ classes for Java verwenden
Hier erfahren Sie, wie Kanalexitprogramme, die in der Programmiersprache C geschrieben wurden, in Verbindung mit einer Java-Anwendung verwendet werden.

In IBM MQ können Sie im MQEnvironment-Objekt oder in der Hashtabelle mit Eigenschaften den Namen eines in der Programmiersprache C geschriebenen Kanalexitprogramms als Zeichenfolge angeben, die an die Felder channelSecurityExit, channelSendExit oder channelReceiveExit übergeben wird. Sie können einen Kanalexit, der in Java geschrieben wurde, jedoch nicht in einer Anwendung verwenden, die in einer anderen Sprache geschrieben wurde.

Geben Sie den Exitprogrammnamen im Format `library(function)` an und stellen Sie sicher, dass die Position des Exitprogramms wie unter Pfad zu Exitsbeschrieben angegeben wird.

Informationen zum Schreiben eines Kanalexits in der Programmiersprache C finden Sie unter „Kanalexitprogramme für Messaging-Kanäle" auf Seite 1013.

Folge von Sende- oder Empfangsexits in IBM MQ classes for Java verwenden
Eine Anwendung der IBM MQ classes for Java kann eine Folge von Kanalsende- oder Kanalempfangsexits verwenden, die nacheinander ausgeführt werden.

Zur Verwendung einer Folge von Sendeexits kann eine Anwendung entweder eine Liste oder eine Zeichenfolge mit den Sendeexits erstellen. Falls eine Liste verwendet wird, kann jedes Element der Liste eines der folgenden Elemente sein:

- Eine Instanz einer benutzerdefinierten Klasse, die die WMQSendExit-Schnittstelle implementiert
- Eine Instanz einer benutzerdefinierten Klasse, die die MQSendExit-Schnittstelle implementiert (bei einem Sendeexit, der in Java geschrieben wurde)
- Eine Instanz der MQExternalSendExit-Klasse (bei einem Sendeexit, der nicht in Java geschrieben wurde)
- Eine Instanz der MQSendExitChain-Klasse
- Eine Instanz der Zeichenfolgeklasse

Eine Liste kann keine andere Liste enthalten.

Die Anwendung kann auf ähnliche Weise eine Folge von Empfangsexits verwenden.

Wenn eine Zeichenfolge verwendet wird, muss sie aus einer oder mehreren durch Kommas getrennten Exitdefinitionen bestehen, die jeweils den Namen einer Java -Klasse oder eines C-Programms im Format `library(function)` enthalten können.

Die Anwendung weist dann vor der Erstellung eines MQQueueManager-Objekts das Listen- oder Zeichenfolgeobjekt dem Feld MQEnvironment.channelSendExit zu.

Die an Exits übergebenen Informationen stehen gänzlich im Kontext der Domäne der Exits. Falls beispielsweise ein Java-Exit und ein C-Exit verkettet werden, hat das Vorhandensein des Java-Exits keinerlei Auswirkung auf den C-Exit.

Exitkettenklassen verwenden

In Versionen vor IBM WebSphere MQ 7.0 wurden zwei Klassen bereitgestellt, die Exitfolgen ermöglichten:

- MQSendExitChain zur Implementierung der MQSendExit-Schnittstelle
- MQReceiveExitChain zur Implementierung der MQReceiveExit-Schnittstelle

Diese Klassen können nach wie vor verwendet werden, die neue Methode wird jedoch bevorzugt. Wenn die Schnittstellen der IBM MQ-Klassen für Java verwendet werden, weist Ihre Anwendung nach wie vor eine Abhängigkeit von `com.ibm.mq.jar` auf. Wenn die neue Schnittstellengruppe im Paket `com.ibm.mq.exits` verwendet wird, besteht keine Abhängigkeit von `com.ibm.mq.jar`.

Zur Verwendung einer Folge von Sendeexits erstellte eine Anwendung eine Liste mit Objekten. Dabei handelt es sich bei jedem Objekt um eines der folgenden Elemente:

- Eine Instanz einer benutzerdefinierten Klasse, die die MQSendExit-Schnittstelle implementiert (bei einem Sendeexit, der in Java geschrieben wurde)
- Eine Instanz der MQExternalSendExit-Klasse (bei einem Sendeexit, der nicht in Java geschrieben wurde)
- Eine Instanz der MQSendExitChain-Klasse

Die Anwendung erstellte ein MQSendExitChain-Objekt durch die Übergabe dieser Liste mit Objekten als Parameter im Konstruktor. Anschließend wies die Anwendung vor der Erstellung eines MQQueueManager-Objekts das MQSendExitChain-Objekt dem Feld MQEnvironment.sendExit zu.

Kanalkomprimierung in IBM MQ classes for Java

Durch die Komprimierung der durch einen Kanal fließenden Daten können Sie die Leistung des Kanals verbessern und den Netzverkehr verringern. IBM MQ classes for Java verwenden die Komprimierungsfunktion, die in IBM MQ integriert ist.

Mithilfe der in IBM MQ bereitgestellten Funktion können Sie die Daten komprimieren, die durch Nachrichtenkanäle und MQI-Kanäle fließen. Bei beiden Kanaltypen können Sie Headerdaten und Nachrichtendaten unabhängig voneinander komprimieren. Standardmäßig werden keine Daten in einem Kanal komprimiert. Sie finden eine vollständige Beschreibung der Kanalkomprimierung, darunter auch Informationen zu ihrer Implementierung in IBM MQ, in den Abschnitten Datenkomprimierung (COMPMSG) und Headerkomprimierung (COMPHDR).

Eine Anwendung der IBM MQ classes for Java gibt durch die Erstellung eines java.util.Collection-Objekts die Techniken an, die für die Komprimierung von Header- oder Nachrichtendaten in einer Clientverbindung verwendet werden können. Jede Komprimierungstechnik ist ein Ganzzahlobjekt in der Objektgruppe. Die Reihenfolge, in der die Anwendung die Komprimierungstechniken zur Objektgruppe hinzufügt, bestimmt die Reihenfolge, in der die Komprimierungstechniken beim Start der Clientverbindung mit dem Warteschlangenmanager vereinbart werden. Die Anwendung kann dann die Objektgruppe dem Feld 'hdrCompList' (für Headerdaten) oder dem Feld 'msgCompList' (für Nachrichtendaten) in der MQEnvironment-Klasse zuweisen. Sobald die Anwendung bereit ist, kann sie die Clientverbindung durch die Erstellung eines MQQueueManager-Objekts starten.

Die folgenden Codefragmente veranschaulichen die beschriebene Methode. Das erste Codefragment zeigt Ihnen, wie Sie eine Komprimierung der Headerdaten implementieren können:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

Das zweite Codefragment zeigt Ihnen, wie Sie eine Komprimierung der Nachrichtendaten implementieren können:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_LZ4HIGH));
:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

Im zweiten Beispiel werden die Komprimierungstechniken beim Start der Clientverbindung in der Reihenfolge RLE und anschließend ZLIBHIGH vereinbart. Die ausgewählte Komprimierungstechnik kann während der Lebensdauer des MQQueueManager-Objekts nicht geändert werden.

Die Komprimierungstechniken für Header- und Nachrichtendaten, die sowohl vom Client als auch vom Warteschlangenmanager in einer Clientverbindung unterstützt werden, werden als Objektgruppen in den Feldern 'hdrCompList' und 'msgCompList' eines MQChannelDefinition-Objekts an einen Kanalexit übergeben. Die tatsächlichen Techniken, die aktuell für die Komprimierung der Header- und Nachrichtendaten bei einer Clientverbindung verwendet werden, werden in den Feldern 'CurHdrCompression' und 'CurMsgCompression' eines MQChannelExit-Objekts an einen Kanalexit übergeben.

Wenn bei einer Clientverbindung die Komprimierung verwendet wird, erfolgt die Komprimierung der Daten, bevor Kanalsendeexits verarbeitet werden, und sie werden nach der Verarbeitung von Kanalempfangsexits extrahiert. Daher befinden sich die an Sende- und Empfangsexits übergebenen Daten in einem komprimierten Zustand.

Weitere Informationen zur Angabe der Komprimierungstechniken sowie Informationen zu den verfügbaren Komprimierungstechniken finden Sie unter Klasse com.ibm.mq.MQEnvironment und Schnittstelle com.ibm.mq.MQC.

TCP/IP-Verbindung in IBM MQ classes for Java gemeinsam nutzen

Mehrere Instanzen eines MQI-Kanals können so festgelegt werden, dass sie eine einzelne TCP/IP-Verbindung gemeinsam nutzen.

In IBM MQ classes for Java steuern Sie mit der Variablen MQEnvironment.sharingConversations die Anzahl der Dialoge, die eine einzelne TCP/IP-Verbindung gemeinsam nutzen können.

Das SHARECNV-Attribut ist ein Best-Effort-Ansatz ('falls möglich') zur gemeinsamen Verbindungsnutzung. Wenn also ein SHARECNV-Wert größer als 0 in Verbindung mit den IBM MQ classes for Java verwendet wird, wird bei einer neuen Verbindungsanforderung nicht immer automatisch eine bereits hergestellte Verbindung gemeinsam genutzt.

Verbindungspooling in IBM MQ classes for Java

IBM MQ classes for Java ermöglichen die Zusammenfassung von Ersatzverbindungen in Pools, damit diese wiederverwendet werden können.

IBM MQ classes for Java bieten eine zusätzliche Unterstützung für Anwendungen, die mehrere Verbindungen mit IBM MQ-Warteschlangenmanagern bearbeiten. Wenn eine Verbindung nicht mehr benötigt wird, kann sie in einen Pool gestellt und später wiederverwendet werden, sie wird also nicht gelöscht. Dies kann eine erhebliche Leistungsverbesserung für Anwendungen und Middleware mit sich bringen, die sich seriell mit beliebigen Warteschlangenmanagern verbinden.

IBM MQ stellt einen Standardverbindungspool zur Verfügung. Anwendungen können diesen Verbindungspool aktivieren oder inaktivieren, indem sie Tokens über die Klasse MQEnvironment registrieren bzw.

deren Registrierung zurücknehmen. Falls der Pool aktiv ist, wenn die IBM MQ classes for Java ein MQQueueManager-Objekt erstellen, durchsuchen sie diesen Standardpool und verwenden eine geeignete Verbindung erneut. Wenn ein MQQueueManager.disconnect()-Aufruf erfolgt, wird die zugrunde liegende Verbindung an den Pool zurückgegeben.

Alternativ können Anwendungen einen MQSimpleConnectionManager-Verbindungspool für eine bestimmte Verwendung erstellen. Anschließend kann die Anwendung entweder diesen Pool bei der Erstellung eines MQQueueManager-Objekts angeben oder diesen Pool an MQEnvironment übergeben, damit er als Standardverbindungspool verwendet wird.

Damit die Verbindungen nicht zu viele Ressourcen belegen, können Sie die Gesamtzahl der von einem MQSimpleConnectionManager-Objekt bearbeitbaren Verbindungen begrenzen. Außerdem können Sie eine Größenbeschränkung für den Verbindungspool festlegen. Die Festlegung von Grenzwerten ist sinnvoll, wenn innerhalb einer JVM Verbindungsnachfragen miteinander in Konflikt stehen.

Die Methode getMaxConnections() gibt standardmäßig den Wert null zurück. Dies bedeutet, dass die Anzahl der Verbindungen, die vom MQSimpleConnectionManager-Objekt bearbeitet werden können, nicht begrenzt ist. Mit der Methode setMaxConnections() können Sie einen Grenzwert festlegen. Wenn Sie einen Grenzwert festlegen und dieser erreicht ist, kann eine Anfrage bezüglich einer weiteren Verbindung eine Ausnahmebedingung (MQException) mit dem Ursachencode MQRC_MAX_CONNS_LIMIT_REACHED auslösen.

Standardverbindungspool in IBM MQ classes for Java steuern
Dieses Beispiel zeigt Ihnen, wie der Standardverbindungspool verwendet wird.

Betrachten Sie die folgende Beispielanwendung MQApp1:

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 nimmt eine Liste von lokalen Warteschlangenmanagern aus der Befehlszeile, stellt zu jedem der Reihe nach eine Verbindung her und führt einige Operationen aus. Wenn in der Befehlszeile jedoch häufig derselbe Warteschlangenmanager aufgelistet wird, ist es effizienter, nur einmal eine Verbindung herzustellen und diese dann viele Male wiederzuverwenden.

IBM MQ classes for Java stellen einen Standardverbindungspool zur Verfügung, den Sie zu diesem Zweck verwenden können. Um den Pool zu aktivieren, verwenden Sie eine der MQEnvironment.addConnectionPoolToken()-Methoden. Wenn Sie den Pool inaktivieren möchten, verwenden Sie MQEnvironment.removeConnectionPoolToken().

Die folgende Beispielanwendung 'MQApp2' weist dieselben Funktionen wie MQApp1 auf, verbindet sich jedoch nur einmal mit jedem Warteschlangenmanager.

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
```

```
            qmgr.disconnect();
        }

        MQEnvironment.removeConnectionPoolToken(token);

    }
}
```

Die erste fett gedruckte Zeile aktiviert den Standardverbindungspool durch die Registrierung eines MQPoolToken-Objekts bei MQEnvironment.

Der MQQueueManager-Konstruktor durchsucht jetzt diesen Pool nach einer geeigneten Verbindung. Er erstellt nur dann eine Verbindung mit dem Warteschlangenmanager, wenn er keine bereits vorhandene Verbindung finden kann. Der Aufruf qmgr.disconnect() gibt die Verbindung zur späteren Wiederverwendung an den Pool zurück. Diese API-Aufrufe sind die gleichen wie bei der Beispielanwendung MQApp1.

Die zweite hervorgehobene Zeile inaktiviert den Standardverbindungspool. Dadurch werden alle im Pool gespeicherten Warteschlangenmanagerverbindungen gelöscht. Dies ist wichtig, da andernfalls die Anwendung mit mehreren Live-Warteschlangenmanagerverbindungen im Pool beendet würde. Diese Situation könnte Fehler verursachen, die in den Warteschlangenmanagerprotokollen angezeigt würden.

Wenn eine Anwendung eine Definitionstabelle für den Clientkanal (CCDT) verwendet, um eine Verbindung zu einem Warteschlangenmanager herzustellen, durchsucht der MQQueueManager-Konstruktor zuerst die Tabelle nach einer geeigneten Definition eines Clientverbindungskanals. Falls eine gefunden wird, durchsucht der Konstruktor den Standardverbindungspool nach einer Verbindung, die für den Kanal verwendet werden kann. Wenn der Konstruktor keine geeignete Verbindung im Pool finden kann, durchsucht er danach die Definitionstabelle für den Clientkanal nach der nächsten geeigneten Definition eines Clientverbindungskanals und fährt wie zuvor beschrieben fort. Wenn der Konstruktor seine Suche in der Definitionstabelle für den Clientkanal abgeschlossen hat und keine geeignete Verbindung im Pool findet, beginnt er in der Tabelle einen zweiten Suchlauf. Während dieser Suche versucht der Konstruktor der Reihe nach, eine neue Verbindung für jede geeignete Definition eines Clientverbindungskanals zu erstellen. Er verwendet die erste Verbindung, die er erstellen kann.

Der Standardverbindungspool speichert bis zu zehn nicht verwendete Verbindungen und hält diese bis zu fünf Minuten lang aktiv. Die Anwendung kann dies ändern (Details hierzu finden Sie unter „Anderen Verbindungspool in IBM MQ classes for Java bereitstellen” auf Seite 419).

Die Anwendung verwendet nicht MQEnvironment für die Bereitstellung eines MQPoolToken, sondern erstellt ein eigenes Token:

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Manche Anwendungen oder Middlewareanbieter stellen Unterklassen von MQPoolToken bereit, um Informationen an einen angepassten Verbindungspool zu übergeben. Diese können erstellt und auf diese Weise an addConnectionPoolToken() übergeben werden, damit zusätzliche Informationen an den Verbindungspool übergeben werden können.

Standardverbindungspool und mehrere Komponenten in IBM MQ classes for Java
Dieses Beispiel zeigt, wie MQPoolTokens aus einer statischen Gruppe von registrierten MQPoolToken-Objekten hinzugefügt oder entfernt werden.

MQEnvironment verfügt über eine statische Gruppe von registrierten MQPoolToken-Objekten. Sie können MQPoolTokens mit den folgenden Methoden aus dieser Gruppe hinzufügen bzw. entfernen:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

Eine Anwendung kann aus vielen Komponenten bestehen, die unabhängig voneinander vorhanden sind und mithilfe eines Warteschlangenmanagers Arbeiten durchführen. In einer solchen Anwendung sollte jede Komponente für ihre Lebensdauer ein MQPoolToken zur MQEnvironment-Gruppe hinzufügen.

Die Beispielanwendung MQApp3 erstellt beispielsweise zehn Threads und startet jeden einzelnen Thread. Jeder Thread registriert sein eigenes MQPoolToken, wartet eine gewisse Zeit und verbindet sich dann mit dem Warteschlangenmanager. Nach der Trennung des Threads entfernt er sein eigenes MQPoolToken.

Der Standardverbindungspool bleibt aktiv, solange sich mindestens ein Token in der Gruppe der MQPool-Tokens befindet, er bleibt also für die Dauer dieser Anwendung aktiv. Die Anwendung muss kein Masterobjekt in der Gesamtsteuerung der Threads aufbewahren.

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

Anderen Verbindungspool in IBM MQ classes for Java bereitstellen
Dieses Beispiel zeigt, wie die Klasse **com.ibm.mq.MQSimpleConnectionManager** für die Bereitstellung eines anderen Verbindungspools verwendet wird.

Diese Klasse stellt grundlegendes Funktionen für das Verbindungspooling bereit und Anwendungen können diese Klasse verwenden, um das Verhalten des Pools anzupassen.

Sobald er instanziiert wurde, kann ein MQSimpleConnectionManager im MQQueueManager-Konstruktor angegeben werden. Der MQSimpleConnectionManager verwaltet dann die Verbindung, die dem erstellten MQQueueManager zugrunde liegt. Wenn der MQSimpleConnectionManager eine geeignete gepoolte Verbindung enthält, wird diese Verbindung wiederverwendet und nach einem Aufruf des Typs MQQueueManager.disconnect() an den MQSimpleConnectionManager zurückgegeben.

Das folgende Codefragment veranschaulicht dieses Verhalten:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
```

```
    : (do something with qmgr2)
    :
    qmgr2.disconnect();
    myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

Die Verbindung, die während des ersten MQQueueManager-Konstruktors aufgebaut wird, wird nach dem Aufruf qmgr.disconnect() in myConnMan gespeichert. Die Verbindung wird dann beim zweiten Aufruf an den MQQueueManager-Konstruktor wiederverwendet.

Die zweite Zeile aktiviert den MQSimpleConnectionManager. Die letzte Zeile inaktiviert den MQSimpleConnectionManager und löscht damit sämtliche Verbindungen, die sich im Pool befinden. Ein MQSimpleConnectionManager befindet sich standardmäßig im Modus MODE_AUTO, der an späterer Stelle in diesem Abschnitt beschrieben wird.

Ein MQSimpleConnectionManager ordnet die Verbindungen, die am häufigsten verwendet werden, zu und löscht die Verbindungen, die am seltensten verwendet werden. Eine Verbindung wird standardmäßig gelöscht, wenn sie fünf Minuten lang nicht verwendet wurde oder wenn der Pool mehr als zehn nicht verwendete Verbindungen enthält. Sie können diese Werte durch das Aufrufen von MQSimpleConnectionManager.setTimeout() ändern.

Sie können einen MQSimpleConnectionManager auch als Standardverbindungspool konfigurieren, der verwendet wird, wenn im MQQueueManager-Konstruktor kein Connection Manager bereitgestellt wird.

Dies wird durch die folgende Anwendung veranschaulicht:

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

Mit den fett gedruckten Zeilen wird ein MQSimpleConnectionManager-Objekt erstellt und konfiguriert. Die Konfiguration bewirkt die folgenden Aktionen:

- Sie beendet Verbindungen, die eine Stunde lang nicht verwendet wurden.
- Sie begrenzt die Anzahl der von myConnMan verwalteten Verbindungen auf 75.
- Sie begrenzt die Anzahl der nicht verwendeten Verbindungen im Pool auf 50.
- Sie legt die Standardeinstellung MODE_AUTO fest. Dies bedeutet, dass der Pool nur aktiv ist, wenn es sich um den Standardverbindungsmanager handelt und die MQPoolTokens-Gruppe von MQEnvironment mindestens ein Token enthält.

Der neue MQSimpleConnectionManager wird dann als Standardverbindungsmanager festgelegt.

In der letzten Zeile ruft die Anwendung MQApp3.main() auf. Dadurch werden mehrere Threads ausgeführt, wobei jeder Thread IBM MQ unabhängig verwendet. Diese Threads verwenden beim Aufbau von Verbindungen myConnMan.

JTA/JDBC-Koordination unter Verwendung von IBM MQ classes for Java

IBM MQ classes for Java unterstützen die Methode MQQueueManager.begin(), die es IBM MQ ermöglicht, als Koordinator für eine Datenbank zu agieren, die einen Treiber bereitstellt, der mit JDBC Typ 2 oder JDBC Typ 4 kompatibel ist.

Diese Unterstützung ist nicht auf allen Plattformen verfügbar. Im Abschnitt Systemvoraussetzungen für IBM MQ ist aufgeführt, welche Plattformen die JDBC-Koordination unterstützen.

Um die XA-JTA-Unterstützung nutzen zu können, müssen Sie die spezielle JTA-Switchbibliothek verwenden. Die Methode für die Verwendung dieser Bibliothek variiert je nachdem, ob Sie Windows oder eine der anderen Plattformen verwenden.

JTA/JDBC-Koordination unter Windows konfigurieren
Die XA-Bibliothek wird als DLL mit einem Namen im Format `jdbcxxx.dll` bereitgestellt.

Die bereitgestellte Datei `jdbcora12.dll` ist mit Oracle 12C kompatibel und für eine Serverinstallation mit IBM MQ für Windows vorgesehen.

Auf Windows-Systemen wird die XA-Bibliothek als vollständige DLL bereitgestellt. Diese DLL hat den Namen `jdbcxxx.dll`. Dabei gibt `xxx` die Datenbank an, für die die Switchbibliothek kompiliert wurde. Diese Bibliothek befindet sich im Verzeichnis `java\lib\jdbc` oder `java\lib64\jdbc` Ihrer Installation der IBM MQ classes for Java. Sie müssen die XA-Bibliothek, die auch als Switchloaddatei beschrieben wird, beim Warteschlangenmanager deklarieren. Verwenden Sie IBM MQ Explorer. Geben Sie die Details zu der Switchloaddatei in der Eigenschaftenanzeige für den Warteschlangenmanager unter dem XA-Ressourcenmanager an. Sie müssen nur den Namen der Bibliothek angeben. For example:

Bei einer Db2-Datenbank müssen Sie das Feld 'SwitchFile' auf folgenden Wert setzen: `dbcdb2`

Bei einer Oracle-Datenbank müssen Sie das Feld 'SwitchFile' auf folgenden Wert setzen: `jdbcora`.

Anmerkungen:

1. Oracle 12C wird für die IBM MQ classes for Java nur unter IBM MQ fürWindows unterstützt.
2. Die unterstützte Version von Oracle 12C ist 12.1.0.1.0 Enterprise Edition und künftige Fixpacks.
3. Für 6-Bit-Versionen der Oracle-Datenbank unter Windows mit 64 Bit ist ein Oracle-Client mit 32 Bit erforderlich.
4. Durch die Verwendung der IBM MQ classes for Java kann IBM MQ als Transaktionskoordinator verwendet werden. Es ist allerdings nicht möglich, an einer Transaktion im Stil einer JTA teilzunehmen.

JTA/JDBC-Koordination auf anderen Plattformen als Windows konfigurieren
Es werden Objektdateien bereitgestellt. Verknüpfen Sie die für Sie geeignete Datei mithilfe der bereitgestellten Makefile und deklarieren Sie diese unter Verwendung der Konfigurationsdatei beim Warteschlangenmanager.

IBM MQ stellt für jedes Datenbankmanagementsystem zwei Objektdateien zur Verfügung. Sie müssen eine Objektdatei verknüpfen, um eine 32-Bit-Switchbibliothek zu erstellen. Außerdem müssen Sie die andere Objektdatei zur Erstellung einer 64-Bit-Switchbibliothek verknüpfen. Bei Db2 lautet der Name jeder Objektdatei `jdbcdb2.o`, während der Name der einzelnen Objektdateien bei Oracle `jdbcora.o` lautet.

Sie müssen jede Objektdatei mithilfe der entsprechenden Objektdatei verknüpfen, die mit IBM MQ bereitgestellt wird. Eine Switchbibliothek erfordert weitere Bibliotheken, die möglicherweise an verschiedenen Position auf unterschiedlichen Systemen gespeichert sind. Eine Switchbibliothek kann zur Suche dieser Bibliotheken jedoch nicht die Umgebungsvariable mit dem Bibliothekspfad verwenden, da die Switchbibliothek vom Warteschlangenmanager geladen wird, der in einer `setuid`-Umgebung (setuid = Definitionsmodus für Benutzer-ID) ausgeführt wird. Daher sorgt die bereitgestellte Makefile dafür, dass eine Switchbibliothek die vollständig qualifizierten Pfadnamen dieser Bibliotheken enthält.

Zur Erstellung einer Switchbibliothek müssen Sie einen **make**-Befehl im folgenden Format eingeben. Wenn Sie eine 32-Bit-Switchbibliothek erstellen möchten, geben Sie den Befehl im Verzeichnis `/java/lib/jdbc` Ihrer IBM MQ-Installation ein. Falls Sie eine 64-Bit-Switchbibliothek erstellen wollen, geben Sie den Befehl im Verzeichnis `/java/lib64/jdbc` ein.

```
make DBMS
```

Dabei steht *DBMS* für das Datenbankmanagementsystem, für das Sie die Switchbibliothek erstellen. Gültige Werte sind db2 für Db2 und `oracle` für Oracle.

Anmerkung:

- Zur Ausführung von 32-Bit-Anwendungen müssen Sie sowohl eine 32-Bit-Switchbibliothek als auch eine 64-Bit-Switchbibliothek für jedes Datenbankmanagementsystem erstellen, das Sie verwenden. Zur Ausführung von 64-Bit-Anwendungen müssen Sie nur eine 64-Bit-Switchbibliothek erstellen. Bei Db2 ist der Name jeder Switchbibliothek `jdbcdb2`, während der Name der einzelnen Switchbibliotheken bei Oracle `jdbcora` lautet. Die Makefiles stellen sicher, dass die 32-Bit- und 64-Bit-Switchbibliotheken in unterschiedlichen IBM MQ-Verzeichnissen gespeichert werden. Eine 32-Bit-Switchbibliothek wird im Verzeichnis `/java/lib/jdbc` gespeichert, während eine 64-Bit-Switchbibliothek im Verzeichnis `/java/lib64/jdbc` gespeichert wird.
- Da Sie Oracle an einer beliebigen Position im System installieren können, verwenden die Makefiles die Umgebungsvariable **ORACLE_HOME**, um festzustellen, wo Oracle installiert ist.
- Wenn IBM MQ an einer anderen Position als der Standardposition installiert ist, ändern Sie den Wert von **MQ_INSTALLATION_PATH** in der Makefile.

Nachdem Sie die Switchbibliotheken für Db2 und/oder Oracle erstellt haben, müssen Sie diese bei Ihrem Warteschlangenmanager deklarieren. Falls die Konfigurationsdatei des Warteschlangenmanagers (qm.ini) bereits `XAResourceManager`-Zeilengruppen für Db2- oder Oracle-Datenbanken enthält, müssen Sie den `SwitchFile`-Eintrag in jeder Zeilengruppe durch einen der folgenden Werte ersetzen:

Bei einer Db2-Datenbank

```
SwitchFile=jdbcdb2
```

Bei einer Oracle-Datenbank

```
SwitchFile=jdbcora
```

Geben Sie nicht den vollständig qualifizierten Pfadnamen der 32-Bit- oder 64-Bit-Switchbibliothek an. Geben Sie nur den Namen der Bibliothek an.

Wenn die Konfigurationsdatei des Warteschlangenmanagers noch keine `XAResourceManager`-Zeilengruppen für Db2- oder Oracle-Datenbanken enthält oder Sie zusätzliche `XAResourceManager`-Zeilengruppen hinzufügen möchten, lesen Sie den Abschnitt IBM MQ verwalten. Dort finden Sie Informationen zur Erstellung einer `XAResourceManager`-Zeilengruppe. Jeder Eintrag `SwitchFile` in einer neuen Zeilengruppe `XAResourceManager` muss jedoch genau wie zuvor für eine Db2 -oder Oracle -Datenbank beschrieben sein. Darüber hinaus müssen Sie den Eintrag `ThreadOfControl=PROCESS` einschließen.

Nachdem Sie die Konfigurationsdatei des Warteschlangenmanagers aktualisiert und sichergestellt haben, dass alle entsprechenden Datenbankumgebungsvariablen festgelegt wurden, können Sie den Warteschlangenmanager erneut starten.

JTA/JDBC-Koordination verwenden
Codieren Sie Ihre API-Aufrufe wie in dem bereitgestellten Beispiel.

Die grundlegende Reihenfolge von API-Aufrufen für eine Benutzeranwendung lautet wie folgt:

```
  qMgr = new MQQueueManager("QM1")
  Connection con = qMgr.getJDBCConnection( xads );
  qMgr.begin()

 < Perform MQ and DB operations to be grouped in a unit of work >

  qMgr.commit() or qMgr.backout();
  con.close()
  qMgr.disconnect()
```

`xads` im Aufruf getJDBCConnection ist eine datenbankspezifische Implementierung der XADataSource-Schnittstelle, die die Details der Datenbank definiert, zu der eine Verbindung hergestellt werden soll. In der Dokumentation zu Ihrer Datenbank finden Sie Informationen darüber, wie Sie ein geeignetes XADataSource-Objekt zur Übergabe an getJDBCConnection erstellen.

Darüber hinaus müssen Sie Ihren Klassenpfad mit den entsprechenden datenbankspezifischen JAR-Dateien für die Durchführung der JDBC-Bearbeitung aktualisieren.

Wenn Sie eine Verbindung zu mehreren Datenbanken herstellen müssen, müssen Sie getJDBCConnection mehrmals aufrufen, damit die Transaktion für mehrere verschiedene Verbindungen ausgeführt wird.

Es gibt zwei Formen von getJDBCConnection, die beide Formen von XADataSource.getXAConnection widerspiegeln:

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

Diese Methoden deklarieren eine Ausnahmebedingung (Exception) in ihren Klauseln 'throws', um Probleme im Zusammenhang mit der JVM-Prüffunktion bei Kunden zu vermeiden, die die JTA-Funktionen nicht verwenden. Die tatsächlich ausgelöste Ausnahmebedingung ist 'javax.transaction.xa.XAException'. Hierfür muss für Programme, die diese Datei früher nicht benötigten, die Datei 'jta.jar' zum Klassenpfad hinzugefügt werden.

Um die JTA/JDBC-Unterstützung zu verwenden, müssen Sie die folgende Anweisung in Ihre Anwendung aufnehmen:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

Bekannte Probleme und Einschränkungen bei der JTA/JDBC-Koordination
Einige der Probleme und Einschränkungen der JTA/JDBC-Unterstützung sind vom verwendeten Datenbankmanagementsystem abhängig. So verhalten sich beispielsweise getestete JDBC-Treiber unterschiedlich, wenn die Datenbank heruntergefahren wird, während eine Anwendung aktiv ist. Wenn die Verbindung zu der von einer Anwendung genutzten Datenbank unterbrochen wird, kann die Anwendung Schritte unternehmen, um eine neue Verbindung zum Warteschlangenmanager und zur Datenbank herzustellen, sodass sie dann diese neuen Verbindungen nutzen kann, um die erforderlichen transaktionsorientierten Arbeitsvorgänge auszuführen.

Da die JTA/JDBC-Unterstützung Aufrufe an JDBC-Treiber ausgibt, kann sich die Implementierung dieser JDBC-Treiber entscheidend auf das Systemverhalten auswirken. Insbesondere verhalten sich die getesteten JDBC-Treiber unterschiedlich, wenn die Datenbank während der Ausführung einer Anwendung beendet wird.

Wichtig: Vermeiden Sie immer ein abruptes Herunterfahen der Datenbank, solange Anwendungen über offene Verbindungen mit ihr verfügen.

Anmerkung: Eine Anwendung der IBM MQ classes for Java muss Verbindungen im Bindungsmodus herstellen, damit IBM MQ als Datenbankkoordinator agieren kann.

Mehrere XAResourceManager-Zeilengruppen
Die Verwendung mehrerer XAResourceManager-Zeilengruppen in einer Warteschlangenmanager-Konfigurationsdatei (`qm.ini`) wird nicht unterstützt. Mit Ausnahme der ersten XAResourceManager-Zeilengruppe werden alle übrigen Zeilengruppen ignoriert.

Db2

Gelegentlich gibt Db2 einen SQL0805N-Fehler zurück. Dieses Problem kann mit dem folgenden CLP-Befehl gelöst werden:

```
DB2 bind @db2cli.lst blocking all grant public
```

Weitere Informationen finden Sie in der Db2-Dokumentation.

Die XAResourceManager-Zeilengruppe muss für die Verwendung von ThreadOfControl=PROCESS konfiguriert werden. Da dies bei Db2 8.1 und höher nicht dem Standardthread der Steuereinstellung

für Db2 entspricht, muss 'toc=p' in der XA-Zeichenfolge zum Öffnen angegeben werden. Im Folgenden finden Sie eine XAResourceManager-Beispielzeilengruppe für Db2 mit JTA/JDBC-Koordination:

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

Dadurch wird nicht verhindert, dass die Java-Anwendungen, die eine JTA/JDBC-Koordination verwenden, selbst als Multithread-Prozesse ausgeführt werden.

Oracle
Wenn die JDBC-Methode Connection.close() nach MQQueueManager.disconnect() aufgerufen wird, wird eine SQL-Ausnahmebedingung generiert. Rufen Sie Connection.close() entweder vor MQQueueManager.disconnect() auf oder übergehen Sie den Aufruf von Connection.close().

Behandlung von Problemen mit Datenbankverbindungen

Wenn eine IBM MQ classes for Java-Anwendung die JTA/JDBC-Unterstützung verwendet, die von IBM MQ bereitgestellt wird, führt sie üblicherweise folgende Schritte aus:

1. Sie erstellt ein neues MQQueueManager-Objekt, das eine Verbindung zu dem Warteschlangenmanager darstellt, der als Transaktionsmanager agieren wird.
2. Sie erstellt ein XADataSource-Objekt, das Details zur Herstellung der Verbindung zu der Datenbank enthält, die in die Transaktion eingetragen wird.
3. Sie ruft die Methode MQQueueManager.getJDBCConnection(XADataSource) auf, die in der zuvor erstellten XADataSource übergeben wird. Dies bewirkt, dass die IBM MQ classes for Java eine Verbindung zur Datenbank herstellen.
4. Sie ruft die Methode MQQueueManager.begin() auf, um die XA-Transaktion zu starten.
5. Sie führt die Messaging- und Datenbankarbeit aus.
6. Wenn alle erforderlichen Arbeiten abgeschlossen sind, ruft sie die Methode MQQueueManager.commit() auf. Dies beendet die XA-Transaktion.
7. Wenn an diesem Punkt eine neue XA-Transaktion erforderlich ist, kann die Anwendung die Schritte 4, 5 und 6 wiederholen.
8. Wenn die Anwendung fertig ist, sollte sie die Datenbankverbindung, die in Schritt 3 erstellt wurde, schließen und dann die Methode MQQueueManager.disconnect() aufrufen, um die Verbindung zum Warteschlangenmanager zu trennen.

Die IBM MQ classes for Java verwalten eine interne Liste aller Datenbankverbindungen, die erstellt wurden, wenn eine Anwendung MQQueueManager.getJDBCConnection(XADataSource) aufruft. Wenn ein Warteschlangenmanager während der Verarbeitung der XA-Transaktion mit der Datenbank kommunizieren muss, findet folgende Verarbeitung statt:

1. Der Warteschlangenmanager ruft die IBM MQ classes for Java auf und übergibt Details des XA-Aufrufs, der an die Datenbank übergeben werden muss.
2. Die IBM MQ classes for Java suchen in der Liste nach der geeigneten Verbindung und nutzen dann diese Verbindung, um den XA-Aufruf an die Datenbank weiterzuleiten.

Wenn die Verbindung zur Datenbank an einem beliebigen Punkt während dieser Verarbeitung verloren geht, sollte die Anwendung Folgendes tun:

1. Alle Arbeiten, die unter der Transaktion ausgeführt wurden, zurücksetzen, indem sie die Methode MQQueueManager.backout() aufruft.
2. Die Datenbankverbindung schließen. Dies sollte dazu führen, dass die IBM MQ classes for Java Details der unterbrochenen Datenbankverbindung aus ihrer internen Liste entfernen.
3. Die Verbindung zum Warteschlangenmanager trennen, indem Sie die Methode MQQueueManager.disconnect() aufruft.

4. Eine neue Verbindung zum Warteschlangenmanager herstellen, indem sie ein neues MQQueueManager-Objekt erstellt.
5. Eine neue Datenbankverbindung herstellen, indem sie die Methode MQQueueManager.getJDBCConnection(XADataSource) aufruft.
6. Die transaktionsorientierte Arbeitsvorgänge erneut ausführen.

Dies ermöglicht es der Anwendung, eine neue Verbindung zum Warteschlangenmanager und zur Datenbank herzustellen und diese Verbindungen dann zur Ausführung der erforderlichen transaktionsorientierten Arbeitsvorgänge zu nutzen.

Unterstützung von Transport Layer Security (TLS) in IBM MQ classes for Java

IBM MQ classes for Java-Clientanwendungen unterstützen die TLS-Verschlüsselung. Für die Verwendung der TLS-Verschlüsselung ist ein JSSE-Provider erforderlich.

IBM MQ classes for Java-Clientanwendungen, die TRANSPORT(CLIENT) verwenden, unterstützen die TLS-Verschlüsselung. TLS stellt eine Kommunikationsverschlüsselung, Authentifizierung und Nachrichtenintegrität bereit. In der Regel wird damit die Kommunikation zwischen zwei Peers im Internet oder innerhalb eines Intranets geschützt.

IBM MQ classes for Java nutzen Java Secure Socket Extension (JSSE) für die Handhabung der TLS-Verschlüsselung und benötigen daher einen JSSE-Provider. JVMs mit JSE v1.4 verfügen bereits über einen integrierten JSSE-Provider. Die Details der Verwaltung und Speicherung von Zertifikaten kann je nach Provider variieren. Informationen hierzu finden Sie in der Dokumentation Ihres JSSE-Providers.

In diesem Abschnitt wird vorausgesetzt, dass Ihr JSSE-Provider ordnungsgemäß installiert und konfiguriert wurde. Außerdem müssen geeignete Zertifikate installiert worden sein, die Ihrem JSEE-Provider zur Verfügung stehen.

Wenn Ihre Clientanwendung der IBM MQ classes for Java für die Verbindung mit einem Warteschlangenmanager eine Definitionstabelle für den Clientkanal (CCDT) verwendet, lesen Sie den Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for Java verwenden” auf Seite 395.

TLS in IBM MQ classes for Java aktivieren

Zur Aktivierung von TLS geben Sie eine Cipher-Suite an. Für die Angabe einer Cipher-Suite haben Sie zwei Möglichkeiten.

TLS wird nur für Clientverbindungen unterstützt. Zur Aktivierung von TLS müssen Sie die Cipher-Suite für die Kommunikation mit dem Warteschlangenmanager angeben. Diese Cipher-Suite muss mit der für den Zielkanal festgelegten CipherSpec übereinstimmen. Außerdem muss die angegebene Cipher-Suite von Ihrem JSEE-Provider unterstützt werden. Da sich Cipher-Suites jedoch von CipherSpecs unterscheiden, haben sie unterschiedliche Namen. „TLS CipherSpecs und Cipher-Suites in IBM MQ classes for Java” auf Seite 430 enthält eine Tabellenzuordnung zwischen den von IBM MQ unterstützten CipherSpecs und ihren Cipher-Suite-Entsprechungen, die in JSSE bekannt sind.

Geben Sie zur Aktivierung von TLS die Cipher-Suite mithilfe der statischen Mitgliedsvariablen 'sslCipherSuite' von MQEnvironment an. Das folgende Beispiel bezieht sich auf den SVRCONN-Kanal SECURE.SVRCONN.CHANNEL, der so eingerichtet wurde, dass er TLS mit der CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256 erfordert:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Obwohl der Kanal über die CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256 verfügt, muss die Java-Anwendung die Cipher-Suite SSL_RSA_WITH_AES_128_CBC_SHA256 angeben. Im Abschnitt „TLS CipherSpecs und Cipher-Suites in IBM MQ classes for Java” auf Seite 430 finden Sie eine Liste der Zuordnungen zwischen den CipherSpecs und den Cipher-Suites.

Eine Anwendung kann eine Cipher-Suite auch durch die Festlegung der Umgebungseigenschaft CMQC.SSL_CIPHER_SUITE_PROPERTY angeben.

Alternativ dazu können Sie die Definitionstabelle für den Clientkanal (Client Channel Definition Table, CCDT) verwenden. Weitere Informationen finden Sie unter „Definitionstabelle für den Clientkanal mit IBM MQ classes for Java verwenden” auf Seite 395

Wenn in Ihrem Fall eine Clientverbindung erforderlich ist, die eine vom IBM Java JSSE FIPS-Provider (IBMJSSEFIPS) unterstützte Cipher-Suite verwendet, kann eine Anwendung das Feld 'sslFipsRequired' in der MQEnvironment-Klasse auf `true` setzen. Alternativ dazu kann die Anwendung die Umgebungseigenschaft CMQC.SSL_FIPS_REQUIRED_PROPERTY festlegen. Der Standardwert ist `false`. Dies bedeutet, dass eine Clientverbindung jede beliebige Cipher-Suite verwenden kann, die von IBM MQ unterstützt wird.

Wenn eine Anwendung mehrere Clientverbindungen verwendet, bestimmt der Wert im Feld 'sslFipsRequired', der bei der ersten Erstellung der Clientverbindung durch die Anwendung verwendet wird, den Wert, der verwendet wird, wenn die Anwendung nachfolgende Clientverbindungen erstellt. Wenn die Anwendung also eine nachfolgende Clientverbindung erstellt, wird der Wert des Feldes 'sslFipsRequired' ignoriert. Sie müssen die Anwendung erneut starten, wenn Sie einen anderen Wert für das Feld 'sslFipsRequired' verwenden möchten.

Für eine erfolgreiche Verbindungsherstellung über TLS müssen für den JSSE-Truststore die Stammzertifikate der Zertifizierungsstelle konfiguriert sein, über die das vom Warteschlangenmanager vorgewiesene Zertifikat authentifiziert werden kann. Ähnlich gilt Folgendes: Wenn SSLClientAuth im SVRCONN-Kanal auf MQSSL_CLIENT_AUTH_REQUIRED gesetzt wurde, muss der JSSE-Schlüsselspeicher ein Identifizierungszertifikat enthalten, das vom Warteschlangenmanager als vertrauenswürdig eingestuft wird.

Zugehörige Verweise

Federal Information Processing Standards (FIPS) für AIX, Linux, and Windows

Definierten Namen des Warteschlangenmanagers in IBM MQ classes for Java verwenden

Der Warteschlangenmanager identifiziert sich selbst mithilfe eines TLS-Zertifikat, das einen definierten Namen (DN) enthält. Eine Clientanwendung der IBM MQ classes for Java kann anhand dieses DN sicherstellen, dass sie mit dem richtigen Warteschlangenmanager kommuniziert.

Ein definiertes Namensmuster wird mit der Variablen 'sslPeerName' von MQEnvironment angegeben. Sie können beispielsweise Folgendes festlegen:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

Die Verbindung kann nur erfolgreich hergestellt werden, wenn der Warteschlangenmanager ein Zertifikat mit einem allgemeinen Namen vorlegt, der mit 'QMGR.' beginnt. und mindestens zwei Namen von Organisationseinheiten, von denen der erste IBM und der zweite WebSpheresein muss.

Wenn sslPeerName festgelegt wird, können Verbindungen nur dann erfolgreich hergestellt werden, wenn ein gültiges Muster festgelegt wird und der Warteschlangenmanager ein entsprechendes Zertifikat vorlegt.

Eine Anwendung kann den definierten Namen des Warteschlangenmanagers auch durch die Festlegung der Umgebungseigenschaft CMQC.SSL_PEER_NAME_PROPERTY angeben. Weitere Informationen zu definierten Namen finden Sie im Abschnitt Definierte Namen.

Zertifikatswiderrufslisten in IBM MQ classes for Java verwenden

Geben Sie die durch die java.security.cert.CertStore-Klasse zu verwendenden Zertifikatswiderrufslisten an. Die IBM MQ classes for Java prüfen dann die Zertifikate anhand der angegebenen Zertifikatswiderrufsliste.

Eine Zertifikatswiderrufsliste (Certificate Revocation List, CRL) ist eine Gruppe von Zertifikaten, die durch die ausstellende Zertifizierungsstelle oder durch die lokale Organisation widerrufen wurden. CRLs werden in der Regel auf LDAP-Servern gehostet. Bei Java 2 v1.4 kann ein CRL-Server zur Verbindungszeit angegeben werden und das vom Warteschlangenmanager vorgelegte Zertifikat wird anhand der CRL geprüft, bevor die Verbindung erlaubt wird. Weitere Informationen zu Zertifikatswiderrufslisten und IBM MQ finden Sie in den Abschnitten Mit Zertifikatswiderrufslisten und Berechtigungswiderrufslisten arbeiten und Auf Zertifikatswiderrufslisten und Berechtigungswiderrufslisten in Verbindung mit IBM MQ classes for Java und IBM MQ classes for JMS zugreifen.

Anmerkung: Damit ein CertStore erfolgreich mit einer auf einem LDAP-Server gehosteten CRL verwendet werden kann, müssen Sie sicherstellen, dass Ihr Java-Software Development Kit (SDK) mit der CRL kompatibel ist. Manche SDKs verlangen, dass die Zertifikatswiderrufsliste RFC 2587 entspricht, in dem ein Schema für LDAP v2 definiert ist. Die meisten LDAP v3-Server verwenden stattdessen RFC 2256.

Die zu verwendenden CRLs werden durch die java.security.cert.CertStore-Klasse angegeben. In der Dokumentation zu dieser Klasse finden Sie ausführliche Informationen zum Erhalt der CertStore-Instanzen. Um einen Zertifikatsspeicher (CertStore) auf Basis eines LDAP-Servers zu erstellen, müssen Sie zuerst eine LDAPCertStoreParameters-Instanz erstellen, die mit den zu verwendenden Einstellungen für den Server und Port initialisiert wird. For example:

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Nachdem Sie eine CertStoreParameters-Instanz erstellt haben, verwenden Sie den statischen Konstruktor für CertStore, um einen CertStore vom Typ LDAP zu erstellen:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

Andere CertStore-Typen (beispielsweise 'Collection') werden ebenfalls unterstützt. Häufig werden mehrere CRL-Server mit identischen CRL-Informationen eingerichtet, um eine Redundanz zu erreichen. Wenn Sie für jeden dieser CRL-Server ein CertStore-Objekt haben, stellen Sie diese alle in eine geeignete Objektgruppe (Collection). Das folgende Beispiel zeigt die CertStore-Objekte, die in eine ArrayList gestellt werden:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

Diese Objektgruppe kann vor der Verbindungsherstellung in der statischen MQEnvironment-Variablen 'sslCertStores' festgelegt werden, um die CRL-Prüfung zu aktivieren:

```
MQEnvironment.sslCertStores = crls;
```

Das Zertifikat, das vom Warteschlangenmanager beim Aufbau der Verbindung vorgelegt wird, wird wie folgt überprüft:

1. Das erste CertStore-Objekt in der durch sslCertStores angegebenen Objektgruppe wird für die Identifizierung eines CRL-Servers verwendet.
2. Es wird versucht, den CRL-Server zu kontaktieren.
3. Wenn der Versuch erfolgreich ist, wird der Server nach einer Übereinstimmung mit dem Zertifikat durchsucht.
   a. Falls festgestellt wird, dass das Zertifikat widerrufen wurde, wird der Suchvorgang beendet und die Verbindungsanforderung schlägt mit dem Ursachencode MQRC_SSL_CERTIFICATE_REVOKED fehl.
   b. Wenn das Zertifikat nicht gefunden werden kann, wird der Suchvorgang beendet und die Verbindung kann fortgesetzt werden.
4. Wenn der Versuch der Kontaktierung des Servers nicht erfolgreich war, wird das nächste CertStore-Objekt für die Identifizierung eines CRL-Servers verwendet und der Prozess wird ab Schritt 2 wiederholt.

   Falls das Objekt der letzte CertStore in der Objektgruppe war oder die Objektgruppe keine CertStore-Objekte enthält, ist der Suchvorgang fehlgeschlagen und die Verbindungsanforderung schlägt mit dem Ursachencode MQRC_SSL_CERT_STORE_ERROR fehl.

Das Objektgruppenobjekt bestimmt die Reihenfolge, in der CertStores verwendet werden.

Die Objektgruppe der CertStores kann auch mithilfe von CMQC.SSL_CERT_STORE_PROPERTY festgelegt werden. Aus praktischen Gründen ermöglicht diese Eigenschaft auch die Angabe eines einzelnen CertStore, der kein Mitglied einer Objektgruppe ist.

Wenn sslCertStores auf null gesetzt ist, erfolgt keine CRL-Prüfung. Diese Eigenschaft wird ignoriert, wenn sslCipherSuite nicht festgelegt ist.

Geheimen Schlüssel in IBM MQ classes for Java erneut vereinbaren
Eine Clientanwendung der IBM MQ classes for Java kann steuern, wann der für die Verschlüsselung in einer Clientverbindung verwendete geheime Schlüssel im Hinblick auf die Gesamtzahl der gesendeten und empfangenen Bytes erneut vereinbart wird.

Die Anwendung kann dies mithilfe einer der folgenden Methoden durchführen. Wenn die Anwendung mehrere dieser Methoden verwendet, gelten die üblichen Vorrangregeln.

- Wenn Sie das Feld "sslResetCount" in der Klasse "MQEnvironment" festlegen.
- Durch Festlegen der Umgebungseigenschaft MQC.SSL_RESET_COUNT_PROPERTY in einem Hashtabellenobjekt. Anschließend weist die Anwendung die Hashtabelle dem Feld `properties` in der MQEnvironment-Klasse zu oder übergibt die Hashtabelle an ein MQQueueManager-Objekt über den zugehörigen Konstruktor.

Der Wert des Felds 'sslResetCount' oder der Umgebungseigenschaft MQC.SSL_RESET_COUNT_PROPERTY stellt die Gesamtzahl der Bytes dar, die vom IBM MQ classes for Java-Client-Code gesendet oder empfangen wurden, bevor der gemeine Schlüssel erneut verhandelt wird. Dabei ist die Anzahl der gesendeten Bytes die Anzahl vor der Verschlüsselung und die Anzahl der empfangenen Bytes die Anzahl nach der Entschlüsselung. Die Anzahl der Byte umfasst auch Steuerinformationen, die vom IBM MQ classes for Java-Client gesendet und empfangen wurden.

Wenn der Rücksetzzähler null ist, was der Standardwert ist, wird der geheime Schlüssel nie neu vereinbart. Der Wert für die Anzahl der Rücksetzungen wird ignoriert, wenn keine Cipher-Suite angegeben wurde.

Angepasste SSLSocketFactory in IBM MQ classes for Java bereitstellen
Wenn Sie eine angepasste JSSE-Socket-Factory verwenden, setzen Sie die MQEnvironment.sslSocketFactory auf das angepasste Factory-Objekt. Die Details variieren je nach den verschiedenen JSEE-Implementierungen.

Verschiedene JSSE-Implementierungen können unterschiedliche Funktionen bereitstellen. Zum Beispiel könnte eine spezialisierte JSSE-Implementierung die Konfiguration eines bestimmten Modells einer Verschlüsselungshardware ermöglichen. Außerdem erlauben manche JSSE-Provider die Anpassung von Schlüsselspeichern und Truststores nach Programm oder die Änderung der Auswahl des Identitätszertifikats aus dem Schlüsselspeicher. In JSSE werden alle diese Anpassungen in einer Factory-Klasse (javax.net.ssl.SSLSocketFactory) abstrahiert.

In Ihrer JSSE-Dokumentation finden Sie Details zur Erstellung einer angepassten SSLSocketFactory-Implementierung. Die Details unterscheiden sich von Provider zu Provider, aber eine typische Reihenfolge von Schritten könnte wie folgt lauten:

1. Erstellung eines SSLContext-Objekts mithilfe einer statischen Methode in SSLContext
2. Initialisierung dieses SSLContext-Objekts mit geeigneten KeyManager- und TrustManager-Implementierungen (diese werden auf Basis ihrer eigenen Factory-Klassen erstellt)
3. Erstellung eines SSLSocketFactory aus dem SSLContext

Wenn Sie über ein SSLSocketFactory-Objekt verfügen, setzen Sie MQEnvironment.sslSocketFactory auf das angepasste Factory-Objekt. For example:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java verwenden diese SSLSocketFactory für die Verbindung mit dem IBM MQ-Warteschlangenmanager. Diese Eigenschaft kann auch mithilfe von CMQC.SSL_SOCKET_FACTORY_PROPERTY

festgelegt werden. Wenn sslSocketFactory auf null gesetzt ist, wird die standardmäßige SSLSocketFactory der JVM verwendet. Diese Eigenschaft wird ignoriert, wenn sslCipherSuite nicht festgelegt ist.

Wenn Sie angepasste SSLSocketFactorys verwenden, berücksichtigen Sie die Auswirkung der gemeinsamen Nutzung einer TCP/IP-Verbindung. Wenn die gemeinsame Verbindungsnutzung möglich ist, wird kein neues Socket der bereitgestellten SSLSocketFactory angefordert. Dies gilt selbst dann, wenn das erstellte Socket sich im Kontext einer nachfolgenden Verbindungsanforderung auf irgendeine Weise unterscheiden würde. Wenn bei einer nachfolgenden Verbindung beispielsweise ein anderes Clientzertifikat vorgelegt werden soll, darf die gemeinsame Verbindungsnutzung nicht zulässig sein.

Änderungen am JSSE-Schlüsselspeicher oder -Truststore in IBM MQ classes for Java vornehmen
Wenn Sie den JSSE-Schlüsselspeicher oder -Truststore ändern, müssen Sie bestimmte Aktionen ausführen, damit die Änderungen in Kraft treten.

Wenn Sie Änderungen am Inhalt des JSSE-Schlüsselspeichers oder -Truststores vornehmen oder die Position der Schlüsselspeicher- bzw. Truststore-Datei ändern, werden die Änderungen nicht automatisch von den Anwendungen der IBM MQ classes for Java übernommen, die zu diesem Zeitpunkt aktiv sind. Damit die Änderungen in Kraft treten, müssen die folgenden Aktionen ausgeführt werden:

- Die Anwendungen müssen alle ihre Verbindungen schließen und nicht verwendete Verbindungen in Verbindungspools löschen.
- Wenn Ihr JSSE-Provider Informationen aus dem Schlüsselspeicher und Truststore zwischenspeichert, müssen diese Informationen aktualisiert werden.

Nachdem diese Aktionen ausgeführt wurden, können die Anwendungen ihre Verbindungen erneut erstellen.

Abhängig vom Design Ihrer Anwendungen und je nachdem, welche Funktion von Ihrem JSSE-Provider bereitgestellt wird, können diese Aktionen unter Umständen ohne Stoppen und Neustart Ihrer Anwendungen ausgeführt werden. Das Stoppen und der Neustart der Anwendungen ist aber gegebenenfalls die einfachste Lösung.

Fehlerbehandlung bei Verwendung von TLS in Verbindung mit IBM MQ classes for Java
Bei der Herstellung einer Verbindung zu einem Warteschlangenmanager über TLS kann IBM MQ classes for Java verschiedene Ursachencodes ausgeben.

Diese werden in der folgenden Liste erläutert:

MQRC_SSL_NOT_ALLOWED
Die sslCipherSuite-Eigenschaft wurde festgelegt, es wurde jedoch eine Bindungsverbindung verwendet. TLS wird jedoch nur von Clientverbindungen unterstützt.

MQRC_JSSE_ERROR
Der JSSE-Provider hat einen Fehler gemeldet, der nicht von IBM MQ behandelt werden konnte. Dies kann auf ein Konfigurationsproblem bei JSSE oder darauf zurückzuführen sein, dass das vom Warteschlangenmanager vorgelegte Zertifikat nicht überprüft werden konnte. Die von JSSE erstellte Ausnahmebedingung kann mithilfe der Methode getCause() in MQException abgerufen werden.

MQRC_SSL_INITIALIZATION_ERROR
Ein MQCONN- oder MQCONNX-Aufruf mit TLS-Konfigurationsoptionen wurde ausgegeben, bei der Initialisierung der TLS-Umgebung ist jedoch ein Fehler aufgetreten.

MQRC_SSL_PEER_NAME_MISMATCH
Das in der Eigenschaft 'sslPeerName' angegebene DN-Muster stimmte nicht mit dem DN überein, der vom Warteschlangenmanager vorgelegt wurde.

MQRC_SSL_PEER_NAME_ERROR
Das in der Eigenschaft 'sslPeerName' angegebene DN-Muster war nicht gültig.

MQRC_UNSUPPORTED_CIPHER_SUITE
Die in sslCipherSuite genannte Cipher-Suite wurde nicht vom JSEE-Provider erkannt. Eine vollständige Liste der vom JSEE-Provider unterstützten Cipher-Suites kann über ein Programm mithilfe der Methode SSLSocketFactory.getSupportedCipherSuites() abgerufen werden. Sie finden eine Liste der

Cipher-Suites, die für die Kommunikation mit IBM MQ verwendet werden können, im Abschnitt „TLS CipherSpecs und Cipher-Suites in IBM MQ classes for Java” auf Seite 430.

MQRC_SSL_CERTIFICATE_REVOKED
Das vom Warteschlangenmanager vorgelegte Zertifikat wurde in einer CRL gefunden, die mit der Eigenschaft 'sslCertStores' angegeben wurde. Aktualisieren Sie den Warteschlangenmanager, damit er vertrauenswürdige Zertifikate verwendet.

MQRC_SSL_CERT_STORE_ERROR
Keiner der bereitgestellten Zertifikatsspeicher (CertStores) konnte nach dem Zertifikat durchsucht werden, das vom Warteschlangenmanager vorgelegt wurde. Die Methode MQException.getCause() gibt den Fehler zurück, der beim Suchvorgang im ersten versuchten CertStore aufgetreten ist. Wenn die kausale Ausnahme 'NoSuchElementException', 'ClassCastException' oder 'NullPointerException' lautet, prüfen Sie, ob die in der Eigenschaft 'sslCertStores' angegebene Objektgruppe mindestens ein gültiges CertStore-Objekt enthält.

TLS CipherSpecs und Cipher-Suites in IBM MQ classes for Java
Ob Anwendungen der IBM MQ classes for Java Verbindungen zu einem Warteschlangenmanager herstellen können, hängt davon ab, welche CipherSpec am Serverende des MQI-Kanals und welche Cipher-Suite am Clientende angegeben ist.

In der folgenden Tabelle sind die von IBM MQ unterstützten CipherSpecs sowie die entsprechenden Cipher-Suites aufgeführt.

Deprecated Überprüfen Sie im Abschnitt Nicht weiter unterstützte CipherSpecs, ob und ggf. ab welchem Upgrade die in der folgenden Tabelle aufgeführten CipherSpecs von IBM MQ nicht mehr unterstützt wurden.

Wichtig: Die aufgeführten Cipher-Suites werden von der mit IBM MQ bereitgestellten IBM Java Runtime Environment (JRE) unterstützt. Sie beinhalten auch die von der Oracle Java-JRE unterstützten Cipher-Suites. Weitere Informationen zur Konfiguration Ihrer Anwendung für die Verwendung einer Oracle Java JRE finden Sie im Abschnitt Anwendung für die Verwendung von IBM Java oder Oracle Java CipherSuite -Zuordnungen konfigurieren.

In der Tabelle ist auch das für die Kommunikation verwendete Protokoll angegeben sowie, ob die Cipher-Suite dem FIPS 140-2-Standard entspricht.

Anmerkung: Unter AIX, Linux, and Windowsstellt IBM MQ die Konformität mit FIPS 140-2 über das Verschlüsselungsmodul IBM Crypto for C (ICC) bereit. Das Zertifikat für dieses Modul wurde in den Langzeitstatus versetzt. Kunden sollten das IBM Crypto for C (ICC) -Zertifikat anzeigen und sich über alle Empfehlungen von NIST im Klaren sein. Ein Ersatz-FIPS 140-3-Modul ist derzeit in Bearbeitung und sein Status kann angezeigt werden, indem in der NIST-CMVP-Module in der Prozessliste nach ihm gesucht wird.

IBM MQ Operator 3.2.0 und das Container-Image des Warteschlangenmanagers ab 9.4.0.0 basieren auf UBI 9. Die Konformität mit FIPS 140-3 steht derzeit an und ihr Status kann angezeigt werden, indem Sie in der NIST CMVP-Module in der Prozessliste nach "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" suchen.

Wenn die Anwendung nicht so konfiguriert ist, dass eine Einhaltung des FIPS 140-2-Standards erzwungen wird, können Cipher-Suites, die als FIPS 140-2-konform gekennzeichnet sind, verwendet werden. Wurde die Anwendung für eine Einhaltung des FIPS 140-2-Standards konfiguriert (siehe die folgenden Hinweise zur Konfiguration), können nur die Cipher-Suites verwendet werden, die als FIPS 140-2-konform gekennzeichnet sind. Bei einem Versuch, eine andere Cipher-Suite zu verwenden, wird ein Fehler zurückgegeben.

Anmerkung: Jede JRE kann über mehrere Provider für Sicherheit über Verschlüsselung verfügen, von denen jeder eine Implementierung derselben Cipher-Suite bereitstellt. Nicht alle Sicherheitsprovider sind allerdings FIPS 140-2-zertifiziert. Wenn für eine Anwendung keine Einhaltung des FIPS 140-2-Standards erzwungen wird, besteht die Möglichkeit, dass eine nicht zertifizierte Implementierung der Cipher-Suite verwendet wird. Nicht zertifizierte Implementierungen arbeiten allerdings unter Umständen nicht FIPS 140-2-konform, auch wenn die Cipher-Suite theoretisch die von diesem Standard geforderten Mindestsicherheitsanforderungen erfüllt. Die folgenden Hinweise enthalten weitere Informationen zur Konfiguration von IBM MQ Java-Anwendungen, damit die Einhaltung des FIPS 140-2-Standards erzwungen wird.

Weitere Informationen zur Konformität mit FIPS 140-2 und Suite-B bei CipherSpecs und Cipher-Suites finden Sie unter Specifying CipherSpecs. Unter Umständen müssen Sie auch Informationen in Zusammenhang mit den US-amerikanischen Federal Information Processing Standards beachten.

Damit damit alle Cipher-Suites verwendet werden können und FIPS 140-2- und/oder Suite-B-zertifiziert arbeiten, ist eine entsprechende JRE erforderlich.IBM Java 7 Serviceaktualisierung 4 Fixpack 2 oder eine höhere Version von IBM JRE bietet die entsprechende Unterstützung für die TLS 1.2 CipherSuites , die in Tabelle 59 auf Seite 431aufgelistet sind.

Um TLS 1.3 -Verschlüsselungen verwenden zu können, muss die JRE, in der Ihre Anwendung ausgeführt wird, TLS 1.3unterstützen.

Anmerkung: Für die Verwendung einiger Cipher-Suites müssen die uneingeschränkten Richtliniendateien in der JRE konfiguriert werden. Ausführlichere Informationen zur Konfiguration der Richtliniendateien in einem SDK oder in einer JRE finden Sie im Abschnitt *IBM SDK Policy files* der *Security Reference for IBM SDK, Java Technology Edition* für die von Ihnen verwendete Version.

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites

CipherSpec **„1" auf Seite 450**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1" auf Seite 450**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1” auf Seite 450**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	nein
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1" auf Seite 450**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	ja
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	nein
TLS_RSA_WITH_3DES_EDE_CBC_SHA „2” auf Seite 450	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	nein „4” auf Seite 450

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	nein „4” auf Seite 450
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	nein „4” auf Seite 450

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	nein „4” auf Seite 450
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	nein „4” auf Seite 450

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1” auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	nein „4” auf Seite 450
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	nein „4” auf Seite 450

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	nein
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1” auf Seite 450**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	nein
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	ja
TLS_AES_128_GCM_SHA256 „3” auf Seite 450	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1" auf Seite 450**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_AES_256_GCM_SHA384 „3" auf Seite 450	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	nein
TLS_CHACHA20_POLY1305_SHA256 „3" auf Seite 450	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec **„1" auf Seite 450**	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_AES_128_CCM_SHA256 „3" auf Seite 450	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	nein
TLS_AES_128_CCM_8_SHA256 „3" auf Seite 450	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	nein
Beliebig „3" auf Seite 450	*ANY	*ANY	Mehrfach	nein
ANY_TLS13 „3" auf Seite 450	*TLS13	*TLS13	TLS V13	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec „1" auf Seite 450	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ANY_TLS12_OR_HIGHER „3" auf Seite 450	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 und höher	nein
ANY_TLS13_OR_HIGHER „3" auf Seite 450	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 und höher	nein

Anmerkungen:

1. Dies ist der Wert, der in einem Kanal in IBM MQkonfiguriert ist, einschließlich einer CCDT (Binary oder JSON).
2. Deprecated Die CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA wird nicht weiter unterstützt. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.
3. Damit TLS- v1.3 -Verschlüsselungen verwendet werden können, muss die Java runtime environment (JRE), die Ihre Anwendung ausführt, TLS v1.3unterstützen.
4. V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0entfernt die IBM Java 8 JRE die Unterstützung für den RSA-Schlüsselaustausch, wenn sie im FIPS-Modus ausgeführt wird.

Cipher-Suites und FIPS-Konformität in einer IBM MQ classes for Java-Anwendung konfigurieren

- Für eine Anwendung, die IBM MQ classes for Java verwendet, gibt es die folgenden beiden Möglichkeiten, die Cipher-Suite für eine Verbindung zu konfigurieren:
  - Setzen Sie das Feld 'sslCipherSuite' in der Klasse 'MQEnvironment' auf den Namen der Cipher-Suite.
  - Setzen Sie die Eigenschaft CMQC.SSL_CIPHER_SUITE_PROPERTY in der Hashtabelle mit Eigenschaften, die an den MQQueueManager-Konstruktor übergeben wurde, auf den Namen der Cipher-Suite.
- Eine Anwendung, die IBM MQ classes for Java verwendet, kann die Konformität mit FIPS 140-2 auf zwei Arten durchsetzen:
  - Setzen Sie das Feld 'sslFipsRequired' in der Klasse 'MQEnvironment' auf 'true'.
  - Setzen Sie die Eigenschaft CMQC.SSL_FIPS_REQUIRED_PROPERTY in der Hashtabelle mit den Eigenschaften, die an den MQQueueManager-Konstruktor übergeben wurde, auf 'true'.

Anwendung für Oracle IBM Java- oder Oracle Java-Cipher-Suite-Zuordnungen konfigurieren

V 9.4.0 Ab IBM MQ 9.4.0kann ein Cipher entweder als CipherSpec oder als CipherSuite -Name definiert werden und wird von IBM MQordnungsgemäß verarbeitet.

Anmerkung: Removed Die Java Systemeigenschaft `com.ibm.mq.cfg.useIBMCipherMappings`, die steuert, welche Zuordnungen in früheren Versionen von IBM MQverwendet werden, wird nicht mehr benötigt und aus dem Produkt unter IBM MQ 9.4.0entfernt.

Einschränkungen bei der Interoperabilität

Je nach verwendetem Protokoll sind einige Cipher-Suites auch mit mehreren IBM MQ-CipherSpecs kompatibel, allerdings wird nur die Cipher-Suite/CipherSpec-Kombination unterstützt, die die in Tabelle 1 angegeben TLS-Version verwendet. Versuche, eine der nicht unterstützten Cipher-Suite/CipherSpecs-Kombinationen zu verwenden, schlagen mit einer entsprechenden Ausnahmebedingung fehl. In Installationen, in denen eine dieser Cipher-Suite/CipherSpec-Kombinationen verwendet wird, sollte zur Verwendung einer unterstützten Kombination übergegangen werden.

In der folgenden Tabelle sind die Cipher-Suites aufgeführt, für die diese Einschränkung gilt.

Tabelle 60. Cipher-Suites und die unterstützten bzw. nicht unterstützten CipherSpecs

CipherSuite	Unterstützte TLS-CipherSpec	Nicht unterstützte SSL-CipherSpec
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA „1" auf Seite 451	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Anmerkung:

1. Deprecated Die CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA wird nicht weiter unterstützt. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.

Anwendungen der IBM MQ classes for Java ausführen

Wenn Sie eine Anwendung schreiben (eine Klasse, die eine main()-Methode enthält) und dabei entweder den Client- oder den Bindungsmodus verwenden, führen Sie Ihr Programm mit dem Java-Interpreter aus.

Verwenden Sie den folgenden Befehl:

```
java -Djava.library.path= library_path MyClass
```

Dabei steht *Bibliothekspfad* für den Pfad der Bibliotheken mit den IBM MQ classes for Java. Weitere Informationen finden Sie in „IBM MQ classes for Java-Bibliotheken" auf Seite 375.

Zugehörige Tasks

Tracing von IBM MQ classes for Java-Anwendungen

Traceerstellung für IBM MQ-Ressourcenadapter ausführen

Umgebungsabhängiges Verhalten von IBM MQ classes for Java

IBM MQ classes for Java ermöglichen Ihnen die Erstellung von Anwendungen, die für verschiedene Versionen von IBM MQ ausgeführt werden können. In dieser Themensammlung wird das Verhalten der Java-Klassen bei den verschiedenen Versionen beschrieben.

IBM MQ classes for Java stellen einen Kern von Klassen zur Verfügung, die in allen Umgebungen eine konsistente Funktion und ein konsistentes Verhalten aufweisen. Funktionen außerhalb dieses Kerns hängen von der Funktionalität des Warteschlangenmanagers ab, mit dem die Anwendung verbunden wird.

Sofern in den vorliegenden Informationen nicht anders angegeben, entspricht das Verhalten der Beschreibung zum jeweiligen Warteschlangenmanager in der MQI-Anwendungsreferenz.

Kernklassen in IBM MQ classes for Java

IBM MQ classes for Java enthalten eine Kerngruppe von Klassen (Core-Klassen), die in allen Umgebungen verwendet werden können.

Die Klassen in der folgenden Gruppe gelten als Kernklassen, die in allen Umgebungen verwendet werden können und lediglich geringfügige Varianten aufweisen. Diese Varianten sind im Abschnitt „Einschränkungen und Varianten bei Kernklassen der IBM MQ classes for Java" auf Seite 453 aufgelistet.

- MQ-Umgebung
- MQException
- MQGetMessageOptions

  Ausgenommen:

  – MatchOptions
  – GroupStatus
  – SegmentStatus
  – Segmentation
- MQManagedObject

  Ausgenommen:

  – inquire()
  – set()
- MQMessage

  Ausgenommen:

  – groupId
  – messageFlags
  – messageSequenceNumber

  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  Ausgenommen:

  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  Ausgenommen:

  - begin( )
  - accessDistributionList()
- MQSimpleConnectionManager
- MQTopic
- MQC

Anmerkung:

1. Manche Konstanten sind nicht im Kern eingeschlossen (Details hierzu finden Sie im Abschnitt „Einschränkungen und Varianten bei Kernklassen der IBM MQ classes for Java” auf Seite 453); verwenden Sie diese nicht in vollständig portierbaren Programmen.
2. Einige Plattformen unterstützen nicht alle Verbindungsmodi. Auf diesen Plattformen können Sie nur die Kernklassen und Optionen verwenden, die sich auf die unterstützten Modi beziehen.

Einschränkungen und Varianten bei Kernklassen der IBM MQ classes for Java

Die Kernklassen verhalten sich im Allgemeinen in allen Umgebungen gleich. Dies gilt selbst dann, wenn die entsprechenden MQI-Aufrufe normalerweise umgebungsspezifische Unterschiede aufweisen. Mit Ausnahme der folgenden geringen Einschränkungen und Varianten ist das Verhalten konsistent, egal ob ein Warteschlangenmanager unter AIX, Linux oder Windowsverwendet wird.

Einschränkungen bei MQGMO_*-Werten in IBM MQ classes for Java
Bestimmte MQGMO_*-Werte werden nicht von allen Warteschlangenmanagern unterstützt.

Die Verwendung der folgenden MQGMO_*-Werte kann dazu führen, dass MQQueue.get() eine MQ-Ausnahmebedingung (MQException) auslöst:

MQGMO_SYNCPOINT_IF _PERSISTENT
MQGMO_MARK_SKIP _BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_ AVAILABLE

MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

Außerdem wird MQGMO_SET_SIGNAL nicht unterstützt, wenn dieser Wert über Java verwendet wird.

Einschränkungen bei MQPMRF_*-Werten in IBM MQ classes for Java
Diese werden nur beim Einreihen von Nachrichten in eine Verteilerliste verwendet und nur von Warteschlangenmanagern unterstützt, die wiederum Verteilerlisten unterstützen. z/OS-Warteschlangenmanager unterstützen beispielsweise keine Verteilerlisten.

Einschränkungen bei MQPMO_*-Werten in IBM MQ classes for Java
Bestimmte MQPMO_*-Werte werden nicht von allen Warteschlangenmanagern unterstützt.

Die Verwendung der folgenden MQPMO_*-Werte kann dazu führen, dass MQQueue.put() oder MQQueueManager.put() eine MQ-Ausnahmebedingung (MQException) auslöst:

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

Einschränkungen und Varianten bei MQCNO_*-Werten in IBM MQ classes for Java
Bestimmte MQCNO_*-Werte werden nicht unterstützt.

- Die automatische Clientwiederverbindung wird von IBM MQ classes for Javanicht unterstützt. Unabhängig davon, welchen Wert für MQCNO_RECONNECT_* Sie festlegen, verhält sich die Verbindung weiterhin, als ob Sie MQCNO_RECONNECT_DISABLED festgelegt hätten.
- MQCNO_FASTPATH wird in Warteschlangenmanagern ignoriert, die MQCNO_FASTPATH nicht unterstützen. Dieser Wert wird auch von Clientverbindungen ignoriert.

Einschränkungen bei MQRO_*-Werten in IBM MQ classes for Java
Die folgenden Berichtsoptionen können festgelegt werden.

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

Weitere Informationen hierzu finden Sie im Abschnitt Bericht.

Miscellaneous differences between IBM MQ classes for Java on z/OS and other platforms
IBM MQ for z/OS behaves differently from IBM MQ on other platforms in some areas.

BackoutCount
A z/OS queue manager returns a maximum BackoutCount of 255, even if the message has been backed out more than 255 times.

Default dynamic queue prefix
When connected to a z/OS queue manager using a bindings connection, the default dynamic queue prefix is CSQ.*. Otherwise, the default dynamic queue prefix is AMQ.*.

MQQueueManager constructor
Client connect is not supported on z/OS. Attempting to connect with client options results in an MQException with MQCC_FAILED and MQRC_ENVIRONMENT_ERROR.

The MQQueueManager constructor might also fail with MQRC_CHAR_CONVERSION_ERROR (if it fails to initialize conversion between the IBM-1047 and ISO8859-1 code pages), or MQRC_UCS2_CON-

VERSION_ERROR (if it fails to initialize conversion between the queue manager's code page and Unicode). If your application fails with one of these reason codes, ensure that the National Language Resources component of Language Environment is installed, and ensure that the correct conversion tables are available.

Conversion tables for Unicode are installed as part of the z/OS C/C++ optional feature. See the *z/OS C/C++ Programming Guide*, SC09-4765, for more information about enabling UCS-2 conversions.

Funktionen außerhalb der Kernklassen von IBM MQ classes for Java

IBM MQ classes for Java enthalten bestimmte Funktionen, die eigens für die Verwendung von API-Erweiterungen konzipiert wurden, die nicht von allen Warteschlangenmanagern unterstützt werden. In diesen Abschnitten wird beschrieben, wie sich diese verhalten, wenn ein Warteschlangenmanager verwendet wird, von dem sie nicht unterstützt werden.

Varianten in der MQQueueManager-Konstruktoroption
Einige MQQueueManager-Konstruktoren schließen ein optionales Ganzzahlargument ein. Einige Werte dieses Arguments werden nicht auf allen Plattformen akzeptiert.

Wenn ein MQQueueManager-Konstruktor ein optionales Ganzzahlargument einschließt, wird es dem MQCNO-Optionsfeld der MQI zugeordnet und für den Wechsel zwischen einer normalen Verbindung und einer Direktaufrufverbindung verwendet. Diese erweiterte Form des Konstruktors wird in allen Umgebungen akzeptiert, wenn nur die Option MQCNO_STANDARD_BINDING oder MQCNO_FASTPATH_BINDING verwendet wird. Alle anderen Optionen führen dazu, dass der Konstruktor mit MQRC_OPTIONS_ERROR fehlschlägt. Die Direktaufrufsoption CMQC.MQCNO_FASTPATH_BINDING wird nur bei einer Bindungsverbindung mit einem Warteschlangenmanager berücksichtigt, von dem sie unterstützt wird. In anderen Umgebungen wird sie ignoriert.

Einschränkungen bei der Methode MQQueueManager.begin()
Diese Methode kann nur für einen IBM MQ -Warteschlangenmanager auf AIX, Linux, and Windows -Systemen im Bindungsmodus verwendet werden. Andernfalls schlägt sie mit MQRC_ENVIRONMENT_ERROR fehl.

Weitere Informationen finden Sie in „JTA/JDBC-Koordination unter Verwendung von IBM MQ classes for Java” auf Seite 420.

Varianten in den MQGetMessageOptions-Feldern
Da manche Warteschlangenmanager die MQGMO-Struktur der Version 2 nicht unterstützen, müssen Sie einige Felder auf ihre Standardwerte setzen.

Wenn Sie einen Warteschlangenmanager verwenden, der die MQGMO-Struktur der Version 2 nicht unterstützt, übernehmen Sie für folgende Fehler die jeweiligen Standardwerte:

GroupStatus
SegmentStatus
Segmentation

Das Feld 'MatchOptions' unterstützt außerdem nur MQMO_MATCH_MSG_ID und MQMO_MATCH_CORREL_ID. Wenn Sie nicht unterstützte Werte in diese Felder stellen, schlägt die nachfolgende Methode MQDestination.get() mit MQRC_GMO_ERROR fehl. Wenn der Warteschlangenmanager die MQGMO-Struktur der Version 2 nicht unterstützt, werden diese Felder nach einer erfolgreichen Ausführung von MQDestination.get() nicht aktualisiert.

Einschränkungen bei Verteilerlisten in IBM MQ classes for Java
Nicht alle Warteschlangenmanager ermöglichen Ihnen das Öffnen einer MQDistributionList.

Die folgenden Klassen werden für die Erstellung von Verteilerlisten verwendet:

MQDistributionList
MQDistributionListItem
MQMessageTracker

Sie können MQDistributionLists und MQDistributionListItems in jeder Umgebung erstellen und füllen, allerdings ermöglichen Ihnen nicht alle Warteschlangenmanager das Öffnen einer MQDistributionList. Insbesondere z/OS-Warteschlangenmanager unterstützen keine Verteilerlisten. Der Versuch, eine MQDistributionList bei Verwendung eines solchen Warteschlangenmanagers zu öffnen, führt zum Fehler MQRC_OD_ERROR..

Varianten in MQPutMessageOptions-Feldern
Wenn ein Warteschlangenmanager keine Verteilerlisten unterstützt, werden bestimmte MQPMO-Felder anders behandelt.

Vier Felder in MQPMO werden als die folgenden Elementvariablen in der MQPutMessageOptions-Klasse wiedergegeben:

knownDestCount
unknownDestCount
invalidDestCount
recordFields

Diese Felder sind in erster Linie für die Verwendung in Verbindung mit Verteilerlisten vorgesehen. Allerdings füllt ein Warteschlangenmanager, der Verteilerlisten unterstützt, auch die DestCount-Felder nach einem MQPUT-Vorgang in einer einzelnen Warteschlange aus. Wenn die Warteschlange beispielsweise in eine lokale Warteschlange aufgelöst wird, wird knownDestCount auf 1 gesetzt, während die übrigen beiden Zählerfelder auf 0 gesetzt werden.

Wenn der Warteschlangenmanager keine Verteilerlisten unterstützt, werden diese Werte wie folgt simuliert:

- Wenn die Methode put() erfolgreich ausgeführt wird, wird unknownDestCount auf 1 gesetzt, während die übrigen Felder auf 0 gesetzt werden.
- Wenn die Methode put() fehlschlägt, wird invalidDestCount auf 1 gesetzt, während die übrigen Felder auf 0 gesetzt werden.

Die Variable 'recordFields' wird in Verbindung mit Verteilerlisten verwendet. Ein Wert kann unabhängig von der Umgebung jederzeit in recordFields geschrieben werden. Er wird ignoriert, wenn das MQPutMessageOptions-Objekt nicht bei einem MQDistributionList.put()-Vorgang, sondern bei einem MQDestination.put()- oder MQQueueManager.put()-Vorgang verwendet wird.

Einschränkungen bei MQMD-Feldern mit IBM MQ classes for Java
Für bestimmte MQMD-Felder im Zusammenhang mit der Nachrichtensegmentierung sollte ihr Standardwert übernommen werden, wenn ein Warteschlangenmanager verwendet wird, der keine Segmentierung unterstützt.

Die folgenden MQMD-Felder sind hauptsächlich bei der Nachrichtensegmentierung relevant:

GroupId
MsgSeqNumber
Offset
MsgFlags
OriginalLength

Wenn eine Anwendung eines dieser MQMD-Felder auf andere Werte als die jeweiligen Standardwerte setzt und anschließend einen put()- oder get()-Aufruf bei einem Warteschlangenmanager ausgibt, der diese nicht unterstützt, gibt put() oder get() eine MQ-Ausnahmebedingung (MQException) mit MQRC_MD_ERROR aus. Ein erfolgreicher put()- oder get()-Vorgang bei einem solchen Warteschlangenmanager führt immer dazu, dass die MQMD-Felder die jeweiligen Standardwerte enthalten. Senden Sie keine gruppierte oder segmentierte Nachricht an eine Java-Anwendung, die für einen Warteschlangenmanager ausgeführt wird, der keine Gruppierung oder Segmentierung von Nachrichten unterstützt.

Falls eine Java-Anwendung versucht, mit get() eine Nachricht aus einem Warteschlangenmanager abzurufen, der diese Felder nicht unterstützt, und die abzurufende physische Nachricht zu einer Gruppe von segmentierten Nachrichten gehört (sie also in den MQMD-Feldern vom Standard abweichende Werte enthält), wird sie ohne Fehler abgerufen. Allerdings werden die MQMD-Felder in der MQMessage nicht

aktualisiert, die MQMessage-Formateigenschaft wird auf MQFMT_MD_EXTENSION gesetzt und die tatsächlichen Nachrichtendaten erhalten als Präfix eine MQMDE-Struktur, die die Werte für die neuen Felder enthält.

Einschränkungen bei IBM MQ classes for Java für den CICS Transaction Server

In einer Umgebung mit CICS Transaction Server for z/OS darf nur der Hauptthread (der erste Thread) CICS- oder IBM MQ-Aufrufe ausgeben.

Beachten Sie, dass die Verwendung von IBM MQ JMS-Klassen in einer CICS Java-Anwendung nicht unterstützt wird.

Es ist daher nicht möglich, MQQueueManager- oder MQQueue-Objekte zwischen den Threads in dieser Umgebung gemeinsam zu nutzen oder einen neuen MQQueueManager in einem untergeordneten Thread zu erstellen.

z/OS Im Abschnitt „Miscellaneous differences between IBM MQ classes for Java on z/OS and other platforms” auf Seite 454 finden Sie einige Einschränkungen und Varianten, die für die IBM MQ classes for Java bei einer Ausführung in Verbindung mit einem z/OS-Warteschlangenmanager gelten. Außerdem werden die Transaktionssteuerungsmethoden in MQQueueManager bei einer Ausführung unter CICS nicht unterstützt. Anwendungen geben nicht MQQueueManager.commit() oder MQQueueManager.backout() aus, sondern verwenden die Synchronisierungsmethoden Task.commit() und Task.rollback() der JCICS-Task. Die Task-Klasse wird von JCICS im Paket com.ibm.cics.server bereitgestellt.

IBM MQ-Ressourcenadapter verwenden

Der Ressourcenadapter ermöglicht Anwendungen, die auf einem Anwendungsserver ausgeführt werden, den Zugriff auf IBM MQ-Ressourcen. Er unterstützt sowohl die eingehende Kommunikation als auch die abgehende Kommunikation.

Inhalt des Ressourcenadapters

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. Zusätzlich zu dem Ressourcenadapter, der Java EE und JMS 2.0unterstützt, stellt IBM MQ 9.3.0 und höher einen Ressourcenadapter bereit, der Jakarta Messagingunterstützt.

JM 3.0 **IBM MQ -Ressourcenadapter für Jakarta Messaging**

Jakarta Connectors Architecture stellt eine Standardmethode für die Verbindung von Anwendungen, die in einer Jakarta EE -Umgebung ausgeführt werden, mit einem Enterprise Information System (EIS) wie IBM MQ oder Db2bereit. Der IBM MQ -Ressourcenadapter für Jakarta Messaging implementiert die Jakarta Connectors 2.0.0 -Schnittstellen und enthält die IBM MQ classes for Jakarta Messaging. Sie ermöglicht Jakarta Messaging -Anwendungen und Message-driven Beans (MDBs), die in einem Anwendungsserver ausgeführt werden, den Zugriff auf die Ressourcen eines IBM MQ -Warteschlangenmanagers. Der Ressourcenadapter unterstützt sowohl die Punkt-zu-Punkt-Domäne als auch die Publish/Subscribe-Domäne.

JMS 2.0 **IBM MQ -Ressourcenadapter für JMS 2.0**

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. Der IBM MQ -Ressourcenadapter für JMS 2.0 implementiert die JCA 1.7 -Schnittstellen und enthält die IBM MQ classes for JMS. Er ermöglicht JMS-Anwendungen und nachrichtengesteuerten Beans (Message-driven Beans, MDBs), die auf einem Anwendungsserver ausgeführt werden, den Zugriff auf die Ressourcen eines IBM MQ-Warteschlangenmanagers. Der Ressourcenadapter unterstützt sowohl die Punkt-zu-Punkt-Domäne als auch die Publish/Subscribe-Domäne.

Der IBM MQ-Ressourcenadapter unterstützt zwei Arten der Kommunikation zwischen einer Anwendung und einem Warteschlangenmanager:

Abgehende Kommunikation
Eine Anwendung startet eine Verbindung mit einem Warteschlangenmanager und sendet dann JMS-Nachrichten an JMS-Ziele und empfängt JMS-Nachrichten von JMS-Zielen. Diese Verarbeitung verläuft synchron.

Eingehende Kommunikation
Eine JMS-Nachricht, die bei einem JMS-Ziel eingeht, wird an eine MDB zugestellt, die die Nachricht asynchron verarbeitet.

Darüber hinaus enthält der Ressourcenadapter die IBM MQ classes for Java. Die Klassen sind automatisch für Anwendungen verfügbar, die in einem Anwendungsserver ausgeführt werden, in dem der Ressourcenadapter implementiert wurde, und ermöglichen Anwendungen, die in diesem Anwendungsserver ausgeführt werden, die Verwendung der API IBM MQ classes for Java , wenn sie auf Ressourcen eines IBM MQ -Warteschlangenmanagers zugreifen.

Die Verwendung der IBM MQ classes for Java innerhalb einer Java EE-Umgebung wird mit Einschränkungen unterstützt. Informationen zu diesen Einschränkungen finden Sie im Abschnitt „IBM MQ classes for Java-Anwendungen in Java EE ausführen” auf Seite 367.

Zu verwendende Version des Ressourcenadapters

Die Version des Ressourcenadapters, die Sie verwenden, hängt davon ab, ob Sie ihn in einem Anwendungsserver implementieren, der Jakarta EE oder Java EEunterstützt:

JM 3.0 **Jakarta EE**
Ab IBM MQ 9.3.0wird Jakarta Messaging 3.0 unterstützt. Der IBM MQ -Ressourcenadapter für Jakarta Messaging muss in einem Anwendungsserver implementiert werden, der Jakarta EEunterstützt.

Java EE 7
Der Ressourcenadapter von IBM MQ 8.0 und höher unterstützt JCA v1.7 und stellt JMS 2.0-Unterstützung bereit. Dieser Ressourcenadapter muss in einem Anwendungsserver von Java EE 7 und höher bereitgestellt werden (siehe „Unterstützungserklärung für IBM MQ-Ressourcenadapter” auf Seite 459).

Sie können den Ressourcenadapter der IBM MQ 8.0 und höher auf jedem beliebigen Anwendungsserver installieren, dessen Kompatibilität mit der Spezifikation der Java Platform, Enterprise Edition 7 zertifiziert ist. Mit dem Ressourcenadapter von IBM MQ 8.0 oder höher kann eine Anwendung über den BINDINGS-oder CLIENT-Transport eine Verbindung zu einem Warteschlangenmanager herstellen.

Wichtig: Der Ressourcenadapter von IBM MQ 8.0 oder höher kann nur auf einem Anwendungsserver bereitgestellt werden, der JMS 2.0 unterstützt.

Ressourcenadapter mit WebSphere Application Server traditional verwenden

Der IBM MQ -Ressourcenadapter ist in WebSphere Application Server traditional 9.0 oder höher vorinstalliert. Deshalb muss kein neuer Ressourcenadapter installiert werden.

JM 3.0 WebSphere Application Server traditional unterstützt derzeit nicht Jakarta EE. Weitere Informationen finden Sie unter IBM MQ resource adapter statement of support.

Anmerkung: Ein Ressourcenadapter von IBM MQ 9.0 oder höher kann im Transportmodus CLIENT oder BINDINGS eine Verbindung zu jedem in Betrieb befindlichen IBM MQ-Warteschlangenmanager herstellen.

Ressourcenadapter mit WebSphere Liberty verwenden

Um eine Verbindung zu IBM MQ von WebSphere Libertyherzustellen, müssen Sie den IBM MQ -Ressourcenadapter verwenden. Da Liberty den IBM MQ-Ressourcenadapter nicht enthält, müssen Sie ihn separat aus Fix Central beziehen.

Die Version des verwendeten Ressourcenadapters hängt davon ab, ob Sie ihn in einer Version von Liberty implementieren, die Jakarta EE oder Java EEunterstützt.

Weitere Informationen zum Herunterladen und Installieren des Ressourcenadapters finden Sie im Abschnitt „Ressourcenadapter in Liberty installieren” auf Seite 467.

Zugehörige Konzepte

„Ressourcenadapter für eingehende Kommunikation konfigurieren” auf Seite 475
Definieren Sie die Eigenschaften eines oder mehrerer ActivationSpec-Objekte, um die eingehende Kommunikation zu konfigurieren.

„Ressourcenadapter für abgehende Kommunikation konfigurieren” auf Seite 494
Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

„IBM MQ classes for JMS/Jakarta Messaging verwenden” auf Seite 85
IBM MQ classes for JMS und IBM MQ classes for Jakarta Messaging sind die Java -Messaging-Provider, die mit IBM MQbereitgestellt werden. Neben der Implementierung der in den Spezifikationen JMS und Jakarta Messaging definierten Schnittstellen fügen diese Messaging-Provider zwei Gruppen von Erweiterungen zur Java -Messaging-API hinzu.

„IBM MQ classes for Java verwenden” auf Seite 364
Verwenden Sie 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.

Zugehörige Verweise

Anwendungsserver für die Verwendung der neuesten Wartungsstufe des Ressourcenadapters konfigurieren

Problembestimmung für den IBM MQ-Ressourcenadapter

WebSphere Application Server-Themen

IBM MQ-Ressourcenadapter warten

JMS-Anwendungen zur Verwendung des IBM MQ-Messaging-Providers in Liberty implementieren

Unterstützungserklärung für IBM MQ-Ressourcenadapter

Der IBM MQ -Ressourcenadapter, den Sie für die Kommunikation zwischen einer Anwendung und einem Warteschlangenmanager verwenden müssen, hängt davon ab, ob Sie die Jakarta Messaging 3.0 -API oder die JMS 2.0 -API verwenden.

JMS 2.0 IBM MQ 8.0 oder höher wird mit einem Ressourcenadapter geliefert, der die Spezifikation JMS 2.0 implementiert. Er kann nur in einem Anwendungsserver implementiert werden, der mit Java Platform, Enterprise Edition 7 (Java EE 7) konform ist und unterstützt daher JMS 2.0. Eine Liste zertifizierter Anwendungsserver für Java Platform, Enterprise Edition wird auf der Oracle-Websiteverwaltet.

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. Zusätzlich zu dem Ressourcenadapter, der Java EE und JMS 2.0unterstützt, stellt IBM MQ 9.3.0 und höher einen Ressourcenadapter bereit, der Jakarta Messagingunterstützt. Die Verwendung der Jakarta Messaging 3.0 -API und der JMS 2.0 -API in derselben Anwendung wird nicht unterstützt. Weitere Informationen finden Sie im Abschnitt IBM MQ-Klassen für JMS verwenden.

Implementierung in WebSphere Liberty

WebSphere Liberty 8.5.5 Fix Pack 6 und höher sowie WebSphere Application Server Liberty 9.0 und höher sind Java EE 7-zertifizierte Anwendungsserver, sodass der IBM MQ 9.0-Ressourcenadapter darin implementiert werden kann.

Wenn Sie den IBM MQ -Ressourcenadapter für Jakarta Messaging mit Libertyverwenden möchten, müssen Sie eine Version von Liberty verwenden, die Jakarta EEunterstützt.

WebSphere Liberty verfügt über die folgenden Features für die Arbeit mit Ressourcenadaptern:

- JM 3.0 Das Feature messaging-3.0 ermöglicht das Arbeiten mit Jakarta Messaging 3.0 -Ressourcenadaptern.
- Die Funktion wmqJmsClient-2.0 ermöglicht die Arbeit mit JMS 2.0-Ressourcenadaptern.
- Die Funktion wmqJmsClient-1.1 ermöglicht die Arbeit mit JMS 1.1-Ressourcenadaptern.

Wichtig:

- JM 3.0 Der IBM MQ -Ressourcenadapter für Jakarta Messaging muss in einer Version von Liberty implementiert sein, die Jakarta EEunterstützt. Dieser Ressourcenadapter kann nicht mit Versionen von Liberty verwendet werden, die die ältere Java EE Spezifikation nicht Jakarta EEunterstützen.
- JMS 2.0 Der Ressourcenadapter IBM MQ 8.0 oder höher, der JMS 2.0 unterstützt, muss mit der Funktion wmqJmsClient-2.0 implementiert werden.

Implementierung in WebSphere Application Server traditional

WebSphere Application Server traditional 9.0 wird mit einem bereits installierten IBM MQ 9.0-Ressourcenadapter geliefert. Deshalb muss kein neuer Ressourcenadapter installiert werden. Der installierte Ressourcenadapter kann im Transportmodus CLIENT oder BINDINGS eine Verbindung zu allen Warteschlangenmanagern herstellen, die auf einer unterstützten Version von IBM MQausgeführt werden. Weitere Informationen finden Sie unter „Konnektivität zu Warteschlangenmanagern von IBM MQ 8.0 oder höher” auf Seite 460.

Wichtig:

- Der Ressourcenadapter von IBM MQ 9.0 kann nicht in WebSphere Application Server traditional-Versionen vor IBM MQ 9.0 bereitgestellt werden, weil diese Versionen nicht Java EE 7-zertifiziert sind.
- JM 3.0 WebSphere Application Server traditional unterstützt derzeit nicht Jakarta EE.

Weitere Informationen zu den Versionen des Ressourcenadapters, die mit WebSphere Application Server vergeliefert werden, finden Sie im technischen Hinweis Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server?

Ressourcenadapter mit anderen Anwendungsservern verwenden

Für alle anderen Java EE 7 oder Jakarta EE -konformen Anwendungsserver können Probleme, die nach dem erfolgreichen Abschluss des IBM MQ Ressourcenadapters Installationsprüftest (IVT) auftreten, an IBM gemeldet werden, um den IBM MQ -Produkttrace und andere IBM MQ Diagnoseinformationen zu untersuchen. Wenn der IVT des IBM MQ -Ressourcenadapters nicht erfolgreich ausgeführt werden kann, werden alle aufgetretenen Probleme wahrscheinlich durch eine falsche Implementierung oder falsche Ressourcendefinitionen verursacht, die anwendungsserverspezifisch sind, und die Probleme müssen mithilfe der Anwendungsserverdokumentation und der Unterstützungsorganisation für diesen Anwendungsserver untersucht werden.

Java-Runtime

Bei der Java Runtime (JRE), die zur Ausführung des Anwendungsservers verwendet wird, muss es sich um eine JRE handeln, die mit dem Client von IBM MQ 9.0 oder höher unterstützt wird. Weitere Informationen finden Sie unter Systemvoraussetzungen für IBM MQ. (Wählen Sie den Versions- und Betriebssystem- oder Komponentenbericht aus, den Sie einsehen möchten, und folgen Sie dann dem **Java**-Link, auf der Registerkarte **Unterstützte Software**.)

Konnektivität zu Warteschlangenmanagern von IBM MQ 8.0 oder höher

Der volle Umfang der JMS 2.0-Funktionalität ist verfügbar, wenn Verbindungen zu einem Warteschlangenmanager für IBM MQ 8.0 oder höher über den Ressourcenadapter hergestellt werden, der in einem Java EE 7-zertifizierten Anwendungsserver implementiert wurde. Um die JMS 2.0-Funktionalität nutzen zu

können, muss der Ressourcenadapter die Verbindung zum Warteschlangenmanager im normalen Modus des IBM MQ-Messaging-Providers herstellen. Weitere Informationen finden Sie im Abschnitt JMS-Eigenschaft **PROVIDERVERSION** konfigurieren.

JM 3.0 Der volle Umfang der Jakarta Messaging 3.0-Funktionalität ist verfügbar, wenn Verbindungen zu einem Warteschlangenmanager für IBM MQ 9.3 oder höher über den Ressourcenadapter hergestellt werden, der in einem Jakarta EE-zertifizierten Anwendungsserver implementiert wurde.

MQ-Erweiterungen

Mit der JMS 2.0-Spezifikation werden Änderungen bestimmter Verhaltensweisen eingeführt. Da IBM MQ 8.0 oder höher diese Spezifikation implementiert, gibt es Änderungen im Verhalten zwischen IBM MQ 8.0 und höher und älteren Versionen des Produkts. In IBM MQ 8.0 oder höher umfassen die IBM MQ classes for JMS Unterstützung für die Java-Systemeigenschaft `com.ibm.mq.jms.SupportMQExtensions`. Wenn diese auf TRUE gesetzt ist, führt dies dazu, dass diese Versionen von IBM MQ diese Verhaltensweisen auf die von IBM WebSphere MQ 7.5 oder früher zurücksetzen. Der Standardwert für die Eigenschaft ist FALSE.

Der Ressourcenadapter von IBM MQ 9.0 oder höher enthält auch eine Ressourcenadaptereigenschaft mit dem Namen `supportMQExtensions` , die dieselbe Wirkung und denselben Standardwert hat wie die `com.ibm.mq.jms.SupportMQExtensions` Java -Systemeigenschaft. Diese Ressourcenadaptereigenschaft ist in der Datei `ra.xml` standardmäßig auf 'false' gesetzt.

Wenn sowohl die Ressourcenadaptereigenschaft als auch die Java-Systemeigenschaft gesetzt sind, hat die Systemeigenschaft Vorrang.

Beachten Sie, dass diese Eigenschaft in dem Ressourcenadapter, der bereits in WebSphere Application Server traditional 9.0 implementiert ist, automatisch auf TRUE gesetzt ist, um die Migration zu unterstützen.

Weitere Informationen finden Sie unter „Eigenschaft SupportMQExtensions” auf Seite 344.

Allgemeine Probleme

Session Interleaving wird nicht unterstützt

Einige Anwendungsserver stellen eine Funktion namens Session Interleaving bereit, d. h., dieselbe JMS-Sitzung kann in mehreren Transaktionen verwendet werden, obwohl sie nur einer einzigen eingetragen ist. Der IBM MQ-Ressourcenadapter unterstützt diese Funktion nicht, was zu folgenden Problemen führen kann:

```
An attempt to put a message to a MQ queue fails with reason code 2072 (MQRC_SYNCPOINT_NOT_AVAILABLE).
```

Aufrufe von xa_close() schlagen mit Ursachencode -3 (XAER_PROTO) fehl und eine FDC mit Test-ID AT040010 wird auf dem IBM MQ-Warteschlangenmanager generiert, auf den vom Anwendungsserver zugegriffen wird. In der Dokumentation Ihres Anwendungsservers finden Sie Informationen zur Deaktivierung dieser Funktion.

Java Transaction API-(JTA-)Spezifikation zur Wiederherstellung von XA-Ressourcen für die XA-Transaktionswiederherstellung

In Abschnitt 3.4.8 der JTA-Spezifikation wird kein bestimmter Mechanismus definiert, mit dem XA-Ressourcen neu erstellt werden, um eine transaktionsorientierte XA-Wiederherstellung durchzuführen. Deshalb ist es Sache jedes einzelnen Transaktionsmanagers (und damit des Anwendungsservers), wie XA-Ressourcen, die an einer XA-Transaktion beteiligt sind, wiederhergestellt werden. Es ist bei einigen Anwendungsservern möglich, dass der IBM MQ 9.0-Ressourcenadapter die anwendungsserverspezifischen Mechanismen, die für die Durchführung der transaktionsorientierten XA-Wiederherstellung verwendet werden, nicht implementiert.

Abgleich von Verbindungen in ManagedConnectionFactory

Ein Anwendungsserver kann die Methode matchManagedConnections für eine ManagedConnectionFactory-Instanz aufrufen, die vom IBM MQ-Ressourcenadapter bereitgestellt wird. Eine ManagedConnection wird nur zurückgegeben, wenn die Methode eine Verbindung findet, die mit den Argumenten

javax.security.auth.Subject und **javax.resource.spi.ConnectionRequestInfo** übereinstimmt, die vom Anwendungsserver an die Methode übergeben wurden.

Einschränkungen beim IBM MQ-Ressourcenadapter

Der IBM MQ-Ressourcenadapter wird auf allen IBM MQ-Plattformen unterstützt. Bei der Verwendung des IBM MQ-Ressourcenadapters sind einige Funktionen von IBM MQ jedoch nicht oder nur eingeschränkt verfügbar.

Bei dem IBM MQ-Ressourcenadapter bestehen folgende Einschränkungen:

- Seit IBM MQ 8.0 ist der Ressourcenadapter ein Java Platform, Enterprise Edition 7-(Java EE 7-)Ressourcenadapter, der JMS 2.0-Funktion bereitstellt. Deshalb muss der Ressourcenadapter von IBM MQ 8.0 oder höher in einem für Java EE 7 oder höher zertifizierten Anwendungsserver installiert werden. Er kann im Transportmodus 'Clients' oder 'Bindings' eine Verbindung mit jedem in Betrieb befindlichen Warteschlangenmanager herstellen.
- Bei Ausführung im WebSphere Liberty-Anwendungsservers werden die stabilisierten IBM MQ classes for Java nicht unterstützt. In anderen Anwendungsservern sollten die IBM MQ classes for Java nicht verwendet werden. Weitere Informationen zu IBM MQ classes for Java in Java EEfinden Sie im IBM technischen Hinweis WebSphere MQ Java -Schnittstellen in J2EE/JEE -Umgebungen verwenden .
- Bei Ausführung im WebSphere Liberty-Anwendungsservers unter z/OS muss die Funktion wmqJmsClient-2.0 verwendet werden. Eine generische JCA-Unterstützung ist für z/OS nicht möglich.
- Der IBM MQ-Ressourcenadapter unterstützt keine Kanalexitprogramme, die in anderen Sprachen als Java geschrieben sind.
- Während der Ausführung eines Anwendungsservers muss der Wert der Eigenschaft sslFipsRequired für alle JCA-Ressourcen 'true' oder für alle JCA-Ressourcen 'false' sein. Dies ist selbst dann erforderlich, wenn die JCA-Ressourcen nicht gleichzeitig verwendet werden. Wenn die Eigenschaft sslFipsRequired für verschiedene JCA-Ressourcen unterschiedliche Werte aufweist, gibt IBM MQ den Ursachencode MQRC_UNSUPPORTED_CIPHER_SUITE aus, selbst wenn keine TLS-Verbindung verwendet wird.
- Sie können nicht mehr als einen Schlüsselspeicher für einen Anwendungsserver angeben. Wenn Verbindungen zu mehr als einem Warteschlangenmanager hergestellt werden, müssen alle Verbindungen denselben Schlüsselspeicher verwenden. Diese Einschränkung gilt nicht für WebSphere Application Server.
- Wenn Sie eine Definitionstabelle für den Clientkanal (CCDT) mit mehreren geeigneten Definitionen eines Clientverbindungskanals verwenden, wählt der Ressourcenadapter bei einem Fehler möglicherweise eine andere Kanaldefinition und damit einen anderen Warteschlangenmanager in der CCDT aus. Dies würde bei der Transaktionswiederherstellung zu Problemen führen. Der Ressourcenadapter ergreift keine Maßnahmen, um die Verwendung einer solchen Konfiguration zu verhindern. Sie sind dafür verantwortlich, Konfigurationen zu vermeiden, die bei der Transaktionswiederherstellung zu Problemen führen können.
- Die Funktion für Verbindungswiederholungen wird für abgehende Verbindungen nicht unterstützt, wenn die Ausführung in einem Java EE -Container (EJB/Servlet) erfolgt. Die Verbindungswiederholung wird überhaupt nicht für abgehende JMS unterstützt, wenn der Adapter in einem JEE -Containerkontext verwendet wird, unabhängig von der Transaktionskonfiguration oder für die Verwendung ohne Transaktionsunterstützung.
- Die erneute Authentifizierung von JMS -Verbindungen gemäß der Definition in Abschnitt 9.1.9 der Spezifikation Java EE Connector Architecture Version 1.7 wird nicht unterstützt. In der Datei `ra.xml` im IBM MQ-Ressourcenadapter muss die Eigenschaft **`reauthentication-support`** auf den Wert `false` gesetzt sein. Ein Versuch des Anwendungsservers, eine JMS-Verbindung erneut zu authentifizieren, führt dazu, dass der IBM MQ-Ressourcenadapter eine javax.resource.spi.SecurityException mit dem Nachrichtencode MQJCA1028 auslöst.

Zugehörige Tasks

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

Zugehörige Verweise

Federal Information Processing Standards (FIPS) für AIX, Linux, and Windows

WebSphere Application Server und der IBM MQ-Ressourcenadapter

Der IBM MQ-Ressourcenadapter wird von Anwendungen verwendet, die ein JMS-Messaging mit dem IBM MQ-Messaging-Provider in WebSphere Application Server ausführen.

Wichtig: Verwenden Sie den Ressourcenadapter IBM MQ nicht mit WebSphere Application Server 6.0 oder WebSphere Application Server 6.1.

WebSphere Application Server traditional 9.0 enthält eine Version des IBM MQ 9.0-Ressourcenadapters. Der Ressourcenadapter von IBM MQ 9.0 oder höher kann nicht in früheren Versionen von WebSphere Application Server bereitgestellt werden, weil diese Versionen nicht Java EE 7-zertifiziert sind.

JM 3.0 WebSphere Application Server traditional unterstützt derzeit nicht Jakarta EE. Weitere Informationen finden Sie unter IBM MQ resource adapter statement of support.

Falls Sie eine JMS-Anwendung für den Zugriff auf die Ressourcen eines IBM MQ-Warteschlangenmanagers über WebSphere Application Server verwenden möchten, verwenden Sie den IBM MQ-Messaging-Provider in WebSphere Application Server. Der IBM MQ-Messaging-Provider enthält eine Version der IBM MQ classes for JMS. Weitere Informationen finden Sie in dem technischen Hinweis Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server ?.

Wichtig: Schließen Sie keine der JAR-Dateien der IBM MQ classes for JMS oder IBM MQ classes for Java in Ihre Anwendung ein. Dies kann zu ClassCastExceptions führen, die möglicherweise nur schwer zu handhaben sind.

Liberty und der IBM MQ-Ressourcenadapter

Der IBM MQ -Ressourcenadapter kann mithilfe des Features Liberty in WebSphere Liberty installiert werden. Welches Feature Sie verwenden, hängt davon ab, welche Version des Ressourcenadapters Sie installieren. Alternativ können Sie den Ressourcenadapter unter bestimmten Einschränkungen mithilfe der generischen Java Platform, Enterprise Edition Connector Architecture-(Java EE JCA-)Unterstützung installieren.

Allgemeine Einschränkungen bei der Installation des Ressourcenadapters in Liberty

Die folgenden Einschränkungen gelten für den Ressourcenadapter, wenn die Funktion wmqJmsClient-1.1 oder wmqJmsClient-2.0 und auch wenn die generische JCA-Unterstützung verwendet wird:

- Die IBM MQ classes for Java werden in Liberty nicht unterstützt. Sie dürfen weder mit der Messaging-Funktion von IBM MQ Liberty noch mit der generischen JCA-Unterstützung verwendet werden. Weitere Informationen finden Sie unter Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.
- Der IBM MQ-Ressourcenadapter hat den Transporttyp BINDINGS_THEN_CLIENT. Dieser Transporttyp wird in der Messaging-Funktion von IBM MQ Liberty nicht unterstützt.
- Vor IBM MQ 9.0 war das Feature Advanced Message Security (AMS) nicht in der Messaging-Funktion von IBM MQ Liberty enthalten. AMS wird jedoch mit einem Ressourcenadapter von IBM MQ 9.0 oder höher unterstützt.

Anmerkung: In IBM MQ -Versionen ab IBM MQ 9.0.0.6 und IBM MQ 9.1.0.1 sollten Sie das Feature transportSecurity-1.0 anstelle des Features ssl-1.0 verwenden.

Weitere Informationen finden Sie unter:

SSL-Kommunikation in Libertyaktivieren
SSL-Standardwerte in Liberty
Transport Security 1.0

Einschränkungen bei Verwendung der Liberty-Funktionen

Mit WebSphere Liberty 8.5.5 Fix Pack 2 bis einschließlich WebSphere Liberty 8.5.5 Fix Pack 5 war nur die Funktion wmqJmsClient-1.1 verfügbar und konnte nur JMS 1.1 verwendet werden. Mit WebSphere Liberty 8.5.5 Fix Pack 6 wurde die Funktion wmqJmsClient-2.0 hinzugefügt, damit JMS 2.0 verwendet werden konnte.

JM 3.0 Ab IBM MQ 9.3.0wird Jakarta Messaging 3.0 unterstützt. Wenn Sie den IBM MQ -Ressourcenadapter für Jakarta Messaging mit Libertyverwenden möchten, müssen Sie eine Version von Liberty verwenden, die Jakarta EEunterstützt. Sie müssen den Ressourcenadapter für Jakarta Messaging mit dem Feature Liberty generic messaging-3.0 verwenden.

Welche Funktion Sie verwenden müssen, hängt davon ab, welche Version des Ressourcenadapters Sie verwenden:

- Der Ressourcenadapter von IBM MQ 8.0.0 Fix Pack 3 und ab IBM MQ 8.0 kann nur mit Funktion wmqJmsClient-2.0 verwendet werden.
- Der Ressourcenadapter von IBM MQ 9.0 kann nur mit Funktion wmqJmsClient-2.0 verwendet werden.
- JM 3.0 Das Feature messaging-3.0 ermöglicht die Arbeit mit Jakarta Messaging 3.0 -Ressourcenadaptern.

Einschränkungen bei Verwendung der generischen JCA-Unterstützung

Wenn Sie die generische JCA-Unterstützung verwenden, gelten folgende Einschränkungen:

- Bei Verwendung der generischen JCA -Unterstützung müssen Sie die Version von JMS angeben. JMS 2.0 und JCA 1.7 dürfen nur mit dem Ressourcenadapter von IBM MQ 8.0.0 Fix Pack 3 und ab IBM MQ 8.0 verwendet werden.
- Es ist nicht möglich, den IBM MQ-Ressourcenadapter unter z/OS unter Verwendung der generischen JCA-Unterstützung auszuführen. Zum Ausführen des IBM MQ -Ressourcenadapters unter z/OSmuss er mit der Funktion wmqJmsClient-1.1 oder wmqJmsClient-2.0 ausgeführt werden.
- Die Position des Ressourcenadapters wird mit dem folgenden XML-Element angegeben:

JM 3.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

JMS 2.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Wichtig: Als Wert für den ID-Tag kann alles AUSSER `wmqJms` angegeben werden. Falls `wmqJms` als ID verwendet wird, kann Liberty den Ressourcenadapter nicht ordnungsgemäß laden. Dies liegt daran, dass `wmqJms` die ID ist, die intern verwendet wird, um auf die spezifische Funktion für IBM MQ zu verweisen. Tatsächlich wird eine NullPointerException erstellt.

Die folgenden Beispiele zeigen einige Snippets aus einer `server.xml` -Datei bei der Ausführung von JMS 2.0:

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

Tipp: Beachten Sie die Verwendung der Funktionen jca-1.7 und jms-2.0 sowie das Fehlen der Funktion wmqJmsClient-2.0.

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Tipp: Beachten Sie die Verwendung von `mqJms` für die ID, was bevorzugt wird. Verwenden Sie nicht `wmqJms`.

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

Tipp: Beachten Sie die Referenz classloaderProviderzurück zum Ressourcenadapter über die ID `mqJms`, damit IBM MQ-spezifische Klassen geladen werden können.

Einschränkungen bei der Traceerstellung mit generischer JCA -Unterstützung

Traceerstellung und Protokollierung sind nicht in das Liberty-Tracesystem integriert. Stattdessen muss der Trace des IBM MQ -Ressourcenadapters mithilfe von Java -Systemeigenschaften oder einer IBM MQ classes for JMS -Konfigurationsdatei aktiviert werden, wie im Abschnitt Traceerstellung für IBM MQ classes for JMS -Anwendungenbeschrieben. Details zum Festlegen von Java -Systemeigenschaften in Libertyfinden Sie in der WebSphere Liberty -Dokumentation.

Um beispielsweise den Trace für den IBM MQ -Ressourcenadapter in Liberty 19.0.0.9zu aktivieren, fügen Sie einen Eintrag zur Datei Liberty `jvm.options`hinzu:

1. Erstellen Sie eine Textdatei mit dem Namen `jvm.options`.
2. Fügen Sie die folgenden JVM-Optionen ein, um die Traceerstellung (eine pro Zeile) in diese Datei zu aktivieren:

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. Wenn Sie diese Einstellungen auf einen einzelnen Server anwenden möchten, speichern Sie `jvm.options` unter:

   ```
   ${server.config.dir}/jvm.options
   ```

   Wenn Sie diese Änderungen auf alle Libertyanwenden möchten, speichern Sie `jvm.options` unter:

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   Dies wird für alle JVMs wirksam, die keine lokal definierte `jvm.options` -Datei haben.
4. Starten Sie den Server erneut, um Änderungen zu aktivieren.

Dies führt dazu, dass der Trace in eine Tracedatei namens `MQRA-WLP_<process identifier>.trc` im Verzeichnis `<path_to_trace_to>` geschrieben wird.

Volle Liberty XA-Unterstützung mit Clientkanaldefinitionstabellen

Bei Verwendung von WebSphere Liberty 18.0.0.2 oder höher können Sie Warteschlangenmanagergruppen in der Definitionstabelle für Clientkanäle (CCDT) in Verbindung mit XA-Transaktionen verwenden. Dies bedeutet, dass es jetzt möglich ist, die durch Warteschlangenmanagergruppen bereitgestellte Lastverteilung und Verfügbarkeit zu nutzen und gleichzeitig die Transaktionsintegrität aufrechtzuerhalten.

Falls bei der Verbindung zu einem Warteschlangenmanager Fehler auftreten, muss der Warteschlangenmanager wieder verfügbar werden, damit die Transaktion aufgelöst werden kann. Die Transaktionswiederherstellung wird von Liberty gesteuert. Gegebenenfalls müssen Sie den Transaktionsmanager so konfigurieren, dass genügend Zeit vorhanden ist, damit die Warteschlangenmanager wieder verfügbar

werden. Weitere Informationen finden Sie unter Transaction manager (transaction) in der WebSphere Liberty-Produktdokumentation.

Dies ist eine clientseitige Funktion, d. h., Sie benötigen einen Ressourcenadapter und keinen Warteschlangenmanager.

IBM MQ-Ressourcenadapter installieren

Der IBM MQ-Ressourcenadapter wird als RAR-Datei (Ressourcenarchiv) bereitgestellt. Installieren Sie die RAR-Datei auf Ihrem Anwendungsserver. Möglicherweise müssen Sie dem Systempfad Verzeichnisse hinzufügen.

Informationen zu diesem Vorgang

Der IBM MQ -Ressourcenadapter wird als RAR-Datei (Ressourcenarchiv) bereitgestellt:

- JM 3.0 Für Jakarta Messaging 3.0heißt diese Datei `wmq.jakarta.jmsra.rar`. Die RAR-Datei enthält die IBM MQ classes for Jakarta Messaging und die IBM MQ-Implementierung der Schnittstellen von Jakarta Connectors Architecture (JCA).
- JMS 2.0 Für JMS 2.0heißt diese Datei `wmq.jmsra.rar`. Die RAR-Datei enthält die IBM MQ classes for JMS und die IBM MQ-Implementierung der Schnittstellen von Java EE Connector Architecture (JCA).

Wenn Sie den Ressourcenadapter im Rahmen der IBM MQ -Produktinstallation installieren, wird die RAR-Datei mit IBM MQ classes for JMS in dem in Tabelle 61 auf Seite 466gezeigten Verzeichnis installiert.

Tabelle 61. IBM MQ -Verzeichnis, das die RAR-Datei für jede Plattform enthält

Plattform	**Directory**
AIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Sie müssen den IBM MQ-Ressourcenadapter verwenden, um von einem Anwendungsserver eine Verbindung mit IBM MQ herzustellen. Abhängig vom verwendeten Anwendungsserver ist der Ressourcenadapter möglicherweise vorinstalliert; wenn nicht, müssen Sie ihn selbst installieren.

Tabelle 62. Installation des Ressourcenadapters in einem Anwendungsserver

Anwendungsserver	**Vorinstalliert oder Installation erforderlich**
WebSphere Application Server traditional 9.0	Der Ressourcenadapter von IBM MQ 9.0 ist in WebSphere Application Server traditional 9.0 vorinstalliert. Deshalb muss kein neuer Ressourcenadapter in WebSphere Application Server traditional 9.0 installiert werden.
WebSphere Liberty	Da WebSphere Liberty den IBM MQ-Ressourcenadapter nicht enthält, müssen Sie ihn separat aus Fix Central beziehen.
Anderer Java EE -oder Jakarta EE -Anwendungsserver	Beziehen Sie den Ressourcenadapter separat von Fix Central, wie für WebSphere Liberty.

Prozedur

- Wenn Sie eine Verbindung zu IBM MQ von WebSphere Libertyoder einem anderen Java EE -oder Jakarta EE -Anwendungsserver herstellen, laden Sie den IBM MQ -Ressourcenadapter herunter und installieren Sie ihn wie in „Ressourcenadapter in Liberty installieren” auf Seite 467beschrieben.
- Linux AIX
  Stellen Sie bei Bindings-Verbindungen auf Systemen mit AIX and Linux sicher, dass sich das Verzeichnis mit den Bibliotheken von Java Native Interface (JNI) im Systempfad befindet.

  Sie finden Angaben zur Position dieses Verzeichnisses, das auch die Bibliotheken der IBM MQ classes for JMS enthält, in der „Java Native Interface-(JNI-)Bibliotheken konfigurieren” auf Seite 101.

  Windows Unter Windows wird dieses Verzeichnis während der Installation von IBM MQ classes for JMS automatisch zum Systempfad hinzugefügt.

  Tipp: Als Alternative zum Festlegen des Systempfads verfügt der IBM MQ-Ressourcenadapter über eine Eigenschaft namens nativeLibraryPath, die zur Angabe der Position der JNI-Bibliothek verwendet werden kann. Das folgende Beispiel zeigt, wie dies in WebSphere Liberty konfiguriert würde:

  ```
  <wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
  ```

  Transaktionen werden sowohl im Clientmodus als auch im Bindungsmodus unterstützt.

Ressourcenadapter in Liberty installieren

Zum Herstellen einer Verbindung zu IBM MQ über WebSphere Libertyoder andere Java EE -oder Jakarta EE -Anwendungsserver müssen Sie den IBM MQ -Ressourcenadapter verwenden. Da Liberty den IBM MQ-Ressourcenadapter nicht enthält, müssen Sie ihn separat aus Fix Central beziehen.

Vorbereitende Schritte

Anmerkung: Die Informationen in diesem Abschnitt gelten nicht für WebSphere Application Server traditional 9.0. Der Ressourcenadapter von IBM MQ 9.0 ist in WebSphere Application Server traditional 9.0 vorinstalliert. Deshalb muss in diesem Fall kein neuer Ressourcenadapter installiert werden.

Stellen Sie vor Beginn dieser Task sicher, dass eine Java runtime environment (JRE) auf Ihrem System installiert ist und die JRE dem Systempfad hinzugefügt wurde.

Das Java-Installationsprogramm, das in diesem Installationsprozess verwendet wird, muss nicht von einem Root oder einem bestimmten Benutzer ausgeführt werden. Die einzige Voraussetzung ist, dass der Benutzer, der es ausführt, Schreibzugriff auf das Verzeichnis hat, in dem die Dateien abgelegt werden sollen.

Wenn für Liberty-Versionen bis einschließlich WebSphere Liberty 8.5.5 Fix Pack 1 eine EJB implementiert wird, die ausschließlich die Konfiguration in `ejb-jar.xml` verwendet, muss für die Version von WebSphere Application Server, die von Liberty verwendet wird, APAR PM89890 angewendet worden sein. Diese Konfigurationsmethode wird für das Installationsprüfprogramm (IVT) des Ressourcenadapters verwendet, d. h., dieser APAR ist erforderlich, damit der Installationsprüftest durchgeführt werden kann.

JM 3.0 Ab IBM MQ 9.3.0wird Jakarta Messaging 3.0 unterstützt. Wenn Sie den IBM MQ -Ressourcenadapter für Jakarta Messaging mit Libertyverwenden möchten, müssen Sie eine Version von Liberty verwenden, die Jakarta EEunterstützt. Sie können beispielsweise das Feature Liberty generic messaging-3.0 verwenden.

Informationen zu diesem Vorgang

Die JAR-Datei für den Ressourcenadapter, die Sie von Fix Central herunterladen können, ist ausführbar. Bei Ausführung der Datei wird die IBM MQ-Lizenzvereinbarung angezeigt, die akzeptiert werden muss. Es wird nach einem Verzeichnis gefragt, in dem der IBM MQ-Ressourcenadapter installiert werden soll. In diesem Verzeichnis werden dann die RAR-Datei und das Installationsprüfprogramm (IVT) des Ressourcenadapters installiert. Sie können entweder den Standardwert akzeptieren oder ein anderes Verzeichnis

angeben, bei dem es sich um das Ressourcenadapterverzeichnis eines Anwendungsservers oder jedes andere Verzeichnis auf Ihrem System handeln kann. Das Verzeichnis wird im Rahmen der Installation erstellt, falls es noch nicht vorhanden ist.

Vor IBM MQ 9.0 hatte der Name der herunterzuladenden Datei das Format *V.R.M.F*-WS-MQ-Java-InstallRA.jar (z. B. 8.0.0.6-WS-MQ-Java-InstallRA.jar). Ab IBM MQ 9.0 lautet das Format des Dateinamens *V.R.M.F*-IBM-MQ-Java-InstallRA.jar (z. B. 9.0.0.0-IBM-MQ-Java-Install-RA.jar).

Nachdem Sie den Ressourcenadapter heruntergeladen und installiert haben, können Sie ihn in WebSphere Liberty konfigurieren.

Vorgehensweise

1. Laden Sie den IBM MQ-Ressourcenadapter von Fix Central herunter.
   a) Klicken Sie auf diesen Link: IBM MQ-Ressourcenadapter.
   b) Suchen Sie den Ressourcenadapter für Ihre Version von IBM MQ in der angezeigten Liste mit den verfügbaren Fixes.

      For example:

      ```
      release level: 9.1.4.0-IBM-MQ-Java-InstallRA
      Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application Ser
      vers
      ```

      Klicken Sie dann auf den Namen der Ressourcenadapterdatei und folgen Sie dem Downloadprozess.
2. Starten Sie die Installation, indem Sie den folgenden Befehl in dem Verzeichnis eingeben, in das Sie die Datei heruntergeladen haben.

   Ab IBM MQ 9.0 hat der Befehl das folgende Format:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   Dabei stehen *V.R.M.F* für die Nummer von Version, Release, Modifikationsstufe sowie Fixpack und *V.R.M.F*-IBM-MQ-Java-InstallRA.jar für den Namen der Datei, die bei Fix Central heruntergeladen wurde.

   Um den IBM MQ-Ressourcenadapter für Release IBM MQ 9.1.4 zu installieren, müssten Sie beispielsweise folgenden Befehl verwenden:

   ```
   java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
   ```

   Anmerkung: Um diese Installation ausführen zu können, muss eine JRE auf Ihrer Maschine installiert und im Systempfad angegeben sein.

   Wenn Sie den Befehl eingeben, werden folgende Informationen angezeigt:

   ```
   Bevor Sie IBM MQ 9.1verwenden, extrahieren oder installieren können, müssen Sie akzeptieren.
   die Bedingungen von 1. IBM Internationale Nutzungsbedingungen für die Bewertung von
   Programme 2. IBM Internationale Nutzungsbedingungen für Programmpakete und weitere
   license information. Please read the following license agreements carefully.

   The license agreement is separately viewable using the
   --viewLicenseAgreement option.
   Press Enter to display the license terms now, or 'x' to skip.
   ```
3. Prüfen und akzeptieren Sie die Lizenzbedingungen:
   a) Drücken Sie die Eingabetaste, um die Lizenz anzuzeigen.

      Alternativ können Sie x drücken, um die Lizenzanzeige zu überspringen.

      Nach der Anzeige der Lizenz oder unverzüglich nach der Auswahl von x wird folgende Nachricht angezeigt, die Sie darüber informiert, dass bei Bedarf zusätzliche Lizenzbedingungen angezeigt werden können:

```
Additional license information is separately viewable using the
--viewLicenseInfo option.
Press Enter to display additional license information now, or 'x' to skip.
```

b) Drücken Sie die Eingabetaste, um die zusätzlichen Lizenzbedingungen anzuzeigen.

Alternativ können Sie x drücken, um die Anzeige der zusätzlichen Lizenzbedingungen zu überspringen.

Nach der Anzeige der zusätzlichen Lizenzbedingungen oder unverzüglich nach der Auswahl von x wird folgende Nachricht angezeigt, in der Sie aufgefordert werden, die Lizenzvereinbarung zu akzeptieren:

```
By choosing the "I Agree" option below, you agree to the terms of the
license agreement and non-IBM terms, if applicable. Wenn Sie diesen
agree, select "I do not Agree".

Select [1] I Agree, or [2] I do not Agree:
```

c) Wählen Sie 1 aus, um die Lizenzvereinbarung zu akzeptieren und mit der Auswahl des Installationsverzeichnisses fortzufahren.

Alternativ können Sie 2 auswählen, um die Installation sofort zu beenden.

Wenn Sie 1 ausgewählt haben, wird folgende Nachricht angezeigt, in der Sie aufgefordert werden, ein Zielinstallationsverzeichnis auszuwählen:

```
Enter directory for product files or leave blank to accept the default value.
The default target directory is H:\Liberty\WMQ
Target directory for product files?
```

4. Geben Sie das Installationsverzeichnis für den Ressourcenadapter an:

- Wenn der Ressourcenadapter an der Standardposition installiert werden soll, drücken Sie die Eingabetaste, ohne einen Wert anzugeben.
- Wenn Sie den Ressourcenadapter an einer anderen Position als der Standardposition installieren möchten, geben Sie den Namen des Verzeichnisses an, in dem der Ressourcenadapter installiert werden soll, und drücken Sie dann die Eingabetaste.

Nachdem die Dateien an der ausgewählten Position installiert wurden, wird eine Bestätigungsnachricht wie im folgenden Beispiel angezeigt:

```
Extracting files to H:\Liberty\WMQ\wmq
Successfully extracted all product files.
```

Während der Installation wird im ausgewählten Installationsverzeichnis ein neues Verzeichnis namens `wmq` erstellt. Anschließend werden folgende Dateien im Verzeichnis `wmq` installiert:

- Das Installationsprüftestprogramm `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) oder `wmq.jmsra.ivt` (JMS 2.0).
- Die RAR-Datei IBM MQ ,`wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 oder `wmq.jmsra.rar` (JMS 2.0).

5. JMS 2.0

Optional: Konfigurieren Sie den Java EE 7 -Ressourcenadapter (JMS 2.0) in WebSphere Liberty Profile.

Sie müssen die folgenden Schritte ausführen, um den Ressourcenadapter in Liberty zu konfigurieren. Weitere Informationen finden Sie in der WebSphere Application Server-Produktdokumentation.

a) Fügen Sie die Funktion wmqJmsClient-2.0 zur Datei `server.xml` hinzu, um mit dem IBM MQ-Ressourcenadapter arbeiten zu können.

Weitere Informationen finden Sie unter „Zu verwendende Version des Ressourcenadapters” auf Seite 458.

b) Fügen Sie einen Verweis auf die Datei `wmq.jmsra.rar` (JMS 2.0) hinzu, die Sie installiert haben.

Eine Beispielkonfiguration für die Unterstützung von Servlets und MDBs und mit JNDI könnte wie folgt aussehen:

```
<featureManager>
   <feature>wmqJmsClient-2.0</feature>
   <feature>servlet-3.0</feature>
   <feature>jmsMdb-3.1</feature>
```

```
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

6. JM 3.0

   Optional: Konfigurieren Sie den Jakarta EE 9 -Ressourcenadapter (Jakarta Messaging 3.0) in WebSphere Liberty Profile.

   Sie müssen die folgenden Schritte ausführen, um den Ressourcenadapter in Liberty zu konfigurieren. Weitere Informationen finden Sie in der WebSphere Application Server-Produktdokumentation.

   a) Fügen Sie die Funktion wmqJmsClient-3.0 zur Datei `server.xml` hinzu, um die Arbeit mit dem IBM MQ -Ressourcenadapter zuzulassen.

      Weitere Informationen finden Sie unter „Zu verwendende Version des Ressourcenadapters” auf Seite 458.

   b) Fügen Sie einen Verweis auf die Datei `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) hinzu, die Sie installiert haben.

   Eine Beispielkonfiguration für die Unterstützung von Servlets und MDBs und mit JNDI könnte wie folgt aussehen:

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

   Anmerkung: Wenn Sie Open Libertyanstelle von WebSphere Liberty Profileverwenden, müssen Sie die generische Unterstützungsfunktion des Ressourcenadapters "messagingClient-3.0" anstelle von "wmqJmsClient-3.0" verwenden. Andere Aspekte der Konfiguration unterscheiden sich. Weitere Details finden Sie in der Dokumentation zu Open Liberty .

IBM MQ-Ressourcenadapter konfigurieren

Um den IBM MQ-Ressourcenadapter zu konfigurieren, definieren Sie verschiedene Java Platform, Enterprise Edition Connector Architecture-(JCA-)Ressourcen und optional Systemeigenschaften. Sie müssen den Ressourcenadapter auch so konfigurieren, dass das IVT-Programm (Installation Verification Test = Installationsprüftest) ausgeführt wird. Dies ist wichtig, weil der IBM Service Sie möglicherweise auffordert, dieses Programm auszuführen, um anzuzeigen, dass ein Anwendungsserver eines anderen Anbieters als IBM richtig konfiguriert ist.

Vorbereitende Schritte

Diese Aufgabe setzt voraus, dass Sie bereits mit JMS und IBM MQ classes for JMS vertraut sind. Viele der Eigenschaften, mit denen der IBM MQ-Ressourcenadapter konfiguriert wird, sind äquivalent zu den Objekteigenschaften der IBM MQ classes for JMS und weisen dieselben Funktionen auf.

Informationen zu diesem Vorgang

Jeder Anwendungsserver stellt eine eigene Gruppe von Verwaltungsschnittstellen zur Verfügung. Einige Anwendungsserver stellen grafische Benutzerschnittstellen für die Definition von JCA-Ressourcen bereit, während bei anderen der Administrator XML-Implementierungspläne schreiben muss. Daher würde es den Rahmen dieser Dokumentation sprengen, für jeden Anwendungsserver Informationen zur jeweiligen Konfiguration des IBM MQ-Ressourcenadapters bereitzustellen.

In den folgenden Schritten geht es deshalb nur um das, was Sie konfigurieren müssen. In der Dokumentation Ihres Anwendungsservers finden Sie Informationen zur Konfiguration eines JCA-Ressourcenadapters.

Prozedur

Definieren Sie JCA-Ressourcen in folgenden Kategorien:

- Definieren Sie die Eigenschaften des Objekts ResourceAdapter.

  Diese Eigenschaften, die die globalen Eigenschaften des Ressourcenadapters, z. B. die Stufe des Diagnosetracings, darstellen, werden im Abschnitt „Eigenschaften des ResourceAdapter-Objekts konfigurieren” auf Seite 472 beschrieben.
- Definieren Sie die Eigenschaften eines ActivationSpec-Objekts.

  Diese Eigenschaften legen fest, wie eine MDB für die eingehende Kommunikation aktiviert wird. Weitere Informationen finden Sie unter „Ressourcenadapter für eingehende Kommunikation konfigurieren” auf Seite 475.
- Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts.

  Der Anwendungsserver verwendet diese Eigenschaften, um ein JMS-ConnectionFactory-Objekt für die abgehende Kommunikation zu erstellen. Weitere Informationen finden Sie unter „Ressourcenadapter für abgehende Kommunikation konfigurieren” auf Seite 494.
- Definieren Sie die Eigenschaften eines verwalteten Zielobjekts.

  Der Anwendungsserver verwendet diese Eigenschaften, um ein JMS-Warteschlangenobjekt oder JMS-Themenobjekt für die abgehende Kommunikation zu erstellen. Weitere Informationen finden Sie unter „Ressourcenadapter für abgehende Kommunikation konfigurieren” auf Seite 494.
- Optional: Definieren Sie einen Implementierungsplan für den Ressourcenadapter.

  Die RAR-Datei des IBM MQ-Ressourcenadapters enthält eine Datei namens `META-INF/ra.xml`, die wiederum einen Implementierungsdeskriptor für den Ressourcenadapter enthält. Dieser Implementierungsdeskriptor wird vom XML-Schema unter https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd definiert. Er enthält Informationen zum Ressourcenadapter sowie zu den von ihm bereitgestellten Services. Ein Anwendungsserver benötigt möglicherweise auch einen Implementierungsplan für den Ressourcenadapter. Dieser Implementierungsplan gilt speziell für den Anwendungsserver.

Geben Sie je nach Bedarf JVM-Systemeigenschaften an:

- Wenn Sie Transport Layer Security (TLS) verwenden, geben Sie wie im folgenden Beispiel die Positionen der Schlüsselspeicherdatei und der Truststore-Datei als JVM-Systemeigenschaften an:

  ```
  java ... -Djavax.net.ssl.keyStore=
  key_store_location
           -Djavax.net.ssl.trustStore=trust_store_location
           -Djavax.net.ssl.keyStorePassword=key_store_password
  ```

  Diese Eigenschaften können keine Eigenschaften eines ActivationSpec- oder ConnectionFactory-Objekts sein und Sie können nicht mehr als einen Schlüsselspeicher für einen Anwendungsserver angeben. Die Eigenschaften gelten für die gesamte JVM und können sich daher auf den Anwendungsserver auswirken, wenn andere Anwendungen, die auf dem Anwendungsserver ausgeführt werden, TLS-Verbindungen verwenden. Der Anwendungsserver könnte diese Eigenschaften auch auf andere Werte zurücksetzen. Weitere Informationen zur Verwendung von TLS mit den IBM MQ classes for JMS finden Sie im Abschnitt „TLS mit IBM MQ classes for JMS verwenden” auf Seite 267.
- Optional: Falls erforderlich, konfigurieren Sie den Ressourcenadapter so, dass Warnungen im Standardausgabeprotokoll Ihres Anwendungsservers protokolliert werden.

  Die Protokolle, Warnungen und Fehlernachrichten des Ressourcenadapters verwenden denselben Mechanismus wie die IBM MQ classes for JMS. Weitere Informationen finden Sie unter Protokollierungsfehler für IBM MQ classes for JMS. Dies bedeutet, dass die Nachrichten standardmäßig in eine Datei mit dem Namen `mqjms.log` geschrieben werden. Wenn Sie in der Konfiguration des Ressourcenadapters festlegen möchten, dass zusätzlich Warnungen im Standardausgabeprotokoll Ihres

Anwendungsservers protokolliert werden sollen, legen Sie folgende JVM-Systemeigenschaft für Ihren Anwendungsserver fest:

```
-Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
```

Dies ist dieselbe Eigenschaft wie die, mit der Trace für die IBM MQ classes for JMS gesteuert wird. Wie bei den IBM MQ classes for JMS ist es möglich, eine Systemeigenschaft zu verwenden, die auf die Datei `jms.config` verweist (siehe „Die Konfigurationsdatei IBM MQ classes for JMS/Jakarta Messaging" auf Seite 103). In der Dokumentation Ihres Anwendungsservers finden Sie Informationen zur Festlegung einer JVM-Systemeigenschaft.

Konfigurieren Sie den Ressourcenadapter für die Ausführung des Installationsprüftests:

- Konfigurieren Sie den Ressourcenadapter für die Ausführung des Programms für den Installationsprüftest (Installation Verification Test, IVT), das mit dem IBM MQ-Ressourcenadapter bereitgestellt wird.

  Sie finden Informationen darüber, was Sie für die Ausführung des IVT-Programms konfigurieren müssen, im Abschnitt „Installation des Ressourcenadapters überprüfen" auf Seite 517.

  Dies ist wichtig, weil der IBM Service Sie möglicherweise auffordert, dieses Programm auszuführen, um anzuzeigen, dass ein Anwendungsserver eines anderen Anbieters als IBM richtig konfiguriert ist.

  Wichtig: Sie müssen den Ressourcenadapter konfigurieren, bevor Sie das Programm ausführen können.

Eigenschaften des ResourceAdapter-Objekts konfigurieren

Das ResourceAdapter-Objekt enthält die globalen Eigenschaften des IBM MQ-Ressourcenadapters, beispielsweise die festgelegte Diagnosetracestufe. Zur Einrichtung dieser Eigenschaften verwenden Sie die Funktionen Ihres Ressourcenadapters, wie in der Dokumentation zu Ihrem Anwendungsserver beschrieben.

Das ResourceAdapter-Objekt verfügt über zwei Eigenschaftsgruppen:

- Eigenschaften, die dem Diagnosetracing zugeordnet sind
- Eigenschaften, die dem Verbindungspool zugeordnet sind, der vom Ressourcenadapter verwaltet wird

Wie diese Eigenschaften definiert werden, richtet sich nach den von Ihrem Anwendungsserver bereitgestellten Verwaltungsschnittstellen. Wenn Sie WebSphere Application Server traditionalverwenden, lesen Sie „WebSphere Application Server traditional-Konfiguration" auf Seite 474 oder wenn Sie WebSphere Libertyverwenden, lesen Sie den Abschnitt „WebSphere Liberty-Konfiguration" auf Seite 474. Bei anderen Anwendungsservern lesen Sie die Produktdokumentation zum jeweiligen Anwendungsserver.

Weitere Informationen zur Definition von Eigenschaften, die dem Diagnosetrace zugeordnet sind, finden Sie im Abschnitt Traceverarbeitung für den IBM MQ-Ressourcenadapter.

Der Ressourcenadapter verwaltet einen internen Verbindungspool mit JMS-Verbindungen, die zur Übermittlung von Nachrichten an MDBs verwendet werden. Tabelle 63 auf Seite 472 listet die Eigenschaften des ResourceAdapter-Objekts auf, die dem Verbindungspool zugeordnet sind.

Tabelle 63. Eigenschaften des ResourceAdapter-Objekts, die dem Verbindungspool zugeordnet sind

Name der Eigenschaft	Typ	Standardwert	Beschreibung
maxConnections	Zeichenfolge	50	Die maximale Anzahl der Verbindungen mit einem IBM MQ-Warteschlangenmanager und die maximale Anzahl der implementierten MDBs.
connectionConcurrency	Zeichenfolge	1	Die maximale Anzahl der MDBs für die gemeinsame Nutzung einer JMS-Verbindung. Eine gemeinsame Nutzung von Verbindungen ist nicht möglich, sodass diese Eigenschaft immer den Wert 1 hat.

Tabelle 63. Eigenschaften des ResourceAdapter-Objekts, die dem Verbindungspool zugeordnet sind (Forts.)

Name der Eigenschaft	Typ	Standardwert	Beschreibung
reconnectionRetryCount	Zeichenfolge	5	Die maximale Anzahl der Versuche, die vom Ressourcenadapter unternommen werden, um die Verbindung mit einem IBM MQ-Warteschlangenmanager wiederherzustellen, falls eine Verbindung fehlschlägt.
reconnectionRetryInterval	Zeichenfolge	300 000	Die in Millisekunden angegebene Zeit, die der Ressourcenadapter wartet, bevor er versucht, die Verbindung mit einem IBM MQ-Warteschlangenmanager wiederherzustellen.
startupRetryCount	Zeichenfolge	0	Die standardmäßige Anzahl der Versuche, beim Systemstart eine MDB-Verbindung herzustellen, wenn der Warteschlangenmanager beim Start des Anwendungsservers nicht aktiv ist.
startupRetryInterval	Zeichenfolge	30 000	Die standardmäßige Ruhezeit zwischen Verbindungsversuchen beim Startvorgang (in Millisekunden).
supportMQExtensions	Zeichenfolge	false	Setzt das Verhalten von IBM MQ JMS auf das Verhalten vor JMS 2.0 zurück. Weitere Informationen finden Sie unter „Eigenschaft SupportMQExtensions” auf Seite 344.
nativeLibraryPath	Zeichenfolge	<empty>	Der Pfad, über den die JNI-Bibliothek von IBM MQ geladen wird, um Verbindungen im Bindungsmodus zu ermöglichen. Windows Unter Windows muss der Systempfad auch das Installationsverzeichnis der zutreffenden IBM MQ-Installation enthalten.

Wenn eine MDB auf dem Anwendungsserver implementiert wird, wird eine neue JMS-Verbindung erstellt und es wird ein Dialog mit dem Warteschlangenmanager gestartet. Dies setzt voraus, dass die maximale Anzahl der durch die Eigenschaft 'maxConnection' angegebenen Verbindungen nicht überschritten wird. Daher entspricht die maximale Anzahl MDBs der maximalen Anzahl der Verbindungen. Wenn die Anzahl der implementierten MDBs dieses Maximum erreicht, schlagen alle Versuche, eine weitere MDB zu implementieren, fehl. Wenn eine MDB gestoppt wird, kann ihre Verbindung von einer anderen MDB verwendet werden.

Im Allgemeinen müssen Sie den Wert der Eigenschaft 'maxConnections' erhöhen, wenn viele MDBs implementiert werden sollen.

Die Eigenschaften 'reconnectionRetryCount' und 'reconnectionRetryInterval' steuern das Verhalten des Ressourcenadapters, wenn Verbindungen mit einem IBM MQ-Warteschlangenmanager fehlschlagen, da beispielsweise ein Netzausfall vorliegt. Wenn eine Verbindung unterbrochen wird, setzt der Adapter die Übermittlung von Nachrichten an alle MDBs, die über die Verbindung versorgt werden, für einen Intervall aus, der durch die Eigenschaft reconnectionRetryInterval festgelegt wird. Der Ressourcenadapter versucht anschließend, die Verbindung mit dem Warteschlangenmanager wiederherzustellen. Wenn der Versuch fehlschlägt, unternimmt der Ressourcenadapter in einem Intervall, das durch die Eigenschaft 'reconnectionRetryInterval' festgelegt ist, weitere Versuche zur Wiederherstellung der Verbindung, bis der Grenzwert, der durch die Eigenschaft 'reconnectionRetryCount' festgelegt ist, erreicht wird. Schlagen alle Versuche fehl, wird die Übermittlung dauerhaft gestoppt, bis die MDBs manuell erneut gestartet werden.

Im Allgemeinen ist für das ResourceAdapter-Objekt keine Verwaltung erforderlich. Wenn Sie jedoch das Diagnosetracing auf Systemen mit AIX and Linux aktivieren möchten, können Sie die folgenden Eigenschaften festlegen:

```
traceEnabled:    true
traceLevel:      10
```

Diese Eigenschaften haben keine Auswirkungen, wenn der Ressourcenadapter nicht gestartet wurde. Dies ist beispielsweise der Fall, wenn Anwendungen, die IBM MQ-Ressourcen nutzen, ausschließlich im Client-Container ausgeführt werden. In dieser Situation können Sie die Eigenschaften für das Diagnosetracing als Systemeigenschaften der Java Virtual Machine (JVM) festlegen. Sie können die Eigenschaften mit dem Attribut -D im Befehl **java** festlegen, wie im folgenden Beispiel gezeigt:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

Sie müssen nicht alle Eigenschaften des ResourceAdapter-Objekts definieren. Für Eigenschaften, die nicht angegeben wurden, werden deren Standardwerte verwendet. In einer verwalteten Umgebung sollten die beiden Methoden zur Angabe von Eigenschaften nicht kombiniert werden. Werden sie kombiniert, haben die JVM-Systemeigenschaften Vorrang vor den Eigenschaften des ResourceAdapter-Objekts.

WebSphere Application Server traditional-Konfiguration

Dieselben Eigenschaften sind für den Ressourcenadapter in WebSphere Application Server traditional verfügbar, sie sollten jedoch in der Eigenschaftenanzeige des Ressourcenadapters festgelegt werden (siehe Einstellungen für JMS-Provider in der Produktdokumentation zu WebSphere Application Server traditional). Die Trace-Einstellungen werden im Abschnitt 'Diagnose' der WebSphere Application Server traditional-Konfiguration festgelegt. Weitere Informationen finden Sie im Abschnitt Mit Diagnoseprovidern arbeiten in der Produktdokumentation zu WebSphere Application Server traditional.

WebSphere Liberty-Konfiguration

Der Ressourcenadapter wird, wie im folgenden Beispiel gezeigt, mithilfe von XML-Elementen in der Datei `server.xml` konfiguriert:

JM 3.0

```
<featureManager>
...
    <feature>messaging-3.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

JMS 2.0

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

Die Trace-Funktion wird durch Hinzufügen des folgenden XML-Elements aktiviert:

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

Ressourcenadapter für eingehende Kommunikation konfigurieren

Definieren Sie die Eigenschaften eines oder mehrerer ActivationSpec-Objekte, um die eingehende Kommunikation zu konfigurieren.

Die Eigenschaften eines ActivationSpec-Objekts legen fest, wie eine MDB (Message-driven Bean, nachrichtengesteuerte Bean) JMS-Nachrichten aus einer IBM MQ-Warteschlange empfängt. Das Transaktionsverhalten der MDB wird in ihrem Implementierungsdeskriptor definiert.

Ein ActivationSpec-Objekt verfügt über zwei Eigenschaftsgruppen:

- Eigenschaften, die für die Erstellung einer JMS-Verbindung mit einem IBM MQ-Warteschlangenmanager verwendet werden
- Eigenschaften, die für die Erstellung eines JMS-Verbindungskonsumenten verwendet werden, der Nachrichten asynchron übermittelt, sobald sie bei einer angegebenen Warteschlange eintreffen

Auf welche Weise Sie die Eigenschaften eines ActivationSpec-Objekts definieren, hängt von der Verwaltungsschnittstelle ab, die von Ihrem Anwendungsserver bereitgestellt wird.

Eigenschaften zur Herstellung einer JMS-Verbindung mit einem IBM MQ-Warteschlangenmanager

Alle Eigenschaften in Tabelle 64 auf Seite 475 sind optional.

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
applicationName	Zeichenfolge	• Der aufrufende Klassenname (falls verfügbar), der so angepasst wird, dass er nicht mehr als 28 Zeichen enthält. Falls er nicht verfügbar ist, wird die Zeichenfolge `WebSphere MQ Client for Java` verwendet.	Der Name, unter dem eine Anwendung beim Warteschlangenmanager registriert wird. Dieser Anwendungsname wird vom **`DISPLAY CONN MQSC/PCF`** -Befehl angezeigt (wobei das Feld **`APPLTAG`** heißt). oder in der Anzeige IBM MQ Explorer **Application Connections** (in der das Feld den Namen **App name** hat).
brokerCCDurSubQueue [1]	Zeichenfolge	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Verbindungskonsument permanente Subskriptionsnachrichten empfängt
brokerCCSubQueue [1]	Zeichenfolge	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Verbindungskonsument nicht permanente Subskriptionsnachrichten empfängt
brokerControlQueue [1]	Zeichenfolge	• **SYSTEM.BROKER.CONTROL.QUEUE** • Ein Warteschlangenname	Der Name der Steuerwarteschlange des Brokers
brokerQueueManager [1]	Zeichenfolge	• **"" (leere Zeichenfolge)** • Name eines Warteschlangenmanagers	Der Name des Warteschlangenmanagers, in dem der Broker ausgeführt wird

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
brokerSubQueue [1]	Zeichenfolge	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Konsument nicht permanenter Nachrichten die Nachrichten empfängt
brokerVersion [1]	Zeichenfolge	• **unspecified** - Nachdem der Broker von V6 auf V7 migriert wurde, legen Sie diese Eigenschaft fest, damit RFH2-Header nicht mehr verwendet werden. Nach der Migration ist diese Eigenschaft nicht mehr relevant. • **V1** - Für die Verwendung eines IBM MQ-Publish/Subscribe-Brokers. Dieser Wert ist der Standardwert, falls TRANSPORT auf BIND oder CLIENT gesetzt ist. • **V2** - Für die Verwendung eines Brokers von IBM Integration Bus im nativen Modus. Dieser Wert ist der Standardwert, falls TRANSPORT auf DIRECT oder DIRECTHTTP gesetzt ist.	Die Version des verwendeten Brokers
ccdtURL	Zeichenfolge	• **Null** • Eine URL (Uniform Resource Locator)	Eine URL, die den Namen und die Position der Datei angibt, welche die Definitionstabelle für den Clientkanal enthält und festlegt, wie auf die Datei zugegriffen werden kann
CCSID	Zeichenfolge	• **819** • ID des codierten Zeichensatzes, die von der Java Virtual Machine (JVM) unterstützt wird	Die ID des codierten Zeichensatzes für eine Verbindung
Kanal	Zeichenfolge	• **SYSTEM.DEF.SVRCONN** • Der Name eines MQI-Kanals	Der Name des zu verwendenden MQI-Kanals
cleanupInterval [1]	int	• **3 600 000** • Eine positive Ganzzahl	Das in Millisekunden angegebene Intervall zwischen den Hintergrundausführungen des Publish/Subscribe-Bereinigungsdienstprogramms

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
cleanupLevel [1]	Zeichenfolge	• **SAFE** • KEINE • STRONG • FORCE • NONDUR	Die Bereinigungsstufe für einen brokerbasierten Subskriptionsspeicher
clientID	Zeichenfolge	• **Null** • Eine Client-ID	Die Client-ID für eine Verbindung
cloneSupport	Zeichenfolge	• **DISABLED** - Nur eine Instanz eines permanenten Topic-Subskribenten kann jeweils ausgeführt werden. • ENABLED - Zwei oder mehr Instanzen desselben permanenten Topic-Subskribenten können gleichzeitig ausgeführt werden. Allerdings muss jede Instanz auf einer separaten Java Virtual Machine (JVM) ausgeführt werden.	Geben Sie an, ob zwei oder mehr Instanzen desselben permanenten Topic-Subskribenten gleichzeitig ausgeführt werden können.
connectionFactoryLookup	Zeichenfolge	• **Null** • Der JNDI-Name eines ConnectionFactory-Objekts	Wenn diese Eigenschaft gesetzt ist, sucht die Aktivierungsspezifikation im JNDI-Namensbereich des Anwendungsservers nach dem JMS-ConnectionFactory-Objekt mit dem angegebenen JNDI-Namen und verwendet anschließend die Eigenschaften dieses Objekts zur Herstellung einer JMS-Verbindung mit einem IBM MQ-Warteschlangenmanager. Hierbei gilt eine Ausnahme. Die einzige Eigenschaft der Aktivierungsspezifikation, die zum Aufbau der JMS-Verbindung verwendet wird, ist die Client-ID. Weitere Informationen finden Sie unter „Eigenschaften ' connectionFactoryLookup' und 'destinationLookup' der Aktivierungsspezifikation” auf Seite 490.

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
connectionNameList	Zeichenfolge	• **localhost(1414)** • Eine Zeichenfolge mit Elementen, die jeweils durch Kommas voneinander getrennt sind. Dabei hat jedes Element folgendes Format: `HOSTNAME(PORT)` Dabei steht *HOSTNAME* entweder für einen DNS-Namen oder für eine IP-Adresse.	Eine Liste mit TCP/IP-Verbindungsnamen, die für die eingehende Kommunikation verwendet werden. Falls angegeben, setzt **connectionNameList** die Eigenschaften **hostname** und **port** außer Kraft. Mit dieser Eigenschaft wird eine Verbindung zu Mehrinstanz-Warteschlangenmanagern wiederhergestellt. **connectionNameList** weist eine ähnliche Form wie **localAddress** auf, darf jedoch nicht damit verwechselt werden. **localAddress** gibt die Merkmale der lokalen Kommunikation an, während **connectionNameList** angibt, wie ein ferner Warteschlangenmanager zu erreichen ist.
dynamicallyBalanced [4]	Boolesch	• **false** • true	Gibt an, ob diese MDB eine Anforderung zum Empfangen von Nachrichten von einem anderen Warteschlangenmanager als Teil eines Lastausgleich für Anwendungen in einem Uniform-Cluster erhalten kann.
failIfQuiesce	Boolesch	• **wahr** • false	Gibt an, ob Aufrufe bestimmter Methoden fehlschlagen, wenn der Warteschlangenmanager gerade stillgelegt wird
headerCompression	Zeichenfolge	• **Ohne** • SYSTEM - Es wird eine RLE-Komprimierung des Nachrichtenheaders durchgeführt.	Eine Liste der Verfahren, die zum Komprimieren von Headerdaten in einer Verbindung verwendet werden können

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
hostName	Zeichenfolge	• **localhost** • Ein Hostname • Eine IP-Adresse	Der Hostname oder die IP-Adresse des Systems, auf dem sich der Warteschlangenmanager befindet. Die Eigenschaften **hostname** und **port** werden durch die Eigenschaft **connectionNameList** außer Kraft gesetzt, falls diese angegeben ist.
localAddress	Zeichenfolge	• **Null** • Eine Zeichenfolge im folgenden Format: `[ host_name ][(low_port [, high_port ])]` Dabei steht *Hostname* für einen Hostnamen oder eine IP-Adresse. *Niedriger_Port* und *hoher_Port* sind TCP-Portnummern und die Klammern bezeichnen eine optionale Komponente.	Für eine Verbindung mit einem Warteschlangenmanager gibt diese Eigenschaft entweder eine oder beide der folgenden Informationen an: • die zu verwendende lokale Netzschnittstelle • den zu verwendenden lokalen Port bzw. Portbereich **localAddress** weist eine ähnliche Form wie **connectionNameList** auf, darf jedoch nicht damit verwechselt werden. **localAddress** gibt die Merkmale der lokalen Kommunikation an, während **connectionNameList** angibt, wie ein ferner Warteschlangenmanager zu erreichen ist.
messageCompression	Zeichenfolge	• **Ohne** • Eine Liste mit einem oder mehreren der folgenden Werte, die durch Leerzeichen voneinander getrennt sind: RLE ZLIBFAST ZLIBHIGH V 9.4.0 LZ4FAST V 9.4.0 LZ4HIGH	Eine Liste der Verfahren, die zum Komprimieren von Nachrichtendaten in einer Verbindung verwendet werden können

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
messageRetention [1]	Boolesch	• **true** (wahr) - nicht benötigte Nachrichten verbleiben in der Eingabewarteschlange • false (falsch) - nicht benötigte Nachrichten werden in Übereinstimmung mit ihren jeweiligen Dispositionsoptionen bearbeitet	Gibt an, ob der Verbindungskonsument nicht erwünschte Nachrichten in der Eingabewarteschlange behält
messageSelection [1]	Zeichenfolge	• **Client** • BROKER	Legt fest, ob die Nachrichtenauswahl durch IBM MQ classes for JMS oder durch den Broker erfolgt. Die Nachrichtenauswahl durch den Broker wird nicht unterstützt, wenn für 'brokerVersion' der Wert '1' festgelegt wurde.
Kennwort	Zeichenfolge	• **Null** • Ein Kennwort	Das Standardkennwort, das beim Erstellen einer Verbindung mit dem Warteschlangenmanager verwendet wird
pollingInterval [1]	int	• **5000** • Jede beliebige positive Ganzzahl.	Wenn sich bei den einzelnen Nachrichtenlistenern innerhalb einer Sitzung keine geeignete Nachricht in der zugehörigen Warteschlange befindet, ist dieser Wert das maximale Intervall in Millisekunden, das verstreicht, bevor die einzelnen Nachrichtenlistener erneut versuchen, eine Nachricht aus der zugehörigen Warteschlange abzurufen. Wenn regelmäßig keine geeigneten Nachrichten für die Nachrichtenlistener innerhalb einer Sitzung verfügbar sind, sollten Sie einen höheren Wert für diese Eigenschaft angeben. Diese Eigenschaft ist nur relevant, wenn für TRANSPORT der Wert BIND oder CLIENT festgelegt ist.

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
port	int	• **1414** • Eine TCP-Portnummer	Der Port, an dem der Warteschlangenmanager empfangsbereit ist. Die Eigenschaften **hostname** und **port** werden durch die Eigenschaft **connectionNameList** außer Kraft gesetzt, falls diese angegeben ist.
providerVersion	Zeichenfolge	• **unspecified** • Eine Zeichenfolge in einem der folgenden Formate – V.R.M.F – V.R.M – V.R – V Dabei sind V, R, M und F Ganzzahlen größer-gleich null.	Die Version, das Release, die Modifikationsstufe und das Fixpack des Warteschlangenmanagers, zu dem die MDB eine Verbindung herstellen möchte.
queueManager	Zeichenfolge	• **"" (leere Zeichenfolge)** • Name eines Warteschlangenmanagers	Der Name des Warteschlangenmanagers, zu dem eine Verbindung hergestellt werden soll
receiveExit [3]	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen besteht, die durch Kommas voneinander getrennt sind. Dabei gibt jedes Element den vollständig qualifizierten Namen einer Klasse an, die die Schnittstelle der IBM MQ classes for Java (MQReceiveExit) implementiert.	Gibt ein Empfangsexitprogramm oder eine Folge von Empfangsexitprogrammen für den Kanal an, die nacheinander ausgeführt werden sollen.
receiveExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen von Benutzerdaten besteht, die durch Kommas voneinander getrennt sind	Die Benutzerdaten, die an die Empfangsexitprogramme des Kanals gesendet werden, wenn diese aufgerufen werden

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
rescanInterval [1]	int	• **5000** • Jede beliebige positive Ganzzahl.	Wenn ein Nachrichtenkonsument in der Punkt-zu-Punkt-Domäne einen Nachrichtenselektor verwendet, um auszuwählen, welche Nachrichten er empfangen möchte, durchsucht IBM MQ classes for JMS die Warteschlange IBM MQ in der durch das Attribut **MsgDeliverySequence** der Warteschlange festgelegten Reihenfolge nach geeigneten Nachrichten. Wenn die IBM MQ classes for JMS eine geeignete Nachricht finden und diese an den Konsumenten übermitteln, setzen die IBM MQ classes for JMS die Suche nach der nächsten geeigneten Nachricht ab der aktuellen Position in der Warteschlange fort. Die IBM MQ classes for JMS durchsuchen die Warteschlange auf diese Weise, bis das Ende der Warteschlange erreicht oder das durch den Wert dieser Eigenschaft festgelegte Intervall in Millisekunden abgelaufen ist. In jedem Fall kehren die IBM MQ classes for JMS zum Anfang der Warteschlange zurück, um die Suche fortzusetzen, und es beginnt ein neues Zeitintervall.
securityExit [3]	Zeichenfolge	• **Null** • Der vollständig qualifizierte Name einer Klasse, die die Schnittstelle der IBM MQ classes for Java (MQSecurityExit) implementiert	Gibt ein Sicherheitsexitprogramm für den Kanal an
securityExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge mit Benutzerdaten	Die Benutzerdaten, die an ein Sicherheitsexitprogramm des Kanals übergeben werden, wenn dieses aufgerufen wird

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
sendExit [3]	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen besteht, die durch Kommas voneinander getrennt sind. Dabei gibt jedes Element den vollständig qualifizierten Namen einer Klasse an, die die Schnittstelle der IBM MQ classes for Java (MQSendExit) implementiert.	Gibt ein Sendeexitprogramm oder eine Folge von Sendeexitprogrammen für den Kanal an, die nacheinander ausgeführt werden sollen
sendExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen von Benutzerdaten besteht, die durch Kommas voneinander getrennt sind	Die Benutzerdaten, die an die Sendeexitprogramme des Kanals gesendet werden, wenn diese aufgerufen werden
shareConvAllowed	Boolesch	• NO-Eine Clientverbindung kann ihren Socket nicht gemeinsam nutzen. • **YES** -Eine Clientverbindung kann ihren Socket gemeinsam nutzen.	Gibt an, ob eine Clientverbindung ihr Socket gemeinsam mit anderen JMS-Verbindungen der höchsten Ebene in demselben Prozess mit demselben Warteschlangenmanager nutzen kann, wenn die Kanaldefinitionen übereinstimmen
sparseSubscriptions [1]	Boolesch	• **false** (false) - Subskriptionen empfangen häufige übereinstimmende Nachrichten. • true (wahr) - Subskriptionen empfangen seltene übereinstimmende Nachrichten. Für diesen Wert ist es erforderlich, dass die Subskriptionswarteschlange zum Durchsuchen geöffnet werden kann.	Steuert die Richtlinie für den Nachrichtenabruf eines TopicSubscriber-Objekts.
sslCertStores	Zeichenfolge	• **Null** • Eine Zeichenfolge mit einer oder mehreren LDAP-URLs, die durch Leerzeichen getrennt sind. Jede LDAP-URL hat folgendes Format: `ldap://host_name [: port ]` Dabei steht *Hostname* für einen Hostnamen oder eine IP-Adresse. *Port* ist eine TCP-Portnummer und die Klammern bezeichnen eine optionale Komponente.	Die LDAP-Server (LDAP = Lightweight Directory Access Protocol), auf denen die Zertifikatswiderrufslisten (CRLs) für TLS-Verbindungen gespeichert sind
sslCipherSuite	Zeichenfolge	• **Null** • Der Name einer Cipher-Suite	Die Cipher-Suite, die für eine TLS-Verbindung verwendet werden soll

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
sslFipsRequired [2]	Boolesch	• **false** • true	Angabe, ob eine TLS-Verbindung eine vom IBM Java JSSE FIPS-Provider (IBMJSSEFIPS) unterstützte Cipher-Suite verwenden muss
sslPeerName	Zeichenfolge	• **Null** • Eine Vorlage für definierte Namen	Bei einer TLS-Verbindung eine Vorlage, die zur Überprüfung des definierten Namens in dem vom Warteschlangenmanager bereitgestellten digitalen Zertifikat verwendet wird
sslResetCount	int	• **0** • Eine Ganzzahl im Bereich 0 bis 999 999 999	Die Gesamtzahl der von einer TLS-Verbindung gesendeten und empfangenen Bytes vor der erneuten Vereinbarung der von TLS verwendeten geheimen Schlüssel
sslSocketFactory	Zeichenfolge	Eine Zeichenfolge, die den vollständig qualifizierten Klassennamen einer Klasse darstellt, die eine Implementierung der Schnittstelle javax.net.ssl.SSLSocketFactory implementiert. Optional kann ein Argument für die Übergabe an die Konstruktormethode eingeschlossen werden. Dieses Argument wird in Klammern gesetzt.	Alle Verbindungen, die im Bereich des verwalteten Objekts eingerichtet werden, verwenden Sockets, die aus dieser Implementierung der Schnittstelle SSLSocketFactory abgerufen werden.
statusRefreshInterval [1]	int	• **60000** • Jede beliebige positive Ganzzahl.	Das Aktualisierungsintervall der lange aktiven Transaktion (in Millisekunden), die feststellt, wenn ein Subskribent die Verbindung zum Warteschlangenmanager verliert. Diese Eigenschaft ist nur relevant, wenn für **subscriptionStore** der Wert QUEUE festgelegt ist.
subscriptionStore [1]	Zeichenfolge	• **Broker** • MIGRATE • WARTESCHLANGE	Legt fest, ob IBM MQ classes for JMS persistente Daten über aktive Subskriptionen speichern

Tabelle 64. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
transportType	Zeichenfolge	• **Client** • BINDINGS • BINDINGS_THEN_CLIENT	Gibt an, ob eine Verbindung mit einem Warteschlangenmanager den Clientmodus oder den Bindungsmodus verwendet. Bei Angabe von `BINDINGS_THEN_CLIENT` versucht der Ressourcenadapter zuerst, eine Verbindung im Bindungsmodus herzustellen. Schlägt dieser Verbindungsversuch fehl, versucht er, eine Verbindung im Clientmodus herzustellen. z/OS Wenn eine auf einem WebSphere Application Server for z/OS-System ausgeführte Aktivierungsspezifikation mit dem Transportmodus `BINDINGS_THEN_CLIENT` konfiguriert wurde und eine zuvor eingerichtete Verbindung unterbrochen wurde, versuchen alle durch die Aktivierungsspezifikation unternommenen Verbindungsversuche zunächst, die Verbindung im Transportmodus `BINDINGS` herzustellen. Erst wenn der Verbindungsversuch im Transportmodus `BINDINGS` fehlschlägt, versucht die Aktivierungsspezifikation einen Verbindungsaufbau im Transportmodus `CLIENT`.
username	Zeichenfolge	• **Null** • Ein Benutzername	Der beim Herstellen einer Verbindung zu einem Warteschlangenmanager zu verwendende Standardbenutzername
wildcardFormat	Zeichenfolge	• CHAR- Erkennt nur Zeichenplatzhalter (wie Broker Version 1) • **TOPIC** - Erkennt nur Platzhalter auf Themenebene (wie in Broker Version 2)	Gibt die zu verwendende Version der Platzhaltersyntax an

Anmerkungen:

1. Diese Eigenschaft kann mit Version 70 von IBM MQ classes for JMS verwendet werden.
2. Sie finden wichtige Informationen zur Verwendung der Eigenschaft 'sslFipsRequired' im Abschnitt „Einschränkungen beim IBM MQ-Ressourcenadapter” auf Seite 462.
3. Informationen zur Konfiguration des Ressourcenadapters zur Lokalisierung eines Exits finden Sie im Abschnitt „IBM MQ classes for JMS für Verwendung von Kanalexits konfigurieren” auf Seite 294.
4. Die Eigenschaft 'dynamicallyBalanced' wird nicht im Rahmen des Supports für XA-Transaktionen unterstützt. Wenn 'dynamicallyBalanced' den Wert 'true' aufweist, muss die MDB so konfiguriert werden, dass XA-Transaktionen inaktiviert werden.

Eigenschaften zur Erstellung eines JMS-Verbindungskonsumenten

Anmerkung: Die Eigenschaften **destination** und **destinationType** müssen explizit definiert werden. Alle anderen Eigenschaften in Tabelle 65 auf Seite 486 sind optional.

Tabelle 65. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung eines JMS-Verbindungskonsumenten verwendet werden

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
destination	Zeichenfolge	Der Name des Ziels	Das Ziel, von dem Nachrichten empfangen werden sollen. Die Eigenschaft **useJNDI** legt fest, wie der Wert dieser Eigenschaft interpretiert wird.
destinationLookup	Zeichenfolge	• **Null** • Der JNDI-Name eines Zielobjekts	Wenn diese Eigenschaft gesetzt ist, sucht die Aktivierungsspezifikation im JNDI-Namensbereich des Anwendungsservers nach dem JMS-Zielobjekt mit dem angegebenen JNDI-Namen und verwendet anschließend die Eigenschaften dieses Objekts zur Erstellung eines JMS-Verbindungskonsumenten. Diese Einstellung hat Vorrang vor den anderen Eigenschaften der Aktivierungsspezifikation. Weitere Informationen finden Sie unter „Eigenschaften 'connectionFactoryLookup' und 'destinationLookup' der Aktivierungsspezifikation” auf Seite 490.
destinationType	Zeichenfolge	• jakarta.jms.Queue (Jakarta Messaging 3.0) • jakarta.jms.Topic (Jakarta Messaging 3.0) • javax.jms.Queue (JMS 2.0) • javax.jms.Topic (JMS 2.0)	Der Zieltyp (Warteschlange oder Thema)
maxMessages	int	• **1** • Eine positive Ganzzahl	Die maximale Anzahl der Nachrichten, die einer Serversitzung gleichzeitig zugeordnet werden können. Wenn die Aktivierungsspezifikation einer MDB Nachrichten in einer XA-Transaktion zustellt, wird unabhängig von der Einstellung dieser Eigenschaft der Wert 1 verwendet.

Tabelle 65. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung eines JMS-Verbindungskonsumenten verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
maxPoolDepth	int	• **10** • Eine positive Ganzzahl	Die maximale Anzahl der Serversitzungen im Sitzungspool des Servers, die vom Verbindungskonsumenten genutzt werden.
messageSelector	Zeichenfolge	• **Null** • Ein SQL92-Nachrichtenselektorausdruck	Ein Nachrichtenselektorausdruck, der angibt, welche Nachrichten übermittelt werden sollen
nonASFTimeout	int	• **0** • Eine positive Ganzzahl	Ein positiver Wert gibt an, dass eine Zustellung ohne ASF (Anwendungsserverfunktionen) verwendet wird. Der Wert entspricht der Zeit in Millisekunden, die eine Abrufanforderung auf Nachrichten wartet, die möglicherweise noch nicht eingetroffen sind (ein Get-Aufruf mit Wartezeit). Der Standardwert 0 gibt an, dass die ASF-Zustellung verwendet wird. Dieser Parameter ist gültig, wenn: • Die Anwendung wird unter WebSphere Application Server 7.0 oder höher ausgeführt. • Die Anwendung wird in WebSphere Liberty mit der entsprechenden Version des wmqJms-Client-Features ausgeführt. Weitere Informationen finden Sie unter „Liberty und der IBM MQ-Ressourcenadapter" auf Seite 463.
nonASFRollbackEnabled	Boolesch	• **false** (false) - Die Nachricht wird selbst dann verarbeitet, wenn die MDB fehlschlägt • true - Bei einem Fehler innerhalb der MDB wird die Nachricht mit einem Rollback in die Warteschlange zurückgesetzt.	Gibt an, ob die Nachrichtenzustellung innerhalb eines IBM MQ-Synchronisationspunkts erfolgt, wenn die MDB nicht transaktionsorientiert ist. Wird ignoriert, wenn die MDB transaktionsorientiert oder **nonASFTimeout** auf 0 gesetzt ist.
poolTimeout	int	• **300000** • Eine positive Ganzzahl	Die Zeit (in Millisekunden), die eine nicht verwendete Serversitzung im Sitzungspool des Servers geöffnet bleibt, bevor sie wegen Inaktivität geschlossen wird.

Tabelle 65. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung eines JMS-Verbindungskonsumenten verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
readAheadAllowed	int	• **DESTINATION** - Legen Sie fest, ob das Vorauslesen durch die Referenzierung der Warteschlangen- oder Themendefinition zulässig ist. • DISABLED - Das Vorauslesen ist nicht zulässig. • ENABLED - Das Vorauslesen ist zulässig. • QUEUE - Legen Sie fest, ob das Vorauslesen durch die Referenzierung der Warteschlangendefinition zulässig ist. • TOPIC - Legen Sie fest, ob das Vorauslesen durch die Referenzierung der Themendefinition zulässig ist.	Gibt an, ob der Browsing-Thread für die Aktivierungsspezifikation das Vorauslesen verwenden kann, um mehrere Nachrichten vom Ziel in einem internen Puffer zu durchsuchen, bevor die Übergabe an die Serversitzungen für ein Lesen mit Löschen erfolgt. **Anmerkung:** Das Vorauslesen zu erlauben, kann zu einer Zunahme von JMSCC0108-Nachrichten oder einer Verringerung der Leistung führen, oder auch zu beidem, wenn die MDB-Verarbeitungsrate nicht mit der Rate des Browsings von Nachrichten vom Ziel Schritt halten kann.
readAheadClosePolicy	int	• **ALL** - Alle Nachrichten im internen Vorauslesepuffer werden an die MDB zugestellt, bevor diese gestoppt wird. • CURRENT - Nur der aktuelle MDB-Aufruf wird abgeschlossen. In diesem Fall befinden sich möglicherweise noch Nachrichten im internen Vorauslesepuffer, die dann gelöscht werden.	Gibt an, wie mit Nachrichten im internen Vorauslesepuffer verfahren wird, wenn die MDB vom Administrator gestoppt wird.
receiveCCSID	int	• **0** - `Charset.defaultCharset` der JVM verwenden • 1208 - UTF-8 • Eine unterstützte ID des codierten Zeichensatzes (CCSID)	Die Zieleigenschaft, die die Ziel-CCSID für die Nachrichtenkonvertierung des Warteschlangenmanagers festlegt. Der Wert wird ignoriert, wenn **`receiveConversion`** nicht auf `QMGR` gesetzt ist.
receiveConversion	Zeichenfolge	• **CLIENT_MSG** • QMGR	Eine Zieleigenschaft, die bestimmt, ob eine Datenkonvertierung vom Warteschlangenmanager durchgeführt wird.

Tabelle 65. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung eines JMS-Verbindungskonsumenten verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
sharedSubscription	Boolesch	• **False** - Die MDB darf die Subskription nicht als gemeinsame Subskription öffnen. • True - Die MDB soll die Subskription als gemeinsame Subskription öffnen (mit den von JMS 2.0 vorgegebenen Regeln; siehe hierzu die JMS 2.0-Spezifikation auf Java.net).	Bestimmt, wie eine MDB durch eine gemeinsame Subskription gesteuert wird. Weitere Informationen zur Verwendung dieser Eigenschaft finden Sie im Abschnitt „Beispiele für Definition der Eigenschaft sharedSubscription" auf Seite 493.
startTimeout	int	• **10 000** • Eine positive Ganzzahl	Die in Millisekunden angegebene Zeit, innerhalb der die Zustellung einer Nachricht an eine MDB beginnen muss, nachdem die Zustellungsbearbeitung der Nachricht eingeplant wurde. Wenn diese Zeit verstrichen ist, wird die Nachricht mit einem Rollback in die Warteschlange zurückgesetzt.
subscriptionDurability	Zeichenfolge	• **NonDurable** - Eine nicht permanente Subskription wird für die Zustellung von Nachrichten an eine nachrichtengesteuerte Bean (MDB, Message-driven Bean) verwendet, die das Thema subskribiert. • Durable - Eine permanente Subskription wird für die Zustellung von Nachrichten an eine nachrichtengesteuerte Bean (MDB, Message-driven Bean) verwendet, die das Thema subskribiert.	Gibt an, ob eine permanente oder eine nicht permanente Subskription für die Zustellung von Nachrichten an eine nachrichtengesteuerte Bean (MDB, Message-driven Bean) verwendet wird, die das Thema subskribiert.
subscriptionName	Zeichenfolge	• **"" (leere Zeichenfolge)** • Ein Subskriptionsname	Der Name der permanenten Subskription

Tabelle 65. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung eines JMS-Verbindungskonsumenten verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
useJNDI	Boolesch	• **false** (falsch) - Die Eigenschaft 'destination' wird als Name einer Warteschlange oder eines Themas von IBM MQ interpretiert. • true-Die Eigenschaft destination wird als Name eines der folgenden Objekte im JNDI-Namensbereich des Anwendungsservers interpretiert: – jakarta.jms.Queue (Jakarta Messaging 3.0) – jakarta.jms.Topic (Jakarta Messaging 3.0) – javax.jms.Queue (JMS 2.0) – javax.jms.Topic (JMS 2.0)	Deprecated Legt fest, die der Wert der Eigenschaft mit dem Namen 'destination' interpretiert wird **Anmerkung:** Diese Eigenschaft wird ab IBM MQ 9.0 nicht mehr weiter unterstützt. Stattdessen wird die Eigenschaft destinationLookup verwendet.

Konflikte und Abhängigkeiten zwischen Eigenschaften

Ein ActivationSpec-Objekt kann Eigenschaften aufweisen, die miteinander in Konflikt stehen. Beispielsweise können Sie TLS-Eigenschaften für eine Verbindung im Bindungsmodus angeben. In diesem Fall wird das Verhalten durch den Transporttyp und die Nachrichtendomäne bestimmt (entweder Punkt-zu-Punkt oder Publish/Subscribe, je nach Wert der Eigenschaft **destinationType**). Alle Eigenschaften, die nicht auf den angegebenen Transporttyp oder die Nachrichtendomäne zutreffen, werden ignoriert.

Wenn Sie eine Eigenschaft definieren, die eine Definition weiterer Eigenschaften erfordert, Sie diese aber nicht definieren, löst das ActivationSpec-Objekt die Ausnahmebedingung 'InvalidPropertyException' aus, wenn seine Methode 'validate()' während der Implementierung einer MDB aufgerufen wird. Die Ausnahmebedingung wird dem Administrator des Anwendungsservers in einer Weise gemeldet, die vom Anwendungsserver abhängt. Wenn Sie beispielsweise die Eigenschaft 'subscriptionDurability' auf 'Durable' setzen, Sie also permanente Subskriptionen verwenden möchten, müssen Sie auch die Eigenschaft **subscriptionName** definieren.

Wenn die Eigenschaften **ccdtURL** und **channel** beide definiert sind, wird die Ausnahmebedingung 'InvalidPropertyException' ausgelöst. Wenn Sie jedoch nur die Eigenschaft **ccdtURL** definieren, übernehmen Sie für die Eigenschaft **channel** den Standardwert SYSTEM.DEF.SVRCONNwird keine Ausnahme ausgelöst und die durch die Eigenschaft **ccdtURL** angegebene Definitionstabelle für Clientkanäle wird zum Starten einer JMS -Verbindung verwendet.

Eigenschaften ' connectionFactoryLookup' und 'destinationLookup' der Aktivierungsspezifikation

In der JMS 2.0-Spezifikation wurden zwei neue ActivationSpec-Eigenschaften eingeführt. Um bevorzugt vor anderen ActivationSpec-Eigenschaften verwendet zu werden, können die Eigenschaften 'connectionFactoryLookup' und 'destinationLookup' mit dem JNDI-Namen eines verwalteten Objekts bereitgestellt werden.

Wenn beispielsweise eine Verbindungsfactory in JNDI definiert ist und der JNDI -Name dieses Objekts in der Sucheigenschaft connectionFactoryfür eine Aktivierungsspezifikation angegeben ist, werden alle Eigenschaften der Verbindungsfactory, die in JNDI definiert sind, den Eigenschaften in Tabelle 64 auf Seite 475vorgezogen.

Wenn ein Ziel in JNDI definiert ist und der JNDI -Name in der Eigenschaft destinationLookup der ActivationSpecfestgelegt ist, werden die Werte verwendet, die den Werten in Tabelle 65 auf Seite 486bevorzugt werden. Weitere Informationen zur Verwendung dieser beiden Eigenschaften finden Sie im Abschnitt „Eigenschaften ' connectionFactoryLookup' und 'destinationLookup' der Aktivierungsspezifikation” auf Seite 490.

Diese beiden Eigenschaften können zur Angabe der JNDI-Namen von ConnectionFactory- und Zielobjekten verwendet werden, wenn diese Namen bevorzugt vor den durch die Eigenschaften der Aktivierungsspezifikation definierten Namen (siehe Tabelle 64 auf Seite 475 und Tabelle 65 auf Seite 486) verwendet werden sollen.

Beachten Sie unbedingt auch die folgenden Anmerkungen, die genau beschreiben, wie diese Eigenschaften funktionieren.

connectionFactoryLookup
Die aus JNDI abgefragte Verbindungsfactory dient als Quelle für die in Tabelle 64 auf Seite 475 aufgeführten Eigenschaften. Das ConnectionFactory-Objekt wird nicht im eigentlichen Sinne zur Herstellung von JMS-Verbindungen verwendet, sondern seine Eigenschaften werden lediglich abgefragt. Die Eigenschaften der Verbindungsfactory überschreiben die entsprechenden Eigenschaften der Aktivierungsspezifikation. Hiervon ausgenommen ist lediglich die **ClientID**. Wenn diese Eigenschaft in der Aktivierungsspezifikation festgelegt ist, hat deren Wert Vorrang vor der Client-ID der Verbindungsfactory. Grund hierfür ist, dass die gleiche Verbindungsfactory oft für mehrere Aktivierungsspezifikationen verwendet wird. Durch diese Handhabung vereinfacht sich die Verwaltung. Allerdings gibt die JMS 2.0-Spezifikation vor, dass jede aus einer Verbindungsfactory erstellte JMS-Verbindung eine eindeutige **ClientID** aufweisen muss. Aufgrund dieser Spezifikation muss eine Aktivierungsspezifikation in der Lage sein, jeden in der Verbindungsfactory festgelegten ClientID-Wert zu überschreiben. Gibt die Aktivierungsspezifikation jedoch keine **ClientID** vor, kann jeder Wert der Verbindungsfactory verwendet werden.

destinationLookup
Die Aktivierungsspezifikation legt die Eigenschaften **Destination** und **UseJndi** fest. Wenn das **UseJndi**-Flag auf `true` gesetzt ist, gilt der Wert der Eigenschaft 'Destination' als JNDI-Name und JNDI wird nach einem Zielobjekt mit diesem JNDI-Namen durchsucht.

Die Eigenschaft 'destinationLookup' verhält sich genauso. Wenn diese Eigenschaft festgelegt ist, wird JNDI nach einem Zielobjekt mit dem von der Eigenschaft vorgegebenen JNDI-Namen durchsucht. Allerdings hat diese Eigenschaft Vorrang vor der Eigenschaft **useJNDI**.

Deprecated Die Eigenschaft useJNDI ist in IBM MQ 9.0 veraltet, da die Eigenschaft **destinationLookup** die Spezifikation JMS 2.0 oder höher ist, die der Ausführung derselben Funktion entspricht.

ActivationSpec-Eigenschaften ohne Entsprechungen in IBM MQ classes for JMS

Die meisten Eigenschaften eines ActivationSpec -Objekts sind äquivalent zu Eigenschaften von IBM MQ classes for JMS -oder IBM MQ classes for Jakarta Messaging -Objekten oder zu Parametern von IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging -Methoden. Drei Optimierungseigenschaften und eine Eigenschaft für die Benutzerfreundlichkeit haben jedoch keine Entsprechungen in IBM MQ classes for JMS oder IBM MQ classes for Jakarta Messaging:

startTimeout
Die in Millisekunden angegebene Zeit, die der Work Manager des Anwendungsservers wartet, bis Ressourcen verfügbar werden, nachdem der Ressourcenadapter ein Arbeitsobjekt für die Zustellung einer Nachricht an eine MDB geplant hat. Wenn diese Zeit verstrichen ist, bevor die Zustellung der Nachricht begonnen hat, wird das Arbeitsobjekt aufgrund einer Zeitlimitüberschreitung beendet, die Nachricht wird mit einem Rollback in die Warteschlange zurückgesetzt und der Ressourcenadapter kann dann erneut versuchen, die Nachricht zuzustellen. Es wird eine Warnung in den Diagnosetrace geschrieben (falls aktiviert). Dies hat jedoch keine weiteren Auswirkungen auf den Prozess der Nachrichtenzustellung. Normalerweise tritt diese Bedingung nur auf, wenn eine hohe Auslastung auf dem System des Anwendungsservers besteht. Falls die Bedingung regelmäßig auftritt, sollten Sie den Wert dieser Eigenschaft erhöhen, damit dem Work Manager mehr Zeit für die Planung der Nachrichtenzustellung zur Verfügung steht.

maxPoolDepth
Die maximale Anzahl der Serversitzungen im Sitzungspool des Servers, die von einem Verbindungskonsumenten genutzt werden. Wenn eine Serversitzung erstellt wird, startet sie einen Dialog mit einem Warteschlangenmanager. Der Verbindungskonsument verarbeitet eine Serversitzung, um eine Nachricht an eine MDB zuzustellen. Eine höhere Pooltiefe ermöglicht es, dass in Situationen mit einem hohen Umfang mehr Nachrichten gleichzeitig zugestellt werden können, belegt jedoch auch mehr Ressourcen des Anwendungsservers. Falls viele MDBs implementiert werden sollen, wird empfohlen, die Pooltiefe niedriger einzustellen, damit die Arbeitslast auf dem Anwendungsserver überschaubar bleibt. Da jeder Verbindungskonsument einen eigenen Serversitzungspool verwendet, definiert diese Eigenschaft nicht die Gesamtzahl der Serversitzungen, die sämtlichen Verbindungskonsumenten zur Verfügung stehen.

poolTimeout
Die in Millisekunden angegebene Zeit, die eine nicht verwendete Serversitzung im Sitzungspool des Servers geöffnet bleibt, bevor sie wegen Inaktivität geschlossen wird. Eine temporäre Zunahme der Nachrichtenarbeitslast führt dazu, dass zusätzliche Serversitzungen erstellt werden, damit die Arbeitslast verteilt werden kann. Wenn die Nachrichtenarbeitslast aber wieder normal ist, verbleiben die zusätzlichen Serversitzungen im Pool und werden nicht verwendet.

Jedes Mal, wenn eine Serversitzung verwendet wird, wird sie mit einer Zeitmarke markiert. Ein Scavenger-Thread prüft laufend, ob die einzelnen Serversitzungen in dem Zeitraum verwendet wurden, der mit dieser Eigenschaft angegeben wurde. Falls eine Serversitzung nicht verwendet wurde, wird sie geschlossen und aus dem Serversitzungspool entfernt. Eine Serversitzung wird möglicherweise nicht sofort nach Ablauf des angegebenen Zeitraums geschlossen. Diese Eigenschaft gibt den Mindestzeitraum der Inaktivität vor der Entfernung an.

useJNDI
Sie finden eine Beschreibung dieser Eigenschaft in Tabelle 65 auf Seite 486.

MDB implementieren

Wenn Sie eine MDB implementieren möchten, müssen Sie zuerst die Eigenschaften eines ActivationSpec-Objekts definieren und die von der MDB benötigten Eigenschaften angeben. Im folgenden Beispiel sehen Sie eine typische Gruppe von Eigenschaften, die Sie möglicherweise explizit definieren:

JM 3.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   jakarta.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

JMS 2.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

Der Anwendungsserver verwendet die Eigenschaften, um ein ActivationSpec-Objekt zu erstellen, das dann einer MDB zugeordnet wird. Die Eigenschaften des ActivationSpec-Objekts legen fest, wie Nachrichten an die MDB zugestellt werden. Die Implementierung der MDB schlägt fehl, wenn die MDB eine dezentrale Transaktionsverarbeitung erfordert, diese jedoch nicht vom Ressourcenadapter unterstützt werden. Sie finden Informationen darüber, wie Sie den Ressourcenadapter installieren, damit eine dezentrale Transaktionsverarbeitung unterstützt wird, im Abschnitt „IBM MQ-Ressourcenadapter installieren” auf Seite 466.

Falls mehrere MDBs Nachrichten von demselben Ziel empfangen, wird eine in der Punkt-zu-Punkt-Domäne gesendete Nachricht nur von einer MDB empfangen. Dies gilt selbst dann, wenn auch andere MDBs für den Empfang der Nachricht infrage kommen. Insbesondere ist zu beachten, dass nur eine der MDBs die Nachricht empfängt, wenn zwei MDBs unterschiedliche Nachrichtenselektoren verwenden und eine eingehende Nachricht beiden Nachrichtenselektoren entspricht. Da nicht definiert ist, welche MDB für den Empfang einer Nachricht gewählt wird, können Sie sich nicht darauf verlassen, dass eine bestimmte MDB die Nachricht empfängt. Die in der Publish/Subscribe-Domäne gesendeten Nachrichten werden von allen MDBs empfangen, die infrage kommen.

Gelegentlich kann es vorkommen, dass eine Nachricht, die an eine MDB zugestellt wurde, mit einem Rollback in eine IBM MQ-Warteschlange zurückgesetzt wird. Diese Rollback-Operation kann beispielsweise durchgeführt werden, wenn eine Nachricht innerhalb einer Arbeitseinheit zugestellt wird, die dann rückgängig gemacht wird. Eine Nachricht, für die ein Rollback erfolgt ist, wird erneut zugestellt, allerdings kann eine falsch formatierte Nachricht dazu führen, dass eine MDB wiederholt fehlschlägt und daher keine Zustellung möglich ist. Eine solche Nachricht wird als nicht verarbeitbare Nachricht bezeichnet. Sie können in der Konfiguration von IBM MQ festlegen, dass IBM MQ classes for JMS eine nicht verarbeitbare Nachricht automatisch an eine andere Nachricht zur Untersuchung übertragen oder dass die Nachricht gelöscht wird.

Sie finden Details zur Behandlung nicht verarbeitbarer Nachrichten im Abschnitt „Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln” auf Seite 244.

Zugehörige Konzepte

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

Federal Information Processing Standards (FIPS) für AIX, Linux, and Windows

Zugehörige Tasks

JMS-Ressourcen in WebSphere Application Server konfigurieren

Beispiele für Definition der Eigenschaft sharedSubscription

Sie können die Eigenschaft 'sharedSubscription' einer Aktivierungsspezifikation in einer Datei `server.xml` von WebSphere Liberty definieren. Alternativ können Sie die Eigenschaft in einer Message-driven Bean (MDB) mithilfe von Annotationen definieren.

Beispiel: Definition in einer Datei `server.xml` von Liberty

Das folgende Beispiel zeigt, wie eine Aktivierungsspezifikation in einer Datei `server.xml` von WebSphere Liberty definiert wird. Im Beispiel wird eine permanente, gemeinsame Subskription für einen Warteschlangenmanager auf localhost/port 1490 erstellt.

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5" subscriptionName="MySubNa
me"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

Beispiel: Definition in einer MDB

Das folgende Beispiel zeigt, wie die Eigenschaft sharedSubscription in der MDB mithilfe von Annotationen definiert wird:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

Das folgende Beispiel zeigt einen Teil des MDB-Codes, in dem die Annotationsmethode angewendet wird:

```
JM 3.0
/**
 * Message-Driven Bean example using Annotations for configuration
 */
```

```
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

JMS 2.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

Zugehörige Konzepte

Subskribenten und Subskriptionen

Subskriptionspermanenz

„Klone und gemeinsam genutzte Subskriptionen” auf Seite 343

Es gibt zwei Methoden, mehreren Konsumenten Zugriff auf dieselbe Subskription zu erteilen. die Verwendung geklonter Subskriptionen oder die Verwendung gemeinsam genutzter Subskriptionen.

Ressourcenadapter für abgehende Kommunikation konfigurieren

Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

Beispiel für die Verwendung abgehender Kommunikation

Bei Verwendung einer abgehenden Kommunikation startet eine auf einem Anwendungsserver ausgeführte Anwendung eine Verbindung mit einem Warteschlangenmanager und sendet dann Nachrichten an dessen Warteschlangen und empfängt Nachrichten daraus. Diese Verarbeitung verläuft synchron. Die folgende Servlet-Methode doGet() verwendet beispielsweise eine abgehende Kommunikation:

JM 3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
```

```
...
// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

Wenn das Servlet eine HTTP-Abrufanforderung GET empfängt, ruft es ein ConnnectionFactory-Objekt und ein Queue-Objekt aus dem JNDI-Namensbereich ab und verwendet die Objekte für das Senden einer Nachricht an eine IBM MQ-Warteschlange. Das Servlet empfängt dann die gesendete Nachricht.

Ressourcen für die abgehende Kommunikation

Definieren Sie zum Konfigurieren der abgehenden Kommunikation Java EE Connector Architecture -Ressourcen (JCA) in den folgenden Kategorien:

- Eigenschaften eines ConnectionFactory-Objekts, die vom Anwendungsserver für die Erstellung eines JMS-ConnectionFactory-Objekts verwendet werden.
- Eigenschaften eines verwalteten Zielobjekts, die vom Anwendungsserver für die Erstellung eines JMS-Queue-Objekts oder eines JMS-Topic-Objekts verwendet werden.

Auf welche Weise Sie diese Eigenschaften definieren, hängt von der Verwaltungsschnittstelle ab, die von Ihrem Anwendungsserver bereitgestellt wird. ConnectionFactory-, Queue- und Topic-Objekte, die vom Anwendungsserver erstellt wurden, werden an einen JNDI-Namensbereich gebunden, aus dem sie von einer Anwendung abgerufen werden können.

In der Regel definieren Sie ein ConnectionFactory-Objekt für jeden Warteschlangenmanager, zu dem Anwendungen möglicherweise eine Verbindung herstellen. Sie definieren ein Queue-Objekt für jede Warteschlange, auf die Anwendungen in der Punkt-zu-Punkt-Domäne möglicherweise zugreifen möchten. Zudem definieren Sie ein Topic-Objekt für jedes Thema, das Anwendungen möglicherweise veröffentlichen oder subskribieren möchten. Ein ConnectionFactory-Objekt kann domänenunabhängig sein. Alternativ kann es auch domänenspezifisch sein, also ein QueueConnectionFactory-Objekt für die Punkt-zu-Punkt-Domäne oder ein TopicConnectionFactory-Objekt für die Publish/Subscribe-Domäne.

Tipp: Mit JMS 2.0 können mit einer Verbindungsfactory sowohl Verbindungen also auch Kontexte erstellt werden. Daher kann ein Verbindungspool einer Verbindungsfactory zugeordnet sein, die sowohl Verbindungen als auch Kontexte enthält. Allerdings wird empfohlen, eine Verbindungsfactory nur zur Erstellung von Verbindungen oder von Kontexten zu verwenden. Dadurch wird sichergestellt, dass der Verbindungspool für diese Verbindungsfactory nur Objekte eines Typs enthält, wodurch der Pool effizienter wird.

Eigenschaften eines ConnectionFactory-Objekts

In Tabelle 66 auf Seite 496 werden die Eigenschaften eines ConnectionFactory-Objekts aufgelistet. Der Anwendungsserver verwendet diese Eigenschaften zur Erstellung eines JMS-ConnectionFactory-Objekts.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
applicationName	Zeichenfolge	• Der aufrufende Klassenname (falls verfügbar), der so angepasst wird, dass er nicht mehr als 28 Zeichen enthält. Falls er nicht verfügbar ist, wird die Zeichenfolge `WebSphere MQ Client for Java` verwendet.	Der Name, unter dem eine Anwendung beim Warteschlangenmanager registriert wird. Dieser Anwendungsname wird durch den Befehl **`DISPLAY CONN MQSC/PCF`** (wobei das Feld die Bezeichnung **`APPLTAG`** trägt) oder in IBM MQ Explorer **Application Connections** angezeigt (wobei das Feld die Bezeichnung **`App name`** (Anwendungsname) trägt).
brokerCCSub-Warteschlange	Zeichenfolge	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Verbindungskonsument nicht permanente Subskriptionsnachrichten empfängt.
Warteschlange brokerControl	Zeichenfolge	• **SYSTEM.BROKER.CONTROL.QUEUE** • Ein Warteschlangenname	Der Name der Steuerwarteschlange des Brokers.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
brokerPub-Warteschlange	Zeichenfolge	• **SYSTEM.BROKER.DEFAULT.STREAM** • Ein Warteschlangenname	Der Name der Warteschlange, an die veröffentlichte Nachrichten gesendet werden (die Datenstromwarteschlange).
brokerQueue	Zeichenfolge	• **"" (leere Zeichenfolge)** • Name eines Warteschlangenmanagers	Der Namen des Warteschlangenmanagers, für den der Broker aktiv ist.
brokerSub-Warteschlange	Zeichenfolge	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Konsument nicht permanenter Nachrichten die Nachrichten empfängt. Weitere Informationen finden Sie unter der Beschreibung der Eigenschaft BROKERSUBQ.
brokerVersion	Zeichenfolge	• **unspecified** - Nachdem der Broker von V6 auf V7 migriert wurde, legen Sie diese Eigenschaft fest, damit RFH2-Header nicht mehr verwendet werden. Nach der Migration ist diese Eigenschaft nicht mehr relevant. • **V1** - Für die Verwendung eines IBM MQ-Publish/Subscribe-Brokers. Dieser Wert ist der Standardwert, falls TRANSPORT auf BIND oder CLIENT gesetzt ist. • **V2** - Für die Verwendung eines Brokers von IBM Integration Bus im nativen Modus. Dieser Wert ist der Standardwert, falls TRANSPORT auf DIRECT oder DIRECTHTTP gesetzt ist.	Die Version des verwendeten Brokers.
ccdtURL	Zeichenfolge	• **Null** • Eine URL (Uniform Resource Locator)	Eine URL, die den Namen und die Position der Datei angibt, welche die Definitionstabelle für den Clientkanal enthält und festlegt, wie auf die Datei zugegriffen werden kann.
CCSID	Zeichenfolge	• **819** • ID des codierten Zeichensatzes, die von der Java Virtual Machine (JVM) unterstützt wird	Die ID des codierten Zeichensatzes für eine Verbindung.
Kanal	Zeichenfolge	• **SYSTEM.DEF.SVRCONN** • Der Name eines MQI-Kanals	Der Name des zu verwendenden MQI-Kanals.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
cleanupInterval	int	• **3 600 000** • Eine positive Ganzzahl	Das in Millisekunden angegebene Intervall zwischen den Hintergrundausführungen des Publish/Subscribe-Bereinigungsdienstprogramms.
cleanupLevel	Zeichenfolge	• **SAFE** • KEINE • STRONG • FORCE • NONDUR	Die Bereinigungsstufe für einen brokerbasierten Subskriptionsspeicher.
clientID	Zeichenfolge	• **Null** • Eine Client-ID	Die Client-ID für eine Verbindung.
cloneSupport	Zeichenfolge	• **DISABLED** - Nur eine Instanz eines permanenten Topic-Subskribenten kann jeweils ausgeführt werden. • ENABLED - Zwei oder mehr Instanzen desselben permanenten Topic-Subskribenten können gleichzeitig ausgeführt werden. Allerdings muss jede Instanz auf einer separaten Java Virtual Machine (JVM) ausgeführt werden.	Geben Sie an, ob zwei oder mehr Instanzen desselben permanenten Topic-Subskribenten gleichzeitig ausgeführt werden können.
connectionNameList	Zeichenfolge	• **localhost(1414)** • Eine Zeichenfolge mit Elementen, die jeweils durch Kommas voneinander getrennt sind. Dabei hat jedes Element folgendes Format: `HOSTNAME(PORT)` Dabei steht *HOSTNAME* entweder für einen DNS-Namen oder für eine IP-Adresse.	Eine Liste mit TCP/IP-Verbindungsnamen, die für die abgehende Kommunikation verwendet werden. **connectionNameList** setzt die Eigenschaften **hostname** und **port** außer Kraft. Mit dieser Eigenschaft wird eine Verbindung zu Mehrinstanz-Warteschlangenmanagern wiederhergestellt. **connectionNameList** weist eine ähnliche Form wie **localAddress** auf, darf jedoch nicht damit verwechselt werden. **localAddress** gibt die Merkmale der lokalen Kommunikation an, während **connectionNameList** angibt, wie ein ferner Warteschlangenmanager zu erreichen ist.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
failIfQuiesce	Boolesch	• **wahr** • false	Gibt an, ob Aufrufe bestimmter Methoden fehlschlagen, wenn der Warteschlangenmanager gerade stillgelegt wird.
headerCompression	Zeichenfolge	• **Ohne** • SYSTEM - Es wird eine RLE-Komprimierung des Nachrichtenheaders durchgeführt.	Eine Liste der Verfahren, die zum Komprimieren von Headerdaten in einer Verbindung verwendet werden können.
hostName	Zeichenfolge	• **localhost** • Ein Hostname • Eine IP-Adresse	Der Hostname oder die IP-Adresse des Systems, auf dem sich der Warteschlangenmanager befindet. Die Eigenschaften **hostname** und **port** werden durch die Eigenschaft **connectionNameList** außer Kraft gesetzt, falls diese angegeben ist.
localAddress	Zeichenfolge	• **Null** • Eine Zeichenfolge im folgenden Format: `[ host_name ][(low_port [, high_port ])]` Dabei steht *Hostname* für einen Hostnamen oder eine IP-Adresse. *Niedriger_Port* und *hoher_Port* sind TCP-Portnummern und die Klammern bezeichnen eine optionale Komponente.	Für eine Verbindung mit einem Warteschlangenmanager gibt diese Eigenschaft entweder eine oder beide der folgenden Informationen an: • die zu verwendende lokale Netzschnittstelle • den zu verwendenden lokalen Port bzw. Portbereich **localAddress** weist eine ähnliche Form wie **connectionNameList** auf, darf jedoch nicht damit verwechselt werden. **localAddress** gibt die Merkmale der lokalen Kommunikation an, während **connectionNameList** angibt, wie ein ferner Warteschlangenmanager zu erreichen ist.
messageCompression	Zeichenfolge	• **Ohne** • Eine Liste mit einem oder mehreren der folgenden Werte, die durch Leerzeichen voneinander getrennt sind: RLE ZLIBFAST ZLIBHIGH V 9.4.0 LZ4FAST V 9.4.0 LZ4HIGH	Eine Liste der Verfahren, die zum Komprimieren von Nachrichtendaten in einer Verbindung verwendet werden können.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
messageSelection	Zeichenfolge	• **Client** • BROKER	Legt fest, ob die Nachrichtenauswahl durch IBM MQ classes for JMS oder durch den Broker erfolgt. Die Nachrichtenauswahl durch den Broker wird nicht unterstützt, wenn für **brokerVersion** der Wert 1 festgelegt wurde.
Kennwort	Zeichenfolge	• **Null** • Ein Kennwort	Das Standardkennwort, das beim Erstellen einer Verbindung mit dem Warteschlangenmanager verwendet wird.
pollingInterval	int	• **5000** • Jede beliebige positive Ganzzahl.	Wenn sich bei den einzelnen Nachrichtenlistenern innerhalb einer Sitzung keine geeignete Nachricht in der zugehörigen Warteschlange befindet, ist dieser Wert das maximale Intervall in Millisekunden, das verstreicht, bevor die einzelnen Nachrichtenlistener erneut versuchen, eine Nachricht aus der zugehörigen Warteschlange abzurufen. Wenn regelmäßig keine geeigneten Nachrichten für die Nachrichtenlistener innerhalb einer Sitzung verfügbar sind, sollten Sie einen höheren Wert für diese Eigenschaft angeben. Diese Eigenschaft ist nur relevant, wenn für **TRANSPORT** der Wert `BIND` oder `CLIENT` festgelegt ist.
port	int	• **1414** • Eine TCP-Portnummer	Der Port, an dem der Warteschlangenmanager empfangsbereit ist. Die Eigenschaften **hostname** und **port** werden durch die Eigenschaft **connectionNameList** außer Kraft gesetzt, falls diese angegeben ist.
providerVersion	Zeichenfolge	• **unspecified** • Eine Zeichenfolge in einem der folgenden Formate – V.R.M.F – V.R.M – V.R – V Dabei sind V, R, M und F Ganzzahlen größer-gleich null.	Die Version, das Release, die Modifikationsstufe und das Fixpack des Warteschlangenmanagers, zu dem die Anwendung eine Verbindung herstellen möchte.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
pubAck-Intervall	int	• **25** • Eine positive Ganzzahl	Die Anzahl an Nachrichten, die von einem Publisher veröffentlicht werden, bevor IBM MQ classes for JMS eine Bestätigung vom Broker anfordert.
queueManager	Zeichenfolge	• **"" (leere Zeichenfolge)** • Name eines Warteschlangenmanagers	Der Name des Warteschlangenmanagers, zu dem eine Verbindung hergestellt werden soll.
receiveExit [3]	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen besteht, die durch Kommas voneinander getrennt sind. Dabei gibt jedes Element den vollständig qualifizierten Namen einer Klasse an, die die Schnittstelle der IBM MQ classes for Java (MQReceiveExit) implementiert.	Gibt ein Empfangsexitprogramm oder eine Folge von Empfangsexitprogrammen für den Kanal an, die nacheinander ausgeführt werden sollen.
receiveExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen von Benutzerdaten besteht, die durch Kommas voneinander getrennt sind	Die Benutzerdaten, die an die Empfangsexitprogramme des Kanals gesendet werden, wenn diese aufgerufen werden.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
rescanInterval	int	• **5000** • Jede beliebige positive Ganzzahl.	Wenn ein Nachrichtenkonsument in der Punkt-zu-Punkt-Domäne einen Nachrichtenselektor für die Auswahl der zu empfangenden Nachrichten verwendet, suchen die IBM MQ classes for JMS in der IBM MQ-Warteschlange in der durch das Warteschlangenattribut **MsgDeliverySequence** festgelegten Reihenfolge nach geeigneten Nachrichten. Wenn die IBM MQ classes for JMS eine geeignete Nachricht finden und diese an den Konsumenten übermitteln, setzen die IBM MQ classes for JMS die Suche nach der nächsten geeigneten Nachricht ab der aktuellen Position in der Warteschlange fort. Die IBM MQ classes for JMS durchsuchen die Warteschlange auf diese Weise, bis das Ende der Warteschlange erreicht oder das durch den Wert dieser Eigenschaft festgelegte Intervall in Millisekunden abgelaufen ist. In jedem Fall kehren die IBM MQ classes for JMS zum Anfang der Warteschlange zurück, um die Suche fortzusetzen, und es beginnt ein neues Zeitintervall.
securityExit [3]	Zeichenfolge	• **Null** • Der vollständig qualifizierte Name einer Klasse, die die Schnittstelle der IBM MQ classes for Java (MQSecurityExit) implementiert	Gibt ein Sicherheitsexitprogramm für den Kanal an.
securityExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge mit Benutzerdaten	Die Benutzerdaten, die an ein Sicherheitsexitprogramm des Kanals übergeben werden, wenn dieses aufgerufen wird.
sendCheckCount	int	• **0** • Jede beliebige positive Ganzzahl.	Die Anzahl der Sendeaufrufe, die zwischen den Prüfungen auf Fehler innerhalb einer einzelnen JMS-Sitzung ohne Transaktion bei der asynchronen PUT-Operation zugelassen werden sollen.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
sendExit [3]	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen besteht, die durch Kommas voneinander getrennt sind. Dabei gibt jedes Element den vollständig qualifizierten Namen einer Klasse an, die die Schnittstelle der IBM MQ classes for Java (MQSendExit) implementiert.	Gibt ein Sendeexitprogramm oder eine Folge von Sendeexitprogrammen für den Kanal an, die nacheinander ausgeführt werden sollen.
sendExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen von Benutzerdaten besteht, die durch Kommas voneinander getrennt sind	Die Benutzerdaten, die an die Sendeexitprogramme des Kanals gesendet werden, wenn diese aufgerufen werden.
shareConvAllowed	Boolesch	• NO-Eine Clientverbindung kann ihren Socket nicht gemeinsam nutzen. • **YES** -Eine Clientverbindung kann ihren Socket gemeinsam nutzen.	Gibt an, ob eine Clientverbindung die zugehörigen Sockets gemeinsam mit anderen JMS-Verbindungen der höchsten Ebene aus dem gleichen Prozess mit dem gleichen Warteschlangenmanager nutzen kann, wenn die Kanaldefinitionen übereinstimmen.
sparseSubscriptions	Boolesch	• **false** (false) - Subskriptionen empfangen häufige übereinstimmende Nachrichten. • true (wahr) - Subskriptionen empfangen seltene übereinstimmende Nachrichten. Für diesen Wert ist es erforderlich, dass die Subskriptionswarteschlange zum Durchsuchen geöffnet werden kann.	Diese Eigenschaft steuert die Richtlinie für den Nachrichtenabruf eines TopicSubscriber-Objekts.
sslCertStores	Zeichenfolge	• **Null** • Eine Zeichenfolge mit einer oder mehreren LDAP-URLs, die durch Leerzeichen getrennt sind. Jede LDAP-URL hat folgendes Format: `ldap://host_name [: port ]` Dabei steht *Hostname* für einen Hostnamen oder eine IP-Adresse. *Port* ist eine TCP-Portnummer und die Klammern bezeichnen eine optionale Komponente.	Die LDAP-Server (LDAP = Lightweight Directory Access Protocol), auf denen die Zertifikatswiderrufslisten (CRLs) für TLS-Verbindungen gespeichert sind.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
sslCipherSuite	Zeichenfolge	• **Null** • Der Name einer Cipher-Suite	Die Cipher-Suite, die für eine TLS-Verbindung verwendet werden soll.
sslFipsRequired [2]	Boolesch	• **false** • true	Angabe, ob eine TLS-Verbindung eine vom IBM Java JSSE FIPS-Provider (IBMJSSEFIPS) unterstützte Cipher-Suite verwenden muss.
sslPeerName	Zeichenfolge	• **Null** • Eine Vorlage für definierte Namen	Bei einer TLS-Verbindung eine Vorlage, die zur Überprüfung des definierten Namens in dem vom Warteschlangenmanager bereitgestellten digitalen Zertifikat verwendet wird.
sslResetCount	int	• **0** • Eine Ganzzahl im Bereich 0 bis 999 999 999	Die Gesamtzahl der von einer TLS-Verbindung gesendeten und empfangenen Bytes vor der erneuten Vereinbarung der von TLS verwendeten geheimen Schlüssel.
sslSocketFactory	Zeichenfolge	Eine Zeichenfolge, die den vollständig qualifizierten Klassennamen einer Klasse darstellt, die eine Implementierung der Schnittstelle javax.net.ssl.SSLSocketFactory implementiert. Optional kann ein Argument für die Übergabe an die Konstruktormethode eingeschlossen werden. Dieses Argument wird in Klammern gesetzt.	Alle Verbindungen, die im Bereich des verwalteten Zielobjekts eingerichtet werden, verwenden Sockets, die aus dieser Implementierung der Schnittstelle SSLSocketFactory abgerufen werden.
Intervall statusRefresh	int	• **60000** • Jede beliebige positive Ganzzahl.	Das Aktualisierungsintervall der lange aktiven Transaktion (in Millisekunden), die feststellt, wenn ein Subskribent die Verbindung zum Warteschlangenmanager verliert. Diese Eigenschaft ist nur relevant, wenn für **SUBSTORE** der Wert QUEUE festgelegt ist.
subscriptionStore	Zeichenfolge	• **Broker** • MIGRATE • WARTESCHLANGE	Legt fest, ob IBM MQ classes for JMS persistente Daten über aktive Subskriptionen speichern.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
targetClientMatching	Boolesch	• **wahr** • false	Gibt an, ob eine Antwortnachricht, die an die Warteschlange gesendet wurde, welche im JMSReplyTo-Headerfeld angegeben ist, nur dann einen MQRFH2-Header aufweist, wenn die eingehende Nachricht über einen MQRFH2-Header verfügt. Sie können diese Eigenschaft auch für eine Aktivierungsspezifikation konfigurieren. Weitere Informationen finden Sie unter „Eigenschaft targetClientMatching für eine Aktivierungsspezifikation konfigurieren” auf Seite 515.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
temporaryModel	Zeichenfolge	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • Jede beliebige Zeichenfolge.	Der Name der Modellwarteschlange, anhand der temporäre JMS-Warteschlangen erstellt werden. Verwenden Sie `SYSTEM.DEFAULT.MODEL.QUEUE`, wenn beide der folgenden Punkte zutreffen: • Ihre Anwendung verwendet eine temporäre Warteschlange, die nicht persistente Nachrichten akzeptieren wird. • Immer nur jeweils eine Anwendung erstellt eine temporäre Warteschlange auf dem Warteschlangenmanager, auf den die ConnectionFactory verweist. Beachten Sie, dass die Warteschlange `SYSTEM.DEFAULT.MODEL.QUEUE` immer nur von jeweils einer Anwendung geöffnet werden kann. Verwenden Sie `SYSTEM.JMS.TEMPQ.MODEL` in den folgenden Situationen: • Wenn Ihre Anwendung eine temporäre Warteschlange verwendet, die persistente Nachrichten akzeptieren wird. • Wenn mehrere Anwendungen eine Verbindung zu dem Warteschlangenmanager herstellen können, auf den die ConnectionFactory verweist, und diese Anwendungen gleichzeitig temporäre Warteschlangen erstellen müssen. Definieren Sie in der folgenden Situation eine neue Modellwarteschlange, für die das Attribut **DEFPSIST** auf YES und das Attribut **DEFSOPT** auf SHARED gesetzt ist: • Wenn Ihre Anwendung eine temporäre Warteschlange verwendet, die nicht persistente Nachrichten akzeptieren wird, und sich mehrere Warteschlangenmanager mit dem Warteschlangenmanager verbinden werden, auf den die ConnectionFactory verweist, und diese Anwendungen gleichzeitig temporäre Warteschlangen erstellen müssen. Setzen Sie die Eigenschaft **tem-**

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
tempQPrefix	Zeichenfolge	• **"" (leere Zeichenfolge)** • Ein Präfix, mit dem der Name einer dynamischen IBM MQ-Warteschlange gebildet werden kann. Die Regeln für die Bildung des Präfix entsprechen den Regeln für die Bildung des Inhalts des Felds **DynamicQName** in einem IBM MQ -Objektdeskriptor (Struktur MQOD), aber das letzte nicht leere Zeichen muss ein Stern (*) sein. Wenn der Wert der Eigenschaft eine leere Zeichenfolge ist, verwendet IBM MQ classes for JMS den Wert AMQ.* beim Erstellen einer dynamischen Warteschlange.	Das Präfix, das verwendet wird, um den Namen einer dynamischen IBM MQ-Warteschlange zu bilden.
tempTopicPrefix	Zeichenfolge	Eine beliebige Zeichenfolge ungleich null, die nur aus gültigen Zeichen für eine IBM MQ-Topic-Zeichenfolge besteht	Beim Erstellen temporärer Themen generiert JMS eine Themenzeichenfolge im Format "TEMP/*TEMPTOPICPREFIX*/*eindeutige_ID* " oder, wenn diese Eigenschaft den Standardwert hat, nur "TEMP/*eindeutige_ID*". Durch Angabe eines Werts für die Eigenschaft **TEMPTOPICPREFIX** können Sie spezifische Modellwarteschlangen zum Erstellen der verwalteten Warteschlangen für Subskribenten von temporären Themen unter Verwendung der jeweiligen Verbindung definieren.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
transportType	Zeichenfolge	• **Client** • BINDINGS • BINDINGS_THEN_CLIENT	Gibt an, ob eine Verbindung mit einem Warteschlangenmanager den Clientmodus oder den Bindungsmodus verwendet. Bei Angabe von `BINDINGS_THEN_CLIENT` versucht der Ressourcenadapter zuerst, eine Verbindung im Bindungsmodus herzustellen. Schlägt dieser Verbindungsversuch fehl, versucht er, eine Verbindung im Clientmodus herzustellen. z/OS Wenn eine auf einem WebSphere Application Server for z/OS-System ausgeführte Aktivierungsspezifikation mit dem Transportmodus `BINDINGS_THEN_CLIENT` konfiguriert wurde und eine zuvor eingerichtete Verbindung unterbrochen wurde, versuchen alle durch die Aktivierungsspezifikation unternommenen Verbindungsversuche zunächst, die Verbindung im Transportmodus `BINDINGS` herzustellen. Erst wenn der Verbindungsversuch im Transportmodus `BINDINGS` fehlschlägt, versucht die Aktivierungsspezifikation einen Verbindungsaufbau im Transportmodus `CLIENT`.
username	Zeichenfolge	• **Null** • Ein Benutzername	Der beim Herstellen einer Verbindung zu einem Warteschlangenmanager zu verwendende Standardbenutzername.
wildcardFormat	int	• CHAR- Erkennt nur Zeichenplatzhalter (wie Broker Version 1) • TOPIC - Erkennt nur Platzhalter auf Themenebene (wie in Broker Version 2)	Gibt die zu verwendende Version der Platzhaltersyntax an.

Anmerkungen:

1. Sie finden wichtige Informationen zur Verwendung der Eigenschaft 'sslFipsRequired' im Abschnitt „Einschränkungen beim IBM MQ-Ressourcenadapter” auf Seite 462.
2. Informationen zur Konfiguration des Ressourcenadapters zur Lokalisierung eines Exits finden Sie im Abschnitt „IBM MQ classes for JMS für Verwendung von Kanalexits konfigurieren” auf Seite 294.

Im folgenden Beispiel sehen Sie eine typische Gruppe von Eigenschaften eines ConnectionFactory-Objekts:

```
channel:        SYSTEM.DEF.SVRCONN
```

```
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

Eigenschaften eines verwalteten Zielobjekts

Der Anwendungsserver verwendet die Eigenschaften eines verwalteten Zielobjekts zur Erstellung eines JMS-Queue-Objekts oder eines JMS-Topic-Objekts.

In Tabelle 67 auf Seite 509 werden die Eigenschaften aufgelistet, die sowohl für ein Queue-Objekt als auch für ein Topic-Objekt verwendet werden.

Tabelle 67. Eigenschaften, die sowohl für ein Queue-Objekt als auch für ein Topic-Objekt verwendet werden

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
CCSID	Zeichenfolge	• **1208** • ID des codierten Zeichensatzes, die von der Java Virtual Machine (JVM) unterstützt wird	Die die ID des codierten Zeichensatzes für das Ziel.
encoding	Zeichenfolge	• **NATIVE** • Eine aus drei Zeichen bestehende Zeichenfolge: – Das erste Zeichen gibt die Darstellung von binären Ganzzahlen an: - *N* bezeichnet eine normale Codierung. - *R* bezeichnet eine umgekehrte Codierung. – Das zweite Zeichen gibt die Darstellung von Ganzzahlen im gepackten Dezimalformat an: - *N* bezeichnet eine normale Codierung. - *R* bezeichnet eine umgekehrte Codierung. – Das dritte Zeichen gibt die Darstellung von Gleitkommazahlen an: - *N* bezeichnet eine IEEE-Standardcodierung. - *R* bezeichnet eine umgekehrte IEEE-Codierung. - *3* bezeichnet die zSeries-Codierung. NATIVE ist äquivalent zu der Zeichenfolge NNN.	Die Darstellung von binären Ganzzahlen, Ganzzahlen im gepackten Dezimalformat und Gleitkommazahlen für das Ziel.
expiry	Zeichenfolge	• **APP** - Die Ablaufzeit einer Nachricht wird durch den Nachrichtenproduzenten bestimmt. • UNLIM - Eine Nachricht läuft nie ab. • 0 - Eine Nachricht läuft nie ab. • Eine positive Ganzzahl, die die Ablaufzeit einer Nachricht in Millisekunden darstellt.	Die Ablaufzeit einer Nachricht, die an das Ziel gesendet wird.

Tabelle 67. Eigenschaften, die sowohl für ein Queue-Objekt als auch für ein Topic-Objekt verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
failIfQuiesce	Zeichenfolge	• **wahr** • false	Gibt an, ob ein Zugriffsversuch auf das Ziel fehlschlägt, wenn sich der Warteschlangenmanager im Stilllegungsmodus befindet.

Tabelle 67. Eigenschaften, die sowohl für ein Queue-Objekt als auch für ein Topic-Objekt verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
messageBodyStyle	Zeichenfolge	• **UNSPECIFIED** • JMS • MQ	Sie können die Eigenschaft **`messageBodyStyle`** für JMS-Warteschlangen und -Themen festlegen: UNSPECIFIED (Standardwert) • Beim Senden generieren IBM MQ classes for JMS je nachdem, welcher Wert für WMQ_TARGET_CLIENT angegeben ist, einen MQRFH2-Header und fügen diesen ein oder auch nicht. • Beim Empfangen legen IBM MQ classes for JMS die JMS-Nachrichteneigenschaften in Übereinstimmung mit den Werten im MQRFH2 fest, falls vorhanden. MQRFH2 wird nicht als Bestandteil des JMS-Nachrichtenhauptteils dargestellt. JMS • Beim Senden generieren IBM MQ classes for JMS automatisch einen MQRFH2-Header und fügen den Header in die IBM MQ-Nachricht ein. • Beim Empfangen legen IBM MQ classes for JMS die JMS-Nachrichteneigenschaften in Übereinstimmung mit den Werten im MQRFH2 fest, falls vorhanden. MQRFH2 wird nicht als Bestandteil des JMS-Nachrichtenhauptteils dargestellt. MQ • Beim Senden generieren IBM MQ classes for JMS keinen MQRFH2. • Beim Empfangen stellen IBM MQ classes for JMS den MQRFH2 als Bestandteil des JMS-Nachrichtenhauptteils dar.

Tabelle 67. Eigenschaften, die sowohl für ein Queue-Objekt als auch für ein Topic-Objekt verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
persistence	Zeichenfolge	• **APP** - Die Persistenz einer Nachricht wird durch den Nachrichtenproduzenten bestimmt. • QDEF - Die Persistenz einer Nachricht wird durch das Attribut **DefPersistence** der IBM MQ-Warteschlange bestimmt. • PERS - Eine Nachricht ist persistent. • NON - Eine Nachricht ist nicht persistent. • HIGH-Die Persistenz einer Nachricht wird durch das Attribut **NonPersistentMessageClass** der Warteschlange IBM MQ gemäß der Erläuterung in „Persistente JMS-Nachrichten” auf Seite 265bestimmt.	Die Persistenz einer Nachricht, die an das Ziel gesendet wird.
priority	Zeichenfolge	• **APP** - Die Priorität einer Nachricht wird durch den Nachrichtenproduzenten bestimmt. • QDEF - Die Priorität einer Nachricht wird durch das Attribut **DefPriority** der IBM MQ-Warteschlange bestimmt. • Eine Ganzzahl im Bereich von 0 (niedrigste Priorität) bis 9 (höchste Priorität).	Die Priorität einer Nachricht, die an das Ziel gesendet wird.
putAsyncAllowed	Zeichenfolge	• QUEUE - Legen Sie fest, ob asynchrone Put-Operationen durch die Referenzierung der Warteschlangendefinition zulässig sind. • TOPIC - Legen Sie fest, ob asynchrone Put-Operationen durch die Referenzierung der Themendefinition zulässig sind. • DESTINATION - Legen Sie fest, ob asynchrone Put-Operationen durch die Referenzierung der Warteschlangen- oder Themendefinition zulässig sind. • DISABLED - Asynchrone Put-Operationen sind nicht zulässig. • ENABLED - Asynchrone Put-Operationen sind zulässig.	Gibt an, ob Nachrichtenproduzenten asynchrone Put-Operationen für das Senden von Nachrichten an dieses Ziel verwenden dürfen.

Tabelle 67. Eigenschaften, die sowohl für ein Queue-Objekt als auch für ein Topic-Objekt verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
readAheadAllowed	int	• **DESTINATION** - Legen Sie fest, ob das Vorauslesen durch die Referenzierung der Warteschlangen- oder Themendefinition zulässig ist. • DISABLED - Das Vorauslesen ist nicht zulässig. • ENABLED - Das Vorauslesen ist zulässig. • QUEUE - Legen Sie fest, ob das Vorauslesen durch die Referenzierung der Warteschlangendefinition zulässig ist. • TOPIC - Legen Sie fest, ob das Vorauslesen durch die Referenzierung der Themendefinition zulässig ist.	Gibt an, ob Nachrichtenkonsumenten und Warteschlangenbrowser das Vorauslesen verwenden dürfen, um nicht persistente Nachrichten des Ziels vor dem Empfang in einen internen Puffer zu schreiben.
receiveCCSID	int	• **0** - `Charset.defaultCharset` der JVM verwenden • 1208 - UTF-8 • Eine unterstützte ID des codierten Zeichensatzes (CCSID)	Die Zieleigenschaft, die die Ziel-CCSID für die Nachrichtenkonvertierung des Warteschlangenmanagers festlegt. Der Wert wird ignoriert, wenn **`receiveConversion`** nicht auf `QMGR` gesetzt ist.
receiveConversion	Zeichenfolge	• **CLIENT_MSG** • QMGR	Eine Zieleigenschaft, die bestimmt, ob eine Datenkonvertierung vom Warteschlangenmanager durchgeführt wird.
targetClient	Zeichenfolge	• **JMS** - Das Ziel einer Nachricht ist eine JMS-Anwendung. • MQ -Das Ziel einer Nachricht ist eine Nicht-JMS IBM MQ -Anwendung.	Gibt an, ob das Ziel einer Nachricht, die an die Zieladresse (destination) gesendet wird, eine JMS-Anwendung ist. Eine Nachricht, deren Ziel eine JMS-Anwendung ist, enthält einen MQRFH2-Header.

In Tabelle 68 auf Seite 513 werden die Eigenschaften aufgelistet, die sich speziell auf ein Queue-Objekt beziehen.

Tabelle 68. Eigenschaften, die sich speziell auf ein Queue-Objekt (Warteschlangenobjekt) beziehen

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
baseQueueManagerName	Zeichenfolge	• **"" (leere Zeichenfolge)** • Name eines Warteschlangenmanagers	Der Name des Warteschlangenmanagers, der Eigner der zugrunde liegenden IBM MQ-Warteschlange ist.
baseQueueName	Zeichenfolge	• **"" (leere Zeichenfolge)** • Ein Warteschlangenname	Der Name der zugrunde liegenden IBM MQ-Warteschlange.

In Tabelle 69 auf Seite 514 werden die Eigenschaften aufgelistet, die sich speziell auf ein Topic-Objekt beziehen.

Tabelle 69. Eigenschaften, die sich speziell auf ein Topic-Objekt (Themenobjekt) beziehen

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
baseTopicName	Zeichenfolge	• **"" (leere Zeichenfolge)** • Ein Themenname	Der Name des zugrunde liegenden Themas.
brokerCCDurSubQueue >	Zeichenfolge	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Verbindungskonsument permanente Subskriptionsnachrichten empfängt.
brokerDurSubQueue	Zeichenfolge	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein permanenter Topic-Subskribent die Nachrichten empfängt. Weitere Informationen finden Sie unter der Beschreibung der Eigenschaft **BROKEDURRSUBQ** in der Dokumentation zu IBM MQ Explorer.
brokerPub-Warteschlange	Zeichenfolge	• **Nicht gesetzt** • Ein Warteschlangenname	Der Name der Warteschlange, an die veröffentlichte Nachrichten gesendet werden (die Datenstromwarteschlange). Der Wert dieser Eigenschaft setzt den Wert der Eigenschaft **brokerPubQueue** des ConnectionFactory-Objekts außer Kraft. Legen Sie den Wert dieser Eigenschaft hingegen nicht fest, wird stattdessen der Wert der Eigenschaft **brokerPubQueue** des ConnectionFactory-Objekts verwendet.
brokerPubQueueManager	Zeichenfolge	• **"" (leere Zeichenfolge)** • Name eines Warteschlangenmanagers	Der Name des Warteschlangenmanagers, in dem die Warteschlange definiert ist, an die die zu diesem Thema veröffentlichten Nachrichten gesendet werden.

Tabelle 69. Eigenschaften, die sich speziell auf ein Topic-Objekt (Themenobjekt) beziehen (Forts.)			
Name der Eigenschaft	**Typ**	**Gültige Werte (der Standardwert ist fett dargestellt)**	**Beschreibung**
brokerVersion	Zeichenfolge	• **Nicht gesetzt** • 1 • 2	Die Version des verwendeten Brokers. Der Wert dieser Eigenschaft setzt den Wert der Eigenschaft **brokerVersion** des ConnectionFactory-Objekts außer Kraft. Legen Sie den Wert dieser Eigenschaft hingegen nicht fest, wird stattdessen der Wert der Eigenschaft **brokerVersion** des ConnectionFactory-Objekts verwendet.

Im folgenden Beispiel sehen Sie eine Gruppe von Eigenschaften eines Queue-Objekts:

```
expiry:               UNLIM
persistence:          QDEF
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

Im folgenden Beispiel sehen Sie eine Gruppe von Eigenschaften eines Topic-Objekts:

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

Zugehörige Tasks

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

JMS-Ressourcen in WebSphere Application Server konfigurieren

Zugehörige Verweise

Federal Information Processing Standards (FIPS) für AIX, Linux, and Windows

Eigenschaft targetClientMatching für eine Aktivierungsspezifikation konfigurieren

Sie können die Eigenschaft **targetClientMatching** für eine Aktivierungsspezifikation konfigurieren, sodass ein MQRFH2-Header in Antwortnachrichten eingeschlossen wird, wenn Anforderungsnachrichten keinen MQRFH2-Header enthalten. Dies bedeutet, dass alle Nachrichteneigenschaften, die eine Anwendung in einer Antwortnachricht definiert, beim Senden der Nachricht eingeschlossen werden.

Informationen zu diesem Vorgang

Wenn eine Message-driven Bean-(MDB-)Anwendung Nachrichten, die keinen MQRFH2-Header enthalten, über eine IBM MQ-JCA-Ressourcenadapter-Aktivierungsspezifikation verarbeitet und nachfolgend Antwortnachrichten an das aus dem Feld JMSReplyTo der Anforderungsnachricht erstellte JMS-Ziel sendet, müssen die Antwortnachrichten einen MQRFH2-Header einschließen, selbst wenn die Anforderungsnachrichten keinen enthalten. Andernfalls gehen alle Nachrichteneigenschaften, die die Anwendung in einer Antwortnachricht definiert hat, verloren.

Die Eigenschaft **targetClientMatching** definiert, ob eine Antwortnachricht, die an die Warteschlange gesendet wurde, die im Headerfeld JMSReplyTo einer eingehenden Nachricht angegeben ist, nur dann einen MQRFH2-Header aufweist, wenn die eingehende Nachricht über einen MQRFH2-Header verfügt. Sie können diese Eigenschaft sowohl in WebSphere Application Server traditional als auch in WebSphere Liberty für eine Aktivierungsspezifikation konfigurieren.

Wenn Sie den Wert der Eigenschaft **targetClientMatching** auf `false` setzen, kann ein MQRFH2-Header in eine Antwortnachricht eingeschlossen werden, die an ein JMS-Ziel gesendet wird, das aus dem JMSReplyTo-Header einer eingehenden Anforderungsnachricht erstellt wird, die keinen MQRFH2-Header enthält. Dies liegt daran, dass die Eigenschaft **targetClient** im JMS-Ziel auf den Wert `0` gesetzt ist, was bedeutet, dass Nachrichten einen MQRFH2-Header enthalten. Das Vorhandensein des MQRFH2-Headers in der abgehenden Nachricht ermöglicht das Speichern benutzerdefinierter Nachrichteneigenschaften in der Nachricht, wenn diese an die IBM MQ-Warteschlange gesendet wird.

Wenn die Eigenschaft **targetClientMatching** auf `true` gesetzt ist und eine Anforderungsnachricht keinen MQRFH2-Header enthält, wird kein MQRFH2-Header in die Antwortnachricht eingeschlossen.

Prozedur

- Definieren Sie in WebSphere Application Server traditional über die Administrationskonsole die Eigenschaft **targetClientMatching** als benutzerdefinierte Eigenschaft für die IBM MQ-Aktivierungsspezifikation:
  a) Klicken Sie im Navigationsbereich auf **Resources -> JMS -> Activation specifications** (Ressourcen > JMS > Aktivierungsspezifikationen).
  b) Wählen Sie den Namen der Aktivierungsspezifikation aus, die Sie anzeigen oder ändern möchten.
  c) Klicken Sie auf **Custom properties -> New** (Benutzerdefinierte Eigenschaften > Neu) und geben Sie dann die Details der neuen benutzerdefinierten Eigenschaft ein.

  Geben Sie `targetClientMatching` als Name und `java.lang.Boolean` als Typ der Eigenschaft an und setzen Sie den Wert auf `false`.
- Geben Sie in WebSphere Liberty die Eigenschaft **targetClientMatching** in der Definition einer Aktivierungsspezifikation in der Datei `server.xml` an.

  For example:

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

Zugehörige Konzepte

„Ziele in einer JMS-Anwendung erstellen” auf Seite 231
Anstelle des Abrufs von Zielen als verwaltete Objekte aus einem JNDI-Namensbereich (JNDI = Java Naming and Directory Interface) kann eine JMS-Anwendung eine Sitzung für die dynamische Erstellung von Zielen zur Laufzeit verwenden. Eine Anwendung kann einen Uniform Resource Identifier (URI) zur Identifizierung einer Warteschlange oder eines Themas von IBM MQ und optional zur Angabe einer oder mehrerer Eigenschaften eines Queue- oder Topic-Objekts verwenden.

„Ressourcenadapter für abgehende Kommunikation konfigurieren” auf Seite 494
Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

Message-driven Bean von IBM MQ in WebSphere Liberty angehalten

Die Eigenschaft **maxSequentialDeliveryFailures** für eine Aktivierungsspezifikation legt fest, wie viele aufeinanderfolgende Fehler bei der Nachrichtenübermittlung an eine Message-driven Bean-Instance (MDB-Instance) der Ressourcenadapter maximal toleriert, bevor die MDB angehalten wird.

Vorbereitende Schritte

Sie müssen wissen, welche Ereignisse dazu führen könnten, dass eine MDB in WebSphere Liberty angehalten wird. Der Ressourcenadapter sieht folgende Fälle als Nachrichtenübermittlungsfehler an:

- Eine ungeprüfte Ausnahmebedingung, die von der Methode **onMessage** der MDB ausgelöst wird.
- Eine JMS-Ausnahmebedingung (`JMSException`) bei der Verarbeitung des Ressourcenadapters vor der Übermittlung der Nachricht an die MDB.

- Eine JMS-Ausnahmebedingung (`JMSException`) bei der Verarbeitung des Ressourcenadapters nach der Übermittlung der Nachricht an die MDB.
- Wenn die XA-Transaktion oder lokale Transaktion, die zum Lesen der Nachricht verwendet wird, rückgängig gemacht wird.
- Wenn im Anwendungsserver kein Thread für die Übermittlung der Nachricht an die MDB verfügbar ist.

Informationen zu diesem Vorgang

Der Standardwert der Eigenschaft **maxSequentialDeliveryFailures** lautet *-1*. Bei dieser Einstellung wird die MDB nie angehalten. Alle anderen negativen Werte werden wie *-1* behandelt. Der Wert:

- *0* bedeutet, dass die MDB beim ersten Fehler angehalten wird.
- *1* bedeutet, dass die MDB bei zwei aufeinanderfolgenden Fehlern angehalten wird.
- *2* bedeutet, dass die MDB bei drei aufeinanderfolgenden Fehlern angehalten wird, usw.

Sie können diese Eigenschaft nur in WebSphere Liberty und nur für die Liberty-Version 18.0.0.4 oder höher für eine Aktivierungsspezifikation konfigurieren.

Achtung: Wenn Sie für das Attribut in einer Anwendungsserverumgebung, bei der es sich nicht um Liberty handelt, einen anderen Wert als den Standardwert festlegen, wird der Wert ignoriert und ein Warnhinweis in das Protokoll geschrieben.

Es ist außerdem möglich, den IBM MQ-Ressourcenadapter in WebSphere Liberty als generischen Ressourcenadapter zu installieren. In diesem Fall werden alle Integrationsfähigkeiten von IBM MQ und WebSphere Application Server inaktiviert und der Ressourcenadapter kann nicht erkennen, dass er in Liberty ausgeführt wird. Daher wird für die Einstellung **maxSequentialDeliveryFailures** kein Wert größer/gleich *0* unterstützt. Bei Angabe eines solchen Wertes wird ein Warnhinweis im Protokoll ausgegeben.

Prozedur

- Geben Sie in WebSphere Liberty die Eigenschaft **maxSequentialDeliveryFailures** bei der Definition einer Aktivierungsspezifikation in der Datei `server.xml` an.

  For example:

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

Zugehörige Konzepte

„Ressourcenadapter für abgehende Kommunikation konfigurieren” auf Seite 494
Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

Installation des Ressourcenadapters überprüfen

Das Programm des Installationsprüftests (IVT) für den IBM MQ-Ressourcenadapter wird als EAR-Datei bereitgestellt. Zur Verwendung des Programms müssen Sie es implementieren und einige Objekte als JCA-Ressourcen definieren.

Informationen zu diesem Vorgang

Das IVT-Programm (Installation Verification Test, Installationsprüftest) wird als EAR-Datei mit dem Namen `wmq.jakarta.jmsra.ivt.ear` (Jakarta Messaging 3.0) oder `wmq.jmsra.ivt.ear` (JMS 2.0) bereitgestellt. Diese Datei wird mit IBM MQ classes for JMS in demselben Verzeichnis installiert wie die RAR-Datei des IBM MQ -Ressourcenadapters `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) oder

`wmq.jmsra.rar` (JMS 2.0). Informationen zur Position, an der diese Dateien installiert werden, finden Sie unter „IBM MQ-Ressourcenadapter installieren” auf Seite 466.

Sie müssen das IVT-Programm auf Ihrem Anwendungsserver implementieren. Das IVT-Programm (Programm des Installationsprüftests) enthält ein Servlet und eine MDB, die testet, ob eine Nachricht an eine IBM MQ-Warteschlange gesendet bzw. aus dieser empfangen werden kann. Mit dem IVT-Programm können Sie prüfen, ob der IBM MQ-Ressourcenadapter ordnungsgemäß für die Unterstützung der dezentralen Transaktionsverarbeitung konfiguriert wurde. Wenn Sie den IBM MQ-Ressourcenadapter auf einem nicht von IBM bereitgestellten Anwendungsserver implementieren, fordert Sie der IBM Service unter Umständen auf, mit dem IVT zu überprüfen, ob der Anwendungsserver korrekt konfiguriert ist.

Als Voraussetzung für die Ausführung des IVT-Programms müssen Sie ein ConnectionFactory-Objekt, ein Queue-Objekt und möglicherweise auch ein Aktivierungsspezifikationsobjekt (Activation Specification) als JCA-Ressourcen definieren. Zudem müssen Sie sicherstellen, dass Ihr Anwendungsserver JMS-Objekte aus diesen Definitionen erstellt und diese an einen JNDI-Namensbereich bindet. Die Eigenschaften dieser Objekte können Sie den Host- und Porteinstellungen Ihres eigenen Warteschlangenmanagers angleichen. Nachfolgend sehen Sie ein einfaches Beispiel für diese Eigenschaften:

```
ConnectionFactory object:
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1550
queueManager:         QM1
transportType:        CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:        TEST.QUEUE
```

Das Verfahren zum Definieren der Objekte 'ConnectionFactory', 'Queue' und 'Activation Specification' variiert abhängig von Ihrem Anwendungsserver. Wenn Sie diese Eigenschaften beispielsweise in WebSphere Liberty festlegen, fügen Sie der Datei `server.xml` des Anwendungsservers die folgenden Einträge hinzu:

JMS 3.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550" transport
Type="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

JMS 2.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550" transport
Type="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
```

```
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

Das IVT-Programm erwartet standardmäßig, dass ein ConnectionFactory-Objekt im JNDI-Namensbereich mit dem Namen 'jms/ivt/IVTCF' gebunden wurde. Für ein Queue-Objekt wird der Bindungsname 'jms/ivt/IVTQueue' erwartet. Sie können auch andere Namen verwenden. In diesem Fall müssen Sie jedoch die Namen der Objekte auf der ersten Seite des IVT-Programms eingeben und die EAR-Datei entsprechend ändern.

Nachdem Sie das IVT-Programm implementiert haben und der Anwendungsserver die JMS-Objekte erstellt und an den JNDI-Namensbereich gebunden hat, können Sie das IVT-Programm mit den folgenden Schritten starten.

Vorgehensweise

1. Starten Sie das IVT-Programm durch Eingabe einer URL im folgenden Format in Ihren Webbrowser:

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

   Dabei steht *Host_des_Anwendungsservers* für die IP-Adresse oder den Hostnamen des Systems, auf dem Ihr Anwendungsserver ausgeführt wird. *Port* ist die Nummer des TCP-Ports, an dem der Anwendungsserver empfangsbereit ist. Beispiel:

   ```
   http://localhost:9080/WMQ_IVT/
   ```

   Im Folgenden sehen Sie ein Beispiel für die erste Seite, die vom IVT-Programm angezeigt wird.

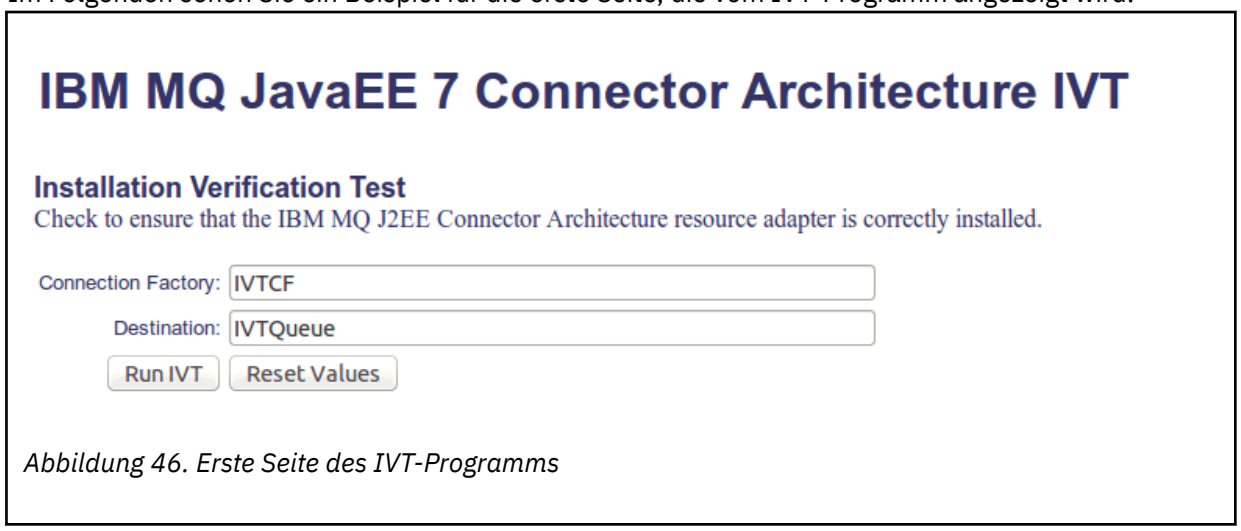
   

   Abbildung 46. Erste Seite des IVT-Programms

2. Wenn Sie den Test ausführen möchten, klicken Sie auf **Run IVT** (IVT ausführen).

   Im Folgenden sehen Sie ein Beispiel für die Seite, die angezeigt wird, wenn der IVT erfolgreich ist.

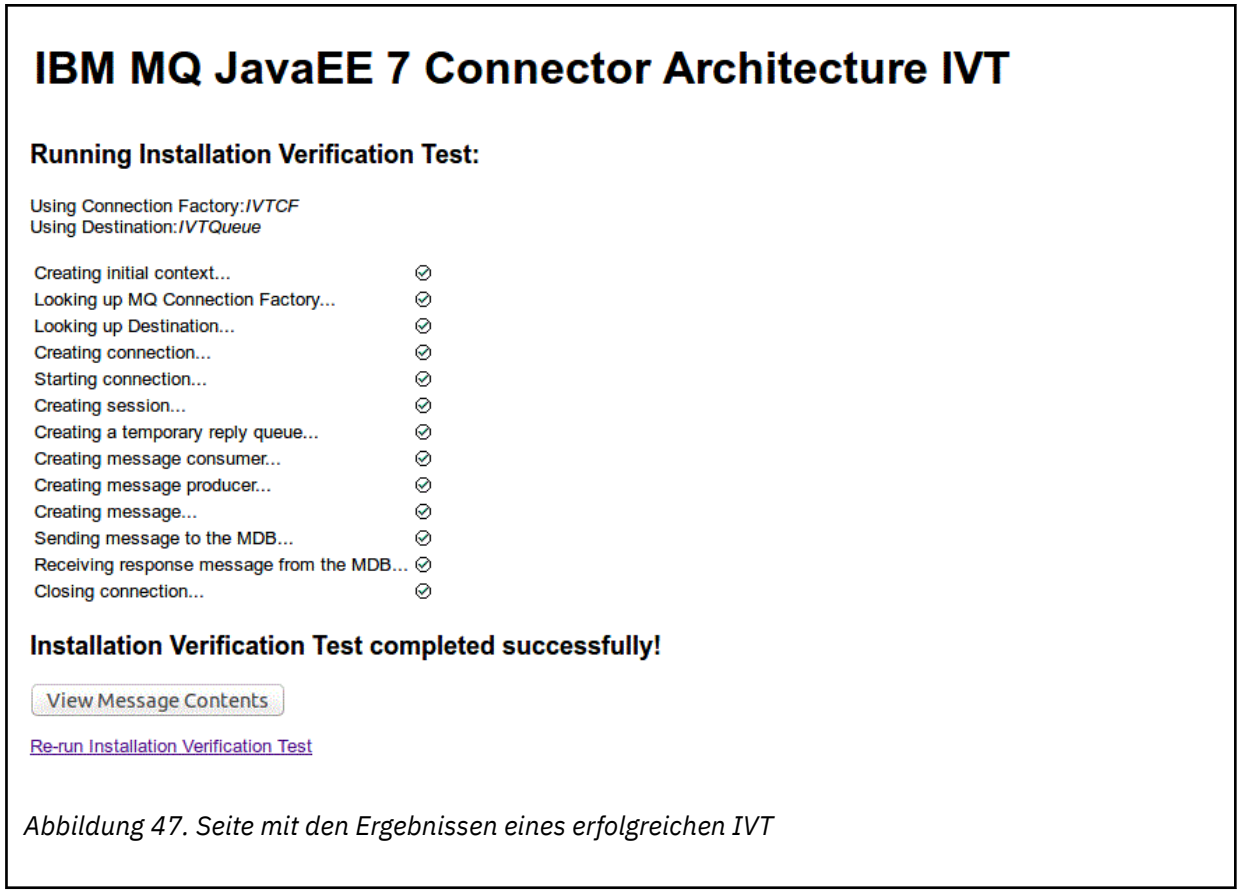


Abbildung 47. Seite mit den Ergebnissen eines erfolgreichen IVT

Im Folgenden sehen Sie ein Beispiel für die Seite, die angezeigt wird, wenn das IVT fehlschlägt. Wenn Sie weitere Informationen zur Fehlerursache erhalten möchten, klicken Sie auf **View Stack Trace** (Stack-Trace anzeigen).

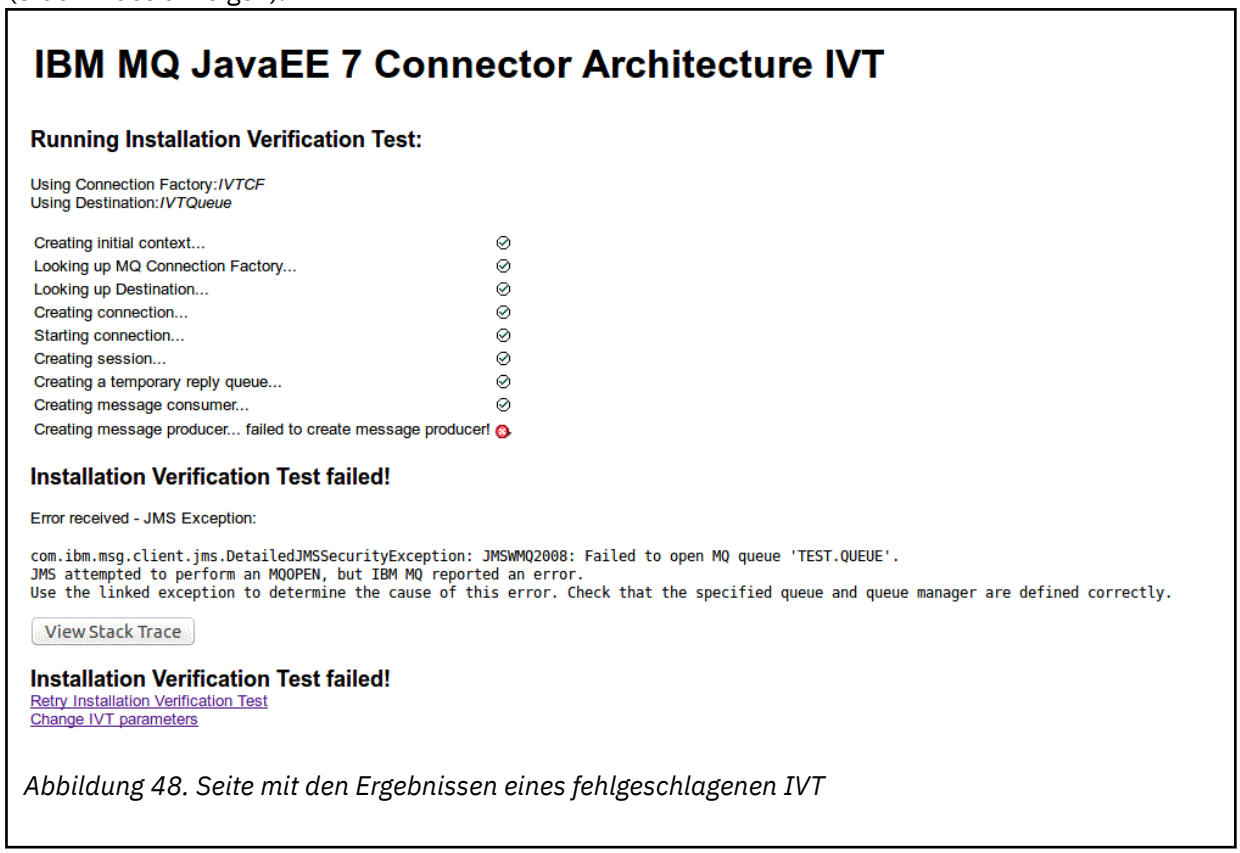


Abbildung 48. Seite mit den Ergebnissen eines fehlgeschlagenen IVT

Windows Ressourcenadapter in GlassFish Server installieren und testen

Um den IBM MQ-Ressourcenadapter in GlassFish Server unter einem Windows-Betriebssystem zu installieren, müssen Sie zunächst eine Domäne erstellen und starten. Danach können Sie den Ressourcenadapter implementieren und konfigurieren und die IVT-Anwendung (Installation Verification Test = Installationsprüftest) implementieren und ausführen.

Vorbereitende Schritte

- Diese Anweisungen beziehen sich auf GlassFish Server Version 4.
- Diese Version von GlassFish Server unterstützt Jakarta EEnicht.

Informationen zu diesem Vorgang

Bei dieser Task wird vorausgesetzt, dass ein GlassFish Server-Anwendungsserver ausgeführt wird und Sie mit den diesbezüglichen Standardverwaltungsaufgaben vertraut sind. Darüber hinaus wird vorausgesetzt, dass Sie über eine IBM MQ-Installation auf Ihrem lokalen System verfügen und mit den diesbezüglichen Standardverwaltungsaufgaben vertraut sind.

Anmerkung: Zur Ausführung der folgenden Aufgabenschritte benötigen Sie eine funktionierende IBM MQ-Installation, in der folgende Objekte konfiguriert sind:

- Einen Warteschlangenmanager mit der Bezeichnung QM, der an Port 1414 gestartet wird, den Kanal SYSTEM.DEF.SVRCONN verwendet und über das Client-Transportprotokoll eine Verbindung herstellt.
- Eine Warteschlange mit dem Namen Q1.

Vorgehensweise

1. Starten Sie das GlassFish Server-Shellprogramm **asadmin**.
   a) Öffnen Sie die Windows-Befehlszeile und navigieren Sie zum Verzeichnis *GlassFish*/bin; *GlassFish* ist hierbei das Installationsverzeichnis von GlassFish Server Version 4.
   b) Geben Sie den Befehl **asadmin** in die Befehlszeile ein.
      Mit dem Befehl **asadmin** wird in der Befehlszeile ein Shellprogramm geöffnet, mit dem Sie eine neue Domäne erstellen können.

   GlassFish Server Version 4 wird auf Ihrem System gestartet.
2. Erstellen und starten Sie eine Domäne.
   a) Verwenden Sie den Befehl **create-domain** mit Angabe von Port und Domänenname, um eine neue Domäne zu erstellen. Geben Sie folgenden Befehl in der Befehlszeile ein:

      ```
      create-domain --adminport port domain_name
      ```

      Dabei steht *Port* für die Portnummer und *Domänenname* für den Namen, den die Domäne verwenden soll.

      Anmerkung: Zum Befehl **create-domain** gibt es viele optionale Parameter. Für diesen Vorgang wird jedoch nur der Parameter `--adminport` benötigt. Weitere Informationen finden Sie in der Produktdokumentation für GlassFish Server Version 4.

      Wenn der von Ihnen angegebene Port belegt ist, wird folgende Nachricht angezeigt:

      `Port für Domänenname Port ist belegt`

      Wenn der von Ihnen angegebene Domänenname bereits verwendet wird, erhalten Sie eine entsprechende Nachricht sowie eine Liste aller Domänennamen, die derzeit nicht verfügbar sind.
   b) Geben Sie auf die Aufforderung, einen Benutzernamen und ein Kennwort einzugeben, die Berechtigungsnachweise für die Anmeldung beim Anwendungsserver über einen Web-Browser ein.

Wenn der Befehl erfolgreich ausgeführt wird, wird eine Nachricht mit einer Zusammenfassung der Domänenerstellung in der Befehlszeile angezeigt, einschließlich der Nachricht `Command create-domain executed successfully.`

Sie haben erfolgreich eine Domäne erstellt.

c) Geben Sie den folgenden Befehl in die Befehlszeile ein, um die Domäne zu starten:

```
start-domain domain_name
```

Dabei steht *Domänenname* für den Domänennamen, den Sie zuvor angegeben haben.

3. Greifen Sie über einen Web-Browser auf den GlassFish-Anwendungsserver zu.
   a) Geben Sie in die Adresszeile eines Web-Browsers folgenden Befehl ein:

```
localhost:port
```

Port ist dabei der Port, den Sie zuvor beim Erstellen der Domäne angegeben haben.

Die GlassFish-Konsole wird angezeigt.

   b) Wenn die GlassFish-Konsole geladen ist und Sie nach einem Benutzernamen und Kennwort gefragt werden, geben Sie die in Schritt 2b angegebenen Berechtigungsnachweise ein.
4. Laden Sie den Ressourcenadapter auf GlassFish Server 4 hoch.
   a) Wählen Sie in der Symbolleiste **Common Tasks** (Allgemeine Tasks) den Menüpunkt **Applications** (Anwendungen) aus, um die Seite **Applications** (Anwendungen) anzuzeigen.
   b) Klicken Sie auf die Schaltfläche **Deploy** (Implementieren), um die Seite **Deploy Applications or Modules** (Anwendungen oder Module implementieren) zu öffnen.
   c) Klicken Sie auf die Schaltfläche **Durchsuchen** und navigieren Sie zur Position der Datei `wmq.jmsra.rar` . Wählen Sie die Datei aus und klicken Sie auf **OK**.
5. Erstellen Sie einen Verbindungspool.
   a) Wählen Sie in der Symbolleiste unter **Resources** (Ressourcen) den Menüpunkt **Connectors** aus.
   b) Wählen Sie anschließend den Menüpunkt **Connector Connection Pools** (Connectorverbindungspools) aus, um die Seite **Connector Connection Pools** (Connectorverbindungspools) zu öffnen.
   c) Klicken Sie auf **New** (Neu), um die Seite **New Connector Connection Pool (Step 1 of 2)** (Neuer Connectorverbindungspool (Schritt 1 von 2)) zu öffnen.
   d) Geben Sie auf der Seite **New Connector Connection Pool (Step 1 of 2)** (Neuer Connectorverbindungspool (Schritt 1 von 2)) den Poolnamen als jms/ivt/IVTCF-Connection-Pool in das Feld **Pool Name** (Poolname) ein.
   e) Wählen Sie im Feld **Resource Adapter** (Ressourcenadapter) `wmq.jmsra` aus.
   f) Geben Sie im Feld **Verbindungsdefinition** den Wert `javax.jms.ConnectionFactory`ein.
   g) Wählen Sie **Next** (Weiter) und anschließend **Finish** (Fertigstellen) aus.
6. Erstellen Sie die Connectorressourcen.
   a) Wählen Sie in der Symbolleiste im Menü **Connectors** die Option **Connector Resource** (Connectorressource) aus, um die Seite **Connector Resources** (Connectorressourcen) zu öffnen.
   b) Wählen Sie **New** (Neu) aus, um die Seite **New Connector Resource** (Neue Connectorressource) zu öffnen.
   c) Geben Sie im Feld **JNDI Name** den Namen `IVTCF` ein.
   d) Geben Sie im Feld **Pool Name** den Namen `jms/ivt/IVTCF-Connection-Pool` ein.
   e) Lassen Sie alle anderen Felder frei.
   f) Klicken Sie für die folgenden Eigenschaft/Wert-Paare jeweils auf **Add Property** (Eigenschaft hinzufügen) und geben Sie den Eigenschaftsnamen und den Wert wie im folgenden Beispiel gezeigt ein:
      - name: host; value: localhost

- name: port; value 1414
- name: channel; value: SYSTEM.DEF.SVRCONN
- name: queueManager; value: QM
- name: transportType; value: CLIENT

Anmerkung: Vergewissern Sie sich, dass Sie die richtigen Werte für Ihre Konfigurationseinstellungen verwenden, die von den Angaben in diesem Beispiel abweichen können.

g) Wählen Sie in der Symbolleiste unter **Connectors** den Menüpunkt **Admin Object Resources** (Verwaltungsobjektressourcen) aus, um die Seite **Admin Object Resources** (Verwaltungsobjektressourcen) zu öffnen.

h) Klicken Sie auf der Seite **Admin Object Resources** (Verwaltungsobjektressourcen) auf **New** (Neu), um die Seite **New Admin Object Resource** (Neue Verwaltungsobjektressource) zu öffnen.

i) Geben Sie im Feld **JNDI Name** den Namen `IVTQueue` ein.

j) Geben Sie im Feld **Resource Adapter** (Ressourcenadapter) `wmq.jmsra` ein.

k) Geben Sie im Feld **Resource Type** (Ressourcentyp) `javax.jms.Queue` ein.

l) Lassen Sie das Feld **Class Name** (Klassenname) unverändert.

m) Klicken Sie für die folgenden Eigenschaft/Wert-Paare jeweils auf **Add Property** (Eigenschaft hinzufügen) und geben Sie den Eigenschaftsnamen und den Wert wie im folgenden Beispiel gezeigt ein:

- name: name; value: IVTQueue
- name: baseQueueManagerName; value QM
- name: baseQueueName; value: Q1

Anmerkung: Vergewissern Sie sich, dass Sie die richtigen Werte für Ihre Konfigurationseinstellungen verwenden, die von den Angaben in diesem Beispiel abweichen können.

n) Klicken Sie auf **OK**.

o) Markieren Sie das Kontrollkästchen **Enabled** (Aktiviert) und klicken Sie dann auf **Enable** (Aktivieren).

7. Implementieren Sie die EAR-Datei `wmq.jmsra.ivt.ear` in GlassFish Server.

a) Klicken Sie in der Symbolleiste auf **Applications** (Anwendungen), um die Seite **Applications** (Anwendungen) aufzurufen.

b) Klicken Sie auf **Deploy** (Implementieren), um die IVT-Anwendung hinzuzufügen.

c) Navigieren Sie im Feld **Location** (Speicherposition) zur Datei `wmq.jmsra.ivt.ear` und wählen Sie die Datei aus.

d) Wählen Sie im Feld **Virtual Servers** (Virtuelle Server) die Option **server** aus und klicken Sie anschließend auf **OK**.

8. Starten Sie das IVT-Programm.

a) Klicken Sie in der Symbolleiste auf **Applications** (Anwendungen), um die Seite **Applications** (Anwendungen) aufzurufen.

b) Klicken Sie in der Tabelle 'Deployed Applications' (Implementierte Anwendungen) auf die Datei `wmq.jmsra.ivt`.

c) Klicken Sie in der Tabelle 'Modules and Components' (Module und Komponenten) auf die Schaltfläche **Launch** (Starten).

d) Wählen Sie den Link http: aus.

e) Klicken Sie auf **Run IVT** (IVT ausführen).

Sie haben das IVT-Programm gestartet, und wenn der Vorgang erfolgreich war, wird die folgende Ausgabe angezeigt:

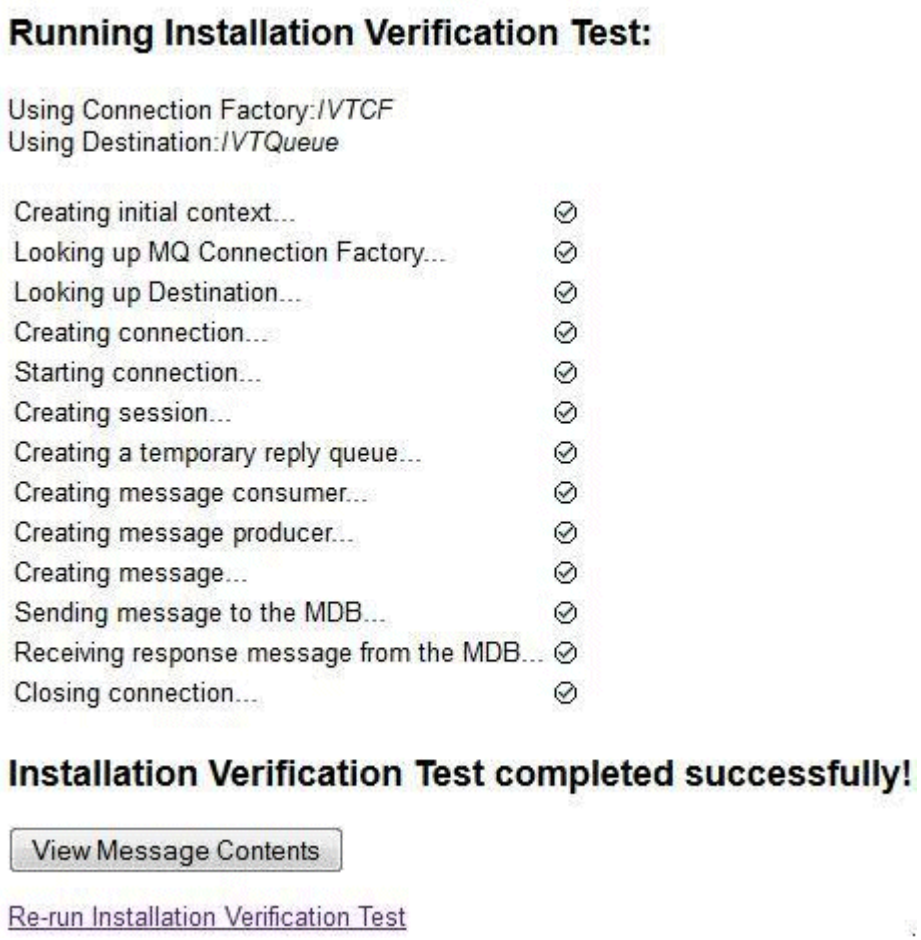


Abbildung 49. Erfolgreiche IVT-Ausgabe

Ressourcenadapter in Wildfly installieren und testen

Wenn Sie den IBM MQ-Ressourcenadapter in Wildfly Version 10 installieren, müssen Sie zuerst einige Konfigurationsdateiänderungen vornehmen, um eine Subsystemdefinition für den IBM MQ-Ressourcenadapter hinzuzufügen. Danach können Sie den Ressourcenadapter implementieren und testen, indem Sie die IVT-Anwendung (Installation Verification Test = Installationsprüftest) installieren und ausführen.

Vorbereitende Schritte

- Diese Anweisungen gelten für Wildfly V10.
- Diese Version von WildFly unterstützt Jakarta EEnicht.

Informationen zu diesem Vorgang

Bei dieser Task wird vorausgesetzt, dass ein WildFly-Anwendungsserver ausgeführt wird und Sie mit den Standardverwaltungsaufgaben für diesen Server vertraut sind. Darüber hinaus wird vorausgesetzt, dass Sie über eine IBM MQ-Installation verfügen und mit Standardverwaltungsaufgaben vertraut sind.

Vorgehensweise

1. Erstellen Sie einen IBM MQ-Warteschlangenmanager mit dem Namen ExampleQM und konfigurieren Sie ihn, wie im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123 beschrieben.

   Beachten Sie bei der Konfiguration des Warteschlangenmanagers folgende Punkte:

   - Der Listener muss an Port 1414 gestartet werden.

- Der zu verwendende Kanal heißt SYSTEM.DEF.SVRCONN.
- Die von der IVT-Anwendung verwendete Warteschlange hat den Namen TEST.QUEUE.

Der Modellwarteschlange SYSTEM.DEFAULT.MODEL.QUEUE muss außerdem DSP- und PUT-Berechtigung erteilt werden, damit diese Anwendung eine temporäre Antwortwarteschlange erstellen kann.

2. Bearbeiten Sie die Konfigurationsdatei *WildFly_Home*/standalone/configuration/standalone-full.xml und fügen Sie das folgende Subsystem hinzu:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-name="com.ibm.mq.connector.outbound.ManagedCon
nectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-name="com.ibm.mq.connector.outbound.ManagedCon
nectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                          jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                <config-property name="baseQueueName">
                    TEST.QUEUE
                </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

3. Implementieren Sie den Ressourcenadapter auf Ihrem Server, indem Sie die Datei wmq.jmsra.rar in das Verzeichnis *WildFly_Home*/standalone/deployments kopieren.
4. Implementieren Sie die IVT-Anwendung, indem Sie die Datei wmq.jmsra.ivt.ear in das Verzeichnis *WildFly_Home*/standalone/deployments kopieren.

5. Starten Sie den Anwendungsserver, indem Sie eine Eingabeaufforderung aufrufen, zum Verzeichnis *WildFly_Home*/bin navigieren und den folgenden Befehl ausführen:

```
standalone.bat -c standalone-full.xml
```

6. Führen Sie die IVT-Anwendung aus.

   Weitere Informationen finden Sie unter „Installation des Ressourcenadapters überprüfen” auf Seite 517. Die Standard-URL für Wildfly lautet `http://localhost:8080/WMQ_IVT/`.

IBM MQ und WebSphere Application Server gemeinsam verwenden

Über den IBM MQ -Messaging-Provider in WebSphere Application Serverkönnen Java Message Service -Messaging-Anwendungen (JMS) Ihr IBM MQ -System als externen Provider von JMS -Messaging-Ressourcen verwenden.

Informationen zu diesem Vorgang

Anwendungen, die in Java geschrieben und unter WebSphere Application Server ausgeführt werden, können die Spezifikation Java Message Service (JMS) verwenden, um Messaging durchzuführen. Die Nachrichtenübertragung (Messaging) in dieser Umgebung kann von einem IBM MQ-Warteschlangenmanager bereitgestellt werden.

Der Vorteil eines IBM MQ-Warteschlangenmanagers liegt darin, dass verbindende JMS-Anwendungen alle Funktionen eines IBM MQ-Netzes nutzen und somit Nachrichten mit Warteschlangenmanagern austauschen können, die auf den verschiedensten Plattformen aktiv sind.

Anwendungen können entweder den *Clienttransport* oder den *Bindungstransport* für das Warteschlangenverbindungsfactory-Objekt verwenden. Für den Bindungstransport muss der Warteschlangenmanager der Anwendung, die eine Verbindung benötigt, lokal zur Verfügung stehen.

Standardmäßig verwenden JMS-Nachrichten, die in IBM MQ-Warteschlangen stehen, einen MQRFH2-Header zum Speichern einiger der JMS-Nachrichtenheaderinformationen. Viele traditionelle IBM MQ-Anwendungen können Nachrichten mit diesen Headern nicht verarbeiten und erfordern ihre eigenen spezifischen Header, z. B. MQCIH für die CICS-Bridge oder MQWIH für IBM MQ-Workflow-Anwendungen. Weitere Informationen zu diesen besonderen Überlegungen finden Sie unter Mapping von JMS-Nachrichten auf IBM MQ-Nachrichten.

Zugehörige Tasks

JMS-Ressourcen in WebSphere Application Server konfigurieren

Anwendungsserver für die Verwendung der neuesten Wartungsstufe des Ressourcenadapters konfigurieren

WebSphere Application Server mit IBM MQ verwenden

IBM MQ und IBM MQ for z/OS können gemeinsam mit oder alternativ zum Standard-Messaging-Provider von WebSphere Application Server verwendet werden.

Der IBM MQ-Messaging-Provider wird als Teil von WebSphere Application Server installiert. Dazu gehört auch eine Version des Ressourcenadapters von IBM MQ sowie derIBM MQ Extended Transactional Client-Funktionalität, die dem Warteschlangenmanager die Teilnahme an vom Anwendungsserver verwalteten XA-Transaktionen ermöglicht. Mittels des Ressourcenadapters können Message-driven Beans (MDBs) so konfiguriert werden, dass sie entweder Aktivierungsspezifikationen oder Listenerports verwenden.

Damit der Anwendungsserver unterstützt wird, muss das Installationsprüftestprogramm für den IBM MQ-Ressourcenadapter auf dem Anwendungsserver bereitgestellt sein und erfolgreich ausgeführt werden. Nach erfolgreicher Ausführung des Installationsprüftestprogramms für den IBM MQ-Ressourcenadapter kann der IBM MQ-Ressourcenadapter eine Verbindung zu allen unterstützten IBM MQ-Warteschlangenmanagern herstellen.

JMS-Verbindungen von WebSphere Application Server zu IBM MQ

Bevor Sie die Versionen von IBM MQ in Betracht ziehen, die mit WebSphere Application Serververwendet werden können, sollten Sie wissen, wie Java Message Service -Anwendungen (JMS), die im Anwendungsserver ausgeführt werden, eine Verbindung zu IBM MQ -Warteschlangenmanagern herstellen können.

JMS-Anwendungen, die auf die Ressourcen eines IBM MQ-Warteschlangenmanagers zugreifen müssen, können dazu einen der folgenden Transporttypen verwenden:

BINDINGS

Dieser Transporttyp kann verwendet werden, wenn der Anwendungsserver und der Warteschlangenmanager auf dem gleichen System und unter demselben Betriebssystemimage installiert sind. Im BINDINGS-Modus findet die gesamte Kommunikation zwischen den beiden Produkten über Interprozesskommunikation (Inter-Process Communication, IPC) statt.

Der IBM MQ-Messaging-Provider verfügt nicht über die nativen Bibliotheken, die für die Verbindung mit einem IBM MQ-Warteschlangenmanager im BINDINGS-Modus erforderlich sind. Zur Verwendung einer Verbindung im BINDINGS-Modus muss IBM MQ auf dem gleichen System installiert sein wie der Anwendungsserver und der Pfad des Ressourcenadapters auf die nativen Bibliotheken muss auf das IBM MQ-Verzeichnis verweisen, in dem sich diese Bibliotheken befinden. Weitere Informationen finden Sie in der Produktdokumentation zu WebSphere Application Server:

- Informationen zu WebSphere Application Server traditional finden Sie unter IBM MQ-Messaging-Provider mit Informationen zu nativen Bibliotheken konfigurieren.
- Informationen zu WebSphere Liberty finden Sie im Abschnitt JMS-Anwendungen in Liberty für die Verwendung des IBM MQ-Messaging-Providers implementieren.

z/OS Wenn Sie unter z/OS eine WebSphere Application Server-Verbindungsfactory mit einem IBM MQ-Warteschlangenmanager im Bindungsmodus verbinden möchten, müssen Sie die richtigen IBM MQ-Bibliotheken in der WebSphere Application Server-STEPLIB-Verkettung angeben. Weitere Informationen bietet der Abschnitt IBM MQ-Bibliotheken und WebSphere Application Server für z/OS STEPLIB in der Produktdokumentation zu WebSphere Application Server.

CLIENT

Beim Client-Transporttyp wird TCP/IP für die Kommunikation zwischen WebSphere Application Server und IBM MQ verwendet. Der CLIENT-Modus wird nicht nur verwendet, wenn sich Anwendungsserver und Warteschlangenmanager auf verschiedenen Systemen befinden, sondern er kann auch verwendet werden, wenn beide auf dem gleichen System und unter demselben Betriebssystemimage installiert sind.

JMS-Anwendungen können als Transporttyp auch BINDINGS_THEN_CLIENT angeben. Bei diesem Transporttyp versucht die Anwendung zunächst, eine Verbindung zum Warteschlangenmanager im BINDINGS-Modus herzustellen - gelingt dies nicht, wird die Verbindung im CLIENT-Modus versucht.

Ermitteln, welche Version des IBM MQ -Ressourcenadapters in WebSphere Application Server installiert ist

Informationen darüber, welche Version des IBM MQ-Ressourcenadapters in WebSphere Application Server installiert ist, finden Sie in der Technote Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server?.

Mit den folgenden Jython- und JACL-Befehlen können Sie herausfinden, welche Version des Ressourcenadapters zurzeit von WebSphere Application Server verwendet wird:

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

Anmerkung: Nach der Eingabe dieses Befehls müssen Sie zweimal die **Eingabetaste** drücken, damit dieser Befehl ausgeführt wird.

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

Ressourcenadapter aktualisieren

Aktualisierungen an dem mit dem Anwendungsserver installierten Ressourcenadapter von IBM MQ werden in Fixpacks zu WebSphere Application Server bereitgestellt. IBM MQ -Ressourcenadapter wird mit **Ressourcenadapter aktualisieren ...** aktualisiert. Die Funktion in der WebSphere Application Server -Administrationskonsole wird nicht empfohlen, da dies bedeutet, dass Aktualisierungen, die in WebSphere Application Server -Fixpacks bereitgestellt werden, keine Auswirkungen haben.

Variable MQ_INSTALL_ROOT

Ab WebSphere Application Server 7.0 wird MQ_INSTALL_ROOT nur zur Lokalisierung nativer Bibliotheken verwendet. Außerdem wird diese Variable durch jeden auf dem Ressourcenadapter konfigurierten Pfad der nativen Bibliotheken überschrieben.

Verbindung zwischen WebSphere Application Server und IBM MQ herstellen

Achtung:

1. Jede unterstützte Version von WebSphere Application Server kann den im Produktpaket enthaltenen IBM MQ-Ressourcenadapter verwenden, um eine Verbindung zu einer unterstützten Version von IBM MQ herzustellen.
2. Wenn der Bindungsmodus verwendet wird, müssen bestimmte Bibliotheken in WebSphere Application Server mit der Version des Warteschlangenmanagers übereinstimmen, zu dem eine Verbindung hergestellt wird:
   - WebSphere Application Server muss so konfiguriert sein, dass die mit IBM MQ 9.4 bereitgestellten nativen Bibliotheken geladen werden. Weitere Informationen finden Sie unter „Java Native Interface-(JNI-)Bibliotheken konfigurieren” auf Seite 101.
   - z/OS Unter z/OS müssen Sie die richtigen IBM MQ-Bibliotheken in der WebSphere Application Server-STEPLIB-Verkettung angeben.

     Details über die benötigten IBM MQ-Bibliotheken finden Sie unter IBM MQ-Bibliotheken und WebSphere Application Server for z/OS STEPLIB.

     Wenn Sie über Bibliotheken für eine Version von IBM MQ in LINKLIST (LINKLST) verfügen, können Sie eine Verbindung zu einer anderen Version von IBM MQ herstellen, indem Sie die Bibliotheken mit STEPLIB überschreiben.
3. Die Version des IBM MQ-Ressourcenadapters ist unabhängig von den nativen (gemeinsam genutzten) Versionen, die durch die Warteschlangenmanagerinstallation bereitgestellt wird.

   Beispielsweise kann WebSphere Application Server 8.5 mit einem IBM MQ 8.0-Ressourcenadapter weiterhin Bindungsverbindungen zu einem IBM MQ 9.0-Warteschlangenmanager mithilfe der nativen IBM MQ 9.0-Bibliotheken verwalten.

Weitere Informationen finden Sie unter „Unterstützungserklärung für IBM MQ-Ressourcenadapter” auf Seite 459.

Die Transporttypen BINDINGS und CLIENT können verwendet werden, um eine Verbindung zu IBM MQ von einer beliebigen Version von WebSphere Application Serverherzustellen. Für den Transporttyp BINDINGS gelten die folgenden Einschränkungen:

- IBM MQ muss auf derselben Maschine wie der Anwendungsserver installiert sein.
- WebSphere Application Server muss so konfiguriert sein, dass die mit IBM MQ bereitgestellten nativen Bibliotheken geladen werden.
- z/OS Wenn Sie unter z/OS eine WebSphere Application Server-Verbindungsfactory mit einem IBM MQ-Warteschlangenmanager im Bindungsmodus verbinden möchten, müssen Sie die richtigen IBM MQ-Bibliotheken in der WebSphere Application Server-STEPLIB-Verkettung angeben.

In der folgenden Tabelle sind die Versionen von WebSphere Application Server aufgeführt, in denen die Ausführung der einzelnen Versionen des IBM MQ -Ressourcenadapters unterstützt wird.

Tabelle 70. Zuordnung von WebSphere Application Server -Versionen zu IBM MQ -Ressourcenadapterversionen.

Version des IBM MQ-Ressourcenadapters	**In welcher Version von WebSphere Application Server kann diese Version des Ressourcenadapters ausgeführt werden?**
Ab IBM MQ 9.0	Der Ressourcenadapter kann in folgenden Versionen ausgeführt werden: • In jeder Java EE 7-kompatiblen Version von WebSphere Liberty. • WebSphere Application Server traditional 9.0
IBM MQ 8.0	Der Ressourcenadapter kann in jeder Java EE 7-kompatiblen Version von WebSphere Liberty ausgeführt werden. Die Ausführung des IBM MQ 8.0-Ressourcenadapters in WebSphere Application Server traditional wird nicht unterstützt. Der bereits in WebSphere Application Server traditional installierte Ressourcenadapter sollte verwendet werden, um eine Verbindung zu IBM MQ 8.0 -Warteschlangenmanagern herzustellen.

Zugehörige Konzepte

„Unterstützungserklärung für IBM MQ-Ressourcenadapter” auf Seite 459
Der IBM MQ -Ressourcenadapter, den Sie für die Kommunikation zwischen einer Anwendung und einem Warteschlangenmanager verwenden müssen, hängt davon ab, ob Sie die Jakarta Messaging 3.0 -API oder die JMS 2.0 -API verwenden.

Zugehörige Informationen

Systemvoraussetzungen für IBM MQ

Anzahl der zwischen WebSphere Application Server und IBM MQ erstellten TCP/IP-Verbindungen bestimmen

Dank der Funktion für gemeinsame Dialognutzung können mehrere Dialoge die gleiche MQI-Kanalinstanz (auch als TCP/IP-Verbindung bezeichnet) gemeinsam nutzen.

Informationen zu diesem Vorgang

Anwendungen, die in WebSphere Application Server 7 und WebSphere Application Server 8 im Normalmodus des IBM MQ-Messaging-Providers ausgeführt werden, verwenden diese Funktion automatisch. Das bedeutet, dass mehrere Anwendungen, die auf der gleichen Anwendungsserverinstanz ausgeführt werden und eine Verbindung zum gleichen IBM MQ-Warteschlangenmanager herstellen, die gleiche Kanalinstanz gemeinsam nutzen können.

Die Anzahl der Dialoge, die eine Kanalinstanz gemeinsam nutzen können, wird durch die IBM MQ-Kanaleigenschaft **SHARECNV** festgelegt. Der Standardwert dieser Eigenschaft ist für Serververbindungskanäle 10.

Durch Ermittlung der Anzahl der von WebSphere Application Server 7 und WebSphere Application Server 8 erstellten Dialoge können Sie auch herausfinden, wie viele Kanalinstanzen erstellt wurden.

Nähere Informationen zu den Modi des IBM MQ-Messaging-Providers finden Sie im Abschnitt Normalmodus für PROVIDERVERSION.

Zugehörige Konzepte

Gemeinsame Dialognutzung verwenden
In einer Umgebung, in der mehrere gemeinsame Datenaustauschvorgänge zulässig sind, kann für mehrere Datenaustauschvorgänge eine einzige Instanz eines MQI-Kanals gemeinsam genutzt werden.

„TCP/IP-Verbindung in IBM MQ classes for JMS gemeinsam nutzen" auf Seite 332
Mehrere Instanzen eines MQI-Kanals können so festgelegt werden, dass sie eine einzelne TCP/IP-Verbindung gemeinsam nutzen.

JMS-Verbindungsfactorys

In WebSphere Application Server ausgeführte Anwendungen, die zur Erstellung von Verbindungen und Sitzungen eine Verbindungsfactory des IBM MQ-Messaging-Providers verwenden, haben aktive Dialoge für jede aus der Verbindungsfactory erstellte JMS-Verbindung sowie für jede aus einer JMS-Verbindung erstellte JMS-Sitzung.

Ein Dialog für jede aus der Verbindungsfactory erstellte JMS-Verbindung

Jeder JMS-Verbindungsfactory ist ein Verbindungspool zugeordnet, der in zwei Bereiche unterteilt ist, dem freien und dem aktiven Pool. Beide Pools sind zunächst einmal leer.

Wenn eine Anwendung aus einer Verbindungsfactory eine JMS-Verbindung anfordert, ermittelt WebSphere Application Server zunächst, ob der freie Pool bereits eine JMS-Verbindung enthält. Wenn ja, wird diese in den aktiven Pool verschoben und der Anwendung zugeteilt. Andernfalls wird eine neue JMS-Verbindung erstellt, in den aktiven Pool verschoben und der Anwendung übergeben. Die maximale Anzahl von Verbindungen, die aus einer Verbindungsfactory erstellt werden kann, wird durch die Eigenschaft **Maximum connections**des Verbindungspools der Verbindungsfactory angegeben. Der Standardwert dieser Eigenschaft ist 10.

Wenn eine Anwendung eine JMS-Verbindung nicht mehr benötigt und diese schließt, wird die Verbindung wieder aus dem aktiven Pool in den freien Pool verschoben, aus dem sie wieder abgerufen werden kann. Dabei legt die Verbindungspooleigenschaft **Unused timeout** (Zeitlimit für nicht verwendete Verbindungen) fest, wie lange eine JMS-Verbindung im freien Pool verbleiben kann, bevor sie getrennt wird. Der Standardwert für diese Eigenschaft ist 1800 Sekunden (30 Minuten).

Bei der Erstellung einer JMS-Verbindung wird ein Dialog zwischen WebSphere Application Server und IBM MQ gestartet. Der Dialog bleibt aktiv, bis die Verbindung geschlossen wird, wenn der Wert der Eigenschaft **Unused timeout** für den freien Pool überschritten wird.

Ein Dialog für jede aus einer JMS-Verbindung erstellte JMS-Sitzung

Jeder JMS-Verbindung, die aus einer Verbindungsfactory des IBM MQ-Messaging-Providers erstellt wird, ist ein JMS-Sitzungspool zugewiesen. Sitzungspools funktionieren genauso wie Verbindungspools. Die maximale Anzahl JMS -Sitzungen, die aus einer einzelnen JMS -Verbindung erstellt werden kann, wird durch die Eigenschaft **Maximum connections**des Sitzungspools der Verbindungsfactory festgelegt. Der Standardwert dieser Eigenschaft ist 10.

Ein Dialog beginnt, wenn eine JMS -Sitzung zum ersten Mal erstellt wird. Der Dialog bleibt aktiv, bis die JMS -Sitzung geschlossen wird, da er länger im freien Pool verbleibt als der Wert der Eigenschaft **Unused timeout** für den Sitzungspool.

Wert der Eigenschaft SHARECNV berechnen

Die maximale Anzahl an Dialogen zwischen einer Verbindungsfactory und IBM MQ kann wie folgt berechnet werden:

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Die Anzahl der Kanalinstanzen, die für diese Anzahl an Dialogen erstellt wird, kann wie folgt berechnet werden:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Ein aus dieser Rechnung verbleibender Rest kann aufgerundet werden.

Für eine einfache Verbindungsfactory, die den Standardwert für die Eigenschaften des Verbindungspools **Maximum connections** und des Sitzungspools **Maximum connections** verwendet, gilt für diese Verbindungsfactory die folgende maximale Anzahl an Dialogen zwischen WebSphere Application Server und IBM MQ :

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

For example:

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

Wenn diese Verbindungsfactory eine Verbindung mit IBM MQ über einen Kanal herstellt, dessen Eigenschaft **SHARECNV** auf 10 gesetzt ist, berechnet sich die Anzahl der erstellten Kanalinstanzen wie folgt:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the chan
nel being used
```

For example:

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

Aktivierungsspezifikationen

Message-driven Bean-Anwendungen, die für die Verwendung einer Aktivierungsspezifikation konfiguriert sind, halten zum einen Dialoge offen, über die die Aktivierungsspezifikation ein JMS-Ziel überwachen kann, zum anderen auch Dialoge für jede Serversitzung, in der eine Message-driven Bean-Instanz zur Verarbeitung von Nachrichten ausgeführt wird.

Folgende Dialoge sind für Message-driven Bean-Anwendungen aktiv, die für die Verwendung einer Aktivierungsspezifikation konfiguriert sind:

- Ein Dialog für die Aktivierungsspezifikation zur Überwachung eines JMS-Ziels auf zutreffende Nachrichten. Dieser Dialog wird beim Start der Aktivierungsspezifikation gestartet und bleibt bis zu deren Beendigung aktiv.
- Ein Dialog für jede Serversitzung, in der eine Message-driven Bean-Instanz zur Verarbeitung von Nachrichten ausgeführt wird.

Die erweiterte Eigenschaft **Maximum server sessions** der Aktivierungsspezifikation gibt an, wie viele Serversitzungen maximal gleichzeitig für eine bestimmte Aktivierungsspezifikation aktiv sein können. Der

Standardwert dieser Eigenschaft ist 10. Serversitzungen werden nach Bedarf erstellt und nach der durch die erweiterte Eigenschaft **Server session pool timeout** (Zeitlimit für Serversitzungspool) angegebenen Inaktivitätszeit wieder geschlossen. Der Standardwert dieser Eigenschaft ist 300000 Millisekunden (5 Minuten).

Dialoge starten bei der Erstellung einer Serversitzung und werden beim Beenden der Aktivierungsspezifikation oder beim Timeout der Serversitzung beendet.

Das bedeutet, dass die maximale Anzahl an Dialogen mit IBM MQ für eine Aktivierungsspezifikation wie folgt berechnet werden kann:

```
Maximum number of conversations = Maximum server sessions + 1
```

Die Anzahl der Kanalinstanzen, die für diese Anzahl an Dialogen erstellt wird, kann wie folgt berechnet werden:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Ein aus dieser Rechnung verbleibender Rest kann aufgerundet werden.

Bei einer einfachen Aktivierungsspezifikation, die den Standardwert für die Eigenschaft **Maximum server sessions** verwendet, wird die maximale Anzahl der Dialoge, die zwischen WebSphere Application Server und IBM MQ für diese Aktivierungsspezifikation bestehen können, folgendermaßen berechnet:

```
Maximum number of conversations = Maximum server sessions + 1
```

For example:

```
    = 10 + 1
    = 11
```

Wenn diese Aktivierungsspezifikation eine Verbindung mit IBM MQ über einen Kanal herstellt, dessen Eigenschaft **SHARECNV** auf 10 gesetzt ist, berechnet sich die Anzahl der erstellten Kanalinstanzen wie folgt:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

For example:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Im ASF-Modus (Application Server Facilities) ausgeführte Listenerports

Im ASF-Modus ausgeführte und von Message-driven Bean-Anwendungen verwendete Listenerports erstellen für jede Serversitzung Dialoge. Ein Dialog überwacht ein Ziel auf zutreffende Nachrichten und ein weiterer Dialog führt eine Message-driven Bean-Instanz zur Verarbeitung von Nachrichten aus. Die Anzahl der Dialoge für jeden Listenerport berechnet sich aus der maximalen Anzahl an Sitzungen.

Standardmäßig werden Listenerports gemäß Spezifikation 1.1 (die die Mechanismen definiert, die Anwendungsserver zur Erkennung von Nachrichten und zur Zustellung derselben zur Verarbeitung an Message-driven Beans verwenden sollten) im ASF-Modus ausgeführt. Message-driven Bean-Anwendungen, die Listenerports in diesem Standardbetriebsmodus verwenden, erstellen Dialoge:

Ein Dialog für den Listenerport zur Überwachung eines Ziels auf zutreffende Nachrichten
Listenerports sind so konfiguriert, dass sie eine JMS-Verbindungsfactory verwenden. Wenn ein Listenerport startet, wird eine Anforderung nach einer JMS-Verbindung aus dem freien Pool der Verbin-

dungsfactory gestellt. Beim Beenden des Listenerports wird die Verbindung wieder in den freien Pool zurückgestellt. Weitere Informationen zur Verwendung des Verbindungspools und wie sich dieser auf die Anzahl der Dialoge mit IBM MQ auswirkt, finden Sie im Abschnitt „JMS-Verbindungsfactorys" auf Seite 530.

Ein Dialog für jede Serversitzung, in der eine Message-driven Bean-Instanz zur Verarbeitung von Nachrichten ausgeführt wird

Die erweiterte Eigenschaft **Maximum sessions** (Maximale Anzahl an Sitzungen) des Listenerports gibt die maximale Anzahl an Serversitzungen an, die für den Listenerport zur selben Zeit aktiv sein können. Der Standardwert dieser Eigenschaft ist 10. Serversitzungen werden nach Bedarf erstellt; dabei werdenJMS-Sitzungen aus dem Sitzungspool der vom Listenerport verwendeten JMS-Verbindung genommen.

Wenn eine Serversitzung für die durch die angepasste Eigenschaft **SERVER.SESSION.POOL.UNUSED.TIMEOUT** (des Nachrichten-Listener-Service) angegebene Dauer inaktiv war, wird die Sitzung geschlossen, wobei die verwendete JMS-Sitzung in den freien Pool des Sitzungspools zurückgestellt wird. Die JMS-Sitzung bleibt so lange im freien Pool des Sitzungspools, bis sie entweder erneut benötigt wird oder bis sie geschlossen wird, weil sie länger im freien Pool inaktiv war, als die Eigenschaft **Unused timeout** (Zeitlimit für nicht verwendete Sitzungen) des Sitzungspools zulässt.

Weitere Informationen zur Verwendung des Sitzungspools und zur Verwaltung der Dialoge zwischen WebSphere Application Server und IBM MQ finden Sie im Abschnitt „JMS-Verbindungsfactorys" auf Seite 530.

Weitere Informationen zur angepassten Eigenschaft **SERVER.SESSION.POOL.UNUSED.TIMEOUT** des Nachrichten-Listener-Service finden Sie im Abschnitt Serversitzungspools für Listenerports überwachen in der Produktdokumentation zu WebSphere Application Server.

Maximale Anzahl an Dialogen zwischen einem einzelnen Listenerport und IBM MQ berechnen

Die maximale Anzahl an Dialogen zwischen einem einzelnen Listenerport und IBM MQ kann wie folgt berechnet werden:

```
Maximum number of conversations = Maximum sessions + 1
```

Die Anzahl der Kanalinstanzen, die für diese Anzahl an Dialogen erstellt wird, kann wie folgt berechnet werden:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Ein aus dieser Rechnung verbleibender Rest kann aufgerundet werden.

Für einen einfachen Listener-Port, der den Standardwert für die Eigenschaft **Maximum sessions** verwendet, wird die maximale Anzahl der Dialoge, die zwischen WebSphere Application Server und IBM MQ für diesen Listener-Port bestehen können, wie folgt berechnet:

```
Maximum number of conversations = Maximum sessions + 1
```

For example:

```
    = 10 + 1
    = 11
```

Wenn dieser Listenerport eine Verbindung mit IBM MQ über einen Kanal herstellt, dessen Eigenschaft **SHARECNV** auf 10 gesetzt ist, berechnet sich die Anzahl der erstellten Kanalinstanzen wie folgt:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

For example:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Listenerports, die nicht im ASF-Modus (Application Server Facilities) ausgeführt werden

Listenerports, die nicht im ASF-Modus ausgeführt werden, können so konfiguriert werden, dass sie das Warteschlangenziel und das Themenziel über Serversitzungen überwachen. Serversitzungen können mehrere Dialoge aufweisen, deren maximale Anzahl von Fall zu Fall berechnet werden kann.

Wenn Listenerports so konfiguriert sind, dass sie nicht im ASF-Modus ausgeführt werden, ändert sich die Art und Weise, wie sie JMS-Ziele überwachen. Message-driven Bean-Anwendungen, die Listenerports im Nicht-ASF-Betriebsmodus verwenden, erstellen einen Dialog für jede Serversitzung, in der eine Message-driven Bean-Instanz zur Verarbeitung von Nachrichten ausgeführt wird. Die erweiterte Eigenschaft **Maximum sessions** (Maximale Anzahl an Sitzungen) des Listenerports gibt die maximale Anzahl an Serversitzungen an, die für den Listenerport zur selben Zeit aktiv sein können. Der Standardwert dieser Eigenschaft ist 10.

Bei Ausführung im Nicht-ASF-Modus erstellt ein Listenerport, der ein Warteschlangenziel überwacht, automatisch die durch die Eigenschaft **Maximum Sessions** (Maximale Anzahl an Sitzungen) des Listenerports angegebene Anzahl an Serversitzungen. Dabei verwenden alle diese Serversitzungen JMS-Sitzungen aus dem Sitzungspool der vom Listenerport verwendeten JMS-Verbindung, um über diese Sitzungen kontinuierlich ein JMS-Ziel auf zutreffende Nachrichten zu überwachen.

Wenn der Listenerport für die Überwachung eines Themenziels konfiguriert ist, wird der Wert von **Maximum sessions** (Maximale Anzahl an Sitzungen) ignoriert und eine einzelne Serversitzung verwendet.

Die von einem im Nicht-ASF-Modus ausgeführten Listenerport verwendete Serversitzung bleibt so lange aktiv, bis der Listenerport gestoppt wird, in welchem Fall auch die JMS-Sitzungen, die vom Listenerport für die JMS-Verbindung verwendet wurden, wieder in den freien Bereich des Sitzungspools zurückgestellt werden.

Weitere Informationen zur Verwendung des Sitzungspools und zur Verwaltung der Dialoge zwischen WebSphere Application Server und IBM MQ finden Sie im Abschnitt „JMS-Verbindungsfactorys” auf Seite 530.

Weitere Informationen zum ASF-und Nicht-ASF-Betriebsmodus mit WebSphere Application Serverund zum Konfigurieren von Listener-Ports für die Verwendung des Nicht-ASF-Modus finden Sie im Abschnitt Nachrichtenverarbeitung im ASF-Modus und im Modus ohne ASF.

Maximale Anzahl an Dialogen für die Überwachung eines Warteschlangenziels berechnen

Die maximale Anzahl an Dialogen zwischen einem einzelnen Listener-Port, der im Nicht-ASF-Modus ausgeführt wird und ein Warteschlangenziel überwacht, und IBM MQ kann mit der folgenden Formel berechnet werden:

```
Maximum number of conversations = Maximum sessions
```

Die Anzahl der Kanalinstanzen, die für diese Anzahl an Dialogen erstellt wird, kann wie folgt berechnet werden:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Ein aus dieser Rechnung verbleibender Rest kann aufgerundet werden.

Für einen einfachen Listenerport, der im Nicht-ASF-Modus ausgeführt wird, den Standardwert von **Maximum sessions** (Maximale Anzahl an Sitzungen) verwendet und ein Warteschlangenziel überwacht,

wird die maximale Anzahl an Dialogen zwischen WebSphere Application Server und IBM MQ wie folgt berechnet:

```
Maximum number of conversations = Maximum sessions
```

For example:

```
     = 10
```

Wenn dieser Listenerport eine Verbindung mit IBM MQ über einen Kanal herstellt, dessen Eigenschaft **SHARECNV** auf 10 gesetzt ist, berechnet sich die Anzahl der erstellten Kanalinstanzen wie folgt:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

For example:

```
     = 10 / 10
     = 1
```

Maximale Anzahl an Dialogen für die Überwachung eines Themenziels berechnen

Für einen Listenerport, der im Nicht-ASF-Modus ausgeführt wird und für die Überwachung eines Themenziels konfiguriert ist, berechnet sich die Anzahl der Dialoge zwischen dem Listenerport und IBM MQ wie folgt:

```
 Maximum number of conversations = 1
```

Die Anzahl der Kanalinstanzen, die für diese Anzahl an Dialogen erstellt wird, kann wie folgt berechnet werden:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Ein aus dieser Rechnung verbleibender Rest kann aufgerundet werden.

Für einen einfachen Listenerport, der im Nicht-ASF-Modus ausgeführt wird, den Standardwert von **Maximum sessions** (Maximale Anzahl an Sitzungen) verwendet und ein Themenziel überwacht, wird die maximale Anzahl an Dialogen zwischen WebSphere Application Server und IBM MQ wie folgt berechnet:

```
Maximum number of conversations = Maximum sessions
```

For example:

```
     = 10
```

Wenn dieser Listenerport eine Verbindung mit IBM MQ über einen Kanal herstellt, dessen Eigenschaft **SHARECNV** auf 10 gesetzt ist, berechnet sich die Anzahl der erstellten Kanalinstanzen wie folgt:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

For example:

```
     = 10 / 10
     = 1
```

Authentifizierungsaliase zum Schutz von WebSphere Application Server-Verbindungen mit IBM MQ konfigurieren

Authentifizierungsaliase sind einer Benutzernamens- und Kennwortkombination zugeordnet, die zum Schutz von WebSphere Application Server-Verbindungen mit IBM MQ verwendet werden können. Auch Verbindungsfactorys können mit Authentifizierungsaliasen konfiguriert werden.

Für Unternehmensanwendungen Authentifizierungsaliase verwenden

Wenn eine Unternehmensanwendung, die in WebSphere Application Server ausgeführt wird, versucht, eine JMS -Verbindung zu IBM MQherzustellen, sucht die Anwendung die Definition einer Verbindungsfactory des IBM MQ -Messaging-Providers im Repository Java Naming Directory Interface (JNDI) des Anwendungsservers.

Wenn sich die Verbindungsfactory-Definition des IBM MQ -Messaging-Providers im JNDI -Repository des Anwendungsservers befindet, wird eine der folgenden Methoden aufgerufen:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Wenn für die Verbindungsfactory ein J2C-Authentifizierungsalias definiert wurde, kann die Benutzernamens- und Kennwortkombination des Alias bei der Herstellung einer Verbindung mit dieser Verbindungsfactory an IBM MQ übergeben werden.

Verbindungsfactorys und Authentifizierungsaliase

Verbindungsfactorys von IBM MQMessaging-Providern enthalten Instruktionen zur Herstellung von Verbindungen mit IBM MQ-Warteschlangenmanagern. Innerhalb von WebSphere Application Server ausgeführte Unternehmensanwendungen können diese Verbindungsfactorys zur Herstellung von JMS-Verbindungen mit IBM MQ verwenden.

WebSphere Application Server speichert Verbindungsfactory-Definitionen in einem Repository, auf das über JNDI zugegriffen werden kann. Eine Verbindungsfactory erhält bei ihrer Erstellung einen JNDI-Namen, über den die Factory auf der Anwendungsserverebene, auf der sie definiert wurde (Zellen-, Knoten- oder Serverebene), eindeutig identifiziert werden kann.

Eine auf WebSphere Application Server-Zellenebene erstellte Verbindungsfactory eines IBM MQ-Messaging-Providers enthält beispielsweise Instruktionen zur Herstellung der Verbindung mit dem Warteschlangenmanager (`myQM`) über BINDINGS-Transport. Diese Verbindungsfactory erhält zur eindeutigen Kennzeichnung den JNDI-Namen `jms/myCF`.

Verbindungsfactorys können auch mit Authentifizierungsaliasen konfiguriert werden. Authentifizierungsaliase sind einer Benutzernamens- und Kennwortkombination zugeordnet. Je nachdem, wie die Verbindungsfactory verwendet wird, können Benutzername und Kennwort im Authentifizierungsalias an IBM MQ übergeben werden, wenn die JMS -Verbindung erstellt wird.

Wichtig: Vor IBM MQ 8.0 führt der Standard-OAM von IBM MQ (Object Authority Manager) allein aus dem Grund eine Berechtigungsprüfung durch, um festzustellen, ob der bei der Herstellung einer Verbindung an IBM MQ übergebene Benutzername auch über die Berechtigung für den Zugriff auf den Warteschlangenmanager verfügt.

Das angegebene Kennwort wird in älteren Versionen nicht überprüft. Um auch in diesen früheren Versionen eine Berechtigungsprüfung durchzuführen und zu überprüfen, ob Benutzername und Kennwort gültig sind, müssen Sie ein IBM MQ-Kanalsicherheitsexit schreiben. Details zur Vorgehensweise finden Sie in „Sicherheitsexitprogramme für Kanäle” auf Seite 1021.

Ab IBM MQ 8.0 überprüft der Warteschlangenmanager neben dem Benutzernamen auch das Kennwort.

Verbindungsfactory verwenden

Informationen zur Verwendung der Verbindungsfactory über ein direktes oder indirektes Lookup finden Sie in den folgenden Abschnitten:

- „Verbindungsfactory über ein direktes Lookup verwenden” auf Seite 539
- „Verbindungsfactory über ein indirektes Lookup verwenden” auf Seite 540

CLIENT-Transport verwenden

Für Verbindungsfactorys, die mit CLIENT-Transport konfiguriert sind, muss angegeben sein, welchen IBM MQ-Serververbindungskanal (SVRCONN) sie für die Verbindung mit dem Warteschlangenmanager verwenden.

Wenn die Eigenschaft MCAUSER (IBM MQ-Kanalagentbenutzer) des Kanals, für den die Verbindungsfactory konfiguriert ist, leer bleibt, kann die Verbindungsfactory mit einem direkten oder indirekten Lookup verwendet werden.

Wenn für die Eigenschaft MCAUSER eine Benutzer-ID eingestellt ist, wird diese Benutzer-ID bei der Herstellung einer Verbindung mit IBM MQ über diese Verbindungsfactory an IBM MQ übergeben, und zwar unabhängig davon, ob die Unternehmensanwendung ein direktes oder ein indirektes Lookup verwendet.

Übersichtstabellen

Die folgenden Tabellen fassen zusammen, welche Benutzer-IDs bei Verwendung des BINDINGS- bzw. des CLIENT-Transports an IBM MQ übergeben werden:

Tabelle 71. BINDINGS-Modus

Konfiguration	**Anwendung ruft `ConnectionFactory.createConnection()` auf**	**Anwendung ruft `ConnectionFactory.createConnection(String username, String password)` auf**
Der Anwendungsimplementierungsdeskriptor enthält keinen Ressourcenverweis auf die Verbindungsfactory	Die Benutzer-ID für den Anwendungsserverprozess wird an IBM MQ übergeben.	Die Benutzer-ID und das Kennwort, die an die Methode `ConnectionFactory.createConnection(String username, String password)` übergeben wurden, werden an IBM MQweitergegeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, und die Eigenschaft **`res-auth`** ist auf "Application" (Anwendung) gesetzt.	Die Benutzer-ID für den Anwendungsserverprozess wird an IBM MQ übergeben.	Die Benutzer-ID und das Kennwort, die an die Methode `ConnectionFactory.createConnection(String username, String password)` übergeben wurden, werden an IBM MQweitergegeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, und die Eigenschaft **`res-auth`** ist auf "Container" gesetzt.	Benutzer-ID und Kennwort des von der Verbindungsfactory verwendeten Authentifizierungsalias werden an IBM MQ übergeben.	Benutzer-ID und Kennwort des von der Verbindungsfactory verwendeten Authentifizierungsalias werden an IBM MQ übergeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, dessen Eigenschaft **`res-auth`** auf "Container" gesetzt ist, und für die Anwendung ist ein Authentifizierungsalias konfiguriert.	Benutzer-ID und Kennwort des von der Anwendung verwendeten Authentifizierungsalias werden an IBM MQ übergeben.	Benutzer-ID und Kennwort des von der Anwendung verwendeten Authentifizierungsalias werden an IBM MQ übergeben.

Tabelle 72. CLIENT-Modus

Konfiguration	**Anwendung ruft `ConnectionFactory.createConnection()` auf**	**Anwendung ruft `ConnectionFactory.createConnection(String username, String password)` auf**
Der Anwendungsimplementierungsdeskriptor enthält keinen Ressourcenverweis auf die Verbindungsfactory, und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER nicht festgelegt ist.	Die Benutzer-ID für den Anwendungsserverprozess wird an IBM MQ übergeben.	Die Benutzer-ID und das Kennwort, die an die Methode `ConnectionFactory.createConnection(String username, String password)` übergeben wurden, werden an IBM MQweitergegeben.
Der Anwendungsimplementierungsdeskriptor enthält keinen Ressourcenverweis auf die Verbindungsfactory, und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER auf eine Benutzer-ID gesetzt ist.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, dessen Eigenschaft **`res-auth`** auf *Application* (Anwendung) gesetzt ist, und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER nicht festgelegt ist.	Die Benutzer-ID für den Anwendungsserverprozess wird an IBM MQ übergeben.	Die Benutzer-ID und das Kennwort, die an die Methode `ConnectionFactory.createConnection(String username, String password)` übergeben wurden, werden an IBM MQweitergegeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, dessen Eigenschaft **`res-auth`** auf *Application* (Anwendung) gesetzt ist, und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER auf eine Benutzer-ID gesetzt ist.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, dessen Eigenschaft **`res-auth`** auf *Container* gesetzt ist, und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER nicht festgelegt ist.	Benutzer-ID und Kennwort des von der Verbindungsfactory verwendeten Authentifizierungsalias werden an IBM MQ übergeben.	Benutzer-ID und Kennwort des von der Verbindungsfactory verwendeten Authentifizierungsalias werden an IBM MQ übergeben.

Tabelle 72. CLIENT-Modus (Forts.)		
Konfiguration	**Anwendung ruft `ConnectionFactory.createConnection()` auf**	**Anwendung ruft `ConnectionFactory.createConnection(String username, String password)` auf**
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, dessen Eigenschaft **`res-auth`** auf *Container* gesetzt ist, und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER auf eine Benutzer-ID gesetzt ist.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, dessen Eigenschaft **`res-auth`** auf *Container* gesetzt ist, für die Anwendung ist ein Authentifizierungsalias konfiguriert und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER nicht festgelegt ist.	Benutzer-ID und Kennwort des von der Anwendung verwendeten Authentifizierungsalias werden an IBM MQ übergeben.	Benutzer-ID und Kennwort des von der Anwendung verwendeten Authentifizierungsalias werden an IBM MQ übergeben.
Der Anwendungsimplementierungsdeskriptor enthält einen Ressourcenverweis auf die Verbindungsfactory, dessen Eigenschaft **`res-auth`** auf *Container* gesetzt ist, für die Anwendung ist ein Authentifizierungsalias konfiguriert und die Verbindungsfactory verwendet einen IBM MQ-Kanal, dessen Eigenschaft MCAUSER auf eine Benutzer-ID gesetzt ist.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.	Die Benutzer-ID, die durch die Eigenschaft MCAUSER des von der Verbindungsfactory verwendeten IBM MQ-Kanals vorgegeben ist, wird an IBM MQ übergeben.

Verbindungsfactory über ein direktes Lookup verwenden

Nachdem für den IBM MQ-Messaging-Provider eine Verbindungsfactory definiert wurde, kann diese Verbindungsfactory durch ein Lookup in einer Unternehmensanwendung gesucht und zur Herstellung einer JMS-Verbindung mit einem IBM MQ-Warteschlangenmanager verwendet werden. Dies kann über ein indirektes Lookup erfolgen.

Bei einem direkten Lookup stellt eine Unternehmensanwendung mit folgendem Methodenaufruf eine Verbindung mit dem JNDI-Repository des Anwendungsservers her:

```
InitialContext ctx = new InitialContext();
```

Sobald die Verbindung mit dem JNDI-Repository besteht, identifiziert die Unternehmensanwendung die Verbindungsfactory-Definition wie folgt mit dem JNDI-Namen der Verbindungsfactory:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

Anmerkungen:

- Bei der Entwicklung der Unternehmensanwendung muss dem Anwendungsentwickler der JNDI-Name der erforderlichen Verbindungsfactory bekannt sein. Da der JNDI-Name in der Anwendung fest codiert ist, muss die Anwendung bei einer Änderung des JNDI-Namens neu erstellt und implementiert werden.
- Wenn eine Verbindungsfactory-Definition auf diese Weise verwendet wird, werden die im Authentifizierungsalias angegebene Benutzernamens- und Kennwortkombination nicht an IBM MQ weitergegeben. Dadurch wird verhindert, dass nicht autorisierte Anwendungen die Verbindungsfactory identifizieren und sich darüber selbst mit geschützten IBM MQ-Systemen verbinden können.

Die an IBM MQ übergebene Benutzernamens- und Kennwortkombination hängt von der Methode ab, mit der die JMS-Verbindung aus der Verbindungsfactory erstellt wird.

Erstellt eine Anwendung eine JMS-Verbindung mit folgender Methode:

```
ConnectionFactory.createConnection()
```

so wird die Identität des Standardbenutzers an IBM MQ übergeben. Dies ist die Kombination aus Benutzername und Kennwort, mit der der Anwendungsserver, unter dem die Unternehmensanwendung ausgeführt wird, gestartet wurde.

Alternativ kann eine Anwendung auch mit folgender Methode eine JMS-Verbindung erstellen:

```
ConnectionFactory.createConnection(String username, String password)
```

Wenn eine Anwendung eine Verbindungsfactory über ein direktes Lookup gefunden hat und danach diese Methode aufruft, werden die in `createConnection()` bereitgestellte Benutzernamens- und Kennwortkombination an IBM MQ übergeben.

Wichtig: Vor IBM MQ 8.0 führt IBM MQ allein aus dem Grund eine Berechtigungsprüfung durch, um festzustellen, ob der übergebene Benutzername auch über die Berechtigung für den Zugriff auf den Warteschlangenmanager verfügt.

Das Kennwort wurde nicht überprüft. Um auch in älteren Versionen eine Berechtigungsprüfung durchzuführen und zu überprüfen, ob Benutzername und Kennwort gültig sind, muss ein IBM MQ-Kanalsicherheitsexit geschrieben werden. Details zur Vorgehensweise finden Sie in „Sicherheitsexitprogramme für Kanäle” auf Seite 1021.

Ab IBM MQ 8.0 überprüft der Warteschlangenmanager neben dem Benutzernamen auch das Kennwort.

Verbindungsfactory über ein indirektes Lookup verwenden

Wenn Sie eine Unternehmensanwendung schreiben, der JNDI-Name der Verbindungsfactory aber nicht bekannt ist, oder wenn die Anwendung auf mehreren Anwendungsservern mit unterschiedlichen Verbindungsfactorys und je nach Anwendungsserver anderen JNDI-Namen installiert werden soll, kann die Verbindungsfactory mithilfe eines Ressourcenverweises gesucht werden. Dies kann über ein indirektes Lookup erfolgen.

Beispiel

Anstatt die Verbindungsfactory direkt über `jms/myCF` zu suchen, enthält eine Unternehmensanwendung einen Ressourcenverweis, dessen lokaler JNDI-Name wie folgt lautet: `jms/myResourceReferenceCF`.

Zur Verwendung dieses JNDI-Namens stellt die Anwendung auf die gleiche Weise wie bei einem direkten Lookup eine Verbindung mit dem JNDI-Repository des Anwendungsservers her:

```
InitialContext ctx = new InitialContext();
```

Anstatt `jms/myCF` nun aber direkt anzugeben, gibt die Anwendung den JNDI-Namen des Ressourcenverweises an:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/myResourceReferen
ceCF");
```

Um den Anwendungsserver zu instruieren, dass die Unternehmensanwendung ein indirektes Lookup ausführt, müssen Sie das Präfix `java:comp/env` des lokalen JNDI-Namens angeben.

Nach der Implementierung der Anwendung ordnet der Benutzer den JNDI-Namen des Ressourcenverweises `jms/myResourceReferenceCF` dem bereits von der Anwendung erstellten JNDI-Namen der Verbindungsfactory zu: `jms/myCF`.

Bei der Ausführung der Anwendung sucht sie nach einer JMS-Verbindungsfactory mit dem lokalen JNDI-Namen, den der Anwendungsserver der folgenden Verbindungsfactory zuordnet: `jms/myCF`. Diese Verbindungsfactory wird dann von der Anwendung zur Herstellung der Verbindung mit IBM MQ verwendet.

Authentifizierungsaliase und indirekte Lookups

Über einen Ressourcenverweis können auch weitere Eigenschaften definiert werden, die das Verhalten der bereitgestellten Verbindungsfactory ändern. Eine der Eigenschaften eines Ressourcenverweises ist **`res-auth`**. Der Wert dieser Eigenschaft legt fest, ob die Unternehmensanwendung bei der Herstellung der Verbindung mit IBM MQ den Authentifizierungsalias der dem Ressourcenverweis zugeordneten Verbindungsfactory verwenden soll (sofern ein Authentifizierungsalias definiert wurde) oder ob die Anwendung Benutzername und Kennwort selbst angibt.

Der Standardwert dieser Eigenschaft ist *Application* (Anwendung). Das heißt, Benutzername und Kennwort, die bei der Herstellung einer Verbindung mit JMS an den Warteschlangenmanager übergeben werden, werden durch die Anwendung bestimmt. Der Authentifizierungsalias der dem Ressourcenverweis zugeordneten Verbindungsfactory wird standardmäßig also nicht verwendet.

Anwendungen können mit einer der folgenden Methoden JMS-Verbindungen erstellen:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Wenn eine Anwendung `ConnectionFactory.createConnection()` verwendet und **`res-auth`** auf *Application* (Anwendung) gesetzt ist, wird an IBM MQ die Identität des Standardbenutzers übergeben. Dies ist die Kombination aus Benutzername und Kennwort, mit der der Anwendungsserver, unter dem die Unternehmensanwendung ausgeführt wird, gestartet wurde.

Wenn eine Anwendung `ConnectionFactory.createConnection(String username, String password)` verwendet und **`res-auth`** auf *Anwendung*gesetzt ist, werden der Benutzername und das Kennwort, die an die Methode übergeben werden, an IBM MQgesendet.

Wenn bei der Herstellung der Verbindung der Authentifizierungsalias der dem Ressourcenverweis zugeordneten Verbindungsfactory verwendet werden soll, muss die Eigenschaft **`res-auth`** auf *Container* gesetzt werden. In diesem Fall werden bei der Herstellung einer Verbindung mit JMS die Details des Authentifizierungsalias verwendet, selbst wenn der `createConnection`-Aufruf Benutzername und Kennwort bereitstellt.

Authentifizierungsalias bei Verwendung eines indirekten Lookups überschreiben

Wenn eine Anwendung einen Ressourcenverweis verwendet, dessen Eigenschaft **`res-auth`** auf *Container* gesetzt ist, können Sie den bei der Herstellung von JMS-Verbindungen verwendeten Authentifizierungsalias überschreiben.

Hierzu muss der Ressourcenverweis zusätzlich die Eigenschaft **`authDataAlias`** enthalten und diese muss einem Authentifizierungsalias zugeordnet sein, der bereits in der Anwendungsserverumgebung, in der die Anwendung implementiert wird, vorhanden ist. Diese Eigenschaft können Sie für jeden Ressourcenverweis festlegen, der mit den Rational-Tools von IBM erstellt wurde.

Mit dieser Methode können Sie bei Verwendung einer durch ein indirektes Lookup ermittelten JMS-Verbindungsfactory einen anderen Authentifizierungsalias verwenden. Ist der angegebene Authentifizierungsalias nicht vorhanden, kann nach der Installation der Unternehmensanwendung ein neuer Authentifizierungsalias festgelegt werden. Weitere Informationen hierzu finden Sie im Abschnitt *Ressourcenverweise* in der Produktdokumentation zu WebSphere Application Server.

Referenzinformationen für WebSphere Application Server 8.5.5

Ressourcenverweise

Referenzinformationen für WebSphere Application Server 8.0

Ressourcenverweise

Referenzinformationen für WebSphere Application Server 7.0

Ressourcenverweise

Lastausgleich für Message-driven Beans bei Verwendung von WebSphere Application Server-Clustern

Bei Verwendung von Message-driven Bean-Anwendungen, die in einem WebSphere Application Server 7.0- und WebSphere Application Server 8.0-Cluster implementiert und für die Ausführung im Normalmodus des IBM MQ-Messaging-Providers konfiguriert sind, wird ein Großteil der Nachrichten von nur einem Cluster-Member verarbeitet. Sie können die Arbeitslast der Cluster-Member jedoch ausgleichen, um die Nachrichtenverarbeitung auf mehrere Member zu verteilen.

IBM MQ enthält eine Funktion mit dem Namen **Asynchronous consume**, die es Anwendungen ermöglicht, Nachrichten asynchron aus einer Warteschlange unter Verwendung von APIs mit dem Namen **MQCB** und **MQCTL**zu konsumieren.

Message-driven Bean-Anwendungen, die in WebSphere Application Server 7.0 und WebSphere Application Server 8.0 im Normalmodus des IBM MQ-Messaging-Providers ausgeführt werden, verwenden diese Funktion automatisch. Wenn die Anwendungen gestartet werden, richten sie einen asynchronen Konsumenten für das JMS -Ziel ein, für dessen Überwachung sie konfiguriert wurden, indem sie **MQCB**aufrufen. Danach gibt die Anwendung durch den Aufruf der **MQCTL**-API bekannt, dass sie bereit für den Empfang von Nachrichten aus dieser JMS-Warteschlange ist.

Wenn die Message-driven Bean-Anwendungen in einem WebSphere Application Server-Cluster implementiert wurden, richtet jedes Cluster-Member einen asynchronen Konsumenten für die JMS-Warteschlange ein, die das Message-driven Bean auf Nachrichten überwacht. Der IBM MQ-Warteschlangenmanager, der als Host für das JMS-Ziel fungiert, ist dann für die Benachrichtigung des Cluster-Members verantwortlich, wenn eine entsprechende Nachricht auf dem JMS-Ziel zu verarbeiten ist.

Wenn WebSphere Application Server eine Verbindung zu einem IBM MQ -Warteschlangenmanager herstellt, werden Nachrichten, die an einem JMS -Ziel ankommen, gleichmäßiger an alle asynchronen Konsumenten verteilt, die an diesem JMS -Ziel registriert sind. Dies bedeutet für Message-driven Bean-Anwendungen in einem WebSphere Application Server 7.0- und WebSphere Application Server 8.0-Cluster, dass die Nachrichten gleichmäßiger auf die Cluster-Member aufgeteilt sind.

Zugehörige Tasks

Eigenschaft JMS **PROVIDERVERSION** konfigurieren

Paket IBM MQ Headers verwenden

Das Paket IBM MQ Headers stellt eine Reihe von Helper-Schnittstellen und -Klassen bereit, mit deren Hilfe Sie die IBM MQ-Header einer Nachricht bearbeiten können. Normalerweise verwenden Sie das Paket IBM MQ Headers, wenn Sie Verwaltungsservices mithilfe des Befehlsservers (unter Verwendung von Programmable Command Format-(PCF-)Nachrichten) ausführen möchten.

Informationen zu diesem Vorgang

Das Paket IBM MQ Headers ist in den Paketen `com.ibm.mq.headers` und `com.ibm.mq.headers.pcf` enthalten. Sie können diese Funktion für jede der beiden alternativen APIs verwenden, die IBM MQ für die Verwendung in Java-Anwendungen bereitstellt:

- IBM MQ classes for Java (auch als IBM MQ Base Java bezeichnet)
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS, auch als IBM MQ JMS bezeichnet)

IBM MQ Base Java-Anwendungen bearbeiten üblicherweise MQMessage-Objekte und die Headers-Unterstützungsklassen interagieren direkt mit diesen Objekten, da sie die IBM MQ Base Java-Schnittstellen nativ verstehen.

In IBM MQ JMS sind die Nutzdaten für eine Nachricht normalerweise ein String- oder Byte-Array-Objekt, das mit DataInput- und DataOutput-Strömen bearbeitet werden kann. Mithilfe des Pakets IBM MQ Headers kann mit diesen Datenströmen interagiert werden. Außerdem ermöglicht es die Bearbeitung aller MQ-Nachrichten, die von IBM MQ JMS-Anwendungen gesendet und empfangen werden.

Obwohl das Paket IBM MQ Headers Referenzen auf das Paket IBM MQ Base Java enthält, ist es deshalb auch zur Verwendung in IBM MQ JMS-Anwendungen vorgesehen und für die Verwendung in Java Platform, Enterprise Edition-(Java EE-)Umgebungen geeignet.

Eine typische Verwendungsmöglichkeit für das Paket IBM MQ Headers ist die Bearbeitung von Verwaltungsnachrichten im Programmable Command Format (PCF), z. B. aus einem der folgenden Gründe:

- Um auf Details einer IBM MQ-Ressource zuzugreifen.
- Um die Größe einer Warteschlange zu überwachen.
- Um den Zugriff auf eine Warteschlange zu blockieren.

Mithilfe von PCF-Nachrichten mit der IBM MQ JMS-API kann diese Art der Verwaltung von anwendungsorientierten Ressourcen aus Java EE-Anwendungen heraus durchgeführt werden, ohne dass dazu auf die IBM MQ Base Java-API zurückgegriffen werden muss.

Prozedur

- Informationen zur Verwendung des Pakets IBM MQ Headers zur Bearbeitung von Nachrichtenheadern für die IBM MQ classes for Java finden Sie im Abschnitt „Mit IBM MQ classes for Java verwenden” auf Seite 543.
- Informationen zur Verwendung des Pakets IBM MQ Headers zur Bearbeitung von Nachrichtenheadern für die IBM MQ classes for JMS finden Sie im Abschnitt „Mit IBM MQ classes for JMS verwenden” auf Seite 544.

Mit IBM MQ classes for Java verwenden

Anwendungen der IBM MQ classes for Java bearbeiten üblicherweise MQMessage-Objekte und die Headers-Unterstützungsklassen interagieren direkt mit diesen Objekten, da sie die Schnittstellen der IBM MQ classes for Java nativ verstehen.

Informationen zu diesem Vorgang

IBM MQ stellt Beispielanwendungen bereit, die zeigen, wie das Paket IBM MQ Headers mit der IBM MQ Base Java-API (IBM MQ classes for Java) verwendet wird.

Die Beispiele zeigen zwei Dinge:

- Wie eine PCF-Nachricht erstellt wird, um eine Verwaltungsaktion auszuführen und die Antwortnachricht auszuwerten.
- Wie diese PCF-Nachricht mithilfe der IBM MQ classes for Java gesendet wird.

Abhängig von der verwendeten Plattform sind diese Beispiele im Unterverzeichnis `pcf` des Verzeichnisses `samples` oder `tools` Ihrer IBM MQ-Installation installiert (siehe „Installationsverzeichnisse für IBM MQ classes for Java” auf Seite 372).

Vorgehensweise

1. Erstellen Sie eine PCF-Nachricht, um eine Verwaltungsaktion auszuführen und die Antwortnachricht auszuwerten.

2. Senden Sie diese PCF-Nachricht mithilfe der IBM MQ classes for Java.

Zugehörige Konzepte

„Behandlung von IBM MQ-Nachrichtenheadern mit IBM MQ classes for Java” auf Seite 401
Es werden Java-Klassen zur Verfügung gestellt, die verschiedene Arten von Nachrichtenheadern darstellen. Darüber hinaus stehen zwei Helper-Klassen zur Verfügung.

„Behandlung von PCF-Nachrichten mit IBM MQ classes for Java” auf Seite 406
Java-Klassen werden für die Erstellung und das Parsing von PCF-strukturierten Nachrichten zur Verfügung gestellt. Darüber hinaus erleichtern sie das Versenden von PCF-Anforderungen und das Sammeln von PCF-Antworten.

Mit IBM MQ classes for JMS verwenden

Um das Paket IBM MQ Headers mit den IBM MQ classes for JMS zu verwenden, sind dieselben grundlegenden Schritte wie für die IBM MQ classes for Java auszuführen. Die Erstellung der PCF-Nachricht und die Auswertung der Antwort können unter Verwendung des Pakets IBM MQ Headers und desselben Beispielcodes auf die gleiche Weise wie für die IBM MQ classes for Java durchgeführt werden.

Informationen zu diesem Vorgang

Um eine PCF-Nachricht über die IBM MQ-API zu senden, müssen die Nachrichtennutzdaten in eine JMS-BytesMessage geschrieben und über die Standard-JMS-APIs gesendet werden. Die einzige Bedingung dabei ist, dass die Nachricht keinen JMS RFH2 oder anderen Header mit spezifischen Werten im MQMD enthalten darf.

Führen Sie die folgenden Schritte aus, um eine PCF-Nachricht zu senden. Die Vorgehensweise zum Erstellen der PCF-Nachricht und Extrahieren von Informationen aus der Antwortnachricht ist dieselbe wie für die IBM MQ classes for Java (siehe „Mit IBM MQ classes for Java verwenden” auf Seite 543).

Vorgehensweise

1. Erstellen Sie ein JMS -Warteschlangenziel, das SYSTEM.ADMIN.COMMAND.QUEUE.

   IBM MQ JMS -Anwendungen senden die PCF-Nachrichten an das SYSTEM.ADMIN.COMMAND.QUEUEund benötigen Zugriff auf ein JMS-Zielobjekt, das diese Warteschlange darstellt. Für das Ziel müssen folgende Eigenschaften festgelegt sein:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   Wenn Sie WebSphere Application Server verwenden, müssen Sie diese Eigenschaften als benutzerdefinierte Eigenschaften für das Ziel definieren.

   Verwenden Sie folgenden Code, um das Ziel programmgesteuert aus einer Anwendung zu erstellen:

   ```
   Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
   ((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
       WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQQueue) q1).setMQMDWriteEnabled(true);
   ```

2. Konvertieren Sie eine PCF-Nachricht in eine JMS-BytesMessage, die die richtigen MQMD-Werte enthält.

   Es muss eine JMS-BytesMessage erstellt und die PCF-Nachricht in die BytesMessage geschrieben werden. Es muss eine Antwortwarteschlange erstellt werden, für die aber keine bestimmten Einstellungen erforderlich sind.

   Der folgende Beispielcodeausschnitt zeigt, wie eine JMS-BytesMessage erstellt und ein com.ibm.mq.headers.pcf.PCFMessage-Objekt in die BytesMessage geschrieben wird. Das PCFMessage-Objekt (pcfCmd) wurde zuvor mit dem Paket IBM MQ Headers erstellt. (Hinweis: Das Paket zum Laden der PCFMessage ist com.ibm.mq.headers.pcf.PCFMessage).

   ```
    // create the JMS Bytes Message
   final BytesMessage msg = session.createBytesMessage();
   ```

```
// Create the wrapping streams to put the bytes into the message payload
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutput dataOutput = new DataOutputStream(baos);

// Set the JMSReplyTo so the answer comes back
msg.setJMSReplyTo(new MQQueue("adminResp"));

// write the pcf into the stream
pcfCmd.write(dataOutput);
baos.flush();
msg.writeBytes(baos.toByteArray());

// we have taken control of the MD, so need to set all
// flags in the MD that we require - main one is the format
msg.setJMSPriority(4);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
  CMQC.MQPER_NOT_PERSISTENT);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
    CMQC.MQRO_PASS_CORREL_ID);
msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

// and send the message
sender.send(msg);
```

3. Senden Sie die Nachricht und empfangen Sie die Antwort über die Standard-JMS-APIs.
4. Konvertieren Sie die Antwortnachricht zur weiteren Verarbeitung in eine PCF-Nachricht.

   Verwenden Sie folgenden Code, um die Antwortnachricht abzurufen und als PCF-Nachricht zu verarbeiten:

```
 // Get the message back
BytesMessage msg = (BytesMessage) consumer.receive();

// get the size of the bytes message & read into an array
int bodySize = (int) msg.getBodyLength();
byte[] data = new byte[bodySize];
msg.readBytes(data);

// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);
```

Zugehörige Konzepte

„JMS-Nachrichten” auf Seite 151
JMS-Nachrichten bestehen aus einem Header, aus Eigenschaften und aus einem Hauptteil. JMS definiert fünf Typen des Nachrichtenhauptteils.

IBM i IBM MQ unter IBM i mit Java und JMS einrichten

Diese Abschnitte geben einen Überblick über die Vorgehensweise zum Einrichten und Testen von IBM MQ mit Java und JMS unter IBM i mithilfe von CL-Befehlen oder der Qshell-Umgebung.

Anmerkung:

- Ab IBM MQ 8.0 sind `ldap.jar`, `jndi.jar` und `jta.jar` Bestandteile des JDK.
- 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.

CL-Befehle verwenden

Der von Ihnen festgelegte Klassenpfad dient zum Testen mit MQ Base Java, JMS mit JNDI und JMS ohne JNDI.

Wenn Sie keine Datei `.profile` unter Ihrem Verzeichnis `/home/Userprofile` verwenden, müssen Sie die unten aufgeführten Umgebungsvariablen auf Systemebene festlegen. Sie können mit dem Befehl **WRKENVVAR** überprüfen, ob sie festgelegt wurden.

1. Geben Sie folgenden Befehl aus, um die Umgebungsvariablen für das gesamte System anzuzeigen: **WRKENVVAR LEVEL(*SYS)**
2. Geben Sie folgenden Befehl aus, um die spezifischen Umgebungsvariablen für Ihren Job anzuzeigen: **WRKENVVAR LEVEL(*JOB)**
3. Wenn der CLASSPATH nicht definiert ist, setzen Sie den folgenden Befehl ab:

JM 3.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

JMS 2.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. Wenn QIBM_MULTI_THREADED nicht gesetzt ist, geben Sie folgenden Befehl aus:

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. Wenn QIBM_USE_DESCRIPTOR_STDIO nicht gesetzt ist, geben Sie folgenden Befehl aus:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. Wenn QSH_REDIRECTION_TEXTDATA nicht gesetzt ist, geben Sie folgenden Befehl aus:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

Qshell-Umgebung verwenden

Bei Verwendung der QSHELL-Umgebung können Sie eine Datei `.profile` in Ihrem Verzeichnis `/home/Userprofile` einrichten. Weitere Informationen finden Sie in der Dokumentation zu Qshell Interpreter (qsh).

Geben Sie in der Datei `.profile` Folgendes an. Beachten Sie, dass die CLASSPATH-Anweisung in einer einzelnen Zeile stehen muss oder wie gezeigt durch das Zeichen \ auf mehrere Zeilen verteilt werden muss.

JM 3.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
```

```
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS 2.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

Stellen Sie sicher, dass sich die Bibliothek QMQMJAVA in der Bibliotheksliste befindet, indem Sie den Befehl **DSPLIBL** ausgeben.

Wenn die Bibliothek QMQMJAVA nicht in der Liste enthalten ist, fügen Sie sie mit folgendem Befehl hinzu: **ADDLIBLE LIB (QMQMJAVA)**

IBM i IBM MQ unter IBM i mit Java testen

Vorgehensweise zum Testen von IBM MQ mit Java mithilfe des Beispielprogramms MQIVP

IBM MQ-Basis-Java testen

Gehen Sie wie folgt vor:

1. Vergewissern Sie sich, dass der Warteschlangenmanager gestartet wurde und den Status ACTIVE hat, indem Sie folgenden Befehl ausgeben:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Vergewissern Sie sich, dass der Serververbindungskanal JAVA.CHANNEL erstellt wurde, indem Sie folgenden Befehl ausgeben:

   ```
   WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

   a. Wenn der JAVA.CHANNEL nicht vorhanden ist, geben Sie folgenden Befehl aus:

   ```
   CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

3. Vergewissern Sie sich, dass der Warteschlangenmanager-Listener für Port 1414 bzw. den Port, den Sie verwenden, ausgeführt wird, indem Sie den Befehl **WRKMQMLSR** ausgeben.

   a. Wenn kein Listener für den Warteschlangenmanager gestartet wurde, geben Sie folgenden Befehl aus:

   ```
   STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
   ```

Beispieltestprogramm MQIVP ausführen

1. Starten Sie die Qshell aus der Befehlszeile, indem Sie den Befehl STRQSH ausgeben.

2. Vergewissern Sie sich, dass der richtige CLASSPATH gesetzt ist, indem Sie den Befehl **export** ausgeben, und geben Sie dann den Befehl **cd** wie folgt aus:

```
cd /qibm/proddata/mqm/java/samples/wmqjava/samples
```

3. Führen Sie das **java**-Programm aus, indem Sie folgenden Befehl ausgeben:

```
java MQIVP
```

Sie können die Taste ENTER drücken, wenn Sie zur Eingabe von

- Verbindungstyp
- IP-Adresse
- Name des Warteschlangenmanagers

aufgefordert werden, um die Standardwerte zu übernehmen. Dies bestätigt die Produktbindungen, die in der Bibliothek QMQMJAVA eingesehen werden können.

Sie erhalten eine Ausgabe wie die im folgenden Beispiel. Beachten Sie, dass der Copyrightvermerk von der verwendeten Version des Produkts abhängt.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM MQ Java-Clientverbindung testen

Sie müssen Folgendes angeben:

- Verbindungstyp
- IP-Adresse
- Port
- Serververbindungskanal
- Warteschlangenmanager

Sie erhalten eine Ausgabe wie die im folgenden Beispiel. Beachten Sie, dass der Copyrightvermerk von der verwendeten Version des Produkts abhängt.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
```

```
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i IBM MQ unter IBM i mit JMS testen

Vorgehensweise zum Testen von IBM MQ mit JMS mit und ohne JNDI

JMS ohne JNDI mit dem Beispielprogramm IVTRun testen

Gehen Sie wie folgt vor:

1. Vergewissern Sie sich, dass der Warteschlangenmanager gestartet wurde und den Status ACTIVE hat, indem Sie folgenden Befehl ausgeben:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Starten Sie die Qshell aus der Befehlszeile, indem Sie den Befehl **STRQSH** ausgeben.
3. Wechseln Sie mit dem Befehl **cd** wie folgt das Verzeichnis:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

4. Führen Sie die Scriptdatei aus:

   ```
   IVTRun -nojndi [-m qmgrname]
   ```

Sie erhalten eine Ausgabe wie die im folgenden Beispiel. Beachten Sie, dass die Copyrightvermerke von den verwendeten Versionen der Produkte abhängen:

```
 IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
```

```
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

IBM MQ JMS-Clientmodus ohne JNDI testen

Gehen Sie wie folgt vor:

1. Vergewissern Sie sich, dass der Warteschlangenmanager gestartet wurde und den Status ACTIVE hat, indem Sie folgenden Befehl ausgeben:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Vergewissern Sie sich, dass der Serververbindungskanal erstellt wurde, indem Sie folgenden Befehl ausgeben:

   ```
   WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
   MQMNAME(QMGRNAME)
   ```

3. Vergewissern Sie sich, dass der Listener für den richtigen Port gestartet wurde, indem Sie den Befehl **WRKMQMLSR** ausgeben.
4. Starten Sie die Qshell aus der Befehlszeile, indem Sie den Befehl **STRQSH** ausgeben.
5. Überprüfen Sie, ob der CLASSPATH richtig ist, indem Sie den Befehl **export** ausgeben.
6. Wechseln Sie mit dem Befehl **cd** wie folgt das Verzeichnis:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

7. Führen Sie die Scriptdatei aus:

   ```
   IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
   ```

Sie erhalten eine Ausgabe wie die im folgenden Beispiel. Beachten Sie, dass die Copyrightvermerke von den verwendeten Versionen der Produkte abhängen.

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
```

```
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

IBM MQ JMS mit JNDI testen

Vergewissern Sie sich, dass der Warteschlangenmanager gestartet wurde und den Status ACTIVE hat, indem Sie folgenden Befehl ausgeben:

```
WRKMQM MQMNAME(QMGRNAME)
```

Beispieltestscript IVTRun verwenden

Gehen Sie wie folgt vor:

1. Nehmen Sie die entsprechenden Änderungen an der Datei `JMSAdmin.config` vor. Geben Sie zum Bearbeiten der Datei den Befehl **EDTF** (Edit File) in der IBM i-Befehlszeile aus:

   ```
   EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
   ```

   .

   a. Entfernen Sie die Kommentarzeichen aus folgender Zeile, um LDAP für Weblogic zu verwenden:

      ```
      INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
      ```

   b. Entfernen Sie die Kommentarzeichen aus folgender Zeile, um LDAP für WebSphere Application Server zu verwenden:

      ```
      INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
      ```

   c. Entfernen Sie die Kommentarzeichen aus folgender Zeile, um das Dateisystem zu testen:

      ```
      INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
      ```

   d. Stellen Sie sicher, dass die richtige PROVIDER_URL ausgewählt wurde, indem Sie die Kommentarzeilen aus der entsprechenden Zeile entfernen.

   e. Setzen Sie alle anderen Zeilen mit dem Symbol # auf Kommentar.

   f. Wenn Sie alle Änderungen vorgenommen haben, drücken Sie **F2=Save** und **F3=Exit**.

2. Starten Sie die Qshell aus der Befehlszeile, indem Sie den Befehl **STRQSH** ausgeben.

3. Überprüfen Sie, ob der CLASSPATH richtig ist, indem Sie den Befehl **export** ausgeben.
4. Wechseln Sie mit dem Befehl **cd** wie folgt das Verzeichnis:

```
cd /qibm/proddata/mqm/java/bin
```

5. Starten Sie das Script **IVTSetup**, um die verwalteten Objekte (*MQQueueConnectionFactory* und *MQQueue*) zu erstellen, indem Sie den Befehl **IVTSetup** ausgeben.
6. Führen Sie das Script IVTRun aus, indem Sie folgenden Befehl ausgeben:

```
IVTRun -url providerURL [-icf initCtxFact]
```

Sie erhalten eine Ausgabe wie die im folgenden Beispiel. Beachten Sie, dass die Copyrightvermerke von den verwendeten Versionen der Produkte abhängen.

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
MQSeries classes for Java Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
```

```
IVT completed OK
IVT finished
$
```

Java-Anwendungsentwicklung mit einem Maven-Repository

Wenn Sie eine Java-Anwendung für IBM MQ mithilfe eines Maven-Repositorys entwickeln, damit Abhängigkeiten automatisch installiert werden, müssen Sie vor der Verwendung der IBM MQ-Schnittstellen keine explizite Installation durchführen.

Zentrales Maven-Repository

Bei Maven handelt es sich um ein Tool, mit dem Anwendungen erstellt werden und auch ein Repository zum Speichern von Artefakten bereitgestellt wird, auf das Ihre Anwendung möglicherweise zugreifen will.

Das Maven-Repository (oder zentrales Repository) verfügt über eine Struktur, mit der Dateien wie beispielsweise JAR-Dateien eindeutige Versionen erhalten, die anschließend mit einem gängigen Verfahren zur Benennung erkannt werden können. Erstellungstools können mit diesen Namen anschließend die Abhängigkeiten für Ihre Anwendung extrahieren. In der Definition Ihrer Anwendung, die bei Verwendung von Maven als Build-Tool als POM-Datei bezeichnet wird, benennen Sie die Abhängigkeiten und der Buildprozess weiß, was von dort aus zu tun ist.

IBM MQ-Clientdateien

Kopien der IBM MQ-Java-Clientschnittstellen sind im zentralen Repository unter `com.ibm.mq GroupId` verfügbar. Sie finden die Datei `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) und die Datei `com.ibm.mq.allclient.jar` (JMS 2.0). Diese Dateien werden normalerweise für eigenständige Programme verwendet. Sie finden auch die Datei `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) und die Datei `wmq.jmsra.rar` (JMS 2.0), die in Java EE -Anwendungsservern verwendet werden kann. `jakarta.client.jar` und `allclient.jar` enthalten sowohl IBM MQ classes for JMS als auch IBM MQ classes for Java.

Wichtig: Die Verwendung des Formats *jar-with-dependencies* des Apache-Maven-Assembly-Plug-ins zum Erstellen einer Anwendung, die die verschiebbaren JAR-Datei von IBM MQ einschließt, wird nicht unterstützt.

In einer mit dem Maven-Befehl verarbeiteten Datei `pom.xml` fügen Sie Abhängigkeiten für diese JAR-Dateien hinzu; dies ist in den folgende Beispielen gezeigt:

- JM 3.0 Zur Anzeige der Beziehung zwischen Ihrem Anwendungscode und der Datei `com.ibm.mq.jakarta.client.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 Zur Anzeige der Beziehung zwischen Ihrem Anwendungscode und der Datei `com.ibm.mq.allclient.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- JM 3.0 Für die Verwendung des Jakarta EE-Ressourcenadapters:

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
```

```
    <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 Für die Verwendung des JMS 2.0 Java EE -Ressourcenadapters:

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

Ein Beispiel für ein einfaches Projekt in Eclipse für die Ausführung eines JMS -Projekts finden Sie im IBM Developer -Artikel Developing Java applications for MQ jetzt einfacher mit Maven.

C++-Anwendungen entwickeln

In IBM MQ werden C++-Klassen bereitgestellt, die IBM MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen.

In IBM WebSphere MQ 7.0 werden Erweiterungen an den IBM MQ-Programmierschnittstellen nicht für die C++-Klassen angewendet.

IBM MQ C++ stellt die folgenden Funktionen bereit:

- Automatische Initialisierung von IBM MQ-Datenstrukturen
- Just-in-time-Verbindung zum Warteschlangenmanager und Öffnung der Warteschlange
- Impliziter Abschluss der Warteschlange und Verbindungsabbau des Warteschlangenmanagers
- Übertragung und Empfang von Headern für nicht zustellbare Nachrichten
- Übertragung und Empfang von IMS-Bridge-Headern
- Übertragung und Empfang von Headern für Referenznachrichten
- Empfang von Auslösenachrichten
- Übertragung und Empfang von CICS bridge-Headern
- Übertragung und Empfang von Arbeitsheadern
- Clientkanaldefinition.

In den folgenden Booch-Klassendiagrammen wird gezeigt, dass alle Klassen umfassend parallel zu den IBM MQ-Entitäten in der prozeduralen MQI sind (z. B. bei Verwendung von C), die über Kennungen oder Datenstrukturen verfügen. Alle Klassen übernehmen Daten aus der ImqError-Klasse (siehe ImqError C++-Klasse), wodurch jedem Objekt eine Fehlerbedingung zugeordnet werden kann.

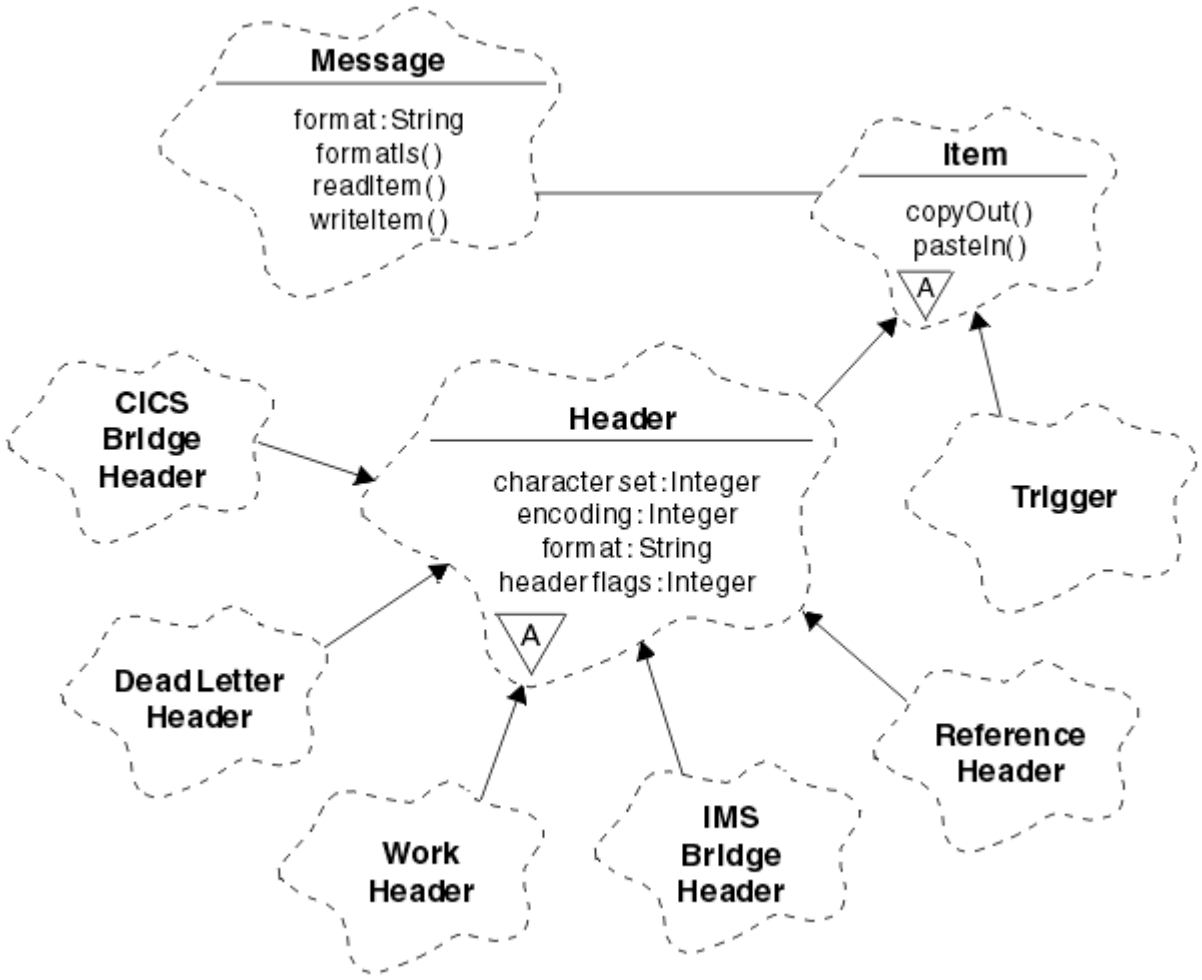


Abbildung 50. IBM MQ C++-Klassen (Behandlung von Elementen)

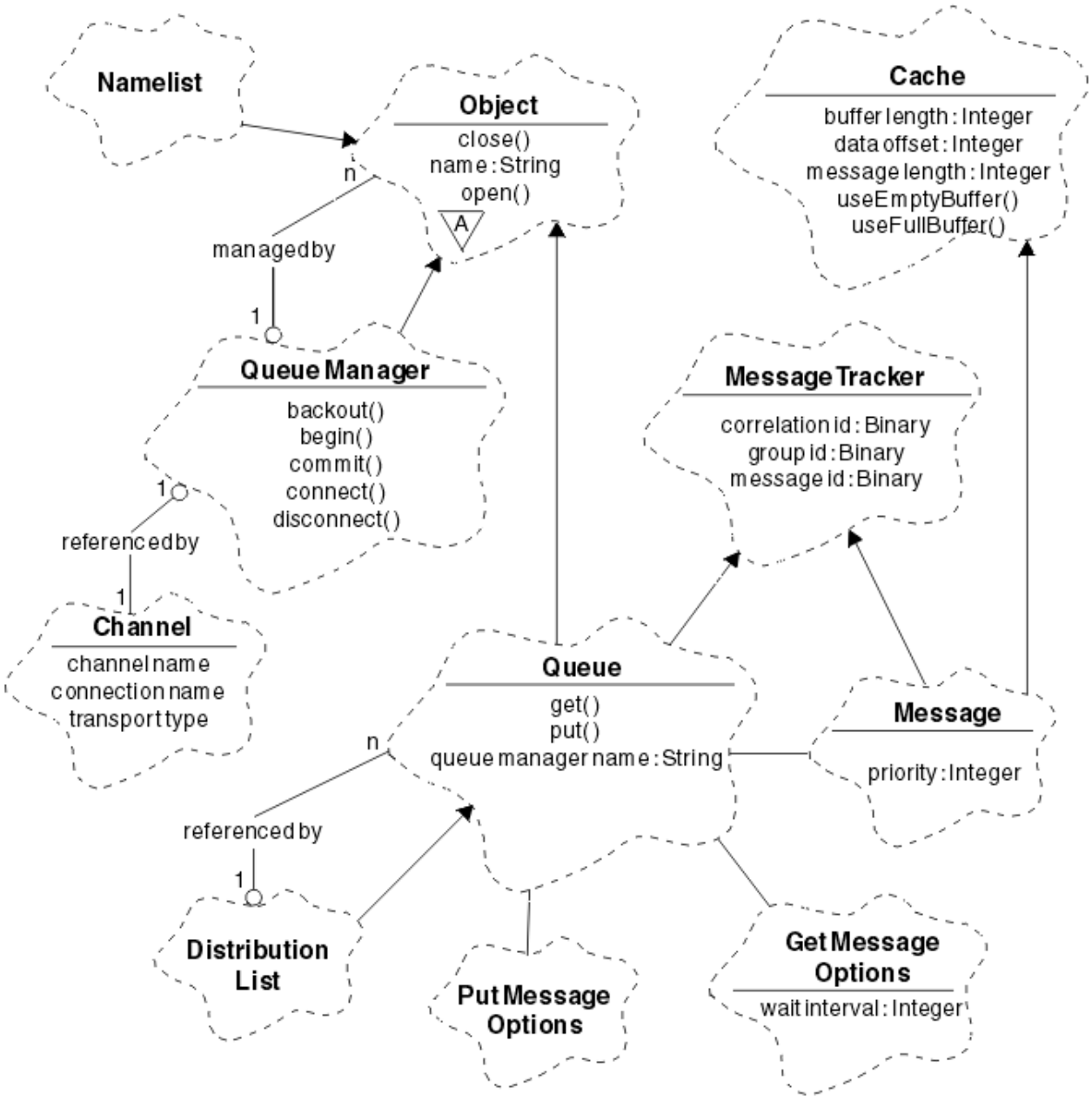


Abbildung 51. IBM MQ C++-Klassen (Warteschlangenmanagement)

Zur Interpretation von Booch-Klassendiagrammen beachten Sie die folgenden Konventionen:

- Methoden und beachtenswerte Attribute werden unter *Klasse* gezeigt.
- Ein kleines Dreieck innerhalb einer Wolke zeigt eine *abstrakte Klasse* an.
- Die *Übernahme* wird durch einen Pfeil auf die übergeordnete Klasse dargestellt.
- Eine einfache Linie zwischen Wolken zeigt eine *kooperative Beziehung* zwischen Klassen an.
- Eine Linie mit einer Zahl zeigt eine *referenzielle Beziehung* zwischen zwei Klassen an. Die Zahl gibt die Anzahl der Objekte an, die jeweils an einer bestimmten Beziehung mitwirken können.

Die folgenden Klassen und Datentypen werden in den C++-Methodensignaturen der Warteschlangenmanagementklassen (siehe Abbildung 51 auf Seite 556) und der Klassen zur Elementverarbeitung (siehe Abbildung 50 auf Seite 555) verwendet:

- Die Klasse 'ImqBinary' (siehe C++-Klasse 'ImqBinary'), die Bytefeldgruppen wie beispielsweise MQBYTE24 einbindet.

- Der Datentyp 'ImqBoolean', der als **typedef unsigned char ImqBoolean** definiert ist.
- Die Klasse 'ImqString' (siehe C++-Klasse 'ImqString'), die Zeichenbereiche wie beispielsweise MQCHAR64 einbindet.

Entitäten mit Datenstrukturen werden in geeigneten Objektklassen zusammengefasst. Mit diesen Methoden wird auf einzelne Datenstrukturfelder zugegriffen (siehe Querverweis auf C++ und MQI).

Entitäten mit Kennungen werden in die Hierarchie der Klasse 'ImqObject' gestellt (siehe C++-Klasse 'ImqObject') und stellen der MQI eingebundene Schnittstellen bereit. Objekte dieser Klassen zeigen intelligentes Verhalten, mit dem die Anzahl erforderlicher Methodenaufrufe für die prozedurale MQI reduziert werden kann. Sie können beispielsweise Warteschlangenmanagerverbindungen bei Bedarf erstellen und löschen oder eine Warteschlange mit den entsprechenden Optionen öffnen und wieder schließen.

Die Klasse 'ImqMessage' (siehe C++-Klasse 'ImqMessage') bindet die MQMD-Datenstruktur ein und fungiert außerdem als Speicher für Benutzerdaten und *Elemente* (siehe „Nachrichten in C++ lesen” auf Seite 567), indem zwischengespeicherte Pufferfunktionen bereitgestellt werden. Sie können Puffer mit fester Länge für Benutzerdaten bereitstellen und den Puffer vielfach verwenden. Das im Puffer enthaltene Datenvolumen kann von einer Verwendung zur nächsten unterschiedlich sein. Das System kann alternativ einen Puffer mit flexibler Länge bereitstellen und verwalten. Die Größe des Puffers (der verfügbare Umfang für den Empfang von Nachrichten) und der tatsächlich verwendete Umfang (die Anzahl der Byte für die Übertragung oder die Anzahl der tatsächlich empfangenen Byte) werden zu wichtigen Überlegungen.

Zugehörige Konzepte

Technische Übersicht

„Beispielprogramme in C++” auf Seite 557
In vier bereitgestellten Beispielprogrammen wird das Abrufen und Einreihen von Nachrichten veranschaulicht.

„Überlegungen zur Programmiersprache C++” auf Seite 561
In dieser Themensammlung werden die Aspekte zur Verwendung der Programmiersprache C++ sowie die Konventionen ausführlich beschrieben, die Sie beim Schreiben von Anwendungsprogrammen berücksichtigen müssen, in denen die Message Queue Interface (MQI) verwendet wird.

„Nachrichtendaten in C++ vorbereiten” auf Seite 566
Nachrichtendaten werden in einem Puffer vorbereitet, der vom System oder der Anwendung geliefert werden kann. Jede Methode hat ihre Vorteile. Es werden Beispiele für die Verwendung eines Puffers aufgeführt.

„Anwendungen für IBM MQ entwickeln” auf Seite 5
Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

Zugehörige Verweise

„IBM MQ C++-Programme erstellen” auf Seite 572
Die URL unterstützter Compiler wird zusammen mit den zu verwendenden Befehlen zum Kompilieren, Verbinden und Ausführen von C++-Programmen und Beispielen zu IBM MQ-Plattformen aufgelistet.

Querverweise zwischen C++ und MQI

IBM MQ-C++-Klassen

Beispielprogramme in C++

In vier bereitgestellten Beispielprogrammen wird das Abrufen und Einreihen von Nachrichten veranschaulicht.

Folgende Beispielprogramme werden bereitgestellt:

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

Die Beispielprogramme befinden sich in den unter Tabelle 73 auf Seite 558 gezeigten Verzeichnissen.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Tabelle 73. Position der Beispielprogramme

Umgebung	Verzeichnis mit Quelle	Verzeichnis mit erstellten Programme
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
V 9.4.0 AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ca (siehe Hinweis „1” auf Seite 558)
IBM i	`/QIBM/ProdData/mqm/samp/`	(siehe Hinweis „2” auf Seite 558)
Linux	*MQ_INSTALLATION_PATH*/samp	--
Windows	*MQ_INSTALLATI-ON_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn (siehe Hinweis „3” auf Seite 558)
z/OS	thlqual.SCSQCPPS	

Anmerkungen:

1. V 9.4.0 AIX Programme, die mit dem XLC 17-Compiler erstellt wurden, befinden sich im Ordner "`ca`", Programme, die mit dem XLC 16-Compiler erstellt wurden, im Ordner "`ia`".
2. IBM i Programme, die mithilfe des ILE C++-Compilers für IBM i erstellt wurden, befinden sich in der Bibliothek QMQM. Die Quellendateien befinden sich in `/QIBM/ProdData/mqm/samp`.
3. Windows Programme, die mithilfe von Microsoft Visual Studio Visual Studio erstellt wurden, befinden sich in *MQ_INSTALLATION_PATH*`\tools\cplus\samples\bin\vn`. Weitere Informationen zu diesen Compilern finden Sie unter „C++-Programme unter Windows erstellen” auf Seite 578.

Beispielprogramm HELLO WORLD (imqwrld.cpp)

In diesem C++-Beispielprogramm wird gezeigt, wie Sie ein ein reguläres Datagramm (C-Struktur) mithilfe der ImqMessage-Klasse einreihen und abrufen.

In diesem Beispielprogramm wird gezeigt, wie Sie ein ein reguläres Datagramm (C-Struktur) mithilfe der ImqMessage-Klasse einreihen und abrufen. Dabei werden einige Methodenaufrufe verwendet, wobei implizite Methodenaufruf wie **öffnen**, **schließen** und **trennen** genutzt werden.

Auf allen Plattformen außer z/OS

Wenn Sie eine Serververbindung zu IBM MQ verwenden, folgen Sie einer der folgenden Prozeduren:

- Um die vorhandene Standardwarteschlange SYSTEM.DEFAULT.LOCAL.QUEUE zu verwenden, führen Sie das Programm **`imqwrlds`** aus, ohne Parameter zu übergeben
- Um eine temporär dynamisch zugeordnete Warteschlange zu verwenden, führen Sie das Programm **`imqwrlds`** aus und übergeben den Namen der Standardmodellwarteschlange SYSTEM.DEFAULT.MODEL.QUEUE.

Wenn Sie eine Clientverbindung zu IBM MQ verwenden, folgen Sie einer der folgenden Prozeduren:

- Richten Sie die Umgebungsvariable MQSERVER ein (weitere Informationen finden Sie im Abschnitt *MQSERVER*) und führen Sie **imqwrldc**aus. Oder:
- Führen Sie **imqwrldc** aus, indem Sie die Parameter **queue-name**, **queue-manager-name**und **channel-definition**übergeben, wobei ein typisches **channel-definition** SYSTEM.DEF.SVRCONN/TCP/*Hostname* (1414)

unter z/OS

z/OS

Erstellen Sie einen Batch-Job mithilfe der Musterjobsteuerungssprache **imqwrldr** und führen Sie diesen aus.

Weitere Informationen finden Sie unter z/OS-Batch, RRS-Batch und CICS .

Beispielcode

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
```

```
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );

      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

Beispielprogramme SPUT (imqsput.cpp) und SGET (imqsget.cpp)

Mit diesen C++-Programmen werden Nachrichten in eine benannte Warteschlange eingereiht und von dort abgerufen.

In diesen Beispielen wird die Verwendung der folgenden Klassen gezeigt:

- ImqError (siehe ImqError C++ class)
- ImqMessage (siehe ImqMessage C++ class)
- ImqObject (siehe ImqObject C++ class)
- ImqQueue (siehe ImqQueue C++ class)
- ImqQueueManager (weitere Informationen erhalten Sie unter ImqQueueManager C++-Klasse)

Folgen Sie den entsprechenden Anweisungen, um die Programme auszuführen.

Auf allen Plattformen außer z/OS

1. Führen Sie **imqsputs** *Warteschlangenname* aus.
2. Geben Sie Textzeilen in der Konsole ein. Diese Zeilen werden als Nachrichten in die angegebene Warteschlange eingereiht.
3. Geben Sie eine Nullzeile ein, um die Eingabe abzuschließen.
4. Führen Sie **imqsgets** *Warteschlangenname* aus, um alle Zeilen abzurufen und diese in der Konsole anzuzeigen.

z/OS Weitere Informationen finden Sie im Abschnitt „Building C++ programs on z/OS Batch, RRS Batch and CICS” auf Seite 580.

unter z/OS

z/OS

1. Erstellen Sie einen Batch-Job mithilfe der Musterjobsteuerungssprache **imqsputr** und führen Sie diesen aus. Die Nachrichten werden aus dem SYSIN-Datensatz gelesen.
2. Erstellen Sie einen Batch-Job mithilfe der Musterjobsteuerungssprache **imqsgetr** und führen Sie diesen aus. Die Nachrichten werden aus der Warteschlange abgerufen und an den SYSPRINT-Datensatz gesendet.

Beispielprogramm DPUT (imqdput.cpp)

In diesem Beispielprogramm in C++ werden Nachrichten in eine Verteilerliste eingereiht, die aus zwei Warteschlangen besteht.

In DPUT wird die Verwendung der Klasse 'ImqDistributionList' gezeigt (siehe C++-Klasse 'ImqDistributionList'). Dieses Beispiel wird unter z/OS nicht unterstützt.

1. Führen Sie **imqdputs** *Warteschlangenname-1 Warteschlangenname-2* aus, um Nachrichten in die zwei benannten Warteschlangen einzureihen.
2. Führen Sie **imqsgets** *Warteschlangenname-1* und **imqsgets** *Warteschlangenname-2* aus, um die Nachrichten aus diesen Warteschlangen abzurufen.

Überlegungen zur Programmiersprache C++

In dieser Themensammlung werden die Aspekte zur Verwendung der Programmiersprache C++ sowie die Konventionen ausführlich beschrieben, die Sie beim Schreiben von Anwendungsprogrammen berücksichtigen müssen, in denen die Message Queue Interface (MQI) verwendet wird.

C++ Headerdateien

Headerdateien werden als Teil der MQI-Definition als Unterstützung zum Schreiben von IBM MQ-Anwendungsprogrammen in der Programmiersprache C++ bereitgestellt.

Diese Headerdateien werden in der folgenden Tabelle zusammengefasst.

Tabelle 74. C/C++ Headerdateien	
Dateiname	**Inhalt**
IMQI.HPP	C++ MQI Klassen (schließt CMQC.H und IMQTYPE.H ein)
IMQTYPE.H	Definiert den Datentyp **ImqBoolean**
CMQC.H	MQI-Datenstrukturen und Manifestkonstanten

Um die Portierbarkeit von Anwendungen zu verbessern, codieren Sie bei der **#include**-Vorprozessoranweisung den Namen der Headerdatei in Kleinbuchstaben:

```
#include <imqi.hpp> // C++ classes
```

C++-Methoden und Attribute

Bei Methodennamen ist Groß-/Kleinschreibung gemischt. Für Parameter und Rückgabewerte müssen verschiedene Überlegungen angestellt werden. Auf Attribute wird nach Erfordernis mittels Set- und Get-Methoden zugegriffen.

Parameter von Methoden, die *const* sind, sind nur für die Eingabe gedacht. Parameter mit Signaturen einschließlich eines Zeigers (*) oder einer Referenz (&) werden nach Referenz übergeben. Rückgabewerte, die keine Zeiger oder Referenz enthalten, werden nach Wert übergeben; im Falle von zurückgegebenen Objekten sind diese neue Entitäten, für die der Aufrufende verantwortlich ist.

Einige Methodensignaturen schließen Elemente ein, die einen Standard erhalten, wenn nichts angegeben wurde. Solche Elemente befinden sich immer am Ende von Signaturen und werden von einem Gleichheitszeichen (=) angegeben; der Wert nach dem Gleichheitszeichen ist der Standardwert, der angewendet wird, wenn das Element übergangen wird.

Bei allen Methodennamen ist Groß-/Kleinschreibung gemischt, mit Kleinbuchstaben am Anfang. Jedes Wort, mit Ausnahme des ersten innerhalb eines Methodennamens, fängt mit einem Großbuchstaben an. Abkürzungen werden nicht verwendet, außer ihre Bedeutung ist weitgehend bekannt. Zu den verwendeten Abkürzungen gehören beispielsweise *id* (für Identität) und *sync* (für Synchronisierung).

Auf Objektattribute wird mittels Set- und Get-Methoden zugegriffen. Eine Set-Methode beginnt mit dem Wort *set*; eine Get-Methode hat kein Präfix. Wenn ein Attribut *schreibgeschützt* ist, gibt es keine Set-Methode.

Attribute werden während der Objekterstellung zu einem gültigen Status initialisiert, und der Status eines Objekts ist immer konsistent.

Datentypen in C++

Alle Datentypen werden von der C-Anweisung **typedef** definiert.

Der Typ **ImqBoolean** wird in IMQTYPE.H als **unsigned char** definiert und kann die Werte WAHR und FALSCH aufweisen. Sie können **ImqBinary**-Klassenobjekte anstelle von **MQBYTE**-Arrays verwenden und **ImqString**-Klassenobjekte anstelle von **char ***. Viele Methoden geben Objekte anstelle von **char**- oder **MQBYTE**-Zeigern zurück, um die Speicherverwaltung zu erleichtern. Für alle Rückgabewerte ist der Aufrufende verantwortlich und im Falle eines zurückgegebenen Objekts kann die Speicherung mittels Löschen verworfen werden.

Bearbeiten von Binärzeichenfolgen in C++

Zeichenfolgen von Binärdaten werden als Objekte der **ImqBinary**-Klasse deklariert. Objekte dieser Klasse können mit den vertrauten C-Operatoren kopiert, verglichen und festgelegt werden. Es wird Beispielcode bereitgestellt.

Das folgende Codemuster zeigt Operationen bei einer binären Zeichenfolge:

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                            // Assign.
if ( correlationId == id ) {                   // Compare.
...
```

Bearbeiten von Zeichenfolgen in C++

Zeichendaten werden oft in **ImqString**-Klassenobjekten zurückgegeben, die mit einem Konvertierungsoperator zu **char *** umgesetzt werden können. Die ImqString-Klasse enthält Methoden zur Unterstützung der Verarbeitung von Zeichenfolgen.

Wenn Zeichendaten mithilfe von MQI C++ Methoden akzeptiert oder zurückgegeben werden, enden die Zeichendaten stets auf null und können eine beliebige Länge aufweisen. Es gelten jedoch bestimmte Einschränkungen seitens IBM MQ, die dazu führen können, dass Informationen abgeschnitten werden. Um die Speicherverwaltung zu vereinfachen, werden Zeichendaten häufig in **ImqString**-Klassenobjekten zurückgegeben. Diese Objekte können mit dem bereitgestellten Konvertierungsoperator zu **char *** umgesetzt werden und in zahlreichen Situationen, in denen **char *** erforderlich ist, für *Schreibschutz*-Zwecke verwendet werden.

Anmerkung: Das **char ***-Konvertierungsergebnis eines **ImqString**-Klassenobjekts kann null sein.

Obwohl C-Funktionen bei **char** * verwendet werden können, gibt es besondere Methoden der **ImqString**-Klasse, die vorzuziehen sind; **operator length** ( ) entspricht funktional **strlen** und **storage** ( ) zeigt den Speicher, der den Zeichendaten zugeordnet ist.

Anfangsstatus von Objekten in C++

Alle Objekte haben einen konsistenten Anfangsstatus, der sich in ihren Attributen widerspiegelt. Die Anfangswerte werden in den Klassenbeschreibungen definiert.

C von C++ verwenden

Wenn Sie C-Funktionen von einem C++-Programm verwenden, beziehen Sie zutreffende Header ein.

Im folgenden Beispiel ist `string.h` in einem C++-Programm zu sehen:

```
extern "C" {
#include <string.h>
}
```

C++-Notationskonventionen

In diesem Beispiel wird gezeigt, wie Methoden aufgerufen und Parameter deklariert werden.

Dieses Codebeispiel verwendet die Methoden und Parameter **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*)

Deklarieren und verwenden Sie die Parameter wie folgt:

```
ImqQueueManager * pmanager ;    // Queue manager
ImqQueue * pqueue ;             // Message queue
ImqMessage msg ;                // Message
char szBuffer[ 100 ];          // Buffer for message data

pmanager = new ImqQueueManager ;
pqueue = new ImqQueue ;
pqueue -> setName( "myreplyq" );
pqueue -> setConnectionReference( pmanager );
```

```
msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );

if ( pqueue -> get( msg ) ) {
  long lDataLength = msg.dataLength( );

...
}
```

Implizite Operationen in C++

Es können mehrere Operationen implizit *genau zu dem Zeitpunkt* auftreten, um die Vorbedingungen für eine erfolgreiche Ausführung einer Methode zu erfüllen. Bei diesen impliziten Operationen handelt es sich um Verbinden, Öffnen, erneut Öffnen, Schließen und Trennen. Sie können Verbinden- und Öffnen-implizites Verhalten mithilfe von Klassenattributen kontrollieren.

Verbinden

Ein ImqQueueManager-Objekt wird automatisch für Methoden verbunden, die zu einem MQI-Aufruf führen (weitere Informationen erhalten Sie unter C++ und MQI-Querverweis).

Offen

Ein ImqObject-Objekt wird automatisch für Methoden geöffnet, die einen MQGET-, MQINQ-, MQPUT- oder MQSET-Aufruf zur Folge haben. Verwenden Sie die Methode **openFor**, um einen oder mehrere relevante Werte für **Öffnen-Optionen** anzugeben.

Erneut öffnen

Ein ImqObject-Objekt wird automatisch für Methoden erneut geöffnet, die einen MQGET-, MQINQ-, MQPUT- oder MQSET-Aufruf zur Folge haben, bei dem das Objekt bereits geöffnet ist, aber bestehende **Öffnen**-Optionen nicht vorhanden sind, um einen erfolgreichen MQI-Aufruf zu ermöglichen. Das Objekt wird vorübergehend mit einem temporären Wert zum **Schließen von Optionen** von MQCO_NONE geschlossen. Verwenden Sie die Methode **openFor**, um für dieeine relevante **Öffnen-Option** hinzuzufügen.

Erneut öffnen kann unter bestimmten Umständen Probleme verursachen:

- Eine temporäre dynamische Warteschlange wird gelöscht, wenn sie geschlossen wird, und kann nicht mehr erneut geöffnet werden.
- Eine Warteschlange, die für eine ausschließliche Eingabe geöffnet wurde (entweder explizit oder standardmäßig) kann von anderen in einem Fenster während des Schließens und erneuten Öffnens aufgerufen werden.
- Wird eine Warteschlange geschlossen, geht die Position des Anzeigecursors verloren. Dieser Sachverhalt verhindert zwar nicht das Schließen und erneute Öffnen, dafür allerdings die nachfolgende Verwendung des Cursors, bis MQGMO_BROWSE_FIRST wieder verwendet wird.
- Der Kontext der letzten abgerufenen Nachricht geht verloren, wenn eine Warteschlange geschlossen wird.

Wenn einer dieser Umstände auftritt oder absehbar ist, vermeiden Sie ein erneutes Öffnen, indem Sie explizit die passenden **Öffnen-Optionen** einstellen, bevor ein Objekt (entweder explizit oder implizit) geöffnet wird.

Die explizite Einstellung der **Öffnen-Optionen** für komplexe Warteschlangenhandhabungssituationen führt zu einer besseren Leistung und vermeidet die Probleme, die mit der Verwendung von erneutem Öffnen einhergehen.

Schließen

Ein ImqObject wird automatisch zu einem Zeitpunkt geschlossen, bei dem der Objektzustand nicht mehr funktionsfähig ist, beispielsweise wenn eine ImqObject-Verbindungsreferenz beschädigt oder wenn ein ImqObject-Objekt gelöscht wird.

Verbindung trennen

Ein ImqQueueManager wird automatisch zu einem Zeitpunkt getrennt, bei dem die Verbindung nicht mehr funktionsfähig ist, beispielsweise wenn eine ImqObject-Verbindungsreferenz beschädigt oder wenn ein ImqQueueManager-Objekt gelöscht wird.

Binärzeichenfolgen und andere Zeichenfolgen in C++

Die ImqString-Klasse umfasst das konventionelle Datenformat *char **. Die ImqBinary-Klasse umfasst das binäre Byte-Array. Einige Methoden, die Zeichendaten festlegen, könnten die Daten abschneiden.

Methoden, die Zeichendaten (**char ***) festlegen, machen zwar stets eine Kopie der Daten, doch manche Methoden schneiden möglicherweise die Kopie ab, da bestimmte Grenzwerte seitens IBM MQ gelten.

Die ImqString-Klasse (weitere Informationen erhalten Sie unter ImqString C++-Klasse) umfasst das konventionelle **char *** und unterstützt:

- Vergleich
- Verkettung
- Kopieren
- Ganzzahl-zu-Text- und Text-zu-Ganzzahl-Konvertierung
- Token-Extraktion (Wort)
- Umsetzung in Großbuchstaben

Die ImqBinary-Klasse (weitere Informationen erhalten Sie unter ImqBinary C++-Klasse) umfasst binäre Byte-Arrays beliebiger Größe. Im Besonderen wird sie verwendet, um die folgenden Attribute zu enthalten:

- **Abrechnungstoken** (MQBYTE32)
- **Verbindungstag** (MQBYTE128)
- **Korrelations-ID** (MQBYTE24)
- **Funktionstoken** (MQBYTE8)
- **Gruppen-ID** (MQBYTE24)
- **Instanz-ID** (MQBYTE24)
- **Nachrichten-ID** (MQBYTE24)
- **Nachrichtentoken** (MQBYTE16)
- **Transaktionsinstanz-ID** (MQBYTE16)

Dabei gehören diese Attribute zu Objekten der folgenden Klasse:

- ImqCICSBridgeHeader (weitere Informationen erhalten Sie unter ImqCICSBridgeHeader C++-Klasse)
- ImqGetMessageOptions (weitere Informationen erhalten Sie unter ImqGetMessageOptions C++-Klasse)
- ImqIMSBridgeHeader (weitere Informationen erhalten Sie unter ImqIMSBridgeHeader C++-Klasse)
- ImqMessageTracker (weitere Informationen erhalten Sie unter ImqMessageTracker C++-Klasse)
- ImqQueueManager (weitere Informationen erhalten Sie unter ImqQueueManager C++-Klasse)
- ImqReferenceHeader (weitere Informationen erhalten Sie unter ImqReferenceHeader C++-Klasse)
- ImqWorkHeader (weitere Informationen erhalten Sie unter ImqWorkHeader C++-Klasse)

Die ImqBinary-Klasse bietet auch Unterstützung für Vergleich und Kopieren.

Nicht unterstützte Funktionen in C++

Die IBM MQ C++-Klassen und Methoden sind unabhängig von der IBM MQ-Plattform. Sie könnten deshalb einige Funktionen bieten, die auf bestimmten Plattformen nicht unterstützt werden.

Wenn Sie versuchen, eine Funktion auf einer Plattform zu verwenden, auf der sie nicht unterstützt wird, wird sie zwar von IBM MQ erkannt, aber nicht von den Programmiersprachenbindungen von C++. IBM MQ meldet den Fehler wie jeden anderen MQI-Fehler bei Ihrem Programm.

Messaging in C++

In dieser Themensammlung wird erläutert, wie Nachrichten in C++ vorbereitet, gelesen und geschrieben werden.

Nachrichtendaten in C++ vorbereiten

Nachrichtendaten werden in einem Puffer vorbereitet, der vom System oder der Anwendung geliefert werden kann. Jede Methode hat ihre Vorteile. Es werden Beispiele für die Verwendung eines Puffers aufgeführt.

Wenn Sie eine Nachricht senden, werden Nachrichtendaten zuerst in einem Puffer vorbereitet, der von einem ImqCache-Objekt verwaltet wird (weitere Informationen erhalten Sie unter ImqCache-C++-Klasse). Ein Puffer wird (durch Vererbung) mit jedem ImqMessage-Objekt verknüpft (weitere Informationen erhalten Sie unter ImqMessage-C++-Klasse): Er kann von der Anwendung (entweder mit der **useEmptyBuffer**- oder der **useFullBuffer**-Methode) oder automatisch vom System bereitgestellt werden. Der Vorteil der Bereitstellung des Nachrichtenpuffers durch die Anwendung besteht darin, dass in vielen Fällen keine Datenkopien erforderlich sind, da die Anwendung vorbereitete Datenbereiche direkt verwenden kann. Der Nachteil ist, dass der bereitgestellte Puffer eine feste Länge hat.

Der Puffer kann wiederverwendet und die Anzahl der übertragenen Bytes jedes Mal variiert werden, indem vor der Übertragung die Methode **setMessageLength** verwendet wird.

Bei einer automatischen Bereitstellung durch das System wird die Anzahl der verfügbaren Bytes vom System verwaltet, und Daten können in den Meldungspuffer kopiert werden, indem beispielsweise die ImqCache-Methode **write** oder die ImqMessage-Methode **writeItem** verwendet wird. Der Nachrichtenpuffer nimmt bei Bedarf zu. Wenn der Puffer wächst, kommt es zu keinem Verlust von zuvor geschriebenen Daten. Eine umfangreiche oder mehrteilige Nachricht kann in sequenziellen Teilen geschrieben werden.

In den folgenden Beispielen werden vereinfachte Nachrichtensendungen gezeigt.

1. Verwenden von vorbereiteten Daten in einem vom Benutzer bereitgestellten Puffer

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. Verwenden von vorbereiteten Daten in einem vom Benutzer bereitgestellten Puffer, wenn die Puffergröße die Datengröße überschreitet

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. Kopieren von Daten zu einem vom Benutzer bereitgestellten Puffer

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. Kopieren von Daten zu einem vom System bereitgestellten Puffer

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. Kopieren von Daten zu einem vom System bereitgestellten Puffer mithilfe von Objekten (Objekte legen sowohl das Nachrichtenformat als auch den Inhalt fest)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

Nachrichten in C++ lesen

Ein Puffer kann von der Anwendung oder dem System bereitgestellt werden. Daten sind direkt vom Puffer aufrufbar oder können sequenziell gelesen werden. Für jeden Nachrichtentyp gibt es eine Klasse. Es wird Beispielcode aufgeführt.

Beim Erhalt von Daten kann die Anwendung oder das System einen geeigneten Nachrichtenpuffer bereitstellen. Der gleiche Puffer kann sowohl für mehrere Übertragungen als auch für mehrere Empfänge für ein bestimmtes ImqMessage-Objekt verwendet werden. Wenn der Nachrichtenpuffer automatisch bereitgestellt wird, wächst er entsprechend der Länge der empfangenen Daten. Allerdings ist es möglich, dass ein von der Anwendung bereitgestellter Nachrichtenpuffer nicht groß genug ist, um die empfangenen Daten zu enthalten. Dann könnte entweder Abschneiden oder ein Fehler auftreten, abhängig von den für Nachrichtenempfang verwendeten Optionen.

Auf ankommende Daten kann direkt vom Nachrichtenpuffer zugegriffen werden, in welchem Fall die Datenlänge die Gesamtmenge der eingehenden Daten angibt. Alternativ können eingehende Daten sequenziell vom Nachrichtenpuffer gelesen werden. In diesem Fall widmet sich der Datenzeiger dem nächsten Byte der ankommenden Daten, und Datenzeiger sowie Datenlänge werden jedes Mal aktualisiert, wenn Daten gelesen werden.

Elemente sind Teile einer Nachricht, die sich alle im Benutzerbereich des Nachrichtenpuffers befinden und sequenziell und gesondert verarbeitet werden müssen. Außer regulären Benutzerdaten kann ein Element ein Header einer nicht zustellbaren Nachricht oder eine Auslösenachricht sein. Elemente sind immer mit Nachrichtenformaten verknüpft; Nachrichtenformate sind **nicht** immer mit Elementen verknüpft.

Es gibt für jedes Element eine Objektklasse, die einem erkennbaren IBM MQ-Nachrichtenformat entspricht. Es gibt eine für einen Header einer nicht zustellbaren Nachricht und eine für eine Auslösenachricht. Es gibt keine Objektklasse für Benutzerdaten. Das heißt, sobald die erkennbaren Formate ausgeschöpft worden sind, wird die Verarbeitung des Restes dem Anwendungsprogramm überlassen. Klassen für Benutzerdaten können durch das Anpassen der ImqItem-Klasse geschrieben werden.

Das folgende Beispiel zeigt einen Nachrichtenempfang, der mehrere potenzielle Elemente, die den Benutzerdaten in einer imaginären Situation vorangehen können, mit einbezieht. Benutzerdaten, die keine Elemente sind, werden als etwas definiert, das nach Elementen auftritt, die identifiziert werden können. Ein automatischer Puffer (der Standard) wird verwendet, um eine beliebige Menge von Nachrichtendaten zu enthalten.

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
```

```
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }

    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
      }

      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                               */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */

      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

In diesem Beispiel ist FMT_USERCLASS eine Konstante, die für einen Namen im 8-Zeichen-Format steht, der mit einer Objektklasse `UserClass` verknüpft ist und von der Anwendung definiert wird.

`UserClass` wird von der ImqItem-Klasse abgeleitet (weitere Informationen erhalten Sie unter ImqItem-C++-Klasse) und implementiert die virtuellen **copyOut**- und **pasteIn**-Methoden dieser Klasse.

Die nächsten zwei Beispiele zeigen Code der ImqDeadLetterHeader-Klasse (weitere Informationen erhalten Sie unter ImqDeadLetterHeader-C++-Klasse). Im ersten Beispiel ist der angepasste, eingebundene Nachrichten-*Schreib*code zu sehen.

```
// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
  ImqBoolean bSuccess ;
```

```
  if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
    ImqCache cacheData( msg ); // Preserve original message content.
    // Note original message attributes in the dead-letter header.
    setEncoding( msg.encoding( ) );
    setCharacterSet( msg.characterSet( ) );
    setFormat( msg.format( ) );

    // Set the message attributes to reflect the dead-letter header.
    msg.setEncoding( MQENC_NATIVE );
    msg.setCharacterSet( MQCCSI_Q_MGR );
    msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
    // Replace the existing data with the dead-letter header.
    msg.clearMessage( );
    if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
      // Append the original message data.
      bSuccess = msg.write( cacheData.messageLength( ),
                            cacheData.bufferPointer( ) );
    } else {
      bSuccess = FALSE ;
    }
  } else {
    bSuccess = FALSE ;
  }
  // Reflect and cache error in this object.
  if ( ! bSuccess ) {
    setReasonCode( msg.reasonCode( ) );
    setCompletionCode( msg.completionCode( ) );
  }

  return bSuccess ;
}
```

Im zweiten Beispiel ist der angepasste, eingebundene Nachrichten-*Lese*code zu sehen.

```
// Read a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
  ImqBoolean bSuccess = FALSE ;

  // First check that the eye-catcher is correct.
  // This is also our guarantee that the "character set" is correct.
  if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
    // Next check that the "encoding" is correct, as the MQDLH
    // contains numeric data.
    if ( msg.encoding( ) == MQENC_NATIVE ) {

      // Finally check that the "format" is correct.
      if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
        char * pszBuffer = (char *) & omqdlh ;
        // Transfer the MQDLH from the message and move pointer on.
        if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
          // Update the encoding, character set and format of the
          // message to reflect the remaining data.
          msg.setEncoding( encoding( ) );
          msg.setCharacterSet( characterSet( ) );
          msg.setFormat( format( ) );
        } else {

          // Reflect the cache error in this object.
          setReasonCode( msg.reasonCode( ) );
          setCompletionCode( msg.completionCode( ) );
        }
      } else {
        setReasonCode( MQRC_INCONSISTENT_FORMAT );
        setCompletionCode( MQCC_FAILED );
      }
    } else {
      setReasonCode( MQRC_ENCODING_ERROR );
      setCompletionCode( MQCC_FAILED );
    {
  } else {
    setReasonCode( MQRC_STRUC_ID_ERROR );
    setCompletionCode( MQCC_FAILED );
  }

  return bSuccess ;
}
```

Mit einem automatischen Puffer ist der Pufferspeicher *flüchtig*. Das heißt, Pufferdaten könnten nach jedem **get**-Methodenaufruf an einem anderen physischen Standort gehalten werden. Verwenden Sie deshalb bei jeder Referenzierung von gepufferten Daten die **bufferPointer**- oder **dataPointer**-Methoden, um auf Nachrichtendaten zuzugreifen.

Sie könnten von einem Programm einen festgelegten Bereich für den Empfang von Nachrichtendaten bereitstellen lassen. Rufen Sie in diesem Fall die **useEmptyBuffer**-Methode auf, bevor Sie die **get**-Methode verwenden.

Die Verwendung eines feststehenden, nichtautomatischen Bereichs grenzt Nachrichten auf eine maximale Größe ein, d. h. es ist wichtig, die MQGMO_ACCEPT_TRUNCATED_MSG-Option des ImqGetMessageOptions-Objekts zu berücksichtigen. Wenn diese Option nicht angegeben wird (Standard), kann der MQRC_TRUNCATED_MSG_FAILED-Ursachencode erwartet werden. Wird diese Option angegeben, könnte der MQRC_TRUNCATED_MSG_ACCEPTED-Ursachencode je nach Design der Anwendung erwartet werden.

Das nächste Beispiel zeigt, wie ein feststehender Speicherbereich verwendet werden kann, um Nachrichten zu erhalten:

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

In diesem Codefragment kann der Puffer stets direkt mittels *pszBuffer* abgefragt werden, anstatt die **bufferPointer**-Methode zu verwenden. Allerdings ist es besser, die **dataPointer**-Methode für den allgemeinen Zugriff zu verwenden. Die Anwendung (nicht das ImqCache-Klassenobjekt) muss einen benutzerdefinierten (nichtautomatischen) Puffer löschen.

Achtung: Beim Angeben eines Nullzeigers und der Länge null mit **useEmptyBuffer** wird kein Puffer fester Länge mit der Länge null benannt, wie erwartet werden könnte. Diese Kombination wird als Anforderung zum Ignorieren von vorherigen benutzerdefinierten Puffern interpretiert und stattdessen auf die Verwendung eines automatischen Puffers zurückzugreifen.

Nachrichten in die Warteschlange für nicht zustellbare Nachrichten in C++ schreiben

Beispielprogrammcode für das Schreiben einer Nachricht in die Warteschlange für nicht zustellbare Nachrichten.

Ein typischer Fall einer mehrteiligen Nachricht ist eine, die einen Header einer nicht zustellbaren Nachricht enthält. Die Daten von einer Nachricht, die nicht verarbeitet werden kann, werden an den Header einer nicht zustellbaren Nachricht angehängt.

```
ImqQueueManager mgr ;        // The queue manager.
ImqQueue queueIn ;           // Incoming message queue.
ImqQueue queueDead ;         // Dead-letter message queue.
ImqMessage msg ;             // Incoming and outgoing message.
ImqDeadLetterHeader header ; // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );
```

```
// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

Nachrichten zur IMS-Brücke in C++ schreiben

Beispielprogrammcode für das Schreiben einer Nachricht zur IMS-Brücke.

Nachrichten, die an die IBM MQ - IMS-Brücke gesendet wurden, verwenden möglicherweise einen besonderen Header. Der IMS-Bridge-Header wird regulären Nachrichtendaten als Präfix vorangestellt.

```
ImqQueueManager    mgr;         // The queue manager.
ImqQueue           queueBridge; // IMS bridge message queue.
ImqMessage         msg;         // Outgoing message.
ImqIMSBridgeHeader header;      // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2,       /* ? */ );         // Total message length.
msg.write( 2,       /* ? */ );         // IMS flags.
msg.write( 7,       /* ? */ );         // Transaction code.
msg.write( /* ? */, /* ? */ );         // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.

// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

Nachricht zur CICS bridge in C++ schreiben

Beispielprogrammcode für das Schreiben einer Nachricht zur CICS bridge.

Nachrichten, die zur IBM MQ for z/OS mithilfe der CICS bridge gesendet wurden, erfordern einen besonderen Header. Der CICS bridge-Header wird regulären Nachrichtendaten als Präfix vorangestellt.

```
ImqQueueManager mgr ;         // The queue manager.
ImqQueue queueIn ;            // Incoming message queue.
ImqQueue queueBridge ;        // CICS bridge message queue.
ImqMessage msg ;              // Incoming and outgoing message.
ImqCicsBridgeHeader header ;  // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
```

```
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

Nachrichten mit einem Work-Header C++ schreiben

Beispielprogrammcode für das Schreiben einer Nachricht, die für eine Warteschlange bestimmt ist, die von z/OS Workload Manager verwaltet wird.

Nachrichten, die an IBM MQ for z/OS gesendet werden und für eine Warteschlange bestimmt sind, die vom z/OS Workload Manager gewartet wird, erfordern einen besonderen Header. Der Work-Header wird regulären Nachrichtendaten als Präfix vorangestellt.

```
ImqQueueManager mgr ;             // The queue manager.
ImqQueue queueIn ;                // Incoming message queue.
ImqQueue queueWLM ;               // WLM managed queue.
ImqMessage msg ;                  // Incoming and outgoing message.
ImqWorkHeader header ;            // Work header information

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the WLM managed queue.
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

IBM MQ C++-Programme erstellen

Die URL unterstützter Compiler wird zusammen mit den zu verwendenden Befehlen zum Kompilieren, Verbinden und Ausführen von C++-Programmen und Beispielen zu IBM MQ-Plattformen aufgelistet.

Eine Liste der Compiler für jede unterstützte Plattform und IBM MQ-Version finden Sie im Abschnitt Systemvoraussetzungen für IBM MQ.

Der Befehl, mit dem Sie das IBM MQ C++-Programm kompilieren und verbinden müssen, hängt von der Installation und den Anforderungen ab. Nachfolgende Beispiele zeigen typische Compiler- und Linker-Befehle für einige der Compiler, die auf verschiedensten Plattformen die Standardinstallation von IBM MQ verwenden.

AIX C++-Programme unter AIX erstellen

IBM MQ C++-Programme auf AIX mit dem Compiler der XL C Enterprise Edition erstellen.

V 9.4.0 Weitere Informationen zur unterschiedlichen Zuordnung von Compileroptionen zwischen XLC 16-und XLC 17 -Compilern finden Sie unter Zuordnung von Optionen.

Deprecated V 9.4.0 Die Unterstützung für den Compiler XL C/C++ für AIX 16 unter AIX wird ab IBM MQ 9.4.0nicht mehr verwendet.

Client

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

32-Bit Unthread-Anwendung

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32-Bit Thread-Anwendung

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

64-Bit Unthread-Anwendung

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

64-Bit Thread-Anwendung

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

V 9.4.0 32-Bit-Anwendung ohne Threads (XLC 17)

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

V 9.4.0 32-Bit-Thread-Anwendung (XLC 17)

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

V 9.4.0 64-Bit-Anwendung ohne Threads (XLC 17)

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

V 9.4.0 64-Bit-Thread-Anwendung (XLC 17)

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

Server

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

32-Bit Unthread-Anwendung

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32-Bit Thread-Anwendung

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

64-Bit Unthread-Anwendung

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

64-Bit Thread-Anwendung

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

V 9.4.0 **32-Bit-Anwendung ohne Threads (XLC 17)**

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

V 9.4.0 **32-Bit-Thread-Anwendung (XLC 17)**

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

V 9.4.0 **64-Bit-Anwendung ohne Threads (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

V 9.4.0 **64-Bit-Thread-Anwendung (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

IBM i C++-Programme unter IBM i erstellen

IBM MQ C++-Programme auf IBM i mit dem ILE C++-Compiler erstellen.

IBM ILE C++ für IBM i ist ein nativer Compiler für Programme in C++. Nachfolgend wird beschrieben, wie mit diesem Compiler IBM MQ C++-Anwendungen erstellt werden. Hierbei wird das Beispielprogramm *Hello World!* von IBM MQ als Muster verwendet.

1. Installieren Sie den ILE C++ für IBM i-Compiler gemäß den Anweisungen des Handbuchs *Read Me first!* (Achtung, bitte zuerst lesen/Wichtige Hinweise), das dem Produkt beiliegt.
2. Stellen Sie sicher, dass die QCXXN-Bibliothek in Ihrer Bibliotheksliste ist.
3. Erstellen Sie das Beispielprogramm HELLO WORLD:
   a. Erstellen Sie ein Modul:

```
CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
TERASPACE(*YES)
```

   Die Quelle für die C++-Beispielprogramme ist in `/QIBM/ProdData/mqm/samp` zu finden und die einzuschließenden Dateien in `/QIBM/ProdData/mqm/inc`.

   Alternativ kann die Quelle in der Bibliothek `SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)` gefunden werden.

b. Binden Sie dies mit von IBM MQ bereitgestellten Serviceprogrammen, um ein Programmobjekt zu erzeugen:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
```

Um eine Thread-Anwendung zu erstellen, verwenden Sie die simultan verwendbaren Serviceprogramme:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
```

c. Führen Sie das Beispielprogramm HELLO WORLD unter Verwendung von SYSTEM.DEFAULT.LOCAL.QUEUE aus:

```
CALL PGM(MYLIB/IMQWRLD)
```

Linux C++-Programme unter Linux erstellen

IBM MQ C++-Programme auf Linux mit dem GNU g++-Compiler erstellen.

System p

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client: System p

32-Bit Unthread-Anwendung

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

32-Bit Thread-Anwendung

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

64-Bit Unthread-Anwendung

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64-Bit Thread-Anwendung

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

Server: System p

32-Bit Unthread-Anwendung

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
```

```
-limqs23gl
-limqb23gl -lmqm
```

32-Bit Thread-Anwendung

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

64-Bit Unthread-Anwendung

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64-Bit Thread-Anwendung

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client: IBM Z

32-Bit Unthread-Anwendung

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

32-Bit Thread-Anwendung

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

64-Bit Unthread-Anwendung

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64-Bit Thread-Anwendung

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Server: IBM Z

32-Bit Unthread-Anwendung

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

32-Bit Thread-Anwendung

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

64-Bit Unthread-Anwendung

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64-Bit Thread-Anwendung

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64 (32 Bit)

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client: x86-64 (32 Bit)

32-Bit Unthread-Anwendung

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATI
ON_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

32-Bit Thread-Anwendung

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

64-Bit Unthread-Anwendung

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L MQ_INSTALLATI
ON_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

64-Bit Thread-Anwendung

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L MQ_INSTALLATI
ON_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Server: x86-64 (32 Bit)

32-Bit Unthread-Anwendung

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

32-Bit Thread-Anwendung

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

64-Bit Unthread-Anwendung

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L MQ_INSTALLATI
ON_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

64-Bit Thread-Anwendung

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L MQ_INSTALLATI
ON_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows C++-Programme unter Windows erstellen

Erstellen Sie IBM MQ-C++-Programme unter Windows mithilfe des C++-Compilers von Microsoft Visual Studio.

Achtung: Bei den von IBM MQ bereitgestellten Bibliotheken handelt es sich um dynamische Bibliotheken und nicht um statische Bibliotheken. IBM MQ stellt sogenannte „`import libraries`“ bereit, die Sie nur während der Kompilierzeit verwenden können. Während der Laufzeit müssen Sie die dynamischen Bibliotheken verwenden.

Aus IBM MQ 8.0.0 Fix Pack 4 stellt IBM MQ weiterverteilbare Clients bereit, die Bibliotheken enthalten, die für die Ausführung von IBM MQ-Anwendungen erforderlich sind. Diese Bibliotheken können paketiert und mit Clientanwendungen weiterverteilt werden. Weitere Informationen finden Sie im Abschnitt Weiterverteilbare Clients unter Windows.

Bibliotheksdateien (.lib) und DLL-Dateien zur Nutzung mit 32-Bit-Anwendungen sind in *MQ_INSTALLATION_PATH*`/Tools/Lib` installiert. Dateien zur Nutzung mit 64-Bit-Anwendungen sind in *MQ_INSTALLATION_PATH*`/Tools/Lib64` installiert. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

Server

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

Universal C Runtime installieren

Wenn Sie Windows 8.1 oder Windows Server 2012 R2 verwenden, müssen Sie das Update für Universal C Runtime (Universal CRT) von Microsoft installieren. Diese Runtime ist Bestandteil von Windows 10 und Windows Server 2016.

Bei dem Update für Universal CRT handelt es sich um das Microsoft-Update KB3118401. Ob diese Aktualisierung vorhanden ist, können Sie überprüfen, indem Sie in Ihrem Verzeichnis `C:\Windows\System32` nach einer Datei namens `ucrtbase.dll` suchen. Falls die Datei nicht vorhanden ist, kön-

nen Sie das Update von folgender Microsoft-Webseite herunterladen: https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401.

Der Versuch, ein IBM MQ-Programm oder ein Programm, das Sie selbst mit Microsoft Visual Studio 2017 kompiliert haben, auszuführen, ohne dass die Runtime installiert ist, führt zu Fehlern wie dem folgenden:

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

Runtimes für Microsoft Visual Studio 2012-Programme bereitstellen

Wenn Sie ein IBM MQ-Programm mit Microsoft Visual Studio 2012 kompiliert haben, müssen Sie darauf achten, dass das IBM MQ-Installationsprogramm nicht die C/C++-Runtimes von Microsoft Visual Studio 2012 installiert. Falls Ihre frühere Version von IBM MQ auf demselben Computer installiert wurde, sind die Microsoft Visual Studio 2012-Runtimes aus dieser Installation verfügbar.

Wenn Sie jedoch ein Programm verwenden, das mit Microsoft Visual Studio 2012 erstellt wurde, und keine frühere Version von IBM MQ installiert wurde, müssen Sie eine der folgenden Aktionen ausführen:

- Laden Sie **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** von Microsoftherunter und installieren Sie es.
- Führen Sie eine Neukompilierung Ihres Programms mit Microsoft Visual Studio 2017 oder einer anderen Version von Microsoft Visual Studio durch, für die die Runtimes installiert sind.

C++-Clientbibliotheken, die mithilfe des Microsoft Visual Studio 2015-Compilers erstellt wurden

IBM MQ stellt C++-Clientbibliotheken bereit, die mit dem C++-Compiler von Microsoft Visual Studio 2015 und dem C++-Compiler von Microsoft Visual Studio 2017 erstellt wurden.

Es werden sowohl 32-Bit- als auch 64-Bit-Versionen der IBM MQ C++-Bibliotheken bereitgestellt. Während die 32-Bit-Bibliotheken im Ordner `bin\vs2015` installiert sind, befinden sich die 64-Bit-Bibliotheken unter den Ordnern `bin64\vs2015`.

IBM MQ ist standardmäßig für die Verwendung der Bibliotheken von Microsoft Visual Studio 2017 konfiguriert. Wenn die Bibliotheken von Microsoft Visual Studio 2015 verwendet werden sollen, müssen Sie die Umgebungsvariable `MQ_PREFIX_VS_LIBRARIES` auf `MQ_PREFIX_VS_LIBRARIES=vs2015` setzen, bevor Sie IBM MQ installieren bzw. bevor Sie den Befehl **setmqenv** oder **setmqinst** verwenden.

Anders benannte IBM MQ-C++-Bibliotheken verwenden

IBM MQ stellt einige zusätzliche C++-Clientbibliotheken bereit, die anders benannt sind. Diese Bibliotheken werden mit den C++-Compilern von Microsoft Visual Studio 2015 und Microsoft Visual Studio 2017 erstellt. Sie werden zusätzlich zu den vorhandenen C++-Bibliotheken bereitgestellt, die ebenfalls mit dem C++-Compiler von Microsoft Visual Studio 2017 erstellt werden. Da diese zusätzlichen IBM MQ-C++-Bibliotheken andere Namen haben, können Sie IBM MQ-C++-Anwendungen, die mit IBM MQ-C++ erstellt und mit Microsoft Visual Studio 2017 und früheren Versionen des Produkts kompiliert wurden, auf demselben Computer ausführen.

Die zusätzlichen Microsoft Visual Studio 2017-Bibliotheken haben folgende Namen:

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

Die zusätzlichen Microsoft Visual Studio 2015-Bibliotheken haben folgende Namen:

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`

- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

Es werden sowohl 32-Bit- als auch 64-Bit-Versionen dieser Bibliotheken bereitgestellt. Die 32-Bit-Bibliotheken werden unter dem Ordner `bin` installiert, die 64-Bit-Bibliotheken unter dem Ordner `bin64`. Entsprechende Importbibliotheken werden in den Verzeichnissen `Tools\lib` und `Tools\lib64` installiert.

Falls Ihre Anwendung Dateien `imq*vs2015.lib` verwendet, müssen Sie diese mit dem Microsoft Visual Studio 2015-Compiler kompilieren. Um IBM MQ-C++-Anwendungen, die mit Microsoft Visual Studio 2015 kompiliert werden, oder Anwendungen, die mit einer früheren Version des Produkts kompiliert werden, auf demselben Computer auszuführen, muss die Umgebungsvariable PATH so wie in den folgenden Beispielen mit einem Präfix versehen werden:

- Für 32-Bit-Anwendungen:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- Für 64-Bit-Anwendungen:

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

Zugehörige Konzepte
Windows: Änderungen ab IBM MQ 8.0

z/OS Building C++ programs on z/OS Batch, RRS Batch and CICS

Build IBM MQ C++ programs on z/OS for the Batch, RRS batch or CICS environments and run the sample programs.

You can write C++ programs for three of the environments that IBM MQ for z/OS supports:

- Batch
- RRS batch
- CICS

Compile, prelink and link

Create an z/OS application by compiling, pre-linking, and link-editing your C++ source code.

IBM MQ C++ for z/OS is implemented as z/OS DLLs for the IBM C++ for z/OS language. Using DLLs, you concatenate the supplied definition sidedecks with the compiler output at pre-link time. This allows the linker to check your calls to the IBM MQ C++ member functions.

Note: There are three sets of sidedecks for each of the three environments.

To build an IBM MQ for z/OS C++ application, create and run JCL. Use the following procedure:

1. If your application runs under CICS, use the CICS-supplied procedure to translate CICS commands in your program.

   In addition, for CICS applications you need to:

   a. Add the SCSQLOAD library to the DFHRPL concatenation.

   b. Define the CSQCAT1 CEDA group using the member IMQ4B100 in the SCSQPROC library.

   c. Install CSQCAT1.
2. Compile the program to produce object code. The JCL for your compilation must include statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

Be sure to specify the `/cxx` compiler option.

Note: The name **thlqual** is the high level qualifier of the IBM MQ installation library on z/OS.

3. Pre-link the object code created in step “2” on page 580, including the following definition sidedecks, which are supplied in **thlqual**.SCSQDEFS:
   a. imqs23dm and imqb23dm for batch
   b. imqs23dr and imqb23dr for RRS batch
   c. imqs23dc and imqb23dc for CICS

   These are the corresponding DLLs.
   a. imqs23im and imqb23im for batch
   b. imqs23ir and imqb23ir for RRS batch
   c. imqs23ic and imqb23ic for CICS
4. Link-edit the object code created in step “3” on page 581, to produce a load module, and store it in your application load library.

To run batch or RRS batch programs, include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB or JOBLIB data set concatenation.

To run a CICS program, first get your system administrator to define it to CICS as an IBM MQ program and transaction. You can then run it in the usual way.

Run the sample programs

The programs are described in “Beispielprogramme in C++” on page 557.

The sample applications are supplied in source form only. The files are:

Table 75. z/OS sample program files

Sample	Source program (in library thlqual.SCSQCPPS)	JCL (in library thlqual.SCSQPROC)
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

To run the samples, compile and link-edit them as with any C++ program (see “Building C++ programs on z/OS Batch, RRS Batch and CICS” on page 580). Use the supplied JCL to construct and run a batch job. You must initially customize the JCL, by following the commentary included with it.

z/OS Building C++ programs on z/OS UNIX System Services

Build IBM MQ C++ programs on z/OS UNIX System Services (z/OS UNIX).

To build an application under the z/OS UNIX shell, you must give the compiler access to the IBM MQ include files (located in `thlqual.SCSQC370` and `hlqual.SCSQHPPS`), and link against two of the DLL sidedecks (located in `thlqual.SCSQDEFS`). At runtime, the application needs access to the IBM MQ data sets `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH`, and one of the language specific data sets, such as `thlqual.SCSQANLE` [6].

[6] You can link with any of the sidedecks listed in "Pre-link the object code to run your z/OS UNIX in any of the three environments, “Building C++ programs on z/OS Batch, RRS Batch and CICS” on page 580

Compiling

1. Copy the sample into the file system using the TSO **oput** command, or use FTP. The rest of this example assumes that you have copied the sample into a directory called `/u/fred/sample`, and named it `imqwrld.cpp`.
2. Log into the z/OS UNIX shell, and change to the directory where you placed the sample.
3. Set up the C++ compiler so that it can accept the DLL sidedeck and .cpp files as input:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. Compile and link the sample program. The following command links the program with the batch sidedecks; the RRS batch sidedecks can be used instead. The \ character is used to split the command over more than one line. Do not enter this character; enter the command as a single line:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

For more information on the TSO **oput** command, refer to the z/OS UNIX Command Reference.

You can also use the make utility to simplify building C++ programs. Here is a sample makefile to build the HELLO WORLD C++ sample program. It separates the compiling and linking stages. Set up the environment as in step “3” on page 582 before running make.

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

Refer to z/OS UNIX System Services Programming Tools for more information on using make.

Running

1. Log into the z/OS UNIX shell, and change to the directory where you built the sample.
2. Set up the STEPLIB environment variable to include the IBM MQ data sets:

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. Run the sample:

```
/u/fred/sample:> ./imqwrld
```

.NET-Anwendungen entwickeln

IBM MQ classes for .NET ermöglichen .NET -Anwendungen, eine Verbindung zu IBM MQ als IBM MQ MQI client oder direkt zu einem IBM MQ -Server herzustellen.

Vorbereitende Schritte

V 9.4.0 Deprecated V 9.4.0 Ab IBM MQ 9.4.0sind in IBM MQ classes for .NETdie Methoden WriteObject(), ReadObject(), CreateObjectMessage () und die Klassen ObjectMessage und XmsObject-MessageImpl , die für die Serialisierung und Deserialisierung von Daten verwendet werden, veraltet.

V 9.4.0 V 9.4.0 Removed Die mit .NET Standard 2.0erstellte IBM MQ .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

Informationen zu diesem Vorgang

Bei IBM MQ classes for .NET handelt es sich um eine Gruppe von Klassen, mit denen .NET-Anwendungen mit IBM MQ interagieren können. Sie stellen die verschiedenen Komponenten von IBM MQ dar, die von Ihrer Anwendung verwendet werden, z. B. Warteschlangenmanager, Kanäle und Nachrichten. Weitere Informationen zu diesen Klassen finden Sie unter IBM MQ .NET -Klassen und -Schnittstellen.

V 9.4.0 IBM MQ 9.4.0 stellt eine IBM MQ .NET -Clientbibliothek bereit, die für .NET 6 als Zielframework erstellt wurde. Weitere Informationen finden Sie unter „Installieren von IBM MQ classes for .NET” auf Seite 584.

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0unterstützt IBM MQ .NET 8 -Anwendungen, die IBM MQ classes for .NETverwenden. Weitere Informationen finden Sie unter „Installieren von IBM MQ classes for .NET” auf Seite 584.

Wenn Sie Anwendungen haben, die Microsoft .NET Framework verwenden und die Funktionen von IBM MQnutzen möchten, müssen Sie IBM MQ classes for .NET Frameworkverwenden. Weitere Informationen finden Sie unter „Installieren von IBM MQ classes for .NET Framework” auf Seite 590.

Weitere Informationen zu den Unterschieden zwischen IBM MQ classes for .NET Framework und IBM MQ classes for .NETfinden Sie unter „Installieren von IBM MQ classes for .NET” auf Seite 584.

Von IBM MQ .NET verwaltete Anwendungen können automatisch Verbindungen zwischen Clusterwarteschlangenmanagern ausgleichen. Sowohl die IBM MQ classes for .NET -als auch die IBM MQ classes for .NET Framework -Bibliotheken werden unterstützt. Weitere Informationen finden Sie unter Informationen zu Uniform-Clustern und Automatische Verteilung der Anwendungslast.

Die objektorientierte IBM MQ .NET-Schnittstelle unterscheidet sich von der MQI-Schnittstelle darin, dass sie Methoden von Objekten anstelle der MQI-Verben verwendet. Die prozedurale IBM MQ-Anwendungsprogrammierschnittstelle ist um Verben wie die in der folgenden Liste gezeigten aufgebaut:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Alle diese Verben verwenden als Parameter ein Handle für das IBM MQ-Objekt, das sie bearbeiten sollen. Da .NET objektorientiert ist, wird dieser Prozess von der .NET-Programmierschnittstelle umgekehrt. Ihr Programm besteht aus einer Reihe von IBM MQ-Objekten, die Sie durch Aufrufen von Methoden für diese Objekte bearbeiten können. Sie können Programme in allen Sprachen schreiben, die von .NET unterstützt werden.

Bei der Verwendung der prozedurorientierten Schnittstelle trennen Sie die Verbindung zu einem Warteschlangenmanager mit dem Aufruf 'MQDISC(*Hconn*, CompCode, Reason)'. Dabei steht *Hconn* für die Kennung des Warteschlangenmanagers. In der .NET-Schnittstelle wird der Warteschlangenmanager von einem Objekt der Klasse MQQueueManager dargestellt. Sie trennen die Verbindung zum Warteschlangenmanager mit dem Aufruf der Methode 'Disconnect()' in dieser Klasse.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

Zugehörige Konzepte

Technische Übersicht

„Anwendungen für IBM MQ entwickeln” auf Seite 5

Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

Zugehörige Tasks

Anfordern von Unterstützung beim IBM Support

Fehlerbehebung bei Problemen in IBM MQ .NET

„Microsoft Windows Communication Foundation -Anwendungen mit IBM MQ entwickeln" auf Seite 1327
Der angepasste WCF-Kanal (Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

„XMS .NET-Anwendungen entwickeln" auf Seite 646
IBM MQ Message Service Client (XMS) for .NET (kurz: XMS .NET) stellt eine Anwendungsprogrammierschnittstelle (API) namens XMS bereit, die dieselben Schnittstellen besitzt wie die API für Java Message Service (kurz: JMS). IBM MQ Message Service Client (XMS) for .NET enthält eine vollständig verwaltete Implementierung von XMS, die von jeder mit .NET kompatiblen Sprache verwendet werden kann.

Windows Linux Installieren von IBM MQ classes for .NET

IBM MQ classes for .NET, einschließlich Beispiele, werden mit IBM MQ unter Windows und Linux installiert.

Voraussetzungen und Installation

V 9.4.0 IBM MQ 9.4.0 stellt eine IBM MQ .NET -Clientbibliothek bereit, die für .NET 6 als Zielframework erstellt wurde. Ab IBM MQ 9.4.0ist Microsoft .NET 6.0 die mindestens erforderliche Version für aktive Anwendungen, die IBM MQ -Bibliotheken verwenden, die mit .NET 6 als Zielframework erstellt wurden. Die IBM MQ .NET -Clientbibliothek, die mit .NET 6 als Zielframework erstellt wurde, ist unter *MQ_INSTALLATION_PATH*`/bin` unter Windows und unter *MQ_INSTALLATION_PATH*`/lib64` unter Linuxverfügbar.

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0unterstützt IBM MQ .NET 8 -Anwendungen, die IBM MQ classes for .NETverwenden. Wenn Sie eine .NET 6 -Anwendung verwenden, können Sie diese Anwendung ausführen, ohne dass eine erneute Kompilierung erforderlich ist, indem Sie eine kleine Bearbeitung in der Datei `runtimeconfig` vornehmen, um `targetframeworkversion` auf `"net8.0"`zu setzen.

V 9.4.0 Deprecated V 9.4.0 Ab IBM MQ 9.4.0sind in IBM MQ classes for .NETdie Methoden WriteObject(), ReadObject(), CreateObjectMessage () und die Klassen ObjectMessage und XmsObjectMessageImpl , die für die Serialisierung und Deserialisierung von Daten verwendet werden, veraltet.

V 9.4.0 V 9.4.0 Removed Die mit .NET Standard 2.0erstellte IBM MQ .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

Die neueste Version von IBM MQ classes for .NET wird standardmäßig als Teil der Standardinstallation von IBM MQ im Feature *Java and .NET Messaging and Web Services* installiert.

Windows Quellen für weitere Informationen zu Voraussetzungen und zur Installation unter Windows:

- Informationen zu den Softwarevoraussetzungen für die Ausführung von IBM MQ classes for .NETfinden Sie unter Voraussetzungen für IBM MQ classes for .NET .
- Installationsanweisungen finden Sie im Abschnitt IBM MQ -Server unter Windows installieren oder IBM MQ-Client auf Windows-Systemen installieren.

Linux Quellen für weitere Informationen zu Voraussetzungen und zur Installation unter Linux:

- Informationen zu den Softwarevoraussetzungen für die Ausführung von IBM MQ classes for .NETfinden Sie unter Voraussetzungen für IBM MQ classes for .NET .
- RPM-Installationsanweisungen finden Sie im Abschnitt IBM MQ-Client auf Linux-Systemen installieren.
- Informationen zu Linux Ubuntu (mit Debian-Paketen) finden Sie im Abschnitt IBM MQ-Client auf Linux-Systemen installieren.

Die IBM MQ classes for .NET Standard-Bibliothek `amqmdnetstd.dll` ist im NuGet-Repository zum Download verfügbar. Weitere Informationen finden Sie unter „IBM MQ classes for .NET aus dem NuGet -Repository herunterladen” auf Seite 589.

Bibliothek `amqmdnetstd.dll`

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0ist die Bibliothek `amqmdnetstd.dll` , die mit .NET 6 als Zielframework erstellt wurde, an den folgenden Positionen verfügbar:

- Windows Unter Windows: *MQ_INSTALLATION_PATH*`\bin`. Die Beispielanwendungen werden in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`installiert.
- Linux Unter Linux: *MQ_INSTALLATION_PATH*`\lib64`. Die .NET -Beispiele befinden sich in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

Achtung: V 9.4.0 V 9.4.0 Removed Ab IBM MQ 9.4.0werden IBM MQ .NET -Clientbibliotheken, die mit .NET Standard 2.0 als Zielframework erstellt wurden, entfernt. Diese Bibliotheken sind in IBM MQ 9.3.1veraltet.

Stabilized LTS Die Bibliothek `amqmdnet.dll` für .NET Framework wird weiterhin bereitgestellt, aber sie ist stabilisiert, d. h., es werden keine neuen Funktionen mehr hinzugefügt. Um die neuesten Funktionen nutzen zu können, müssen Sie eine Migration auf die Bibliothek `amqmdnetstd.dll` durchführen. Sie können die Bibliothek `amqmdnet.dll` jedoch unter IBM MQ 9.1 oder höheren Long Term Support- oder Continuous Delivery-Releases weiterhin verwenden.

V 9.4.0 V 9.4.0 Es folgen zwei Szenarios, die nach dem Entfernen der `netstandard2.0` -Bibliotheken auftreten können:

- Wenn Sie eine IBM MQ classes for .NET Framework -Anwendung verwenden, die mit `netstandard2.0` -Bibliotheken wie `amqmdnetstd.dll`erstellt wurde, müssen Sie Ihre Anwendung mit den Microsoft.NET Framework 4.7.2 -Bibliotheken wie `amqmdnet.dll`erneut erstellen, damit Ihre Anwendung erfolgreich ausgeführt werden kann. Wenn Sie Ihre Anwendung nicht erneut erstellen, erhalten Sie möglicherweise ein System.IO.Unexceptionable -Nachricht ohne Ausnahme:

  ```
  Ausnahmebedingung abgefangen: System.IO.FileLoadException: Could not load file or assembly
  'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' or one of its
  dependencies. Die Manifestdefinition der loktierten Baugruppe stimmt nicht mit der Assembly-Re-
  ferenz überein. (Ausnahme von HRESULT: 0x80131040)
  Dateiname: 'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e'
     bei SimplePut.SimplePut.PutMessages()
     bei SimplePut.SimplePut.Main (String [] args) in C:\SampleCode\Program.cs:line 132
  ```

- Wenn Sie eine .NET 6 -Anwendung verwenden, die mit `netstandard2.0` -Bibliotheken erstellt wurde, müssen Sie diese Bibliotheken nur durch dieselben .NET 6 -Bibliotheken im Ordner `bin` des Anwendungslaufzeitverzeichnisses ersetzen. Es ist keine Neuerstellung erforderlich.

  Anmerkung: Die .NET 6 -Ersatzbibliothek sollte immer dieselbe oder eine höhere Version als die ersetzte `netstandard2.0` -Bibliothek haben.

`dspmqver`-Befehl

Mit dem Befehl **`dspmqver`** können Versions- und Buildinformationen für die .NET Core-Komponente angezeigt werden.

Funktionsvergleich zwischen IBM MQ classes for .NET Framework und IBM MQ classes for .NET

In der folgenden Tabelle sind die Features für IBM MQ classes for .NET Framework im Gegensatz zu den Features für IBM MQ classes for .NET

Tabelle 76. Unterschiede zwischen IBM MQ classes for .NET Framework und IBM MQ classes for .NET .

Funktion	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET**
Klassennamen (APIs)	Alle Klassen bleiben in jedem Netz gleich.	Alle Klassen bleiben in jedem Netz gleich.
Betriebssystem	Windows	Windows Dockerisierte Container Linux macOS
`app.config`-Datei (Konfigurationsdatei zum Aktivieren von Trace im erneut verteilbaren Client)	Mit der Datei `app.config` wird der Trace für das weiterverteilbare Paket und den eigenständigen IBM MQ .NET -Client aktiviert. Weitere Informationen zu den Variablen, die Sie für den Trace verwenden, einschließlich **MQTRACEPATH** und **MQTRACELEVEL**, finden Sie im Abschnitt Trace für einen IBM MQ classes for .NET Framework -Client mit einer Anwendungskonfigurationsdatei erstellen .	`app.config` wird nicht unterstützt. Verwenden Sie Umgebungsvariablen.

Tabelle 76. Unterschiede zwischen IBM MQ classes for .NET Framework und IBM MQ classes for .NET . (Forts.)

Funktion	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
Trace	Bei einer vollständigen Clientinstallation von IBM MQkönnen Sie mit dem Befehl **strmqtrc** den Trace für IBM MQ classes for .NET Frameworkaktivieren. Für weiterverteilbare Clients wird auch die Datei `app.config` verwendet, um den Trace zu aktivieren. Weitere Informationen finden Sie unter Traceerstellung für IBM MQ .NET -Anwendungen. V 9.4.0 Ab IBM MQ 9.4.0können Sie den Trace aktivieren und inaktivieren, indem Sie die Datei `mqclient.ini` verwenden und die entsprechenden Eigenschaften der Trace-Zeilengruppe festlegen. Sie können die Traceerstellung auch dynamisch mit der Datei `mqclient.ini` aktivieren und inaktivieren. Weitere Informationen hierzu finden Sie im Abschnitt Traceerstellung für IBM MQ .NET -Anwendungen mit mqclient.ini.	Die Umgebungsvariable **MQDOTNET_TRACE_ON** wird verwendet, um den Trace für weiterverteilbare Clients zu aktivieren. Werte, die gleich und kleiner als 0 sind, aktivieren den Trace nicht. Der Wert 1 aktiviert die Traceerstellung auf Standardstufe. Ein Wert größer als 1 aktiviert die detaillierte Traceerstellung. Wenn Sie diese Umgebungsvariable auf einen anderen Wert wie eine Zeichenfolge setzen, wird der Trace nicht aktiviert. Siehe Traceerstellung für IBM MQ .NET -Anwendungen mit Umgebungsvariablen. Die Umgebungsvariable **MQDOTNET_TRACE_ON** prüft, ob das Traceverzeichnis IBM MQ verfügbar ist. Wenn das Traceverzeichnis verfügbar ist, wird die Tracedatei im Traceverzeichnis generiert. Wenn IBM MQ jedoch nicht installiert ist, wird die Tracedatei in das aktuelle Arbeitsverzeichnis kopiert. Andere Umgebungsvariablen wie **MQERRORPATH**, **MQLOGLEVEL**, **MQSERVER**usw., die für IBM MQ classes for .NET Frameworkverwendet werden, können auf dieselbe Weise verwendet werden und funktionieren. V 9.4.0 Ab IBM MQ 9.4.0können Sie den Trace aktivieren und inaktivieren, indem Sie die Datei `mqclient.ini` verwenden und die entsprechenden Eigenschaften der Trace-Zeilengruppe festlegen. Sie können die Traceerstellung auch dynamisch mit der Datei `mqclient.ini` aktivieren und inaktivieren. Weitere Informationen hierzu finden Sie im Abschnitt Traceerstellung für IBM MQ .NET -Anwendungen mit mqclient.ini.
Transportmodi	Verwaltet, nicht verwaltet und Bindungen	Verwaltet

Tabelle 76. Unterschiede zwischen IBM MQ classes for .NET Framework und IBM MQ classes for .NET . (Forts.)		
Funktion	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET**
TLS	Der Windows-Schlüsselspeicher wird für die Speicherung der Zertifikate verwendet.	Windows Unter Windows muss der Schlüsselspeicher für die Speicherung der Zertifikate verwendet werden. Zulässige Werte sind *USER oder *SYSTEM. Auf Basis der Eingabe sucht der IBM MQ .NET-Client im Windows-Keystore des aktuellen Benutzers oder im gesamten System. Linux Unter Linux wird empfohlen, die Klasse X509Store für die Installation von Zertifikaten zu verwenden, und .NET Core installiert Zertifikate an der folgenden Position: "`.dotnet/corefx/cryptography/x509stores`".
CCDT (Client Channel Definition Table)	Unterstützt	Unterstützt und die Einstellungen des CCDT-Pfads sind mit .NET Framework-Klassen identisch.
Automatische Neuverbindung des Clients	Unterstützt	Unterstützt
Dezentrale Transaktionsverarbeitung	Unterstützt	Nicht unterstützt
Installation von DLLs (Dynamic Link Library) im Global Assembly Cache (GAC)	Dlls werden als Teil der IBM MQ-Installation im GAC installiert.	Dlls werden nicht als Teil der IBM MQ-Installation im GAC installiert.

Anmerkung: Windows Windows-Sicherheits-IDs (SIDs):

Die Authentifizierung auf Domänenebene wird für IBM MQ classes for .NET (Bibliotheken.NET Standard und .NET 6) nicht unterstützt. Für die Authentifizierung wird die angemeldete Benutzer-ID verwendet.

Anwendungen für IBM MQ .NET Core unter macOS entwickeln

macOS

IBM MQ .NET Core-Anwendungen können unter macOS entwickelt werden.

Da die IBM MQ-.NET-Bibliotheken nicht zusammen mit dem macOS-Toolkit ausgeliefert werden, müssen Sie sie von einem IBM MQ-Client unter Windows oder Linux nach macOS kopieren. Anschließend können Sie mithilfe dieser Bibliotheken IBM MQ .NET Core -Anwendungen unter macOSentwickeln.

Die neu entwickelten Anwendungen können anschließend in Windows- oder Linux-Umgebungen ausgeführt werden.

Zugehörige Konzepte

„Installieren von IBM MQ classes for .NET Framework” auf Seite 590
IBM MQ classes for .NET Framework, einschließlich Beispiele, werden mit IBM MQinstalliert. Microsoft.NET Framework unter Windows ist als Voraussetzung erforderlich.

„Installieren von IBM MQ classes for XMS .NET” auf Seite 651

IBM MQ classes for XMS .NET, einschließlich Beispiele, werden mit IBM MQ unter Windows und Linuxinstalliert.

Windows Linux IBM MQ classes for .NET aus dem NuGet -Repository herunterladen

Die IBM MQ classes for .NET -Werte können aus dem NuGet -Repository heruntergeladen werden, sodass sie einfach von .NET -Entwicklern verarbeitet werden können.

Informationen zu diesem Vorgang

Bei NuGet handelt es sich um den Paketmanager für die Entwicklungsplattformen von Microsoft, einschließlich .NET. Die NuGet-Client-Tools ermöglichen das Erzeugen und Verarbeiten von Paketen. Ein NuGet-Paket ist eine einzelne komprimierte Datei mit der Erweiterung `.nupkg`, die kompilierten Code (DLLs), weitere zu diesem Code gehörende Dateien sowie eine beschreibende Manifestdatei mit Informationen wie beispielsweise der Versionsnummer des Pakets enthält.

Sie können das NuGet-Paket `IBMMQDotnetClient` mit der Bibliothek `amqmdnetstd.dll` aus NuGet Gallery herunterladen, d. h. aus dem zentralen Paketrepository, das von allen Paketautoren und -konsumenten verwendet wird.

Anmerkung: V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0enthält das Paket NuGet Bibliotheken, die mit .NET 6 als Zielframework erstellt wurden.

Removed Die mit .NET Standard 2.0erstellte IBM MQ .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0unterstützt IBM MQ .NET 8 -Anwendungen, die IBM MQ classes for .NETverwenden. Wenn Sie eine .NET 6 -Anwendung verwenden, können Sie diese Anwendung ausführen, ohne dass eine erneute Kompilierung erforderlich ist, indem Sie eine kleine Bearbeitung in der Datei `runtimeconfig` vornehmen, um `targetframeworkversion` auf `"net8.0"`zu setzen.

Zum Herunterladen des Pakets `IBMMQDotnetClient` gibt es drei Möglichkeiten:

- Mithilfe von Microsoft Visual Studio. NuGet wird als Erweiterung für Microsoft Visual Studio verteilt. Ab Microsoft Visual Studio 2012 ist NuGet standardmäßig vorinstalliert.
- Aus der Befehlszeile mithilfe des NuGet-Paketmanagers oder der .NET-Befehlszeilenschnittstelle.
- Über einen Web-Browser.

Wie beim Redistributable Package aktivieren Sie den Trace mit der Umgebungsvariablen **MQDOTNET_TRACE_ON**.

Prozedur

- Führen Sie zum Herunterladen des Pakets `IBMMQDotnetClient` über die Benutzerschnittstelle des Paketmanagers in Microsoft Visual Studio die folgenden Schritte aus:
  a) Klicken Sie mit der rechten Maustaste auf das .NET-Projekt und klicken Sie anschließend auf **Manage Nuget-Packages** (Nuget-Pakete verwalten).
  b) Klicken Sie auf die Registerkarte **Durchsuchen** und suchen Sie nach `"IBMMQDotnetClient"`.
  c) Wählen Sie das Paket aus und klicken Sie auf **Install** (Installieren).
  Während der Installation stellt der Paketmanager Informationen zum Fortschritt in Form von Konsolenanweisungen bereit.
- Wählen Sie zum Herunterladen des Pakets `IBMMQDotnetClient` über die Befehlszeile eine der folgenden Optionen aus:
  - Geben Sie bei Verwendung des NuGet-Paketmanagers den folgenden Befehl ein:

```
Install-Package IBMMQDotnetClient -Version 9.1.4.0
```

Während der Installation stellt der Paketmanager Informationen zum Fortschritt in Form von Konsolenanweisungen bereit. Sie können die Ausgabe an eine Protokolldatei weiterleiten.

- Geben Sie bei Verwendung der .NET-Befehlszeilenschnittstelle den folgenden Befehl ein:

```
dotnet add package IBMMQDotnetClient --version 9.1.4
```

- Bei der Verwendung eines Web-Browsers laden Sie das Paket `IBMMQDotnetClient` unter https://www.nuget.org/packages/IBMMQDotnetClient herunter.

Zugehörige Konzepte

Lizenzinformationen für IBM MQ Client for .NET

Zugehörige Tasks

„IBM MQ classes for XMS .NET aus dem NuGet-Repository herunterladen” auf Seite 654
Die IBM MQ classes for XMS .NET sind im NuGet-Repository für den Download verfügbar, damit sie auf einfache Weise von .NET Developers verarbeitet werden können.

Windows Installieren von IBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework, einschließlich Beispiele, werden mit IBM MQinstalliert. Microsoft.NET Framework unter Windows ist als Voraussetzung erforderlich.

Die aktuellste Version von IBM MQ classes for .NET Framework ist standardmäßig als Teil der IBM MQ-Standardinstallation in der Funktion *Java and .NET Messaging and Web Services* installiert. Installationsanweisungen finden Sie in den Abschnitten IBM MQ-Server unter Windows installieren oder IBM MQ-Client auf Windows-Systemen installieren.

Ab IBM MQ 9.3.0müssen Sie zum Ausführen von IBM MQ classes for .NET Framework Microsoft.NET Framework V4.7.2 oder höher installieren.

Vorhandene Anwendungen, die mit Microsoft.NET Framework V3.5 kompiliert wurden, können ohne erneute Kompilierung ausgeführt werden, indem der folgende Tag in der Datei `app.config` der Anwendung hinzugefügt wird:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

Anmerkung: Wenn Microsoft .NET Framework V4.7.2 oder höher nicht vor der Installation von IBM MQinstalliert wird, wird die IBM MQ -Produktinstallation ohne Fehler fortgesetzt, aber IBM MQ classes for .NET ist nicht verfügbar. Wenn das .NET Framework installiert wird, nachdem Sie IBM MQ installiert haben, müssen die IBM MQ.NET-Assemblys durch Ausführung des Scripts *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` registriert werden, wobei *WMQ-Installationsverzeichnis* für das Verzeichnis steht, in dem IBM MQ installiert ist. Mit diesem Script werden die erforderlichen Assemblys im Global Assembly Cache (GAC) installiert. Im Verzeichnis %TEMP% wird eine Gruppe von Dateien `amqi*.log` erstellt, in denen die ausgeführten Aktionen dokumentiert werden. Es ist nicht erforderlich, das Script `amqiRegisterdotNet.cmd` erneut auszuführen, wenn für .NET ein Upgrade auf V4.7.2 oder höher von einer früheren Version durchgeführt wird, beispielsweise von .NET V3.5.

Wenn Sie die IBM MQ classes for .NET zuvor in einer Umgebung mit mehreren Installationen als Support-Pack installiert haben, kann IBM MQ erst installiert werden, nachdem das Support-Pack deinstalliert wurde. Das Feature IBM MQ classes for .NET , das mit IBM MQ installiert wird, enthält dieselben Funktionen wie das Support-Pack.

Es werden auch Beispielanwendungen bereitgestellt, einschließlich Quellendateien; siehe „Beispielanwendungen für .NET” auf Seite 591.

Informationen zur Verwendung des angepassten IBM MQ-Kanals für Microsoft WCF mit .NET finden Sie im Abschnitt „Microsoft Windows Communication Foundation -Anwendungen mit IBM MQ entwickeln” auf Seite 1327.

Zugehörige Konzepte

„Installieren von IBM MQ classes for .NET” auf Seite 584
IBM MQ classes for .NET, einschließlich Beispiele, werden mit IBM MQ unter Windows und Linux installiert.

Zugehörige Tasks

Traceerstellung für IBM MQ .NET-Anwendungen

Optionen für Verbindung von IBM MQ classes for .NET zu einem Warteschlangenmanager

Die Verbindung von IBM MQ classes for .NET zu einem Warteschlangenmanager kann auf drei Arten hergestellt werden. Prüfen Sie, welcher Verbindungstyp für Ihre Anforderungen am besten geeignet ist.

Verbindungen über Clientbindungen

Wenn Sie die IBM MQ classes for .NET als IBM MQ MQI client verwenden möchten, können Sie diese mit dem IBM MQ MQI client auf der IBM MQ-Servermaschine oder auf einem separaten System installieren. Eine Clientverbindung kann XA- oder Nicht-XA-Transaktionen verwenden.

Verbindungen über Serverbindungen

Im Serverbindungsmodus kommunizieren IBM MQ classes for .NET nicht über ein Netz, sondern verwenden die Warteschlangenmanager-API. Dadurch wird eine bessere Leistung für IBM MQ-Anwendungen bereitgestellt als bei der Verwendung von Netzverbindungen.

Zum Verwenden von Verbindungen über Bindungen müssen Sie IBM MQ classes for .NET auf dem IBM MQ-Server installieren.

Verwaltete Clientverbindung

Bei einer Verbindung in diesem Modus wird ein IBM MQ-Client mit einem IBM MQ-Server verbunden, der auf dem lokalen oder einem fernen System ausgeführt wird.

Die IBM MQ classes for .NET, für die in diesem Modus eine Verbindung hergestellt wird, verbleiben im verwalteten .NET-Code und geben keine Anrufe an native Services aus. Weitere Informationen zum verwalteten Code finden Sie in der Microsoft-Dokumentation.

Die Nutzung des verwalteten Clients unterliegt einer Reihe von Einschränkungen. Weitere Informationen hierzu finden Sie unter „Verwaltete Clientverbindungen” auf Seite 608.

Beispielanwendungen für .NET

Um Ihre eigenen .NET-Anwendungen auszuführen, verwenden Sie die Anweisungen für die Prüfprogramme, mit dem die Namen der Beispielanwendungen durch Ihren Anwendungsnamen ersetzt werden.

Folgende Beispielanwendungen werden bereitgestellt:

- Eine Anwendung zum Einreihen von Nachrichten
- Eine Anwendung zum Abrufen von Nachrichten
- Eine 'Hello World'-Anwendung
- Eine Publish/Subscribe-Anwendung
- Eine Anwendung, in der Nachrichteneigenschaften verwendet werden

Diese Beispielanwendungen werden alle in der Programmiersprache C# bereitgestellt und einige sind außerdem in C++ und Visual Basic vorhanden. Sie können Anwendungen in jeder Sprache schreiben, die von .NET unterstützt wird.

Das "Put message"-Programm SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)
Dieses Programm zeigt, wie eine Nachricht in eine benannte Warteschlange eingereiht wird. Das Programm enthält drei Parameter:

- Der Name einer Warteschlange (erforderlich), z. B. SYSTEM.DEFAULT.LOCAL.QUEUE
- Der Name eines Warteschlangenmanagers (optional)
- Die Definition eines Kanals (optional), z. B. SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Wenn kein Warteschlangenmanagername angegeben ist, wird standardmäßig der lokale Warteschlangenmanager verwendet. Wenn ein Kanal definiert ist, hat er das gleiche Format wie die Umgebungsvariable MQSERVER.

Das "Get message"-Programm SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
Dieses Programm zeigt, wie eine Nachricht aus einer benannten Warteschlange abgerufen wird. Das Programm enthält drei Parameter:

- Der Name einer Warteschlange (erforderlich), z. B. SYSTEM.DEFAULT.LOCAL.QUEUE
- Der Name eines Warteschlangenmanagers (optional)
- Die Definition eines Kanals (optional), z. B. SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Wenn kein Warteschlangenmanagername angegeben ist, wird standardmäßig der lokale Warteschlangenmanager verwendet. Wenn ein Kanal definiert ist, hat er das gleiche Format wie die Umgebungsvariable MQSERVER.

Das "Hello World"-Programm (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
In diesem Programm wird erklärt, wie eine Nachricht in eine Warteschlange eingereiht und aus einer Warteschlange abgerufen wird. Das Programm enthält drei Parameter:

- Der Name einer Warteschlange (optional), z. B. SYSTEM.DEFAULT.LOCAL.QUEUE oder SYSTEM.DEFAULT.MODEL.QUEUE
- Der Name eines Warteschlangenmanagers (optional)
- Eine Kanaldefinition (optional), z. B. SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Wenn kein Warteschlangenname angegeben ist, wird standardmäßig der Name SYSTEM.DEFAULT.LOCAL.QUEUE verwendet. Wenn kein Warteschlangenmanagername angegeben ist, wird standardmäßig der lokale Warteschlangenmanager verwendet.

Das "Publish/subscribe"-Programm (MQPubSubSample.cs)
In diesem Programm wird die Verwendung von IBM MQ-Publish/Subscribe gezeigt. Es wird nur in der Programmiersprache C# bereitgestellt. Das Programm enthält zwei Parameter:

- Der Name eines Warteschlangenmanagers (optional)
- Eine Kanaldefinition (optional)

Das "Message properties"-Programm (MQMessagePropertiesSample.cs)
In diesem Programm wird die Verwendung von Nachrichteneigenschaften gezeigt. Es wird nur in der Programmiersprache C# bereitgestellt. Das Programm enthält zwei Parameter:

- Der Name eines Warteschlangenmanagers (optional)
- Eine Kanaldefinition (optional)

Sie können Ihre Installation prüfen, indem Sie diese Anwendungen kompilieren und ausführen.

Installationspositionen

Abhängig von der Programmiersprache, in der sie geschrieben sind, sind die Beispielanwendungen an den folgenden Positionen installiert. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsput.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsget.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

Managed C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqswrld.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsput.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsget.cpp

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsget.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb

Beispielanwendungen erstellen

Zum Erstellen der Beispielanwendungen wird für jede Sprache eine Stapeldatei bereitgestellt.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

Die Datei 'bldcssamp.bat' enthält eine Zeile für jedes Beispiel, und dies ist alles, was zum Erstellen dieses Beispielprogramms erforderlich ist:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

Managed C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

Die Datei 'bldmcpsamp.bat' enthält eine Zeile für jedes Beispiel, und dies ist alles, was zum Erstellen dieses Beispielprogramms erforderlich ist:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Wenn Sie diese Anwendungen unter Microsoft Visual Studio 2003/.NET SDKv1.1 kompilieren möchten, ersetzen Sie den Kompilierbefehl

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

durch

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

Die Datei 'bldvbsamp.bat' enthält eine Zeile für jedes Beispiel, und dies ist alles, was zum Erstellen dieses Beispielprogramms erforderlich ist:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

Beispiele für Verwendung von IBM MQ mit Microsoft .NET Core

IBM MQ unterstützt .NET Core für IBM MQ .NET -Anwendungen in Windows -Umgebungen. IBM MQ classes for .NET Standard, einschließlich Beispiele, wird standardmäßig im Rahmen der IBM MQ-Standardinstallation installiert.

Die Beispielanwendungen für IBM MQ .NET werden im Verzeichnis `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base` installiert. Es wird auch ein Script bereitgestellt, mit dem die Beispiele kompiliert werden können.

Zum Erstellen der Beispiele können Sie die bereitgestellten Dateien `build.bat` erstellen. Für jedes Beispiel befindet sich unter Windows eine Datei `build.bat` an der folgenden Position:

- `MQ\tools\dotnet\samples\cs\core\base\SimpleGet`
- `MQ\tools\dotnet\samples\cs\core\base\SimplePut`

Linux IBM MQ unterstützt auch Core für Anwendungen in Linux -Umgebungen.

Weitere Informationen zur Verwendung von IBM MQ mit Microsoft .NET Core finden Sie unter „Installieren von IBM MQ classes for .NET” auf Seite 584.

Warteschlangenmanager für das Akzeptieren von TCP/IP-Clientverbindungen konfigurieren

Sie können einen Warteschlangenmanager so konfigurieren, dass eingehende Verbindungsanforderungen von den Clients akzeptiert werden.

Informationen zu diesem Vorgang

In dieser Task werden die grundlegenden Schritte bei der Konfiguration eines Warteschlangenmanagers zum Akzeptieren von TCP/IP-Clientverbindungen erläutert. Für ein Produktionssystem müssen Sie beim Konfigurieren von Warteschlangenmanagern außerdem die Auswirkungen auf die Sicherheit beachten.

Vorgehensweise

1. Definieren Sie einen Serververbindungskanal:
   a. Starten Sie den Warteschlangenmanager.
   b. Definieren Sie einen Beispielkanal mit der Bezeichnung NET.CHANNEL:

   ```
   DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
   DESCR('Sample channel for IBM MQ classes for .NET')
   ```

   Wichtig: Dieses Beispiel bezieht sich ausschließlich auf die Verwendung in einer Sandbox-Umgebung, da keine Überlegungen zu Auswirkungen auf die Sicherheit eingeschlossen sind. In einem Produktionssystem können Sie TLS oder einen Sicherheitsexit verwenden. Weitere Informationen finden Sie unter IBM MQ schützen.
2. Starten Sie einen Listener:

   ```
   runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
   ```

Anmerkung: Die eckigen Klammern zeigen optionale Parameter an. Der Parameter *qmname* (Warteschlangenmanagername) ist für den Standardwarteschlangenmanager nicht erforderlich und der Parameter *portnum* (Portnummer) ist nicht erforderlich, wenn der Standardport (1414) verwendet wird.

Dezentrale Transaktionsverarbeitung in .NET

Bei der dezentralen oder globalen Transaktionsverarbeitung können Clientanwendungen mehrere verschiedene Datenquellen in mindestens zwei vernetzten Systemen in eine Transaktion einschließen.

Bei der dezentrale Transaktionsverarbeitung koordiniert ein Transaktionsmanager die Transaktion zwischen mindestens zwei Ressourcenmanagern und verwaltet diese.

Bei den Transaktionen kann es sich um einen Prozess mit einer einphasigen oder zweiphasigen Festschreibung handeln. Die einphasige Festschreibung ist ein Prozess, bei dem nur ein Ressourcenmanager an der Transaktion beteiligt ist, und bei dem zweiphasigen Festschreibungsprozess sind mehrere Ressourcenmanager an der Transaktion beteiligt. Im zweiphasigen Festschreibungsprozess sendet der Transaktionsmanager einen Vorbereitungsaufruf, um zu überprüfen, ob alle Ressourcenmanager auf die Festschreibung vorbereitet sind. Wenn er von allen Ressourcenmanagern eine Bestätigung empfängt, wird der Festschreibungsaufruf abgesetzt. Andernfalls wird für die gesamte Transaktion ein Rollback ausgeführt. Weitere Einzelheiten finden Sie im Abschnitt Transaktionsmanagement und Unterstützung. Die Ressourcenmanager sollten die Transaktionsmanager über ihre Beteiligung an der Transaktion informieren. Wenn der Ressourcenmanager den Transaktionsmanager über seine Beteiligung informiert, erhält der Ressourcenmanager Callbacks vom Transaktionsmanager, wenn die Transaktion festgeschrieben oder rückgängig gemacht wird.

Die IBM MQ .NET-Klassen unterstützen bereits die dezentrale Transaktionsverarbeitung für Verbindungen im nicht verwalteten Modus und im Serverbindungsmodus. In diesen Modi delegieren IBM MQ .NET-Klassen alle Aufrufe an den erweiterten Transaktionsclient für die Programmiersprache C, der die Transaktionsverarbeitung im Auftrag von .NET verwaltet.

Die IBM MQ.NET-Klassen unterstützen jetzt die dezentrale Transaktionsverarbeitung im verwalteten Modus, in dem IBM MQ .NET-Klassen den Namensbereich 'System.Transactions' für die Unterstützung der dezentrale Transaktionsverarbeitung verwenden. Durch die System.Transactions-Infrastruktur wird die transaktionsorientierte Programmierung einfach und effizient, da Transaktionen unterstützt werden, die in allen Ressourcenmanagern (einschließlich IBM MQ) aufgerufen werden. Die IBM MQ .NET-Anwendung kann Nachrichten mithilfe der .NET-Programmierung für implizite Transaktionen oder dem Programmiermodell für explizite Transaktionen einreihen und abrufen. Bei impliziten Transaktionen werden die Transaktionsgrenzen vom Anwendungsprogramm erstellt, das entscheidet, wann die Transaktion festgeschrieben, rückgängig gemacht (für explizite Transaktionen) oder abgeschlossen werden soll. Bei expliziten Transaktionen müssen Sie explizit angeben, ob Sie die Transaktion festschreiben, rückgängig machen oder abschließen möchten.

IBM MQ.NET verwendet den dezentralen Transaktionskoordinator von Microsoft (MS-DTC) als Transaktionsmanager, mit dem die Transaktion zwischen mehreren Ressourcenmanagern koordiniert und verwaltet wird. IBM MQ wird als Ressourcenmanager verwendet. Beachten Sie, dass Sie TLS nicht zusammen mit XA-Transaktionen verwenden können. Sie müssen stattdessen eine Definitionstabelle für Clientkanäle (CCDT) verwenden. Weitere Informationen hierzu finden Sie im Abschnitt Erweiterten transaktionsorientierten Client mit TLS-Kanälen verwenden.

IBM MQ.NET folgt dem X/Open Distributed Transaction Processing-Modell (DTP). Das X/Open Distributed Transaction Processing-Modell ist ein Modell für die dezentrale Transaktionsverarbeitung, das von der Open Group, einem Konsortium aus Softwareanbietern, aufgestellt wurde. Dieses Modell ist ein Standard für die meisten kommerziellen Softwareanbieter im Bereich der Transaktionsverarbeitung und der Datenbankdomänen. Die meisten kommerziellen Produkte für die Transaktionsverwaltung unterstützen das X/DTP-Modell.

Transaktionsmodi

- „Dezentrale Transaktionsverarbeitung im verwalteten Modus von .NET” auf Seite 597
- Dezentrale Transaktionen im nicht verwalteten Modus

Transaktionen in verschiedenen Szenarios koordinieren

- Eine Verbindung ist möglicherweise an mehreren Transaktionen beteiligt, aber es ist jeweils nur eine Transaktion aktiv.
- Während einer Transaktion wird der Aufruf MQQueueManager.Disconnect berücksichtigt. In diesem Fall wird die Transaktion dazu aufgefordert, einen Rollback durchzuführen.
- Während einer Transaktion wird der Aufruf MQQueue.Close oder MQTopic.Close berücksichtigt. In diesem Fall wird die Transaktion dazu aufgefordert, einen Rollback durchzuführen.
- Die Transaktionsgrenzen werden vom Anwendungsprogramm erstellt, das entscheidet, wann die Transaktion festgeschrieben, rückgängig gemacht (für explizite Transaktionen) oder abgeschlossen (für implizite Transaktionen) werden soll.
- Wenn der Client während einer Transaktion mit einem unerwarteten Fehler unterbrochen wird, bevor ein PUT- oder GET-Aufruf im Aufruf einer Warteschlange oder eines Themas ausgegeben werden kann, wird die Transaktion rückgängig gemacht und eine MQ-Ausnahme ausgegeben.
- Wenn der Ursachencode MQCC_FAILED während eines PUT- oder GET-Aufrufs beim Aufrufen einer Warteschlange oder eines Themas zurückgegeben wird, wird eine MQ-Ausnahme mit dem Ursachencode ausgegeben und die Transaktion wird rückgängig gemacht. Wenn vom Transaktionsmanager bereits ein Aufruf zur Vorbereitung ausgegeben wurden, gibt IBM MQ .NET die Anforderung zur Vorbereitung anschließend zurück, indem das Rückgängigmachen der Transaktion erzwungen wird. Anschließend verursacht der DTC des Transaktionsmanagers einen Rollback der aktuellen Arbeit mit allen Ressourcenmanagern in aktuellen Umgebungstransaktionen.
- Wenn während einer Transaktion, an der mehrere Ressourcenmanager beteiligt sind, einige umgebungsbedingte Ursachen dazu führen, dass der PUT- oder GET-Aufruf unendlich blockiert ist, wartet der Transaktionsmanager bis zu einem festgelegten Zeitpunkt. Nachdem das Zeitlimit überschritten ist, löst er einen Rollback der aktuellen Arbeit mit allen Ressourcenmanagern in aktuellen Umgebungstransaktionen aus. Wenn die unendliche Wartezeit in der Vorbereitungsphase auftritt, wird das Zeitlimit des Transaktionsmanagers möglicherweise überschritten oder der Transaktionsmanager löst einen unbestätigten Aufruf in der Ressource aus, wodurch die Transaktion rückgängig gemacht wird.
- Anwendungen, die Transaktionen verwenden, müssen Nachrichten mit PUT- oder GET-Aufrufen unter SYNC_POINT ausgeben. Wenn der PUT- oder GET-Aufruf für eine Nachricht in einem Transaktionskontext ausgegeben wird, der sich nicht unter SYNC_POINT befindet, schlägt der Aufruf mit dem Ursachencode MQRC_UNIT_OF_WORK_NOT_STARTED fehl.

Unterschiede im Verhalten zwischen der verwalteten und nicht verwalteten Transaktionsunterstützung mit dem Namensbereich 'Microsoft.NET System.Transactions'

Verschachtelte Transaktionen verfügen über einen Transaktionsbereich, der sich in einem anderen Transaktionsbereich befindet

- Der vollständig verwaltete Client für IBM MQ .NET unterstützt den verschachtelten Transaktionsbereich
- Der nicht verwaltete Client für IBM MQ .NET unterstützt den verschachtelten Transaktionsbereich nicht

Abhängige Transaktionen aus System.Transactions

- Der vollständig verwaltete Client für IBM MQ .NET unterstützt die Funktion für abhängige Transaktionen, die von System.Transactions bereitgestellt wird.
- Der nicht verwaltete Client für IBM MQ .NET unterstützt die Funktion für abhängige Transaktionen nicht, die von System.Transactions bereitgestellt wird.

Produktbeispiele

Die Produktbeispiele `SimpleXAPut` und `SimpleXAGet` sind unter `WebSphere MQ\tools\dotnet\samples\cs\base` verfügbar. Bei den Beispielen handelt es sich um C#-Anwendungen, in denen die Verwendung von MQPUT und MQGET in der dezentralen Transaktionsverarbeitung mithilfe des Namensbereichs 'SystemTransactions' gezeigt wird. Weitere Informationen zu diesen Beispielen finden Sie

unter „Einfache Nachrichten zum Abrufen und Einreihen in einem Transaktionsbereich erstellen” auf Seite 600.

Dezentrale Transaktionsverarbeitung im verwalteten Modus von .NET

IBM MQ .NET-Klassen verwenden den Namensbereich 'System.Transactions' für die Unterstützung der dezentralen Transaktionsverarbeitung im verwalteten Modus. Im verwalteten Modus wird die dezentrale Transaktionsverarbeitung auf allen Servern, die in einer Transaktion aufgeführt sind, vom MS-DTC koordiniert und verwaltet.

In den IBM MQ .NET-Klassen wird ein explizites Programmiermodell auf Basis der Klasse 'System.Transactions.Transaction' und ein implizites Programmiermodell mithilfe der Klasse 'System.Transactions.TransactionScope' bereitgestellt, in der die Transaktionen automatisch von der Infrastruktur verwaltet werden.

Implizite Transaktion

Im folgenden Codeteil wird beschrieben, wie eine IBM MQ .NET-Anwendung eine Nachricht mithilfe der .NET-Programmierung für eine implizite Transaktion einreiht.

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

Erläuterung des Codeflusses bei einer impliziten Transaktion

Der Code erstellt den Bereich *TransactionScope* und reiht die Nachricht in dem Bereich ein. Anschließend wird *Complete* aufgerufen, um den Transaktionskoordinator über den Abschluss der Transaktion zu informieren. Der Transaktionskoordinator gibt jetzt *prepare* und *commit* aus, um die Transaktion abzuschließen. Bei der Ermittlung eines Problems wird ein *Rollback* aufgerufen.

Explizite Transaktion

Im folgenden Code wird beschrieben, wie eine IBM MQ .NET-Anwendung Nachrichten mithilfe des .NET-Programmiermodells für eine explizite Transaktion einreiht.

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

Erläuterung des Codeflusses bei einer expliziten Transaktion

Der Codeteil erstellt eine Transaktion mit der Klasse *CommitableTransaction*. Es wird eine Nachricht in diesen Bereich eingereiht und anschließend wird explizit *commit* aufgerufen, um die Transaktion abzuschließen. Wenn Probleme auftreten, wird ein *Rollback* aufgerufen.

Dezentrale Transaktionsverarbeitung im nicht verwalteten Modus von .NET

IBM MQ.NET-Klassen unterstützten nicht verwaltete Verbindungen (Client) über den erweiterten Transaktionsclient und COM+/MTS als Transaktionskoordinator. Dabei wird das Programmiermodell für die impli-

zite oder explizite Transaktion verwendet. Im nicht verwalteten Modus delegieren IBM MQ .NET-Klassen alle Aufrufe an den erweiterten Transaktionsclient für die Programmiersprache C, der die Transaktionsverarbeitung im Auftrag von .NET verwaltet.

Die Transaktionsverarbeitung wird von einem externen Transaktionsmanager gesteuert, der die globale Arbeitseinheit unter der Kontrolle der API des Transaktionsmanagers koordiniert. Die Verben MQBEGIN, MQCMIT und MQBACK sind nicht verfügbar.IBM MQ .NET-Klassen stellen diese Unterstützung über ihren nicht verwalteten Transportmodus (C-Client) bereit. Weitere Informationen finden Sie im Abschnitt XA-konforme Transaktionsmanager konfigurieren.

Der Microsoft Transaction Server (MTS) wurde als System zur Transaktionsverarbeitung entwickelt, mit dem unter Windows NT die gleichen Funktionen wie auf CICS-, Tuxedo- und anderen Plattformen verfügbar sind. Wenn der MTS installiert ist, wird Windows NT ein separater Service mit der Bezeichnung 'Microsoft Distributed Transaction Coordinator' (MSDTC) hinzugefügt. MSDTC koordiniert die Transaktionen, die sich auf verschiedene Datenspeicher oder Ressourcen erstrecken. Für die Ausführung muss in jedem Datenspeicher ein eigener proprietärer Ressourcenmanager implementiert werden.

IBM MQ wird durch die Implementierung einer Schnittstelle (proprietäre Ressourcenmanagerschnittstelle) mit MSDTC kompatibel. In dieser Schnittstelle werden DTC XA-Aufrufe den IBM MQ(X/Open)-Aufrufen zugeordnet. IBM MQ übernimmt die Rolle eines Ressourcenmanagers.

Wenn eine Komponente wie beispielsweise COM+ den Zugriff auf IBM MQ anfordert, überprüft die COM-Komponente normalerweise über das entsprechende MTS-Kontextobjekt, ob eine Transaktion erforderlich ist. Wenn eine Transaktion erforderlich ist, informiert die COM-Komponente den DTC und startet automatisch eine integrale IBM MQ-Transaktion für diese Operation. Anschließend arbeitet die COM-Komponente mit diesen Daten über die MQMTS-Software, indem Sie Nachrichten bei Bedarf einreiht und abruft. Die aus der COM-Komponente erhaltene Objektinstanz ruft die Methode 'SetComplete' oder 'SetAbort' auf, nachdem alle Aktionen zu den Daten abgeschlossen wurden. Wenn die Anwendung die Methode 'SetComplete' ausgibt, signalisiert der Aufruf der DTC, dass die Anwendung die Transaktion abgeschlossen hat und der DTC den Prozess zur zweiphasigen Festschreibung fortsetzen kann. Der DTC gibt anschließend Aufrufe an den MQMTS aus, der wiederum Aufrufe an IBM MQ ausgibt, um die Transaktion festzuschreiben oder rückgängig zu machen.

IBM MQ .NET-Anwendung mit einem nicht verwalteten Client schreiben

Für eine Ausführung im Kontext von COM+ muss eine .NET-Klasse Werte von System.EnterpriseServices.ServicedComponent übernehmen. Zur Erstellung von Assemblys, die Servicekomponenten verwenden, gelten die folgenden Regeln und Empfehlungen:

Anmerkung: Die folgenden Schritte sind nur im System.EnterpriseServices-Modus relevant.

- Die in COM+ gestarteten Klassen und Methoden müssen öffentlich sein (keine internen Klassen und keine geschützten oder statischen Methoden).
- Attribute für Klassen und Methoden: Durch das Attribut 'TransactionOption' wird die Transaktionsebene der Klasse festgelegt, durch die angegeben wird, ob die Transaktionen inaktiviert, unterstützt oder erforderlich sind. Das Attribut 'AutoComplete' in der Methode 'ExecuteUOW()' weist die COM+-Komponente zum Festschreiben der Transaktion an, falls keine nicht behandelte Ausnahme ausgelöst wurde.
- Assembly einen starken Namen zuweisen: Die Assembly muss einen starken Namen haben und im Global Assembly Cache (GAC) registriert sein. Die Assembly wird explizit in der COM+-Komponente oder beim ersten Start einer Instanz der Komponente nach der Registrierung im GAC registriert.
- Registrierung einer Assembly in COM+: Bereiten Sie die Assembly vor, damit sie für COM-Clients verfügbar ist. Erstellen Sie anschließend mit dem Tool zur Registrierung der Assembly (regasm.exe) eine Typbibliothek:

  ```
  regasm UnmanagedToManagedXa.dll
  ```

- Registrieren Sie die Baugruppe in GAC `gacutil /i UnmanagedToManagedXa.dll`.

- Registrieren Sie die Assembly mit dem Tool für das .NET-Serviceinstallationsprogramm (regsvcs.exe) in COM+. Zeigen Sie die Typbibliothek an, die mit 'regasm.exe' erstellt wurde:

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- Die Assembly wird im GAC implementiert und später durch eine Registrierung beim Start in der COM+-Komponente registriert. Das .NET-Framework achtet darauf, dass die Registrierung nach der ersten Ausführung des Codes vorgenommen wird.

In den folgenden Abschnitten finden Sie ein Beispiel für den Codefluss, in dem das Modell 'System.EnterpriseServices' und 'System.Transactions' mit COM+ verwendet werden:

Beispiel für einen Codefluss, in dem das Modell 'System.EnterpriseServices' verwendet wird

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]    [System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOpti
on.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
                                     MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
   xa.ExecuteUOW();
}
```

Beispiel für einen Codefluss, in dem 'System.Transactions' für Interaktionen mit COM+ verwendet wird

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                 opts,     EnterpriseServicesInteropOption.Full)
  {
```

```
        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                          MQC.MQOO_INPUT_SHARED +
                                          MQC.MQOO_OUTPUT +
                                          MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

Einfache Nachrichten zum Abrufen und Einreihen in einem Transaktionsbereich erstellen

C#-Anwendungen für das Produktbeispiel sind in IBM MQ verfügbar. Mit diesen einfachen Anwendungen wird das Einreihen und Abrufen von Nachrichten in einem Transaktionsbereich dargestellt. Am Ende der Task können Sie Nachrichten aus einer Warteschlange oder einem Thema abrufen und darin einreihen.

Vorbereitende Schritte

Der MSDTC-Service muss ausgeführt werden und für XA-Transaktionen aktiviert sein.

Informationen zu diesem Vorgang

Das Beispiel ist eine einfache Anwendung, `SimpleXAPut` und `SimpleXAGet`. Bei den Programmen `SimpleXAPut` und `SimpleXAGet` handelt es sich um C#-Anwendungen, die in IBM MQ verfügbar sind. `SimpleXAPut` zeigt die Verwendung von MQPUT in der dezentralen Transaktionsverarbeitung mithilfe des Namensbereichs 'SystemTransactions'. `SimpleXAGet` zeigt die Verwendung von MQGET in der dezentralen Transaktionsverarbeitung mithilfe des Namensbereichs 'SystemTransactions'.

`SimpleXAPut` befindet sich im Verzeichnis `MQ\tools\dotnet\samples\cs\base`

Vorgehensweise

Die Anwendungen können mit den Befehlszeilenparametern aus `tools\dotnet\samples\cs\base\bin` ausgeführt werden.

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n numberOfMsgs]
```

Es folgt eine Auflistung der Parameter:

`-destinationURI`

Dies kann eine Warteschlange oder ein Thema sein. Geben Sie für eine Warteschlange `queue://queueName` und für ein Thema `topic://topicNamean`.

`-host`

Dies kann ein Hostname (z. B. localhost) oder eine IP-Adresse sein.

`-port`

Der Port, auf dem der Warteschlangenmanager ausgeführt wird.

`-channel`

Der Kanal, der für die Verbindung verwendet wird. Der Standardwert ist SYSTEM.DEF.SVRCONN.

-transaction

Das Ergebnis der Transaktion, z. B. Festschreiben oder Rollback.

-mode

Der Transportmodus, z. B. verwalteter oder nicht verwalteter Modus.

-numberOfMsgs

Die Anzahl der Nachrichten. Der Standardwert ist 1.

Beispiel

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

Transaktionen in IBM MQ .NET wiederherstellen

In diesem Abschnitt wird der Prozess zum Wiederherstellen von Transaktionen in IBM MQ .NET XA im verwalteten Modus beschrieben.

Informationen zu diesem Vorgang

Bei der verteilten Transaktionsverarbeitung können die Transaktionen erfolgreich abgeschlossen werden. Es sind jedoch Szenarios möglich, in denen eine Transaktion aus verschiedenen Gründen fehlschlagen kann. Dies umfasst Systemfehler, Hardwarefehler, Netzfehler, falsche oder ungültige Daten, Anwendungsfehler oder natürliche bzw. von Menschen verursachte Katastrophen. Transaktionsfehler können nicht verhindert werden. Das System für verteilte Transaktionen muss in der Lage sein, diese Fehler zu verarbeiten. Es muss Fehler beim Auftreten ermitteln und diese beheben. Dieser Prozess wird als Transaktionswiederherstellung bezeichnet.

Ein wichtiger Aspekt bei der dezentralen Transaktionsverarbeitung ist die Wiederherstellung von unvollständigen oder unbestätigten Transaktionen. Es ist unumgänglich, die Wiederherstellung auszuführen, während die Arbeitseinheit einer bestimmten Transaktion bis zur Wiederherstellung gesperrt ist. Microsoft.NET stellt in der zugehörigen Klassenbibliothek 'System.Transactions' die Option zum Wiederherstellen von unvollständigen oder unbestätigten Transaktionen bereit. Diese Unterstützung bei der Wiederherstellung erwartet, dass der Ressourcenmanager die Transaktionsprotokolle verwaltet und die Wiederherstellung bei Bedarf ausführt.

Im Transaktionswiederherstellungsmodell von Microsoft .NET wird die Transaktionswiederherstellung vom Transaktionsmanager (System.Transactions und/oder Microsoft Distributed Transaction Coordinator (MS-DTC)) eingeleitet, koordiniert und gesteuert. Die auf dem OLE Tx-Protokoll (das Microsoft XA-Protokoll) basierenden Ressourcenmanager stellen die Optionen zum Konfigurieren des DTC-Dienst bereit, mit dem die Wiederherstellung ausgeführt, koordiniert und gesteuert wird. Dazu müssen Ressourcenmanager den XA-Switch mit dem MS-DTC mithilfe einer nativen Schnittstelle registrieren.

Der XA-Switch stellt die Einstiegspunkte von XA-Funktionen wie 'xa_start', 'xa_end' und 'xa_recover' im Ressourcenmanager zum Distributed Transaction Coordinator her.

Wiederherstellung mit dem Microsoft Distributed Transaction Coordinator (DTC):

Im Microsoft Distributed Transaction Coordinator werden zwei Arten von Wiederherstellungsprozessen bereitgestellt.

Wiederherstellung ohne Daten (Cold Recovery)

Die Wiederherstellung ohne Daten wird ausgeführt, wenn der Transaktionsmanagerprozess fehlschlägt, während eine Verbindung zu einem XA-Ressourcenmanager offen ist. Wenn der Transaktions-

manager erneut gestartet wird, werden die Transaktionsmanagerprotokolle gelesen, die Verbindung zum XA-Ressourcenmanager wird erneut hergestellt und anschließend wird die Wiederherstellung eingeleitet.

Wiederherstellung im laufenden Betrieb (Hot Recovery)

Die Wiederherstellung im laufenden Betrieb wird ausgeführt, wenn der Transaktionsmanager aktiv bleibt, während die Verbindung zwischen dem Transaktionsmanager und dem XA-Ressourcenmanager aufgrund eines Fehlers im XA-Ressourcenmanager oder in Netz fehlschlägt. Nach dem Fehler versucht der Transaktionsmanager regelmäßig, die Verbindung zum XA-Ressourcenmanager wiederherzustellen. Wenn die Verbindung erneut hergestellt werden kann, leitet der Transaktionsmanager die XA-Wiederherstellung ein.

Der Namensbereich 'System.Transactions' stellt die verwaltete Implementierung der dezentralen Transaktionsverarbeitung bereit, die auf dem MS-DTC als Transaktionsmanager basiert. Es werden ähnliche Funktionen wie die der nativen MS-DTC-Schnittstelle bereitgestellt, allerdings in einer vollständig verwalteten Umgebung. Der einzige Unterschied ist die Wiederherstellung der Transaktion. In 'System.Transactions' wird erwartet, dass Ressourcenmanager die Wiederherstellung selbst ausführen und anschließend mit den Transaktionsmanagern (MS-DTC) koordinieren. Ressourcenmanager müssen die Wiederherstellung einer bestimmten unvollständigen Transaktion anfordern. Dies wird anschließend vom Transaktionsmanager akzeptiert und auf Basis der tatsächlichen Ausgabe dieser bestimmten Transaktion koordiniert.

Prozess der Transaktionswiederherstellung für IBM MQ .NET

In diesem Abschnitt ist beschrieben, wie verteilte Transaktionen mit IBM MQ .NET-Klassen wiederhergestellt werden können.

Übersicht

Zum Wiederherstellen einer unvollständigen Transaktion sind die Wiederherstellungsinformationen erforderlich. Die Wiederherstellungsinformationen für die Transaktion müssen von den Ressourcenmanagern im Speicher protokolliert sein.IBM MQ .NET -Klassen folgen einem ähnlichen Pfad. Die Wiederherstellungsinformationen zur Transaktion werden in einer Systemwarteschlange mit der Bezeichnung SYSTEM.DOTNET.XARECOVERY.QUEUE protokolliert.

Der Prozess zur Wiederherstellung der Transaktion in IBM MQ .NET läuft in zwei Stufen ab:

1. Protokollierung der Wiederherstellungsinformationen für die Transaktion in SYSTEM.DOTNET.XARECOVERY.QUEUE
2. Wiederherstellung von Transaktionen mit der Überwachungsanwendung 'WmqDotnetXAMonitor'

SYSTEM.DOTNET.XARECOVERY.QUEUE

Bei SYSTEM.DOTNET.XARECOVERY.QUEUE handelt es sich um eine Systemwarteschlange mit Informationen zur Wiederherstellung unvollständiger Transaktionen. Diese Warteschlange wird erstellt, wenn ein Warteschlangenmanager erstellt wird.

Bei jeder Transaktion wird der Warteschlange SYSTEM.DOTNET.XARECOVERY.QUEUE während der Vorbereitungsphase eine persistente Nachricht mit den Wiederherstellungsinformationen hinzugefügt . Die Nachricht wird gelöscht, wenn die Festschreibung erfolgreich war.

Anmerkung: Die Warteschlange SYSTEM.DOTNET.XARECOVERY.QUEUE darf nicht gelöscht werden.

Anwendung 'WMQDotnetXAMonitor'

IBM MQ Die XA-Überwachungsanwendung 'WmqDotnetXAMonitor' für .NET ist eine von .NET verwaltete Anwendung, die einen Warteschlangenmanager überwacht, Nachrichten in der Warteschlange SYSTEM.DOTNET.XARECOVERY.QUEUE verarbeitet und unvollständige Transaktionen wiederherstellt.

Wenn der Nachrichtenkanalagent die Nachricht nicht in die Zielwarteschlange einreihen kann, wird ein Ausnahmebericht mit der ursprünglichen Nachricht erstellt und in eine Übertragungswarteschlange

eingereiht, damit sie in die in der ursprünglichen Nachricht angegebene Empfangswarteschlange für Antworten gesendet werden kann. (Wenn sich die Warteschlange für Antwortnachrichten auf demselben WS-Manager wie der Nachrichtenkanalmanager befindet, wird die Nachricht direkt in diese Warteschlange gestellt, nicht in eine Übertragungswarteschlange.)

In den folgenden Fällen gilt eine Transaktion als unvollständig und wird wiederhergestellt:

- Wenn die Transaktion vorbereitet ist, aber COMMIT nicht innerhalb des Zeitlimitintervalls abgeschlossen wurde.
- Wenn die Transaktion vorbereitet ist, aber der IBM MQ-Warteschlangenmanager inaktiv ist.
- Wenn die Transaktion vorbereitet ist, aber der Transaktionsmanager inaktiv wird.

  Die XA-Überwachungsanwendung muss aus dem gleichen System ausgeführt werden, auf dem Ihre IBM MQ .NET-Clientanwendung aktiv ist. Wenn Anwendungen, die auf mehreren System ausgeführt werden, eine Verbindung zum gleichen Warteschlangenmanager herstellen, muss die Anwendung 'WmqDotnetXAMonitor' auf allen Systemen ausgeführt werden. Obwohl auf jedem Clientsystem eine Instanz der XA-Überwachungsanwendung zum Wiederherstellen der Anwendung aktiv ist, sollte jede Instanz der XA-Überwachungsanwendung in der Lage sein, die Nachricht zu ermitteln, die einer Transaktion zugeordnet ist, die vom lokalen MS-DTC der aktuellen XA-Überwachungsanwendung koordiniert wird, damit diese wieder eingetragen und abgeschlossen werden kann.

Zugehörige Konzepte

„Transaktionen wiederherstellen: Anwendungsfälle für IBM MQ .NET” auf Seite 603
Es gibt mehrere unterschiedliche Anwendungsfälle, die das Wiederherstellen von Transaktionen nötig machen können.

Zugehörige Tasks

„Anwendung 'WMQDotnetXAMonitor' verwenden” auf Seite 604
Der IBM MQ .NET-Client stellt eine XA-Überwachungsanwendung namens 'WmqDotnetXAMonitor' zur Verfügung, mit der Sie alle unvollständigen verteilten Transaktionen wiederherstellen können. Die Anwendung 'WmqDotnetXAMonitor' stellt eine Verbindung zum Warteschlangenmanager mit unbestätigten Transaktionen her und löst anschließend die Transaktion gemäß den von Ihnen festgelegten Parametern auf.

Transaktionen wiederherstellen: Anwendungsfälle für IBM MQ .NET

Es gibt mehrere unterschiedliche Anwendungsfälle, die das Wiederherstellen von Transaktionen nötig machen können.

- **IBM MQ-Anwendung mit einzelnem DTC-Dienst und einzelner Warteschlangenmanagerinstanz:** Wenn Sie in diesem Anwendungsfall eine Verbindung zum Warteschlangenmanager herstellen und die Arbeitseinheit (Unit of Work, UoW) unter der Transaktion ausführen, stellt die XA-Überwachungsanwendung die Transaktion wieder her und schließt sie ab, falls die Transaktion fehlschlägt und nicht vollständig beendet werden kann.

  In diesem Anwendungsfall wird eine einzelne Instanz der XA-Überwachungsanwendung ausgeführt, da den Transaktionen ein einziger Warteschlangenmanager zugeordnet ist.
- **Mehrere IBM MQ-Anwendungen mit einzelnem DTC-Dienst und einzelner Warteschlangenmanagerinstanz:** In diesem Anwendungsfall sind mehrere IBM MQ-Anwendungen in einem einzelnen DTC-Dienst vorhanden und alle sind mit dem gleichen Warteschlangenmanager verbunden und führen Arbeitseinheiten in den Transaktionen aus.

  Wenn die Transaktionen fehlschlagen und unvollständig sind, werden die Transaktionen, die alle Anwendungen betreffen, von der XA-Überwachungsanwendung wiederhergestellt und abgeschlossen.

  In diesem Anwendungsfall wird eine einzelne XA-Überwachungsanwendungsinstanz ausgeführt, da in den Transaktionen ein einziger Warteschlangenmanager verwendet wird.
- **Mehrere IBM MQ-Anwendungen, mehrere DTC-Dienste und unterschiedliche Warteschlangenmanagerinstanzen:** In diesem Anwendungsfall sind mehrere IBM MQ-Anwendungen in verschiedenen DTC-Diensten vorhanden (d. h. jede Anwendung wird auf einem anderen System ausgeführt) und stellen Verbindungen zu unterschiedlichen Warteschlangenmanagern her.

Wenn ein Fehler auftritt und die Transaktion unvollständig ist, überprüft die Überwachungsanwendung den Wert von 'TransactionManagerWhereabouts' in der Nachricht, um die DTC-Adresse zu ermitteln. Wenn der Wert von 'TransactionManagerWhereabouts' mit der DTC-Adresse übereinstimmt, unter der die Überwachung ausführt wird, wird die Wiederherstellung abgeschlossen. Andernfalls wird die Suche fortgesetzt, bis die der DTC-Adresse entsprechende Nachricht gefunden wird.

In diesem Anwendungsfall wird pro Client (Benutzer oder Computer) nur eine Instanz der XA-Überwachungsanwendung ausgeführt, da jeder Client über einen eigenen Warteschlangenmanager verfügt, der in Transaktionen verwendet wird.

- **Mehrere IBM MQ-Anwendungen, mehrere DTC-Dienste, mehrere identische Warteschlangenmanagerinstanzen:** In diesem Anwendungsfall sind mehrere IBM MQ-Anwendungen in verschiedenen DTC-Diensten vorhanden (d. h. jede Anwendung wird auf einem anderen System ausgeführt) und alle alle stellen Verbindungen zum gleichen Warteschlangenmanager her.

  Wenn ein Fehler auftritt und die Transaktion unvollständig ist, prüft die Überwachungsanwendung den Wert von 'TransactionManagerWhereabouts' in der Nachricht, um sicherzustellen, dass die DTC-Adresse und der DTC-Wert mit dem DTC-Dienst übereinstimmt, in dem die Überwachung ausgeführt wird. Wenn beide Werte übereinstimmen, wird die Wiederherstellung abgeschossen. Andernfalls wird die Suche fortgesetzt, bis die der DTC-Adresse entsprechende Nachricht gefunden wird.

  In diesem Anwendungsfall wird pro Client (Benutzer oder Computer) nur eine einzelne Instanz der XA-Überwachungsanwendung ausgeführt, da jeder Client über seine eigene Warteschlangenmanagerzuordnung verfügt, die in Transaktionen verwendet wird.
- **Mehrere IBM MQ-Anwendungen, einzelner DTC-Dienst, verschiedene Warteschlangenmanagerinstanzen:** In diesem Anwendungsfall sind mehrere IBM MQ-Anwendungen in einem einzelnen DTC-Dienst vorhanden (d. h. auf einem Computer werden mehrere IBM MQ-Anwendungen ausgeführt) und stellen Verbindungen zu unterschiedlichen Warteschlangenmanagern her.

  Wenn die Transaktion fehlschlägt und unvollständig ist, stellt die Überwachungsanwendung die Transaktion wieder her.

  In diesem Anwendungsfall werden so viele Instanzen der Überwachungsanwendung ausgeführt, wie Verbindungen zum Warteschlangenmanager vorhanden sind, da jede Anwendung über einen eigenen Warteschlangenmanager verfügt, der in Transaktionen verwendet wird, und jede Transaktion wiederhergestellt werden muss.

  Anmerkung: Wenn die XA-Überwachungsanwendung nicht im Hintergrund ausgeführt wird, können Sie sie starten.

Zugehörige Konzepte

„Prozess der Transaktionswiederherstellung für IBM MQ .NET” auf Seite 602
In diesem Abschnitt ist beschrieben, wie verteilte Transaktionen mit IBM MQ .NET-Klassen wiederhergestellt werden können.

Zugehörige Tasks

„Anwendung 'WMQDotnetXAMonitor' verwenden” auf Seite 604
Der IBM MQ .NET-Client stellt eine XA-Überwachungsanwendung namens 'WmqDotnetXAMonitor' zur Verfügung, mit der Sie alle unvollständigen verteilten Transaktionen wiederherstellen können. Die Anwendung 'WmqDotnetXAMonitor' stellt eine Verbindung zum Warteschlangenmanager mit unbestätigten Transaktionen her und löst anschließend die Transaktion gemäß den von Ihnen festgelegten Parametern auf.

Anwendung 'WMQDotnetXAMonitor' verwenden

Der IBM MQ .NET-Client stellt eine XA-Überwachungsanwendung namens 'WmqDotnetXAMonitor' zur Verfügung, mit der Sie alle unvollständigen verteilten Transaktionen wiederherstellen können. Die Anwendung 'WmqDotnetXAMonitor' stellt eine Verbindung zum Warteschlangenmanager mit unbestätigten Transaktionen her und löst anschließend die Transaktion gemäß den von Ihnen festgelegten Parametern auf.

Informationen zu diesem Vorgang

Die Anwendung 'WMQDotnetXAMonitor' muss manuell ausgeführt werden. Sie kann zu jedem Zeitpunkt gestartet werden. Sie kann gestartet werden, wenn die Nachrichten in SYSTEM.DOTNET.XARECOVERY.QUEUE angezeigt werden, oder sie kann weiterhin im Hintergrund ausgeführt werden, bevor Sie transaktionsorientierte Arbeitsvorgänge mit den Anwendungen ausführen, die mithilfe der IBM MQ .NET-Klassen geschrieben wurden.

Sie können die Parameterwerte für WMQDotnetXAMonitor entweder über die Befehlszeile oder in einer Anwendungskonfigurationsdatei festlegen. Werte, die in einer Anwendungskonfigurationsdatei bereitgestellt werden, haben Vorrang vor Werten, die über die Befehlszeile festgelegt werden.

Vor IBM MQ 9.3.0ist die Verbindung, die WMQDotnetXAMonitor herstellt, eine nicht sichere Verbindung.

Ab IBM MQ 9.3.0 haben Sie die Möglichkeit, eine sichere Verbindung zum Warteschlangenmanager herzustellen, indem Sie zusätzliche Parameter für WMQDotnetXAMonitor festlegen.

Prozedur

- Informationen zum Bereitstellen von Eingaben für WmqDotNETXAMonitor mithilfe einer Anwendungskonfigurationsdatei finden Sie im Abschnitt „Einstellungen in der Anwendungskonfigurationsdatei für WmqDotNETXAMonitor” auf Seite 607.
- Starten Sie die Anwendung 'WMQDotnetXAMonitor' über die Befehlszeile mit dem folgenden Befehl und den in Ihrem Fall erforderlichen Parametern:

  Vor IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
  ```

  Ab IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
  Repository -s Cipher Spec
  ```

  Sie können die folgenden Parameter angeben:

  – **-m *Warteschlangenmanagername***
  Gibt den Namen des Warteschlangenmanagers an.

  Optional

  -n *Verbindungsname*
  Der Verbindungsname im Format 'Host(Port)'. Der Wert für *Verbindungsname* kann die Namen mehrerer Verbindungen umfassen. Mehrere Verbindungsnamen müssen in einer durch Kommas getrennten Liste angegeben werden. Beispiel: `localhost (1414), localhost (1415), localhost (1416)`. Die Anwendung 'WMQDotnetXAMonitor' führt die Wiederherstellung für alle Verbindungsnamen aus, die in der durch Kommas getrennten Liste angegeben sind.

  -c *ChannelName*
  Der Kanalname.

  -i
  Abschluss von heuristischen Verzweigungen.

  Optional

  -k *SSL-Schlüsselrepository*
  Der Name des SSL-Schlüsselrepositorys. Folgende Werte werden unterstützt:

  - *SYSTEM (Standardwert)
  - *Benutzer

  Optional

-s *CipherSpec*
Die CipherSpec, die Sie festlegen, muss eine der CipherSpecs für die unterstützte Version sein; vorzugsweise ist sie mit der CipherSpecs identisch, die in der Windows-Gruppenrichtlinie angegeben ist. Weitere Informationen finden Sie unter „CipherSpec-Unterstützung für den verwalteten .NET-Client" auf Seite 628.

Zum das Herstellen einer sicheren Verbindung zum Warteschlangenmanager muss dieser Parameter angegeben werden.

-dn *SSL-Peer-Name*
Der SSL-Peer-Name mit dem der definierte Namen (DN) des Zertifikats vom Peerwarteschlangenmanager überprüft wird.

Optional

-cl *Zertifikatsbezeichnung*
Der Name, der das Zertifikat bezeichnet.

Optional

-sn *Abgehende SNI*
Gibt an, ob als Servernamensangabe (Server Name Indication, SNI) beim Starten einer TLS-Verbindung der Name des IBM MQ-Zielkanals oder der Hostname festgelegt werden soll. Für diese Option werden die folgenden Werte unterstützt:

- CHANNEL (Standardwert)
- HOSTNAME
- *

Wenn kein Wert festgelegt ist, wird der Standardwert verwendet, also `CHANNEL`.

Optional

-cr *Zertifikatswiderrufsprüfung*
Gibt an, ob geprüft werden soll, ob die Zertifizierung widerrufen wurde. Für diese Option werden die folgenden Werte unterstützt:

- true
- false (Standardwert)

Optional

-kr *Zähler für Schlüsselzurücksetzung*
Die Gesamtzahl der unverschlüsselten Byte, die über den Kanal gesendet und empfangen werden können, bevor der für die Verschlüsselung verwendete geheime Schlüssel neu vereinbart wird.

Der Standardwert 0 gibt an, dass geheime Schlüssel in keinem Fall neu vereinbart werden.

Optional

Die Anwendung 'WMQDotnetXAMonitor' führt die folgenden Aktionen aus:

1. Sie überprüft die Warteschlangenlänge von SYSTEM.DOTNET.XARECOVERY.QUEUE mit einem Intervall von 100 Sekunden.
2. Wenn die Warteschlangenlänge größer als null ist, durchsucht sie die Warteschlange auf Nachrichten und prüft, ob die Nachrichten die Kriterien der unvollständigen Transaktion erfüllen.
3. Wenn eine der Nachrichten die Kriterien der unvollständigen Transaktion erfüllt, wird sie von der Überwachung zurückgezogen und die Informationen zur Wiederherstellung der Transaktion werden abgerufen.
4. Anschließend wird ermittelt, ob sich die Wiederherstellungsinformationen auf die lokale Instanz von Microsoft Distributed Transaction Coordinator (MS DTC) beziehen. Wenn dies zutrifft, stellt WMQDotnetXAMonitor danach die Transaktion wieder her. Andernfalls wird die nächste Nachricht durchsucht.
5. Die Überwachungsanwendung löst Aufrufe an den Warteschlangenmanager aus, um die unvollständige Transaktion wiederherzustellen.

Einstellungen in der Anwendungskonfigurationsdatei für WmqDotNETXAMonitor
Mithilfe einer Anwendungskonfigurationsdatei können Sie Eingaben für die XA-Überwachungsanwendung 'WmqDotNETXAMonitor' von IBM MQ .NET bereitstellen. Ein Beispiel für eine Anwendungskonfigurationsdatei wird mit IBM MQ .NET geliefert. Diese Beispieldatei kann nach Ihren Anforderungen geändert werden.

Bei der Berücksichtigung der Eingabewerte hat die Anwendungskonfigurationsdatei die höchste Priorität. Wenn Eingabewerte sowohl in der Befehlszeile (siehe „Anwendung 'WMQDotnetXAMonitor' verwenden" auf Seite 604) als auch in der Anwendungskonfigurationsdatei bereitgestellt werden, haben die Werte aus der Anwendungskonfigurationsdatei Vorrang.

Beispielanwendungskonfigurationsdatei für vor IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

Konfigurationsdatei der Beispielanwendung aus IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

Anwendungsprotokoll 'WmqDotNetXAMonitor'
Die Überwachungsanwendung erstellt eine Protokolldatei im Anwendungsverzeichnis zur Protokollierung des Fortschritts der Überwachung und des Status der Transaktionswiederherstellung. Die Protokollierung beginnt mit dem Verbindungsnamen und den Einzelheiten zum Kanal, um den aktuellen Warteschlangenmanager anzuzeigen, für den die Wiederherstellung ausgeführt wird.

Sobald die Wiederherstellung startet, werden die Nachrichten-ID (MessageId) der Nachricht zur Transaktionswiederherstellung, die Transaktions-ID (TransactionId) der unvollständigen Transaktion und das tatsächliche Ergebnis der Transaktion gemäß der Koordination des Transaktionsmanagers protokolliert.

Beispielprotokolldatei:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

IBM MQ .NET-Programme schreiben und implementieren

Für den Zugriff von IBM MQ classes for .NET auf IBM MQ-Warteschlangen schreiben Sie Programme in einer beliebigen von .NET unterstützten Sprache, in der Aufrufe zum Einreihen von Nachrichten in die IBM MQ-Warteschlangen und Aufrufe zum Abrufen der Nachrichten aus den Warteschlangen enthalten sind.

Vorbereitende Schritte

V 9.4.0 Deprecated V 9.4.0 Ab IBM MQ 9.4.0sind in IBM MQ classes for .NETdie Methoden WriteObject(), ReadObject(), CreateObjectMessage () und die Klassen ObjectMessage und XmsObjectMessageImpl , die für die Serialisierung und Deserialisierung von Daten verwendet werden, veraltet.

V 9.4.0 V 9.4.0 Removed Die mit .NET Standard 2.0erstellte IBM MQ .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

Informationen zu diesem Vorgang

In der IBM MQ-Dokumentation sind nur Informationen zu den Programmiersprachen C#, C++ und Visual Basic enthalten.

Die Themen in diesem Abschnitt enthalten Informationen, die Sie beim Schreiben von Anwendungen für die Interaktion mit IBM MQ -Systemen unterstützen. Einzelheiten zu einzelnen Klassen finden Sie unter IBM MQ .NET-Klassen und -Schnittstellen.

Unterschiede bei der Verbindung

Die Vorgehensweise bei der Programmierung von IBM MQ.NET hat einige Auswirkungen auf den Verbindungsmodus, den Sie verwenden möchten.

Bei der Verwendung von IBM MQ classes for .NET als verwalteten Client gibt es eine Reihe von Unterschieden zum standardmäßigen IBM MQ MQI client, da einige Funktionen für einen verwalteten Client nicht verfügbar sind.

IBM MQ.NET ermittelt aus den Einstellungen, die Sie für den Verbindungsnamen, den Kanalnamen, den Anpassungswert NMQ_MQ_LIB und die Eigenschaft MQC.TRANSPORT_PROPERTY angeben, welcher Verbindungstyp verwendet werden soll.

Verwaltete Clientverbindungen

Bei der Verwendung von IBM MQ classes for .NET als verwalteten Client gibt es eine Reihe von Unterschieden zum standardmäßigen IBM MQ MQI client.

Sie folgenden Funktionen sind für einen verwalteten Client nicht verfügbar:

- Kanalkomprimierung
- Kettung von Kanalexits

Wenn Sie versuchen, diese Funktionen bei einem verwalteten Client zu verwenden, wird er eine MQ-Ausnahmebedingung zurückgeben. Wenn der Fehler auf der Clientseite der Verbindung festgestellt wird, wird

der Ursachencode MQRC_ENVIRONMENT_ERROR ausgegeben. Wenn er auf der Serverseite festgestellt wird, wird der Ursachencode vom Server verwendet.

Kanalexits, die für einen nicht verwalteten Client geschrieben werden, funktionieren nicht. Sie müssen speziell für den verwalteten Client neue Exits schreiben. Stellen Sie sicher, dass in Ihrer Definitionstabelle für den Clientkanal (CCDT) keine ungültigen Kanalexits angegeben sind.

Der Name eines verwalteten Kanalexits kann eine Länge von bis zu 999 Zeichen haben. Wenn Sie die Definitionstabelle für den Clientkanal verwenden, um den Namen des Kanalexits anzugeben, ist die Länge allerdings auf 128 Zeichen begrenzt.

Die Kommunikation wird nur über TCP/IP unterstützt.

Wenn Sie einen Warteschlangenmanager mit dem Befehl **endmqm** stoppen, kann das Schließen eines Serververbindungskanals zu einem verwalteten .NET-Client mehr Zeit in Anspruch nehmen als das Schließen von Serververbindungskanälen zu anderen Clients.

Wenn Sie *NMQ_MQ_LIB* auf `managed` gesetzt haben, um die IBM MQ-Problemdiagnose für verwaltete Clients zu verwenden, wird keiner der Parameter -i, -p, -s, -b oder -c des Befehls **strmqtrc** unterstützt.

Eine verwaltete .NET-Anwendung, in der XA-Transaktionen verwendet werden, funktioniert nicht mit einem z/OS-Warteschlangenmanager. Ein verwalteter .NET-Client, der versucht, eine Verbindung zu einem z/OS-Warteschlangenmanager herzustellen, schlägt beim MQOPEN-Aufruf mit dem Fehler MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354) fehl. Eine verwaltete .NET-Anwendung, in der XA-Transaktionen verwendet werden, kann allerdings mit dem verteilten Warteschlangenmanager ausgeführt werden.

Definition des zu verwendenden Verbindungstyps

Der Verbindungstyp wird aus den Einstellungen ermittelt, die für den Verbindungsnamen, den Kanalnamen, den Anpassungswert NMQ_MQ_LIB und die Eigenschaft MQC.TRANSPORT_PROPERTY angegeben sind.

Sie können den Verbindungsnamen folgendermaßen angeben:

- Explizit in einem MQQueueManager-Konstruktor:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Durch das Festlegen der Eigenschaften MQC.HOST_NAME_PROPERTY und MQC.PORT_PROPERTY (optional) in einem Hashtabelleneintrag eines MQQueueManager-Konstruktors:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Explizit als MQEnvironment-Werte:

```
MQEnvironment.Hostname
```

  `MQEnvironment.Port` (optional).
- Durch das Festlegen der Eigenschaften MQC.HOST_NAME_PROPERTY und MQC.PORT_PROPERTY (optional) in der Hashtabelle von MQEnvironment.properties.

Sie können den Kanalnamen folgendermaßen angeben:

- Explizit in einem MQQueueManager-Konstruktor:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Durch das Festlegen der Eigenschaft MQC.CHANNEL_PROPERTY in einem Hashtabelleneintrag eines MQQueueManager-Konstruktors:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Explizit als ein MQEnvironment-Wert:

```
MQEnvironment.Channel
```

- Durch das Festlegen der Eigenschaft MQC.CHANNEL_PROPERTY in der Hashtabelle von MQEnvironment.properties.

Sie können die Transporteigenschaft folgendermaßen angeben:

- Durch das Festlegen der Eigenschaft MQC.TRANSPORT_PROPERTY in einem Hashtabelleneintrag eines MQQueueManager-Konstruktors:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Durch das Festlegen der Eigenschaft MQC.TRANSPORT_PROPERTY in der Hashtabelle von MQEnvironment.properties.

Wählen Sie mittels der folgenden Werte den benötigten Verbindungstyp aus:

MQC.TRANSPORT_MQSERIES_BINDINGS - Verbindung als Server
MQC.TRANSPORT_MQSERIES_CLIENT - Verbindung als Nicht-XA-Client
MQC.TRANSPORT_MQSERIES_XACLIENT - Verbindung als XA-Client
MQC.TRANSPORT_MQSERIES_MANAGED - Verbindung als verwalteter Nicht-XA-Client

Sie können den Anpassungswert NMQ_MQ_LIB wie in der folgenden Tabelle gezeigt festlegen, um den Verbindungstyp explizit auszuwählen.

NMQ_MQ_LIB-Wert	**Verbindungstyp**
mqic.dll	Verbindung als Nicht-XA-Client
mqicxa.dll	Verbindung als XA-Client
mqm.dll	Verbindung als Server oder Nicht-XA-Client
managed	Verbindung als verwalteter Nicht-XA-Client

Anmerkung: Aus Gründen der Kompatibilität mit früheren Releases werden die Werte von 'mqic32.dll' und 'mqic32xa.dll' als Synonyme für 'mqic.dll' und 'mqicxa.dll' akzeptiert. Allerdings sind 'mqm.dll' und 'mqm.pdb' nur Komponenten des Clientpakets ab IBM WebSphere MQ 7.1.

Wenn Sie einen Verbindungstyp auswählen, der in Ihrer Umgebung nicht verfügbar ist, beispielsweise bei der Angabe von 'mqic32xa.dll' ohne XA-Unterstützung, gibt IBM MQ.NET eine Ausnahme aus.

Wenn Sie NMQ_MQ_LIB auf "managed" setzen, verwendet der Client verwaltete Diagnosetests für IBM MQ-Probleme, die .NET-Datenkonvertierung und weitere verwaltete und untergeordnete IBM MQ-Funktionen.

Bei allen anderen Werten für NMQ_MQ_LIB verwendet der .NET-Prozess nicht verwaltete Diagnosetests für IBM MQ-Probleme und die Datenkonvertierung und weitere nicht verwaltete und untergeordnete IBM MQ-Funktionen (vorausgesetzt, ein IBM MQ MQI client oder -Server ist auf dem System installiert).

IBM MQ.NET wählt den Verbindungstyp folgendermaßen aus:

1. Wenn MQC.TRANSPORT_PROPERTY angegeben ist, wird die Verbindung gemäß dem Wert von MQC.TRANSPORT_PROPERTY bereitgestellt.

Allerdings garantiert die MQC.TRANSPORT_PROPERTY-Einstellung MQC.TRANSPORT_MQSERIES_MANAGED nicht, dass der Clientprozess verwaltet ausgeführt wird. Selbst bei dieser Einstellung wird der Client in den folgenden Fällen nicht verwaltet:

- Wenn ein anderer Thread des Prozesses, der nicht auf MQC.TRANSPORT_MQSERIES_MANAGED gesetzt ist, mit MQC.TRANSPORT_PROPERTY verbunden ist.
- Wenn NMQ_MQ_LIB auf "managed" gesetzt ist, werden Diagnosetests für Probleme, die Datenkonvertierung und weitere untergeordnete Funktionen nicht vollständig verwaltet (vorausgesetzt, ein IBM MQ MQI client oder -Server ist auf dem System installiert).

2. Ist ein Verbindungsname ohne Kanalname oder umgekehrt angegeben worden, wird ein Fehler ausgegeben.
3. Sind sowohl Verbindungs- als auch Kanalname angegeben worden:
   - Ist f NMQ_MQ_LIB auf 'mqic32xa.dll' gesetzt, wird eine Verbindung als XA-Client hergestellt.
   - Ist NMQ_MQ_LIB auf 'managed' gesetzt, wird eine Verbindung als verwalteter Client hergestellt.
   - Andernfalls wird eine Verbindung als Nicht-XA-Client hergestellt.
4. Ist NMQ_MQ_LIB angegeben, wird die Verbindung gemäß dem Wert von NMQ_MQ_LIB hergestellt.
5. Ist ein IBM MQ-Server installiert, wird die Verbindung als Server hergestellt.
6. Ist ein IBM MQ MQI client installiert, wird die Verbindung als Nicht-XA-Client hergestellt.
7. Andernfalls wird die Verbindung als verwalteter Client hergestellt.

Windows Projektvorlage von IBM MQ .NET verwenden

Der IBM MQ .NET-Client bietet Ihnen die Möglichkeit, eine Projektvorlage zu verwenden, um Sie bei der Entwicklung Ihrer .NET Core-Anwendungen zu unterstützen.

Vorbereitende Schritte

Sie müssen über Microsoft Visual Studio 2017 oder höher und .NET Core 2.1 auf Ihrem System verfügen.

Sie müssen die .NET-Vorlage aus dem Verzeichnis

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip

in das Verzeichnis

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

kopieren. Dabei gilt Folgendes:

- *&MQ-INSTALLATIONSSTAMMVERZEICHNIS* ist das Stammverzeichnis Ihrer Installation.
- *&BENUTZERAUSGANGSVERZEIHNIS* ist Ihr Ausgangsverzeichnis.

Sie müssen Microsoft Visual Studio stoppen und erneut starten, um die Vorlage zu übernehmen.

Informationen zu diesem Vorgang

Die .NET-Projektvorlage enthält allgemeinen Code, den Sie zur Unterstützung bei der Entwicklung Ihrer Anwendungen verwenden können. Mit dem integrierten Code können Sie eine Verbindung zum IBM MQ-Warteschlangenmanager herstellen und eine Operation zum Einreihen oder Abrufen ausführen, indem Sie einfach die Eigenschaften in dem integrierten Code ändern.

Vorgehensweise

1. Öffnen Sie Microsoft Visual Studio.
2. Klicken Sie auf **File** (Datei), **New** (Neu) und dann **Project** (Projekt).

3. Wählen Sie im Fenster *Neues Projekt erstellen* `IBM MQ .NET Client App (.NET Core)` aus und klicken Sie auf **Weiter**.
4. Ändern Sie im Fenster *Configure your new project* (Neues Projekt konfigurieren) gegebenenfalls den Projektnamen (*Project name*) und klicken Sie auf **Create** (Erstellen), um das .NET-Projekt zu erstellen.

   Die Datei `MQDotnetApp.cs` wird zusammen mit der Projektdatei erstellt. Diese Datei enthält den Code für die Verbindung zum Warteschlangenmanager und führt eine Operation zum Einreihen und Abrufen aus.

   Die Verbindungseigenschaften sind auf Standardwerte gesetzt:

   - MQC.CONNECTION_NAME_PROPERTY ist auf *localhost(1414)* gesetzt.
   - MQC.CHANNEL_PROPERTY ist auf *DOTNET.SVRCONN* gesetzt.

   Die Warteschlange ist auf *Q1* gesetzt. Sie können diese Eigenschaften entsprechend ändern.
5. Kompilieren Sie die Anwendung und führen Sie sie aus.

Zugehörige Konzepte

Komponenten und Funktionen von IBM MQ

.NET-Anwendungslaufzeit - nur Windows

Konfigurationsdateien für IBM MQ classes for .NET

Eine .NET-Clientanwendung kann eine IBM MQ MQI client-Konfigurationsdatei und, falls Sie den Typ einer verwalteten Verbindung verwenden, eine .NET-Anwendungskonfigurationsdatei verwenden. Die Einstellungen in der Anwendungskonfigurationsdatei haben Priorität.

Clientkonfigurationsdatei

Eine IBM MQ classes for .NET-Clientanwendung kann eine Clientkonfigurationsdatei auf die gleiche Weise wie jeder andere IBM MQ MQI client verwenden. Diese Datei heißt gewöhnlich `mqclient.ini`, Sie können jedoch einen anderen Dateinamen angeben. Weitere Informationen zur Clientkonfigurationsdatei finden Sie unter IBM MQ MQI client -Konfigurationsdatei `mqclient.ini`.

Nur die folgenden Attribute in einer IBM MQ MQI client-Konfigurationsdatei sind für IBM MQ classes for .NET relevant. Wen Sie andere Attribute angeben, hat dies keine Auswirkungen.

Tabelle 77. Für IBM MQ classes for .NET relevante Attribute der Clientkonfigurationsdatei

Zeilengruppe	**Attribut**
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ReconDelay
CHANNELS	DefRecon
CHANNELS	MQReconnectTimeout
CHANNELS	ServerConnectionParms
CHANNELS	Put1DefaultAlwaysSync
CHANNELS	PasswordProtection
ClientExitPath	Standardpfad für Exits
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime

Tabelle 77. Für IBM MQ classes for .NET relevante Attribute der Clientkonfigurationsdatei (Forts.)	
Zeilengruppe	**Attribut**
MessageBuffer	UpdatePercentage
Sicherheit	Datei MQIInitialKey
SSL	SSLKeyRepository
SSL	Kennwort für SSLKeyRepository
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

Sie können diese Attribute mithilfe der entsprechenden Umgebungsvariable überschreiben.

Anwendungskonfigurationsdatei

Wenn Sie mit dem Typ der verwalteten Verbindung arbeiten, können Sie auch die Clientkonfigurationsdatei IBM MQ und die entsprechenden Umgebungsvariablen mit der Anwendungskonfigurationsdatei .NET überschreiben.

Die Einstellungen der .NET-Anwendungskonfigurationsdatei werden nur bei der Ausführung über die verwaltete Verbindung akzeptiert und bei anderen Verbindungstypen ignoriert.

Die .NET-Anwendungskonfigurationsdatei und das zugehörige Format werden von Microsoft für die allgemeine Verwendung im .NET-Framework definiert, aber die speziellen Abschnittsnamen, Schlüssel und Werte, die in dieser Dokumentation verwendet werden, sind spezifisch für IBM MQ.

Beim Format der .NET-Anwendungskonfigurationsdatei handelt es sich um eine Anzahl von *Abschnitten*. Jeder Abschnitt enthält mindestens einen *Schlüssel* und jedem Schlüssel ist ein *Wert* zugeordnet. Im folgenden Beispiel werden die Abschnitte, Schlüssel und Werte gezeigt, die in einer .NET-Anwendungskonfigurationsdatei zur Steuerung der Eigenschaft für das TCP/IP-Keepalive verwendet werden:

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

Die Schlüsselwörter, die in den Abschnittsnamen und Schlüsseln der .NET-Anwendungskonfigurationsdatei verwendet werden, stimmen exakt mit den Schlüsselwörtern für die Zeilengruppen und Attribute überein, die in der Clientkonfigurationsdatei definiert sind.

Der Abschnitt `<configSections>` muss das erste untergeordnete Element des Elements `<configuration>` sein.

Weitere Informationen finden Sie in der Microsoft-Dokumentation.

Beispiel eines C#-Codefragments für Verwendung mit .NET

In diesem C#-Codefragment wird gezeigt, dass eine Anwendung eine Verbindung zu einem Warteschlangenmanager herstellt, eine Nachricht in eine Warteschlange einreiht und eine Antwort empfängt.

Das folgende C#-Codefragment zeigt eine Anwendung, die drei Aktionen ausführt:

1. Verbindung zu einem Warteschlangenmanager herstellen

2. Nachricht in SYSTEM.DEFAULT.LOCAL.QUEUE einreihen
3. Nachricht zurückerhalten

Es wird auch gezeigt, die der Verbindungstyp geändert wird.

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }

    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

      // Set up the options on the queue we want to open
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;
```

```
      // Now specify the queue that we want to open,and the open options
      MQQueue system_default_local_queue =
        qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

      // Define an IBM MQ message, writing some text in UTF format
      MQMessage hello_world = new MQMessage();
      hello_world.WriteUTF("Hello World!");

      // Specify the message options
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                           // same as MQPMO_DEFAULT

      // Put the message on the queue
      system_default_local_queue.Put(hello_world, pmo);



      // Get the message back again

      // First define an IBM MQ message buffer to receive the message
      MQMessage retrievedMessage =new MQMessage();
      retrievedMessage.MessageId =hello_world.MessageId;

      // Set the get message options
      MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                          //same as MQGMO_DEFAULT

      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

IBM MQ-Umgebung einrichten

Damit Sie über die Clientverbindung eine Verbindung zu einem Warteschlangenmanager herstellen können, müssen Sie zuerst die IBM MQ-Umgebung einrichten.

Anmerkung: Dieser Schritt ist nicht erforderlich, wenn Sie IBM MQ classes for .NET im Serverbindungsmodus verwenden.

Mit der .NET-Programmierschnittstelle können Sie den Anpassungswert NMQ_MQ_LIB verwenden, es wird aber auch eine MQEnvironment-Klasse eingeschlossen. Mit dieser Klasse können Sie Einzelheiten angeben, die während des Verbindungsversuchs verwendet werden sollen, wie beispielsweise die Einträge in der folgenden Liste:

- Kanalname
- Hostname
- Portnummer

- Kanalexits
- SSL-Parameter
- Benutzer-ID und Kennwort

Vollständige Informationen zur MQEnvironment-Klasse finden Sie unter MQEnvironment.NET -Klasse .

Verwenden Sie zur Angabe des Kanalnamens und des Hostnamens folgenden Code:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

Der Client versucht standardmäßig, eine Verbindung zu einem IBM MQ-Listener auf Port 1414 herzustellen. Wenn Sie einen anderen Port angeben möchten, verwenden Sie folgenden Code:

```
MQEnvironment.Port = nnnn;
```

Verbindung zu einem Warteschlangenmanager herstellen und trennen

Nach der Konfiguration der IBM MQ-Umgebung können Sie eine Verbindung zu einem Warteschlangenmanager herstellen.

Um eine Verbindung zu einem Warteschlangenmanager herzustellen, erstellen Sie eine neue Instanz der Klasse 'MQQueueManager':

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Um die Verbindung zu einem Warteschlangenmanager zu trennen, rufen Sie die Methode `Disconnect` auf dem Warteschlangenmanager aus:

```
queueManager.Disconnect();
```

Sie müssen über die Abfrageberechtigung (`inq`) für den Warteschlangenmanager verfügen, wenn Sie versuchen, eine Verbindung zum Warteschlangenmanager herzustellen. Ohne die Berechtigung zum Abfragen schlägt der Verbindungsversuch fehl.

Wenn Sie die `Disconnect`-Methode aufrufen, werden alle offenen Warteschlangen und Prozesse, auf die Sie über diesen Warteschlangenmanager zugegriffen haben, geschlossen. Es wird jedoch aus programmiertechnischen Gründen empfohlen, die Ressourcen explizit zu schließen, wenn Sie mit ihrer Verwendung fertig sind. Zum Schließen der Ressourcen verwenden Sie die `Close`-Methode in dem Objekt, das der jeweiligen Ressource zugeordnet ist.

Die Methoden `Commit` und `Backout` auf einem Warteschlangenmanager ersetzen die MQCMIT- und MQBACK-Aufrufe, die mit der prozeduralen Schnittstelle verwendet werden.

Zugriff auf Warteschlangen und Themen

Sie können mithilfe von MQQueueManager-Methoden oder geeigneten Konstruktoren auf Warteschlangen und Themen zugreifen.

Verwenden Sie für den Zugriff auf Warteschlangen die Methoden der MQQueueManager-Klasse. Die Objektdeskriptorstruktur MQOD wird in den Parametern dieser Methoden komprimiert. Verwenden Sie beispielsweise folgenden Code, wenn Sie eine Warteschlange auf einem Warteschlangenmanager öffnen möchten, der durch ein MQQueueManager-Objekt dargestellt wird und die Bezeichnung 'queueManager' hat:

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
```

```
                                        "dynamicQName",
                                        "altUserId");
```

Der Parameter *options* ist identisch mit dem Options-Parameter im MQOPEN-Aufruf.

Die AccessQueue-Methode gibt ein neues Objekt der MQQueue-Klasse zurück.

Wenn Sie die Warteschlange nicht mehr benötigen, schließen Sie sie mit der Close()-Methode wie im folgenden Beispiel:

```
queue.Close();
```

Mit IBM MQ .NET können Sie auch einen MQQueue-Konstruktor verwenden, um eine Warteschlange zu erstellen. Die Parameter entsprechen den Parametern für die Methode 'accessQueue', wobei zusätzlich ein Warteschlangenmanagerparameter verwendet wird, mit dem das instanziierte MQQueueManager-Objekt angegeben wird, das verwendet werden soll. For example:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

Wenn Sie ein Warteschlangenobjekt auf diese Weise erstellen, können Sie Ihre eigenen Unterklassen von MQQueue schreiben.

Auf ähnliche Weise können Sie mit den Methoden der MQQueueManager-Klasse auch auf Themen zugreifen. Zum Öffnen eines Themas verwenden Sie die Methode 'AccessTopic()'. Dadurch wird ein neues Objekt der MQTopic-Klasse zurückgegeben. Wenn Sie das Thema nicht mehr benötigen, schließen Sie es mit der Methode 'Close()' der MQTopic-Klasse.

Sie können ein Thema auch mit einem MQTopic-Konstruktor erstellen. Es gibt eine Reihe von Konstruktoren für Themen; weitere Informationen dazu finden Sie im Abschnitt MQTopic.NET-Klasse.

Nachrichten bearbeiten

Nachrichten werden mit den Methoden der Warteschlangen- oder Themenklasse behandelt. Wenn Sie eine neue Nachricht generieren möchten, erstellen Sie ein neues MQMessage-Objekt.

Mit der Put()-Methode der MQQueue- oder MQTopic-Klasse können Sie Nachrichten in Warteschlangen oder Themen einreihen. Mit der Get()-Methode der MQQueue- oder MQTopic-Klasse können Sie Nachrichten aus Warteschlangen oder Themen abrufen. Im Gegensatz zur prozeduralen Schnittstelle, in der Byte-Feldgruppen mit MQPUT und MQGET eingereiht und abgerufen werden, werden in den IBM MQ classes for .NET Instanzen der MQMessage-Klasse eingereiht und abgerufen. Die MQMessage-Klasse umfasst den Datenpuffer, der die eigentlichen Nachrichtendaten sowie alle MQMD-Parameter (MQMD=Nachrichtendeskriptor) enthält, die diese Nachricht beschreiben.

Wenn Sie eine neue Nachricht generieren möchten, erstellen Sie eine neue Instanz der MQMessage-Klasse und verwenden die WriteXXX-Methoden, um Daten in den Nachrichtenpuffer zu schreiben.

Bei der Erstellung der neuen Nachrichteninstanz werden alle MQMD-Parameter automatisch auf ihre Standardwerte gesetzt, wie in Anfangswerte und Sprachdeklarationen für MQMD definiert. Die Put()-Methode von MQQueue hat als weiteren Parameter eine Instanz der MQPutMessageOptions-Klasse. Diese Klasse stellt die MQPMO-Struktur dar. Im folgenden Beispiel wird eine Nachricht erstellt und in eine Warteschlange eingereiht:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);
```

```
// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.Put(myMessage,pmo);
```

Die Get()-Methode von MQQueue gibt eine neue Instanz von MQMessage zurück, die für die Nachricht steht, die gerade aus der Warteschlange abgerufen wurde. Sie hat als weiteren Parameter eine Instanz der MQGetMessageOptions-Klasse. Diese Klasse stellt die MQGMO-Struktur dar.

Sie müssen keine maximale Nachrichtenlänge angeben, da die Get()-Methode die Größe des internen Puffers automatisch an die ankommende Nachricht anpasst. Verwenden Sie die ReadXXX-Methoden der MQMessage-Klasse, um auf die Daten in der Antwortnachricht zuzugreifen.

Das folgende Beispiel zeigt, wie eine Nachricht aus einer Warteschlange abgerufen wird:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

Sie können das von den Lese- und Schreibmethoden (read und write) verwendete Zahlenformat ändern, indem Sie die Elementvariable *encoding* entsprechend festlegen.

Sie können den Zeichensatz ändern, der für Lese- und Schreibzeichenfolgen verwendet wird, indem Sie die Elementvariable *characterSet* entsprechend festlegen.

Weitere Einzelheiten finden Sie unter MQMessage.NET-Klasse.

Anmerkung: Die WriteUTF()-Methode von MQMessage verschlüsselt automatisch die Länge der Zeichenfolge und die enthaltenen Unicode-Bytes. Wenn Ihre Nachricht von einem anderen .NET-Programm gelesen wird (unter Verwendung von readUTF()), ist dies die einfachste Art, Zeichenfolgeinformationen zu senden.

Nachrichteneigenschaften bearbeiten

Mittels Nachrichteneigenschaften können Nachrichten ausgewählt oder Informationen zu einer Nachricht abgerufen werden, ohne auf den Nachrichtenheader zugreifen zu müssen. Die MQMessage-Klasse enthält Methoden zum Abrufen und Festlegen von Eigenschaften.

Mittels Nachrichteneigenschaften ermöglichen Sie es einer Anwendung, die zu verarbeitenden Nachrichten auszuwählen oder Informationen zu einer Nachricht abzurufen, ohne auf den MQMD- oder den MQRFH2-Header zugreifen zu müssen. Sie erleichtern auch die Kommunikation zwischen IBM MQ- und JMS-Anwendungen. Weitere Informationen zu Nachrichteneigenschaften in IBM MQ finden Sie im Abschnitt Nachrichteneigenschaften.

Die MQMessage-Klasse stellt eine Reihe von Methoden zum Abrufen und Festlegen von Eigenschaften auf Basis des Eigenschaftentyps bereit. Die Namen der Abrufmethoden haben das Format 'Get*Property', die Namen der Festlegemethoden das Format 'Set*Property'. Der Stern (*) steht dabei für eine der folgenden Zeichenfolgen:

- Boolesch
- Byte
- Bytes
- Double
- Float
- Ganzzahl
- Int2
- Int4

- Int8
- Lang
- Objekt
- Kurz
- Zeichenfolge

Um beispielsweise die IBM MQ-Eigenschaft 'myproperty' (eine Zeichenfolge) abzurufen, verwenden Sie den Aufruf `message.GetStringProperty('myproperty')`. Optional können Sie einen Eigenschaftsdeskriptor übergeben, der von IBM MQ abgeschlossen wird.

Fehler bearbeiten

Fehler, die aufgrund von IBM MQ classes for .NET auftreten, werden mit den Blöcken `try` und `catch` behandelt.

Die Methoden in der .NET-Schnittstelle geben keinen Beendigungs- und Ursachencode zurück. Stattdessen lösen sie eine Ausnahmebedingung aus, wenn der Beendigungs- und Ursachencode eines IBM MQ-Aufrufs nicht beide null sind. Dadurch wird die Programmlogik vereinfacht, sodass Sie nicht nach jedem Aufruf an IBM MQ die Rückgabecodes prüfen müssen. Sie können entscheiden, an welchen Punkten Sie in Ihrem Programm die Möglichkeit von Fehlern zulassen möchten. An diesen Punkten können Sie Ihren Code in `try` -und `catch` -Blöcke einschließen, wie im folgenden Beispiel gezeigt:

```
try
{
  myQueue.Put(messageA,PutMessageOptionsA);
  myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
  // This block of code is only executed if one of
  // the two put methods gave rise to a non-zero
  // completion code or reason code.
  Console.WriteLine("An error occurred during the put operation:" +
                        "CC = " + ex.CompletionCode +
                        "RC = " + ex.ReasonCode);
  Console.WriteLine("Cause exception:" + ex );
}
```

Attributwerte abrufen und festlegen

Die Klassen MQManagedObject, MQQueue und MQQueueManager enthalten Methoden, mit denen Sie Ihre Attributwerte abrufen und festlegen können. Berücksichtigen Sie, dass die Methoden für MQQueue nur dann funktionieren, wenn Sie beim Öffnen der Warteschlange die entsprechenden Inquire- und Set-Flags angeben.

MQQueueManager- und MQQueue-Klassen übernehmen die gemeinsamen Attribute aus einer Klasse mit der Bezeichnung MQManagedObject. Diese Klasse definiert die Inquire()- und Set()-Schnittstellen.

Wenn Sie ein neues Warteschlangenmanagerobjekt mithilfe des Operators *new* erstellen, wird es automatisch für die Abfrage (inquire) geöffnet. Wenn Sie die Methode AccessQueue() verwenden, um auf ein Warteschlangenobjekt zuzugreifen, wird dieses Objekt nicht automatisch für eine Inquire- oder eine Set-Operation geöffnet. Dies kann zu Problemen mit einigen Typen von fernen Warteschlangen führen. Wenn Sie die Inquire- und Set-Methoden verwenden möchten, um Eigenschaften für eine Warteschlange festzulegen, müssen Sie die entsprechenden Inquire- und Set-Flags im Parameter 'openOptions' der AccessQueue()-Methode angeben.

Die Inquire- und Set-Methoden enthalten drei Parameter:

- selectors-Array
- intAttrs-Array
- charAttrs-Array

Sie benötigen nicht die in MQINQ enthaltenen Parameter SelectorCount, IntAttrCount und CharAttr-Length, da die Länge einer Feldgruppe immer bekannt ist. Das folgende Beispiel zeigt, wie in einer Warteschlange eine Abfrage durchgeführt wird:

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

Alle Attribute dieser Objekte können abgefragt werden. Ein Teil der Attribute ist als Objekteigenschaften verfügbar. Eine Liste der Objektattribute finden Sie im Abschnitt Objektattribute. Informationen zu Objekteigenschaften finden Sie in der entsprechenden Klassenbeschreibung.

Multithread-Programme

Die .NET-Laufzeitumgebung wird grundsätzlich als Multithread-Prozess ausgeführt. Mit IBM MQ classes for .NET kann ein Warteschlangenmanagerobjekt gemeinsam in mehreren Threads verwendet werden, es wird dabei allerdings sichergestellt, dass der gesamte Zugriff auf den Zielwarteschlangenmanager synchronisiert ist.

Stellen Sie sich ein einfaches Programm vor, das eine Verbindung zu einem Warteschlangenmanager herstellt und beim Systemstart eine Warteschlange öffnet. Das Programm zeigt eine einzige Schaltfläche auf dem Bildschirm an. Wenn ein Benutzer auf diese Schaltfläche klickt, ruft das Programm eine Nachricht aus der Warteschlange ab. In dieser Situation läuft die Initialisierung der Anwendung in einem Thread ab, und der Code, der infolge des Klickens auf die Schaltfläche ausgeführt wird, in einem anderen Thread (Thread der Benutzeroberfläche).

Die Implementierung der IBM MQ .NET sorgt dafür, dass bei einer bestimmten Verbindung (MQQueueManager-Objektinstanz) der gesamte Zugriff auf den IBM MQ-Zielwarteschlangenmanager synchronisiert ist. Das Standardverhalten ist, dass ein Thread, der einen Aufruf an einen Warteschlangenmanager ausgeben möchte, blockiert ist, bis alle anderen Aufrufe für diese Verbindung, die gerade verarbeitet werden, abgeschlossen sind. Wenn Sie simultanen Zugriff auf denselben Warteschlangenmanager von mehreren Threads innerhalb Ihres Programms benötigen, erstellen Sie ein neues MQQueueManager-Objekt für jeden Thread, der gleichzeitigen Zugriff anfordert. (Dies ist mit der Ausgabe eines eigenen MQCONN-Aufrufs für jeden einzelnen Thread gleichzusetzen.)

Wenn die Standardverbindungsoptionen von MQC.MQCNO_HANDLE_SHARE_NONE oder MQC.MQCNO_SHARE_NO_BLOCK überschrieben werden, ist der Warteschlangenmanager nicht mehr synchronisiert.

Definitionstabelle für den Clientkanal mit .NET verwenden

Sie können eine Clientkanaldefinitionstabelle (CCDT) mit IBM MQ classes for .NETverwenden. Sie können die Position der CCDT auf verschiedene Arten angeben, abhängig davon, ob Sie eine verwaltete oder nicht verwaltete Verbindung verwenden.

Verbindungstyp für den nicht verwalteten Nicht-XA- oder XA-Client

Bei einem nicht verwalteten Verbindungstyp können Sie die Position der CCDT auf zwei Arten angeben:

- Mithilfe der Umgebungsvariablen MQCHLLIB können Sie das Verzeichnis angeben, in dem sich die Tabelle befindet, und mithilfe von MQCHLTAB können Sie den Dateinamen der Tabelle angeben.
- Die Clientkonfigurationsdatei wird verwendet. Verwenden Sie in der Zeilengruppe CHANNELS die Attribute **ChannelDefinitionDirectory** , um das Verzeichnis anzugeben, in dem sich die Tabelle befindet, und **ChannelDefinitionFile** , um den Dateinamen anzugeben.

Wenn die Position sowohl in der Clientkonfigurationsdatei als auch unter Verwendung von Umgebungsvariablen angegeben wird, müssen die Umgebungsvariablen Priorität haben. Sie können diese Funktion verwenden, um eine Standardposition in der Clientkonfigurationsdatei anzugeben und diese bei Bedarf mit Umgebungsvariablen zu überschreiben.

Verwalteter Clientverbindungstyp

Bei einem verwalteten Verbindungstyp können Sie die Position der CCDT auf drei Arten angeben:

- Mithilfe der .NET-Anwendungskonfigurationsdatei. Verwenden Sie im Abschnitt CHANNELS die Schlüssel **ChannelDefinitionDirectory** , um das Verzeichnis anzugeben, in dem sich die Tabelle befindet, und **ChannelDefinitionFile** , um den Dateinamen anzugeben.
- Mithilfe der Umgebungsvariablen MQCHLLIB können Sie das Verzeichnis angeben, in dem sich die Tabelle befindet, und mithilfe von MQCHLTAB können Sie den Dateinamen der Tabelle angeben.
- Die Clientkonfigurationsdatei wird verwendet. Verwenden Sie in der Zeilengruppe CHANNELS die Attribute **ChannelDefinitionDirectory** , um das Verzeichnis anzugeben, in dem sich die Tabelle befindet, und **ChannelDefinitionFile** , um den Dateinamen anzugeben.

Wenn die Position auf mehr als eine dieser drei Arten angegeben wird, haben die Umgebungsvariablen Priorität vor der Clientkonfigurationsdatei und die .NET-Anwendungskonfigurationsdatei hat Priorität vor den beiden anderen Methoden. Mit dieser Funktion können Sie eine Standardposition in der Clientkonfigurationsdatei angeben und diese bei Bedarf mithilfe der Umgebungsvariablen oder der Anwendungskonfigurationsdatei überschreiben.

Ab IBM MQ 9.3.0verhält sich der .NET -Client genauso wie die C-und Java -Clients und gibt den Fehler MQRC_Q_MGR_NAME_ERROR zurück, wenn eine CCDT mit Warteschlangenmanagergruppierung verwendet wird.

So ermittelt eine .NET-Anwendung, welche Kanaldefinition verwendet werden soll

In der IBM MQ .NET-Clientumgebung kann die Kanaldefinition, die verwendet werden soll, auf unterschiedliche Arten angegeben werden. Es sind mehrere Spezifikationen der Kanaldefinition vorhanden. Eine Anwendung leitet die Kanaldefinition aus einer oder mehreren Quellen ab.

Wenn mehrere Kanaldefinitionen vorhanden sind, wird die Definition, die verwendet wird, nach folgender Priorität ausgewählt:

1. Eigenschaften, die im MQQueueManager-Konstruktor angegeben sind, entweder explizit oder durch die Integration von *MQC.CHANNEL_PROPERTY* in die Hashtabelle für die Eigenschaften
2. Die Eigenschaft *MQC.CHANNEL_PROPERTY* in der Hashtabelle 'MQEnvironment.properties'
3. Die Eigenschaft *Channel* in 'MQEnvironment'
4. Die .NET-Anwendungskonfigurationsdatei, Abschnittsname CHANNELS, Schlüssel 'ServerConnectionParms' (gilt nur für verwaltete Verbindungen)
5. Die Umgebungsvariable *MQSERVER*
6. Die Clientkonfigurationsdatei, Zeilengruppe CHANNELS, Attribut 'ServerConnectionParms'
7. Die Definitionstabelle für den Clientkanal (CCDT). Die Position der CCDT wird in der .NET-Anwendungskonfigurationsdatei angegeben (gilt nur für verwaltete Verbindungen)
8. Die Definitionstabelle für den Clientkanal (CCDT). Die Position der CCDT wird mithilfe der Umgebungsvariablen *MQCHLIB* und *MQCHLTAB* angegeben
9. Die Definitionstabelle für den Clientkanal (CCDT). Die Position der CCDT wird mithilfe der Clientkonfigurationsdatei angegeben

Bei den Punkten 1-3 wird die Kanaldefinition Feld für Feld aus den Werten erstellt, die von der Anwendung bereitgestellt werden. Diese Werte können mit unterschiedlichen Schnittstellen bereitgestellt werden und für jede Schnittstelle können mehrere Werte vorhanden sein. Feldwerte werden der Kanaldefinition gemäß der folgenden Prioritätenreihenfolge hinzugefügt:

1. Der Wert von *connName* im MQQueueManager-Konstruktor
2. Werte von Eigenschaften aus der Hashtabelle 'MQQueueManager.properties'
3. Werte von Eigenschaften aus der Hashtabelle 'MQEnvironment.properties'
4. Werte, die als MQEnvironment-Felder festgelegt sind (z. B. MQEnvironment.Hostname, MQEnvironment.Port)

Bei den Punkten 4-6 wird die gesamte Kanaldefinition als Wert bereitgestellt. Für nicht angegebene Felder in der Kanaldefinition wird die Standardeinstellung des Systems verwendet. Die Werte aus anderen Methoden zum Definieren von Kanälen und die zugehörigen Felder werden nicht mit diesen Spezifikationen zusammengefasst.

Bei den Punkten 7-9 wird die gesamte Kanaldefinition aus der CCDT übernommen. Für Felder, die nicht explizit bei der Definition des Kanals angegeben wurden, werden die Standardeinstellungen des Systems verwendet. Die Werte aus anderen Methoden zum Definieren von Kanälen und die zugehörigen Felder werden nicht mit diesen Spezifikationen zusammengefasst.

Kanalexits in IBM MQ .NET verwenden

Für Clientbindungen können Sie wie bei jeder anderen Clientverbindung Kanalexits verwenden. Für verwaltete Bindungen müssen Sie ein Exitprogramm entwickeln, das die entsprechende Schnittstelle implementiert.

Clientbindungen

Wenn Sie Clientbindungen verwenden, können Sie wie im Abschnitt Kanalexits beschrieben Kanalexits verwenden. Sie können keine Kanalexits verwenden, die für verwaltete Bindungen geschrieben wurden.

Verwaltete Bindungen

Wenn Sie eine verwaltete Verbindung zum Implementieren eines Exits verwenden, definieren Sie eine neue .NET-Klasse, die die entsprechende Schnittstelle implementiert. Im IBM MQ-Paket sind drei Exitschnittstellen definiert:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Anmerkung: Mit diesen Schnittstellen geschriebene Benutzerexits werden in einer nicht verwalteten Umgebung nicht als Kanalexits unterstützt.

Im folgenden Beispiel wird eine Klasse definiert, die alle drei Exitschnittstellen implementiert:

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
     // complete the body of the send exit here
  }


  // This method comes from the receive exit
  byte[] ReceiveExit(MQChannelExit       channelExitParms,
                     MQChannelDefinition channelDefinition,
                     byte[]              dataBuffer
                     ref int             dataOffset
                     ref int             dataLength
                     ref int             dataMaxLength)
  {
     // complete the body of the receive exit here
  }
```

```
  // This method comes from the security exit
  byte[] SecurityExit(MQChannelExit       channelExitParms,
                      MQChannelDefinition channelDefParms,
                      byte[]              dataBuffer
                      ref int             dataOffset
                      ref int             dataLength
                      ref int             dataMaxLength)
  {
     // complete the body of the security exit here
  }

}
```

An jeden Exit wird eine MQChannelExit- und eine MQChannelDefinition-Objektinstanz übergeben. Diese Objekte stellen die MQCXP- und MQCD-Strukturen dar, die in der prozedurgesteuerten Schnittstelle definiert werden.

Die Daten, die von einem Sendeexit gesendet werden sollen, und die Daten, die in einem Sicherheits- oder Empfangsexit empfangen werden, werden in den Parametern des Exits angegeben.

Beim Beginn der Ausführung des Exits handelt es sich bei den Daten am Offset *dataOffset* mit der Länge *dataLength* in der Bytefeldgruppe *dataBuffer* um die Daten, die zum Versand durch einen Sendeexit bereitstehen, und die Daten, die in einem Sicherheits- oder Empfangsexit empfangen wurden. Der Parameter *dataMaxLength* gibt die maximale Länge (aus *dataOffset*) an, die für den Exit im *dataBuffer* verfügbar ist. Hinweis: Bei einem Sicherheitsexit kann der Wert des Datenpuffers null betragen, wenn der Exit zum ersten Mal aufgerufen wird oder die Partnerseite keine Daten senden möchte.

Bei der Rückgabe sollte der Wert von *dataOffset* und *dataLength* so festgelegt werden, dass er auf den Offset und die Länge innerhalb der zurückgegebenen Bytefeldgruppe verweist, die dann von den .NET-Klassen verwendet werden sollte. Bei einem Sendeexit gibt dies die Daten an, die gesendet werden sollen, und für einen Sicherheits- oder Empfangsexit die Daten, die interpretiert werden sollen. Der Exit gibt in der Regel eine Bytefeldgruppe zurück. Ausnahmen können ein Sicherheitsexit sein, der keine Daten sendet, und alle Exits, die mit den INIT- oder TERM- Ursachencodes aufgerufen werden. Deshalb ist der einfachste Exit, der geschrieben werden kann, einer, der lediglich den Datenpuffer zurückgibt:

Der einfachste Exithauptteil, der möglich ist, lautet wie folgt:

```
{
  return dataBuffer;
}
```

Klasse 'MQChannelDefinition'

Die Benutzer-ID und das Kennwort, die mit der verwalteten .NET-Clientanwendung angegeben werden, werden in der IBM MQ .NET-Klasse 'MQChannelDefinition' festgelegt, die an den Clientsicherheitsexit übergeben wird. Der Sicherheitsexit kopiert die Benutzer-ID und das Kennwort in die Felder 'MQCD.RemoteUserIdentifier' und 'MQCD.RemotePassword' (weitere Informationen hierzu finden Sie im Abschnitt „Sicherheitsexit schreiben” auf Seite 1025).

Kanalexits angeben (verwalteter Client)

Wenn Sie beim Erstellen Ihres MQQueueManager-Objekts (im MQEnvironment- oder MQQueueManager-Konstruktor) einen Kanalnamen oder einen Verbindungsnamen angeben, können Sie Kanalexits auf zwei Arten angeben.

Nach Ausführungspriorität sind dies:

1. Übergabe der Hashtabelleneigenschaften MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY oder MQC.RECEIVE_EXIT_PROPERTY im MQQueueManager-Konstruktor.
2. Festlegen der MQEnvironment-Eigenschaften SecurityExit, SendExit oder ReceiveExit.

Wenn Sie keinen Kanalnamen oder Verbindungsnamen angeben, stammen die Kanalexits, die verwendet werden, aus der Kanaldefinition, die aus einer Definitionstabelle für den Clientkanal (Client Channel Definition Table, CCDT) übernommen wurde. Die in der Kanaldefinition gespeicherten Werte können nicht

überschrieben werden. Weitere Informationen zu Kanaldefinitionstabellen finden Sie in den Abschnitten Definitionstabelle für Clientkanal und „Definitionstabelle für den Clientkanal mit .NET verwenden” auf Seite 620.

Die Spezifikation weist immer die Form einer Zeichenfolge mit folgendem Format auf:

```
Assembly_name(Class_name)
```

Klassenname ist der vollständig qualifizierte Name (einschließlich Spezifikation für den Namensbereich) einer .NET-Klasse, die eine der Schnittstellen 'IBM.WMQ.MQSecurityExit', 'IBM.WMQ.MQSendExit' oder 'IBM.WMQ.MQReceiveExit' implementiert (wie erforderlich). *Assembly-Name* ist die vollständig qualifizierte Speicherposition (einschließlich Dateierweiterung) der Assembly, in der sich die Klasse befindet. Die Länge der Zeichenfolge ist auf 999 Zeichen begrenzt, wenn Sie die Eigenschaften von MQEnvironment oder MQQueueManager verwenden. Wenn der Kanalexitname allerdings in der CCDT angegeben ist, ist er auf 128 Zeichen begrenzt. Der .NET-Client-Code lädt und erstellt bei Bedarf eine Instanz der angegebenen Klasse, indem die Spezifikation der Zeichenfolge analysiert wird.

Benutzerdaten des Kanalexits angeben (verwalteter Client)

Den Kanalexits können Benutzerdaten zugeordnet sein. Wenn Sie beim Erstellen Ihres MQQueueManager-Objekts (im MQEnvironment- oder MQQueueManager-Konstruktor) einen Kanalnamen oder einen Verbindungsnamen angeben, können Sie die Benutzerdaten auf zwei Arten angeben.

Nach Ausführungspriorität sind dies:

1. Übergabe der Hashtabelleneigenschaften MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY oder MQC.RECEIVE_USERDATA_PROPERTY im MQQueueManager-Konstruktor.
2. Festlegen der MQEnvironment-Eigenschaften SecurityUserData, SendUserData oder ReceiveUserData.

Wenn Sie keinen Kanalnamen oder Verbindungsnamen angeben, stammen die Werte für die Benutzerdaten des Exits aus der Kanaldefinition, die aus der Definitionstabelle für den Clientkanal (CCDT) übernommen wurde. Die in der Kanaldefinition gespeicherten Werte können nicht überschrieben werden. Weitere Informationen zu Kanaldefinitionstabellen finden Sie in den Abschnitten Definitionstabelle für Clientkanal und „Definitionstabelle für den Clientkanal mit .NET verwenden” auf Seite 620.

Bei der Spezifikation handelt es sich in jedem Fall um eine Zeichenfolge, die auf 32 Zeichen beschränkt ist.

Automatische Wiederherstellung der Clientverbindung in .NET

Sie können Ihren Client so konfigurieren, dass er bei einer nicht erwarteten Verbindungsunterbrechung automatisch erneut eine Verbindung zu einem Warteschlangenmanager herstellt.

Die Verbindung eines Clients zu einem Warteschlangenmanager kann unerwartet unterbrochen werden, wenn beispielsweise der Warteschlangenmanager gestoppt wird oder ein Netz- oder Serverfehler auftritt.

Ohne automatische Wiederherstellung der Clientverbindung wird ein Fehler generiert, wenn die Verbindung fehlschlägt. Mithilfe des Fehlercodes können Sie die Verbindung wiederherstellen.

Ein Client, der die Funktion zur automatischen Wiederherstellung der Clientverbindung verwendet, wird als wiederverbindbarer Client bezeichnet. Um einen wiederverbindbaren Client zu erstellen, geben Sie bestimmte Optionen an, die als Verbindungswiederholungsoptionen bezeichnet werden, während Sie eine Verbindung zum Warteschlangenmanager herstellen.

Wenn es sich bei der Clientanwendung um einen IBM MQ .NET-Client handelt, kann die automatische Wiederherstellung der Clientverbindung festgelegt werden, indem beim Erstellen eines Warteschlangenmanagers mit der `MQQueueManager`-Klasse ein entsprechender Wert für CONNECT_OPTIONS_PROPERTY angegeben wird. Einzelheiten zu den Werten von CONNECT_OPTIONS_PROPERTY finden Sie im Abschnitt Optionen für die Verbindungswiederholung.

Sie können auswählen, ob die Clientanwendung immer eine Verbindung zu einem Warteschlangenmanager mit demselben Namen, zu demselben Warteschlangenmanager oder zu einer Gruppe von Warte-

schlangenmanagern, die in der Clientverbindungstabelle mit demselben QMNAME definiert sind, herstellt oder wiederherstellt (weitere Informationen finden Sie unter Warteschlangenmanagergruppen in CCDT).

Transport Layer Security-(TLS-)Unterstützung für .NET

IBM MQ classes for .NET-Clientanwendungen unterstützen Transport Layer Security-(TLS-)Verschlüsselung. Das TLS-Protokoll ermöglicht Kommunikationssicherheit im Internet sowie die vertrauliche und zuverlässige Kommunikation zwischen Client/Server-Anwendungen.

Zugehörige Konzepte

TLS-Unterstützung für verwalteten IBM MQ.NET-Client

Verschlüsselte Sicherheitsprotokolle: TLS

TLS-Unterstützung für den nicht verwalteten .NET-Client

Die TLS-Unterstützung für den nicht verwalteten .NET -Client basiert auf der MQI in der Programmiersprache C und IBM Global Security Kit (GSKit). Die MQI in C verarbeitet die TLS-Operationen und GSKit implementiert die sicheren TLS-Socketprotokolle.

TLS für den nicht verwalteten .NET-Client aktivieren

TLS wird nur für Clientverbindungen unterstützt. Zur Aktivierung von TLS müssen Sie die CipherSpec für die Kommunikation mit dem Warteschlangenmanager angeben. Diese muss mit der für den Zielkanal festgelegten CipherSpec übereinstimmen.

Geben Sie zur Aktivierung von TLS die CipherSpec mithilfe der statischen Mitgliedsvariablen 'SSLCipherSpec' von MQEnvironment an. Das folgende Beispiel bezieht sich auf den SVRCONN-Kanal SECURE.SVRCONN.CHANNEL, der TLS mit dem CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA anfordert:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Eine Liste der CipherSpecs finden Sie im Abschnitt CipherSpecs angeben.

Die Eigenschaft 'SSLCipherSpec' kann auch mit MQC.SSL_CIPHER_SPEC_PROPERTY in der Hashtabelle mit Verbindungseigenschaften festgelegt werden.

Für eine erfolgreiche Verbindungsherstellung über TLS muss für den Client-Keystore die Stammzertifikatskette der Zertifizierungsstelle konfiguriert sein, über die das vom Warteschlangenmanager vorgewiesene Zertifikat authentifiziert werden kann. Wenn die Eigenschaft 'SSLClientAuth' im SVRCONN-Kanal auf MQSSL_CLIENT_AUTH_REQUIRED gesetzt ist, muss der Client-Keystore entsprechend ein ermittelndes persönliches Zertifikat enthalten, das vom Warteschlangenmanager anerkannt wird.

Definierten Namen des Warteschlangenmanagers verwenden

Der Warteschlangenmanager identifiziert sich selbst mithilfe eines TLS-Zertifikat, das einen *Definierten Namen* (DN) enthält.

Eine Clientanwendung der IBM MQ .NET kann anhand dieses DN sicherstellen, dass sie mit dem richtigen Warteschlangenmanager kommuniziert. Ein definiertes Namensmuster wird mit der Variablen 'sslPeerName' von MQEnvironment angegeben. Sie können beispielsweise Folgendes festlegen:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

Die Verbindung kann nur erfolgreich hergestellt werden, wenn der Warteschlangenmanager ein Zertifikat mit einem allgemeinen Namen vorlegt, der mit 'QMGR.' beginnt. und mindestens zwei Organisationseinheitennamen, von denen der erste IBM und der zweite WEBSPHERE sein muss.

Die Eigenschaft 'SSLPeerName' kann auch mit MQC.SSL_PEER_NAME_PROPERTY in der Hashtabelle mit Verbindungseigenschaften festgelegt werden. Weitere Informationen zu definierten Namen und Regeln zum Festlegen von Peernamen finden Sie im Abschnitt IBM MQ schützen.

Wenn die Eigenschaft 'SSLPeerName' festgelegt ist, können Verbindungen nur dann erfolgreich hergestellt werden, wenn die Eigenschaft für ein gültiges Muster festgelegt ist und der Warteschlangenmanager ein übereinstimmendes Zertifikat übergibt.

Fehlerbehandlung bei Verwendung von TLS

Bei der Herstellung einer Verbindung zu einem Warteschlangenmanager über TLS kann IBM MQ classes for .NET folgende Ursachencodes ausgeben:

MQRC_SSL_NOT_ALLOWED
Die Eigenschaft 'SSLCipherSpec' wurde festgelegt, aber die Verbindung über Bindungen wurde verwendet. TLS wird jedoch nur von Clientverbindungen unterstützt.

MQRC_SSL_PEER_NAME_MISMATCH
Das in der Eigenschaft 'SSLPeerName' angegebene definierte Namensmuster stimmte nicht mit dem vom Warteschlangenmanager bereitgestellten definierten Namen überein.

MQRC_SSL_PEER_NAME_ERROR
Das in der Eigenschaft 'SSLPeerName' angegebene definierte Namensmuster war nicht gültig.

MQRC_KEY_REPOSITORY_ERROR
Die Position des Schlüsselrepositorys ist entweder nicht angegeben, ungültig oder es kann nicht darauf zugegriffen werden.

TLS-Unterstützung für den verwalteten .NET-Client

Der verwaltete .NET -Client verwendet die Microsoft .NET Framework -Bibliotheken, um sichere TLS-Socketprotokolle zu implementieren. Die Microsoft-Klasse System.Net.SecuritySslStream fungiert als Datenstrom über verbundene TCP-Sockets und sendet und empfängt Daten über diese Socketverbindung.

Die mindestens erforderliche .NET Framework -Version ist .NET Framework v3.5. Die Stufe der Verschlüsselungsalgorithmusunterstützung basiert auf der .NET Framework -Version, die von der Anwendung verwendet wird:

- Für Anwendungen, die auf .NET Framework Version 3.5 und 4.0basieren, sind die Secure Socket-Protokolle SSL 3.0 und TSL 1.0verfügbar.
- Für Anwendungen, die auf .NET Framework Version 4.5basieren, sind die SSL-Protokolle 3.0, TLS 1.1 und TLS 1.2verfügbar.

Möglicherweise müssen Sie Anwendungen, die eine höhere TLS-Protokollunterstützung erwarten, auf eine neuere Version des Frameworks umstellen, wie für die Microsoft -Sicherheitsunterstützung in .NET Frameworkdefiniert.

Die Hauptfunktionen der TLS-Unterstützung für den verwalteten .NET-Client lauten wie folgt:

Unterstützung für das TLS-Protokoll
Die TLS-Unterstützung für den verwalteten .NET-Client wird über die .NET-SSLStream-Klasse definiert und ist abhängig vom .NET-Framework der Anwendung. Weitere Informationen finden Sie im Abschnitt „TLS-Protokollunterstützung für den verwalteten .NET-Client” auf Seite 628.

CipherSpec-Unterstützung
Für den verwalteten .NET-Client gelten dieselben TLS-Einstellungen wie für Microsoft.NET-TLS-Datenströme. Weitere Informationen finden Sie in den Abschnitten „CipherSpec-Unterstützung für den verwalteten .NET-Client” auf Seite 628 und „CipherSpec-Zuordnungen für den verwalteten .NET-Client” auf Seite 630.

Schlüsselrepositorys
Das Schlüsselrepository auf Clientseite ist ein Windows-Keystore. Das serverseitige Repository ist ein Repository vom Typ CMS (Cryptographic Message Syntax, Syntax kryptografischer Nachrichten). Weitere Informationen finden Sie im Abschnitt „Schlüsselrepositorys für den verwalteten .NET-Client” auf Seite 632.

Zertifikate
Zur Implementierung der gegenseitigen Authentifizierung von Client und Warteschlangenmanager können Sie selbst signierte TLS-Zertifikate verwenden. Weitere Informationen finden Sie unter „Zertifikate für den verwalteten .NET-Client verwenden” auf Seite 632.

SSLPEERNAME
In .NET können Anwendungen über das optionale Attribut SSLPEERNAME ein DN-Muster (DN = Distinguished Name, definierter Name) angeben. Weitere Informationen finden Sie im Abschnitt „SSLPEERNAME” auf Seite 633.

FIPS-Konformität
Die programmgesteuerte FIPS-Aktivierung wird von der Microsoft.NET-Sicherheitsbibliothek nicht unterstützt. Die FIPS-Aktivierung wird über die Windows-Gruppenrichtlinieneinstellung gesteuert.

NSA Suite B-Konformität
IBM MQ implementiert RFC 6460. Die Microsoft.NET-Implementierung für NSA Suite B lautet 5430. Sie wird ab .NET-Framework 3.5 unterstützt.

Zurücksetzen oder Neuvereinbarung des geheimen Schlüssels
Obwohl die SSLStream-Klasse das Zurücksetzen geheimer Schlüssel oder die Neuvereinbarung von geheimen Schlüsseln nicht unterstützt, ermöglicht der verwaltete .NET -Client für die Konsistenz mit anderen IBM MQ -Clients, dass Anwendungen `SSLKeyResetCount`festlegen können. Weitere Informationen finden Sie unter „Zurücksetzen oder Neuvereinbarung des geheimen Schlüssels für den verwalteten .NET -Client” auf Seite 633.

Widerrufsprüfung
Die SSLStream-Klasse unterstützt die Überprüfung des Zertifikatswiderrufs, die automatisch von der Zertifikatsverkettungsengine durchgeführt wird. Weitere Informationen finden Sie unter „Widerrufsprüfung” auf Seite 634.

Unterstützung für IBM MQ-Sicherheitsexits
Die Klasse `SSLStream` bietet eingeschränkte Unterstützung für IBM MQ -Sicherheitsexits. Das Abfragen lokaler und ferner Zertifikate zum Abrufen von `SSLPeerNamePtr`(Subjekt-DN) und `SSLRemCertIssNamePtr` (Aussteller-DN) ist möglich, da dies in Microsoft.NETunterstützt wird. Der Abruf von Attributen wie DNQ, UNSTRUCTUREDNAME und UNSTRUCTUREDADDRESS wird allerdings nicht unterstützt, diese Werte können somit nicht über die Exits abgerufen werden.

Unterstützung für Verschlüsselungshardware
Verschlüsselungshardware wird für den verwalteten .NET-Client nicht unterstützt.

Unterstützung für TLS1.3 auf verwalteten IBM MQ .NET -und XMS .NET -Clients

V 9.4.0

Ab IBM MQ 9.4.0unterstützen IBM MQ .NET -und XMS .NET -Clients TLS1.3 , vorausgesetzt, das Betriebssystem unterstützt TLS1.3.

Der verwaltete .NET -Client verwendet die Microsoft .NET Framework -Bibliotheken, um sichere TLS-Socketprotokolle zu implementieren. Die Microsoft `System.Net.SecuritySslStream` -Klasse arbeitet als Stream über verbundene TCP-Sockets und sendet und empfängt Daten über diese Socketverbindung.

Unter Windowsverwendet .NET SCHANNEL und unter Linux .NET OpenSSL für die SSL-Kommunikation.

Windows Für IBM MQ .NET -Clientanwendungen, die unter Windowsausgeführt werden

Microsoft hatte angekündigt, dass Windows 11 und Windows Server 2022 standardmäßig TSL1.3 -Verschlüsselungen unterstützen.

TLS_AES_128_GCM_SHA256 und TLS_AES_256_GCM_SHA384 Cipher-Suites sind in beiden Versionen von Windowsstandardmäßig aktiviert.

Achtung:

- TLS_CHACHA20_POLY1305_SHA256 Cipher Suite ist standardmäßig nicht aktiviert, wird aber unterstützt.
- Damit ein IBM MQ .NET -Client mit aktiviertem TLS1.3 erfolgreich eine Verbindung zu einem Warteschlangenmanager herstellen kann, ist IBM Global Security Kit (GSKit) 8.0.55.29 die auf der Warteschlangenmanagerseite erforderliche Mindestversion.

Linux Für IBM MQ .NET -Clientanwendungen, die unter Linuxausgeführt werden

Da .NET OpenSSL unter Linux für die SSL-Übertragung verwendet, ist zur Verwendung von TLS1.3 OpenSSL v1.1.1 die Mindestanforderung.

Da .NET OpenSSL unter Linuxverwendet, sollten außerdem alle von OpenSSL unterstützten Verschlüsselungen auch für .NET funktionieren.

OpenSSL unterstützt die folgenden CipherSpecs für TLS1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

Zugehörige Konzepte

„CipherSpec-Zuordnungen für den verwalteten .NET-Client” auf Seite 630
Die IBM MQ.NET-Schnittstelle unterhält eine Zuordnungstabelle für IBM MQ und Microsoft.NET, anhand derer die Version des TLS-Protokolls bestimmt wird, die der verwaltete Client für eine sichere Verbindung zu einem Warteschlangenmanager verwenden muss.

TLS-Protokollunterstützung für den verwalteten .NET-Client
Die TLS-Unterstützung von IBM MQ.NET basiert auf der .NET-Klasse SSLStream.

Anmerkung: Die TLS-Protokollunterstützung für den verwalteten .NET-Client ist von der von der Anwendung verwendeten Version des .NET-Frameworks abhängig. Weitere Informationen finden Sie in „TLS-Unterstützung für den verwalteten .NET-Client” auf Seite 626.

Damit die Microsoft.NET-Klasse SSLStream TLS initialisieren und einen Handshake mit dem Warteschlangenmanager durchführen kann, ist **SSLProtocol** einer der erforderlichen Parameter, den Sie festlegen müssen. Er dient zur Angabe der TLS-Versionsnummer, wobei es sich um einen der folgenden Werte handeln muss:

- SSL3.0
- TLS1.0
- TLS1.2

Der Wert dieses Parameters ist eng an die Protokollfamilie gekoppelt, zu der die bevorzugte CipherSpec gehört. Wenn SSLStream einen TLS-Handshake mit dem Server (Warteschlangenmanager) startet, wird dabei über die in **SSLProtocol** angegebene TLS-Version die Liste der für die Vereinbarung zu verwendenden CipherSpecs ermittelt.

In IBM MQ.NET sind keine Eigenschaften verfügbar, die von den Anwendungen zur Festlegung dieses Wertes verwendet werden können. Stattdessen verwendet IBM MQ eine Zuordnungstabelle zur internen Zuordnung der definierten CipherSpec zu der Protokollfamilie und ermittelt die zu verwendende SSL-Protokollversion. In dieser Tabelle sind für alle unterstützten CipherSpecs die Zuordnungen zwischen Microsoft.NET und IBM MQ sowie die Protokollversion, zu der diese gehören, aufgeführt. Weitere Informationen finden Sie im Abschnitt „CipherSpec-Zuordnungen für den verwalteten .NET-Client” auf Seite 630.

CipherSpec-Unterstützung für den verwalteten .NET-Client
Die CipherSpec-Einstellungen für eine Anwendung werden während des Handshakes mit dem Server verwendet.

IBM MQ-Clients bieten die Möglichkeit, einen CipherSpec-Wert festzulegen, der beim Handshake mit dem Warteschlangenmanager verwendet wird. IBM MQ-Clients müssen für den Aufbau einer sicheren Verbindung eine gültige CipherSpec festlegen, vorzugsweise die CipherSpec, die in der Windows-Gruppenrichtlinie angegeben ist. Wird in diesem Feld kein Wert angegeben, handelt es sich um einen einfachen Textkanal ohne Sicherheit an den Sockets.

Beim verwalteten IBM MQ.NET-Client beziehen sich die TLS-Einstellungen auf die Microsoft.NET-SSLStream-Klasse. Für SSLStream kann eine CipherSpec oder eine Vorgabenliste von CipherSpecs nur in der Windows-Gruppenrichtlinie, einer computerweiten Einstellung, definiert werden. SSLStream verwendet dann die angegebene CipherSpec oder Vorgabenliste beim Handshake mit dem Server. Falls es weitere IBM MQ-Clients gibt, kann die CipherSpec-Eigenschaft in der Anwendung in der IBM MQ-Kanaldefinition festgelegt werden. Dieselbe Einstellung wird dann auch für die TLS-Verhandlung verwendet. Infolge dieser Einschränkung könnten beim TLS-Handshake alle unterstützten CipherSpecs vereinbart werden, unabhängig davon, was in der IBM MQ-Kanalkonfiguration angegeben ist. Daher wird es vermutlich zum Fehler AMQ9631 auf dem Warteschlangenmanager kommen. Legen Sie zur Vermeidung dieses Fehlers die CipherSpec, die Sie in der Anwendung festgelegt haben, als TLS-Konfiguration in der Windows-Gruppenrichtlinie fest.

Der neue TLS-Client-Code von IBM MQ.NET prüft nur, ob die richtige Protokollversion vereinbart wurde. Die TLS-Protokollversion leitet sich aus der von der Anwendung festgelegten CipherSpec ab und wird für den TLS-Handshake mit dem Server (Warteschlangenmanager) verwendet. Daher ist es auslegungsmäßig erforderlich, die CipherSpec in der Anwendung des verwalteten IBM MQ.NET-Clients festzulegen. Wenn es sich bei der vom IBM MQ-Client festgelegten CipherSpec nicht um die CipherSpec aus den Protokollen SSL 3.0, TLS 1.0 und TLS 1.2 handelt, würde der von IBM MQ verwaltete .NET-Client standardmäßig mit den Verschlüsselungen aus dem Protokoll SSL 3.0 oder TLS 1.0 verhandeln und keinen Fehler melden.

Anmerkung: Wenn die von der Anwendung angegebene CipherSpec IBM MQ nicht bekannt ist, wird sie vom IBM MQ-verwalteten .NET-Client nicht berücksichtigt und der Client vereinbart die Verbindung basierend auf der Gruppenrichtlinie des Windows-Systems.

CipherSpec einstellen

Es gibt drei Möglichkeiten, eine CipherSpec einzustellen:

.NET-MQEnvironment-Klasse
Im folgenden Beispiel sehen Sie, wie eine CipherSpec mit der MQEnvironment-Klasse festgelegt wird.

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

TLS-Eigenschaft 'CipherSpec'
Im folgenden Beispiel sehen Sie, wie zur Festlegung einer CipherSpec dem MQQueueManager-Konstruktor ein Hashtabellenparameter hinzugefügt wird.

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Windows-Gruppenrichtlinie
Wenn eine Cipher-Suite-Liste über die Gruppenrichtlinien-Verwaltungskonsole von Windows konfiguriert ist, muss die SVRCONN-Kanaldefinition eine übereinstimmende CipherSpec angeben. Eine übereinstimmende CipherSpec kann entweder ein generischer Wert sein, z. B. "ANY_TLS12_OR_HIGHER", oder ein bestimmter Wert, der der höchsten Cipher-Suite zugeordnet ist, die aus der sortierten Liste vereinbart würde. Für .NET-Clients wird die Verwendung generischer CipherSpec-Werte empfohlen, da dann die SVRCONN-CipherSpec-Konfiguration nicht geändert werden muss, wenn sich die Reihenfolge in der Clientliste ändert.

CCDT-Nutzung

IBM MQ.NET unterstützt nur Definitionstabellen für Clientkanäle (TAB-Dateien), die sich auf einem lokalen Computer befinden. Vorhandene CCDT-Dateien, in denen ein CipherSpec-Wert festgelegt ist, können für IBM MQ.NET-Verbindungen verwendet werden. Allerdings bestimmt der im Clientverbindungskanal definierte CipherSpec-Wert die TLS-Protokollversion und muss auch der in der Windows-Gruppenrichtlinie festgelegten CipherSpec entsprechen.

Zugehörige Konzepte

„IBM MQ-Umgebung einrichten" auf Seite 615
Damit Sie über die Clientverbindung eine Verbindung zu einem Warteschlangenmanager herstellen können, müssen Sie zuerst die IBM MQ-Umgebung einrichten.

„TLS-Unterstützung für den verwalteten .NET-Client" auf Seite 626
Der verwaltete .NET -Client verwendet die Microsoft .NET Framework -Bibliotheken, um sichere TLS-Socketprotokolle zu implementieren. Die Microsoft-Klasse System.Net.SecuritySslStream fungiert als Datenstrom über verbundene TCP-Sockets und sendet und empfängt Daten über diese Socketverbindung.

Zugehörige Tasks

CipherSpecs angeben

Zugehörige Verweise

.NET-MQEnvironment-Klasse

CipherSpec-Zuordnungen für den verwalteten .NET-Client

Die IBM MQ.NET-Schnittstelle unterhält eine Zuordnungstabelle für IBM MQ und Microsoft.NET, anhand derer die Version des TLS-Protokolls bestimmt wird, die der verwaltete Client für eine sichere Verbindung zu einem Warteschlangenmanager verwenden muss.

Wenn im Kanal SVRCONN eine CipherSpec angegeben ist, versucht der Warteschlangenmanager nach Abschluss des TLS-Handshakes, die CipherSpec mit der von der Clientanwendung verwendeten und vereinbarten CipherSpec abzugleichen. Wenn der Warteschlangenmanager keine übereinstimmende CipherSpec finden kann, schlägt die Kommunikation mit dem Fehler AMQ9631 fehl.

Die IBM MQ.NET-Schnittstelle unterhält eine CipherSpec-Zuordnungstabelle für die Zuordnung zwischen IBM MQ und Microsoft.NET. Anhand dieser Tabelle wird die TLS-Protokollversion ermittelt, die der Client für eine sichere Socketverbindung mit dem Warteschlangenmanager verwenden soll. Entsprechend dem SSLCipherSpec-Wert kann die SSLProtocol-Version TLS 1.0 oder TLS 1.2 lauten, je nachdem, welche Version des Microsoft.NET-Frameworks Sie verwenden.

Stellen Sie sicher, dass Sie den richtigen SSLCipherSpec-Wert angeben, da die Angabe eines falschen Werts dazu führen kann, dass die Protokolle SSL 3.0 oder TLS 1.0 verwendet werden.

Tabelle 78. Zuordnungstabelle für IBM MQ und Microsoft.NET

IBM MQ CipherSpec	Microsoft.NET CipherSpec	TLS-Version
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2

Tabelle 78. Zuordnungstabelle für IBM MQ und Microsoft.NET (Forts.)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**TLS-Version**
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
V 9.4.0 TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3
V 9.4.0 TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3

Anmerkungen:

1. Deprecated Die CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA wird nicht weiter unterstützt. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.

Zugehörige Konzepte

„TLS-Unterstützung für den verwalteten .NET-Client" auf Seite 626
Der verwaltete .NET -Client verwendet die Microsoft .NET Framework -Bibliotheken, um sichere TLS-Socketprotokolle zu implementieren. Die Microsoft-Klasse System.Net.SecuritySslStream fungiert als Datenstrom über verbundene TCP-Sockets und sendet und empfängt Daten über diese Socketverbindung.

Schlüsselrepositorys für den verwalteten .NET-Client
Verwaltete .NET-Clients verwenden als Schlüsselrepository den Windows-Schlüsselspeicher. Im Benutzer- oder Systemschlüsselspeicher müssen Zertifikate und private Schlüssel verfügbar sein, damit sie von der Clientanwendung während eines TLS-Handshakes sowohl für die Identität als auch für die Vertrauensbeziehung genutzt werden können.

Clientseite

In der Anwendung können Sie einen der folgenden Werte für das Schlüsselrepository festlegen:

- "*USER": IBM MQ .NET greift zum Abrufen der Clientzertifikate auf den Zertifikatsspeicher des aktuellen Benutzers zu.
- "*SYSTEM": IBM MQ .NET greift zum Abrufen der Zertifikate auf das lokale Computerkonto zu.

Die Zertifikate des Clients müssen im eigenen Zertifikatsspeicher (My Certificate) des Benutzer- oder Computerkontos gespeichert werden. Alle Serverzertifikate (CA-Zertifikate) müssen im Stammverzeichnis des Zertifikatsspeichers gespeichert werden.

Anmerkung: In einer einzigen Datei können mehrere Zertifikate in den folgenden Formaten gespeichert werden:

- Personal Information Exchange - PKCS #12 (.PFX, .P12)
- Cryptographic Message Syntax Standard - PKCS #7-Zertifikate (.P7B)
- Microsoft Serialized Certificate Store (.SST)

Zertifikate für den verwalteten .NET-Client verwenden
Für Clientzertifikate greift der durch IBM MQ verwaltete .NET-Client auf den Windows-Keystore zu und lädt alle Clientzertifikate, die anhand der Zertifikatsbezeichnung oder der Zeichenfolge abgeglichen wurden.

Bei der Auswahl des Zertifikats für den SSLStream-TLS-Handshake verwendet der durch IBM MQ verwaltete .NET-Client immer das erste übereinstimmende Zertifikat.

Zertifikate anhand der Zertifikatsbezeichnung abgleichen

Wenn Sie die Zertifikatsbezeichnung festlegen, durchsucht der verwaltete IBM MQ .NET-Client den Windows-Zertifikatsspeicher mit der angegebenen Bezeichnung, um das Clientzertifikat zu finden. Alle übereinstimmenden Zertifikate werden geladen und das erste Zertifikat in der Liste wird verwendet. Es gibt zwei Möglichkeiten, die Zertifikatsbezeichnung festzulegen.

- Die Zertifikatsbezeichnung kann in der MQEnvironment-Klasse über MQEnvironment.CertificateLabel festgelegt werden.
- Die Zertifikatsbezeichnung kann auch in den Eigenschaften einer Hashtabelle definiert werden, angegeben als Eingabeparameter mit dem MQQueueManager- Konstruktor, wie im folgenden Beispiel dargestellt.

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

Beim Namen (CertificateLabel) und Wert muss die Groß-/Kleinschreibung beachtet werden.

Zertifikate nach Zeichenfolge abgleichen

Wenn keine Zertifikatsbezeichnung festgelegt ist, wird das Zertifikat gesucht und verwendet, das der Zeichenfolge "ibmwebspheremq" und dem derzeit angemeldeten Benutzer (in Kleinbuchstaben) entspricht.

Zugehörige Tasks

Client sicher mit einem WS-Manager verbinden

Zugehörige Verweise

.NET-MQEnvironment-Klasse

SSLPEERNAME
Das Attribut SSLPEERNAME wird zur Prüfung des definierten Namens (DN) des Zertifikats vom Peer-Warteschlangenmanager aus verwendet.

In IBM MQ.NET können Anwendungen über SSLPEERNAME ein Muster für den definierten Namen angeben, wie im folgenden Beispiel dargestellt.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Wie bei anderen IBM MQ-Clients ist SSLPEERNAME ein optionaler Parameter.

Wenn der Wert SSLPEERNAME nicht definiert ist, nimmt der verwaltete IBM MQ.NET-Client keinerlei Validierung des fernen (Server-)Zertifikats vor und akzeptiert das ferne (Server-)Zertifikat unverändert.

Wie Sie SSLPEERNAME festlegen, hängt davon ab, welches der IBM MQ-Stack-Angebote Sie verwenden.

IBM MQ classes for .NET
Es gibt folgende drei Optionen:

1. Definition von MQEnvironment.SSLPeerName in der MQEnvironment-Klasse.
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. Verwenden Sie den Warteschlangenmanager-Konstruktor `MQQueueManager (String queueManagerName, Hashtable properties)`. Geben Sie SSLPEERNAME in `Hashtable properties` als Option 2 an.

XMS .NET
Definieren Sie den SSL-Peernamen in der Verbindungsfactory:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF
Beziehen Sie SslPeerName als durch Semikolon getrenntes Feld in den URI ein.

Zugehörige Verweise

.NET-MQEnvironment-Klasse

Zurücksetzen oder Neuvereinbarung des geheimen Schlüssels für den verwalteten .NET -Client
Das Zurücksetzen/die Neuvereinbarung des geheimen Schlüssels wird von der SSLStream-Klasse nicht unterstützt. Um jedoch mit anderen IBM MQ -Clients konsistent zu sein, ermöglicht der IBM MQ verwaltete .NET Client Anwendungen, **SSLKeyResetCount**festzulegen.

Bei Erreichen des Grenzwerts trennt IBM MQ.NET die Verbindung zum Warteschlangenmanager. Die Anwendungen werden darüber in Form einer Ausnahme mit dem Ursachencode MQRC_CONNECTION_BROKEN informiert. Die Anwendungen haben die Möglichkeit, die Ausnahme zu bearbeiten und die Verbin-

dungen wiederherzustellen oder die Option MQCNO_RECONNECT zu aktivieren, damit IBM MQ.NET automatisch wieder eine Verbindung zum Warteschlangenmanager herstellt.

Wenn die Funktion für automatische Clientverbindungswiederholung aktiviert ist, werden bei Erreichen des Zählerstandes, bei dem eine Schlüsselzurücksetzung erfolgt, alle vorhandenen Verbindungen abgebrochen und der IBM MQ.NET-Client erstellt alle Verbindungen neu. Weitere Informationen zur automatischen Clientverbindungswiederholung finden Sie im Abschnitt Automatische Clientverbindungswiederholung.

Zugehörige Konzepte

Zurücksetzen von geheimen SSL-und TLS-Schlüsseln

Widerrufsprüfung

Die SSLStream-Klasse unterstützt die Überprüfung des Zertifikatswiderrufs.

Die Überprüfung des Widerrufs wird automatisch von der Zertifikatsverkettungsengine durchgeführt. Dies gilt sowohl für das Online Certificate Status Protocol (OCSP) als auch für die Zertifikatswiderrufslisten (CRLs). Die SSLStream-Klasse verwendet den Zertifikatswiderruf, bei dem nur der im Zertifikat angegebene Server verwendet wird, der Server wird also durch das Zertifikat selbst vorgegeben. HTTP-CDP-Erweiterungen und OCSP-HTTP-Anforderungen können über den HTTP-Proxy-Server weitergeleitet werden.

Wie Sie die Überprüfung des Widerrufs festlegen, hängt davon ab, welches der IBM MQ-Stack-Angebote Sie verwenden.

IBM MQ.NET
Die Widerrufsüberprüfung kann in der Klassendatei `MQEnvironment.cs` durch die Eigenschaft **`MQEnvironment.SSLCertRevocationCheck`** festgelegt werden.

XMS .NET
Die Widerrufsübersprüfung kann im Eigenschaftskontext der Verbindungsfactory definiert werden, wie im folgenden Beispiel dargestellt.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF
Die Widerrufsübersprüfung kann in der URI durch folgende Namenskonvention definiert werden.

```
"SslCertRevocationCheck=true"
```

TLS für das durch IBM MQ verwaltete .NET konfigurieren

Die TLS-Konfiguration für das durch IBM MQ verwaltete .NET beinhaltet die Erstellung der Unterzeichnerzertifikate sowie die Konfiguration der Serverseite, der Clientseite und des Anwendungsprogramms.

Informationen zu diesem Vorgang

Zur Konfiguration von TLS müssen Sie zunächst die entsprechenden Unterzeichnerzertifikate erstellen. Bei den Unterzeichnerzertifikaten kann es sich entweder um selbst signierte oder von einer Zertifizierungsstelle bereitgestellte Zertifikate handeln. Selbst signierte Zertifikate können zwar in einer Entwicklungs-, Test- oder Vorproduktionsumgebung verwendet werden, in einem Produktionssystem sollten sie jedoch nicht verwendet werden. Verwenden Sie in einem Produktionssystem Zertifikate, die Sie von einer vertrauenswürdigen externen Zertifizierungsstelle (CA) erhalten haben.

Vorgehensweise

1. Erstellen Sie die Unterzeichnerzertifikate.
   a) Um selbst signierte Zertifikate zu erstellen, verwenden Sie den Befehl **`runmqakm`** oder V 9.4.0 V 9.4.0 **`runmqktool`** .

Weitere Informationen finden Sie im Abschnitt Selbst signiertes persönliches Zertifikat unter AIX, Linux, and Windows erstellen.

b) Um bei einer Zertifizierungsstelle Zertifikate für den Warteschlangenmanager und die Clients anzufordern, beachten Sie die Anweisungen im Abschnitt Obtaining personal certificates from a certificate authority.

2. Konfigurieren Sie die Serverseite.

a) Konfigurieren Sie TLS auf dem Warteschlangenmanager unter Verwendung von IBM Global Security Kit (GSKit), wie im Abschnitt Sichere Verbindung zwischen Client und Warteschlangenmanager herstellenbeschrieben.

b) Legen Sie die TLS-Attribute des SVRCONN-Kanals fest:

- Setzen Sie **SSLCAUTH** auf REQUIRED oder OPTIONAL.
- Legen Sie für **SSLCIPH** eine entsprechende CipherSpec fest.

Weitere Informationen finden Sie unter „TLS für den nicht verwalteten .NET-Client aktivieren” auf Seite 625.

3. Konfigurieren Sie die Clientseite.

a) Importieren Sie die Clientzertifikate in den Windows-Zertifikatsspeicher (unter dem Benutzer-/Computerkonto).

IBM MQ .NET greift vom Windows-Zertifikatsspeicher auf Clientzertifikate zu, daher müssen Sie Ihre Zertifikate in den Windows-Zertifikatsspeicher importieren, um eine sichere Socketverbindung zu IBM MQ herzustellen. Weitere Informationen zum Zugriff auf den Windows-Keystore sowie zum Importieren der clientseitigen Zertifikate finden Sie im Abschnitt Zertifikate und private Schlüssel importieren und exportieren.

b) Geben Sie die Zertifikatsbezeichnung wie im Abschnitt Sichere Verbindung zwischen Client und Warteschlangenmanager herstellen an.

c) Bearbeiten Sie gegebenenfalls die Windows-Gruppenrichtlinie, um die CipherSpec festzulegen. Starten Sie anschließend den Computer erneut, damit die Aktualisierungen der Windows-Gruppenrichtlinie in Kraft treten.

4. Konfigurieren Sie das Anwendungsprogramm.

a) Legen Sie MQEnvironment oder den SSLCipherSpec-Wert fest, um die Verbindung als eine sichere Verbindung anzugeben.

Der von Ihnen angegebene Wert wird zur Ermittlung des verwendeten Protokolls (TLS) verwendet. Die festgelegte CipherSpec sollte eine der CipherSpecs der unterstützten SSLProtocol-Version sein und kann vorzugsweise mit der CipherSpec identisch sein, die in der Windows-Gruppenrichtlinie angegeben ist. (Welche SSLProtocol-Version unterstützt wird, hängt vom verwendeten .NET-Framework ab. Die SSLProtocol-Version kann TLS 1.0 oder TLS 1.2 sein, je nachdem welche Version des Microsoft .NET-Frameworks Sie verwenden.) TLS 1.0 oder TLS 1.2 sein.

Anmerkung: Wenn die von der Anwendung angegebene CipherSpec IBM MQ nicht bekannt ist, wird sie vom IBM MQ-verwalteten .NET-Client nicht berücksichtigt und der Client vereinbart die Verbindung basierend auf der Gruppenrichtlinie des Windows-Systems.

b) Setzen Sie die Eigenschaft SSLKeyRepository auf "*SYSTEM" oder "*USER".

c) Optional: Legen Sie für SSLPEERNAME den definierten Namen (DN) des Serverzertifikats fest.

d) Geben Sie die Zertifikatsbezeichnung wie im Abschnitt Sichere Verbindung zwischen Client und Warteschlangenmanager herstellen an.

e) Definieren Sie weitere optionale Parameter, die Sie benötigen, wie z. B. KeyResetCount, CertificationRevocationCheck, und aktivieren Sie FIPS.

Beispiele für die Einstellung des TLS-Protokolls und des TLS-Schlüsselrepositorys

Für Basis-.NET können Sie das TLS-Protokoll und das TLS-Schlüsselrepository wie im folgenden Beispiel über die MQEnvironment-Klasse festlegen:

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Alternativ können Sie das TLS-Protokoll und das TLS-Schlüsselrepository wie im folgenden Beispiel durch eine Hashtabelle im MQQueueManager-Konstruktor festlegen.

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Nächste Schritte

Weitere Informationen zu den ersten Schritten bei der Entwicklung von IBM MQ .NET verwalteten TLS-Anwendungen finden Sie unter„Einfache Anwendung schreiben” auf Seite 636.

Zugehörige Verweise

.NET-MQEnvironment-Klasse

KeyResetCount (MQLONG)

Federal Information Processing Standards (FIPS) für AIX, Linux, and Windows

Einfache Anwendung schreiben

Hier finden Sie Tipps zum Schreiben einer einfachen von IBM MQ verwalteten .NET-Anwendung, die das TLS-Protokoll verwendet, einschließlich Beispielen zur Festlegung der SSL-Eigenschaften für Verbindungsfactorys, zum Erstellen einer Warteschlangenmanagerinstanz, einer Verbindung, einer Sitzung und eines Ziels sowie zum Senden einer Testnachricht.

Vorbereitende Schritte

Zunächst müssen Sie TLS für verwaltetes IBM MQ.NET konfigurieren, wie im Abschnitt „TLS für das durch IBM MQ verwaltete .NET konfigurieren” auf Seite 634 beschrieben.

Legen Sie zur Anwendungsprogrammkonfiguration in Basis-.NET SSL-Eigenschaften über die MQEnvironment-Klasse oder durch Angabe einer Hashtabelle als Teil des MQQueueManager-Konstruktors fest.

Legen Sie für die Anwendungsprogrammkonfiguration in XMS .NET die SSL-Eigenschaften im Eigenschaftskontext der Verbindungsfactorys fest.

Vorgehensweise

1. Legen Sie die SSL-Eigenschaften für die Verbindungsfactorys fest, wie in den folgenden Beispielen dargestellt.

   Beispiel für IBM MQ.NET

   ```
   properties = new Hashtable();
   properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
   properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
   properties.Add(MQC.PORT_PROPERTY, port);
   properties.Add(MQC.CHANNEL_PROPERTY, channelName);
   properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
   properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
   properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
   properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
   properties.Add("CertificateLabel", "ibmwebspheremq");
   MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
   ```

Beispiel für XMS .NET

```
cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

2. Erstellen Sie die Warteschlangenmanagerinstanz, die Verbindungen, die Sitzung und das Ziel, wie in den folgenden Beispielen dargestellt.

   Beispiel für MQ .NET

```
queueManager = new MQQueueManager(queueManagerName, properties);
Console.WriteLine("done");

// accessing queue
Console.Write("Accessing queue " + queueName + ".. ");
queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT + MQC.MQOO_FAIL_IF_QUIE
SCING);
Console.WriteLine("done");
```

   Beispiel für XMS .NET

```
connectionWMQ = cf.CreateConnection();
// Create session
sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

// Create destination
destination = sessionWMQ.CreateQueue(destinationName);

// Create producer
producer = sessionWMQ.CreateProducer(destination);
```

3. Senden sie eine Nachricht, wie in den folgenden Beispielen dargestellt.

   Beispiel für MQ .NET

```
// creating a message object
message = new MQMessage();
message.WriteString(messageString);

// putting messages continuously
for (int i = 1; i <= numberOfMsgs; i++)
{
Console.Write("Message " + i + " <" + messageString + ">.. ");
queue.Put(message);
Console.WriteLine("put");
}
```

   Beispiel für XMS .NET

```
textMessage = sessionWMQ.CreateTextMessage();
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. Prüfen Sie die TLS-Verbindung.

   Prüfen Sie den Kanalstatus, um festzustellen, ob die TLS-Verbindung erstellt wurde und ordnungsgemäß funktioniert.

Trace für SSLStream konfigurieren

Um Traceerstellungsereignisse und -nachrichten im Zusammenhang mit der SSLStream-Klasse aufzuzeichnen, müssen Sie der Anwendungskonfigurationsdatei für Ihre Anwendung einen Konfigurationsabschnitt für Systemdiagnosen hinzufügen.

Informationen zu diesem Vorgang

Anmerkung:

Diese Task gilt nur für IBM MQ classes for .NET Framework . Die Anwendungskonfigurationsdatei wird in IBM MQ classes for .NET (Bibliotheken.NET Standard und .NET 6) nicht unterstützt.

Wenn Sie der Anwendungskonfigurationsdatei keinen Konfigurationsabschnitt für Systemdiagnosen hinzufügen, zeichnet der von IBM MQ verwaltete .NET-Client keine Ereignisse, Traces oder Debugging-Punkte in Zusammenhang mit TLS und der SSLStream-Klasse auf.

Anmerkung: Wenn die IBM MQ-Tracefunktion mithilfe von **strmqtrc** gestartet wird, werden nicht alle erforderlichen TLS-Traces erfasst.

Vorgehensweise

1. Erstellen sie eine Anwendungskonfigurationsdatei (App.Config) für Ihr Anwendungsprojekt.
2. Fügen Sie einen Konfigurationsabschnitt für Systemdiagnosen hinzu, wie im folgenden Beispiel dargestellt.

```
<system.diagnostics>
           <sources>
                 <source name="System.Net" tracemode="includehex">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Sockets">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Cache">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
           </sources>
           <switches>
                 <add name="System.Net" value="Verbose"/>
                 <add name="System.Net.Sockets" value="Verbose"/>
                 <add name="System.Net.Cache" value="Verbose"/>
                 <add name="System.Security" value="Verbose"/>
                 <add name="System.Net.Security" value="Verbose"/>
           </switches>

           <sharedListeners>
                 <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace, amqmd
net, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
           </sharedListeners>
         <trace autoflush="true"/>
     </system.diagnostics>
```

Achtung: Das Feld `Version` des Eintrags `add name` muss die Version der verwendeten .net-Datei `amqmdnet.dll` sein.

Zugehörige Tasks

Traceerstellung für IBM MQ classes for .NET Framework -Clients mithilfe einer Anwendungskonfigurationsdatei

Beispielanwendungen für Implementierung von TLS in verwaltetem .NET
Es werden Beispielanwendungen bereitgestellt, die die TLS-Implementierung für verwaltetes .NET in IBM MQ classes for .NET, XMS .NET und im angepassten IBM MQ-Kanal für WCF veranschaulichen.

Aus der folgenden Tabelle geht hervor, wo die Beispielanwendungen gespeichert sind. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Tabelle 79. Position von Beispielanwendungen für Implementierung von TLS in verwaltetem .NET

IBM MQ.NET-Stack-Angebot	**Speicherposition der Beispiele**
Basis-.NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs
XMS .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs
Angepasster IBM MQ-Kanal für WCF	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs

Windows .NET Monitor verwenden

Die .NET Monitor-Anwendung entspricht weitgehend einem IBM MQ-Auslösemonitor.

Wichtig: Sehen Funktionen, die nur mit der Primärinstallation aufWindows für wichtige Informationen.

Sie können .NET-Komponenten erstellen, die bei jedem Empfang einer Nachricht in einer überwachten Warteschlange instanziiert werden und anschließend diese Nachricht verarbeiten. Der .NET-Monitor wird mit dem Befehl **runmqdnm** gestartet und mit dem Befehl **endmqdnm** gestoppt. Einzelheiten zu diesen Befehlen finden Sie unter runmqdnm und endmqdnm.

Zum Verwenden von .NET Monitor schreiben Sie eine Komponente, mit der die in einer amqmdnm.dll-Datei definierte Schnittstelle 'IMQObjectTrigger' implementiert wird.

Komponenten sind entweder transaktionsorientiert oder nicht transaktionsorientiert. Eine transaktionsorientierte Komponente muss Werte von 'System.EnterpriseServices.ServicedComponent' übernehmen und als 'RequiresTransaction' oder 'SupportsTransaction' registriert sein. Sie darf nicht als 'RequiresNew' registriert sein, da .NET Monitor bereits eine Transaktion aufgerufen hat.

Die Komponente empfängt MQQueueManager-, MQQueue- und MQMessage-Objekte aus **runmqdnm**. Sie kann auch eine Zeichenfolge für den Benutzerparameter empfangen, wenn dieser Parameter beim Start von runmqdnm mit der Befehlszeilenoption *-u* angegeben wurde. Beachten Sie, dass Ihre Komponente die Inhalte einer Nachricht empfängt, die auf der überwachten Warteschlange in einem MQMessage-Objekt angekommen ist. Das Herstellen einer Verbindung zum Warteschlangenmanager, das Öffnen der Warteschlange oder das Abrufen der Nachricht selbst ist dazu nicht erforderlich. Die Komponente muss anschließend die Nachricht entsprechend verarbeiten und die Steuerung an .NET Monitor zurückgeben.

Wenn Ihre Komponente als transaktionsorientierte Komponente geschrieben wurde, wird sie so registriert, dass die Transaktion mithilfe der von 'System.EnterpriseServices.ServicedComponent' bereitgestellten Funktionen festgeschrieben oder rückgängig gemacht werden.

Da die Komponente MQQueueManager- und MQQueue-Objekte sowie die Nachricht empfängt, verfügt sie über die vollständigen Kontextinformationen zu dieser Nachricht und kann beispielsweise eine weitere Warteschlange im gleichen Warteschlangenmanager öffnen, ohne eine separate Verbindung zu IBM MQ herstellen zu müssen.

Windows *Beispielcodefragmente*

Dieses Thema enthält zwei Beispiele für Komponenten, mit denen eine Nachricht aus der .NET-Überwachungsanwendung abgerufen und ausgedruckt werden kann. In einem Beispiel wird die transaktionsorientierte Verarbeitung und im anderen die nicht transaktionsorientierte Verarbeitung verwendet. In einem dritten Beispiel werden Dienstprogrammroutinen gezeigt, die für die ersten beiden Beispiele angewendet werden können. Alle Beispiele sind in C# geschrieben.

Beispiel 1: Transaktionsorientierte Verarbeitung

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

Beispiel 2: Nicht transaktionsorientierte Verarbeitung

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
```

```
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

Beispiel 3: Allgemeine Routinen

```
/**********************************************************************/
/* Licensed materials, property of IBM                                */
/* 63H9336                                                            */
/* (C) Copyright IBM Corp. 2005, 2024.                                */
/**********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ------------------------------------------------------------------ */
    /* Default prefix string of the namespace.                             */
    /* ------------------------------------------------------------------ */
    private string prefixText = "dnmsamp";

    /* ------------------------------------------------------------------ */
    /* Constructor that takes the replacement prefix string to use.       */
    /* ------------------------------------------------------------------ */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ------------------------------------------------------------------ */
    /* Display an arbitrary string to the console.                        */
    /* ------------------------------------------------------------------ */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }



 /* ------------------------------------------------------------------ */
    /* Display the content of the message passed to the console.          */
    /* ------------------------------------------------------------------ */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
```

```
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ------------------------------------------------------------------ */
    /* Convert the byte array into a hex string.                          */
    /* ------------------------------------------------------------------ */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }
  }
}
```

IBM MQ .NET-Programme kompilieren

Beispielbefehle zum Kompilieren von .NET-Anwendungen, die in verschiedenen Sprachen geschrieben wurden.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Zum Erstellen einer Anwendung in der Programmiersprache C# mithilfe von IBM MQ classes for .NET verwenden Sie den folgenden Befehl:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe My
Prog.cs
```

Zum Erstellen einer Anwendung in der Programmiersprache Visual Basic mithilfe von IBM MQ classes for .NET verwenden Sie den folgenden Befehl:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Zum Erstellen einer Anwendung in der Programmiersprache Managed C++ mithilfe von IBM MQ classes for .NET verwenden Sie den folgenden Befehl:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

Informationen zu anderen Sprachen finden Sie in der Dokumentation, die vom Anbieter der entsprechenden Sprache bereitgestellt wird.

IBM MQ .NET-Standalone-Client verwenden

Der IBM MQ .NET -Client bietet die Möglichkeit, eine IBM MQ .NET -Assembly zu packen und zu implementieren, ohne die vollständige IBM MQ -Clientinstallation auf Produktionssystemen zur Ausführung Ihrer Anwendungen verwenden zu müssen.

Vorbereitende Schritte

V 9.4.0 Ab IBM MQ 9.4.0basiert die an der Standardposition installierte `amqmdnetstd.dll` -Clientbibliothek auf .NET 6.

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0unterstützt IBM MQ .NET 8 -Anwendungen, die IBM MQ classes for .NETverwenden. Wenn Sie eine .NET 6 -Anwendung verwenden, können Sie diese Anwendung ausführen, ohne dass eine erneute Kompilierung erforderlich ist, indem Sie eine kleine Bearbeitung in der Datei `runtimeconfig` vornehmen, um `targetframeworkversion` auf `"net8.0"`zu setzen.

V 9.4.0 V 9.4.0 Removed Die mit .NET Standard 2.0erstellte IBM MQ .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt IBM MQ 9.4.0entfernt.

Stabilized LTS Die Bibliothek `amqmdnet.dll` wird weiterhin bereitgestellt, aber sie ist stabilisiert, d. h., es werden keine neuen Funktionen mehr hinzugefügt. Um die neuesten Funktionen nutzen zu können, müssen Sie eine Migration auf die Bibliothek `amqmdnetstd.dll` durchführen. Sie können die Bibliothek `amqmdnet.dll` jedoch unter IBM MQ 9.1 Long Term Support -oder Continuous Delivery -Releases weiterhin verwenden.

Informationen zu diesem Vorgang

Sie können Ihre IBM MQ .NET -Anwendungen auf einer Maschine erstellen, auf der der vollständige IBM MQ -Client installiert ist, und später die IBM MQ .NET -Assembly, d. h. `amqmdnetstd.dll`, zusammen mit Ihrer Anwendung packen und auf Produktionssystemen implementieren.

Bei den Anwendungen, die Sie erstellen und bereitstellen, kann es sich um traditionelle .NET -Anwendungen, Services oder Microsoft Azure Web-/Workeranwendungen handeln.

In solchen Implementierungen unterstützt der IBM MQ .NET-Client nur den verwalteten Modus für die Verbindung zu einem Warteschlangenmanager. Serverbindungen und Verbindungen im nicht verwalteten Clientmodus sind nicht verfügbar, da für diese beiden Modi eine vollständige IBM MQ-Clientinstallation benötigt wird. Beim Versuch, diese beiden anderen Modi zu verwenden, kommt es zu einer Ausnahmebedingung der Anwendung.

Prozedur

Verweis auf die IBM MQ .NET-Client-Assembly in Anwendungen

- Verweisen Sie wie bei früheren Releases in Ihrer Anwendung auf die Assembly `amqmdnetstd.dll`.

  Setzen Sie die Eigenschaft **`CopyLocal`** der `amqmdnetstd.dll` -Assembly auf `True` , um sicherzustellen, dass die `amqmdnetstd.dll` -Assembly in das Verzeichnis `bin` der Anwendung kopiert wird. Wenn diese Eigenschaft festgelegt wird, kann das Anwendungs-Packaging-Tool außerdem die erforderlichen binären Dateien für die Implementierung in Produktionssystemen und Microsoft Azure PaaS-Cloudumgebungen packen.

Unterstützung für globale Transaktionen hinzufügen

- Vergewissern Sie sich, dass Ihre Anwendung zusammen mit der Anwendung selbst die Überwachungsanwendung WMQDotnetXAMonitor implementiert.

  Wenn eine Anwendung die IBM MQ .NET-Funktion für verwaltete globale Transaktionen verwendet, muss neben der Anwendung selbst auch WMQDotnetXAMonitor auf dem System implementiert werden. Dieses Dienstprogramm wird zur Wiederherstellung unbestätigter Transaktionen benötigt.

Trace starten und stoppen

- Nur für IBM MQ classes for .NET Framework : Informationen zum Starten und Stoppen des Trace unter Verwendung der Anwendungskonfigurationsdatei und einer IBM MQ -spezifischen Tracekonfigurationsdatei finden Sie unter Traceerstellung für einen IBM MQ Classes for .NET Framework-Client unter Verwendung einer Anwendungskonfigurationsdatei.

  Sie müssen sowohl die Anwendungskonfigurationsdatei als auch eine IBM MQ-spezifische Tracekonfigurationsdatei verwenden, da keine vollständige IBM MQ-Clientinstallation vorhanden ist und die zum Starten und Stoppen des Trace verwendeten Standardtools **strmqtrc** und **endmqtrc** somit nicht verfügbar sind.

  Anmerkungen:

  – Diese Art der Tracegenerierung gilt sowohl für den weiterverteilbaren verwalteten .NET -Client als auch für den eigenständigen .NET -Client. Siehe .NET -Anwendungslaufzeit-nur Windows.

  – Die Anwendungskonfigurationsdatei wird in IBM MQ classes for .NET (Bibliotheken.NET Standard und .NET 6) nicht unterstützt. Zum Aktivieren des Trace für IBM MQ classes for .NET (.NET Standard -und .NET 6 -Bibliotheken) verwenden Sie die Umgebungsvariable **MQDOTNET_TRACE_ON** . Siehe Traceerstellung für IBM MQ .NET -Anwendungen mit Umgebungsvariablen.

- V 9.4.0

  Starten und stoppen Sie den Trace, indem Sie die Datei `mqclient.ini` verwenden und die entsprechenden Eigenschaften der Trace-Zeilengruppe festlegen.

  Siehe Traceerstellung für IBM MQ .NET -Anwendungen mit mqclient.ini.

  Ab IBM MQ 9.4.0können Sie den Trace konfigurieren, indem Sie die Datei `mqclient.ini` verwenden und die entsprechenden Eigenschaften der Zeilengruppe 'Trace' festlegen. Sie können die Traceerstellung auch dynamisch mit der Datei `mqclient.ini` aktivieren und inaktivieren.

Bindungsumleitung in der Anwendungskonfigurationsdatei aktivieren

- Um den Bindungsverweis für die Kompilierzeit der IBM MQ-.NET-Assembly auf eine spätere Version der Assembly zu aktivieren, fügen Sie die Eigenschaft <dependentAssembly> zur Anwendungskonfigurationsdatei hinzu.

  Das folgende Beispielsnippet aus der Datei `app.config` leitet eine Anwendung um, die mit der IBM MQ 8.0.0 Fix Pack 2-Version (`8.0.0.2`) der IBM MQ .NET-Assembly kompiliert wurde, wenn später Fixpack IBM MQ 8.0.0 Fix Pack 3 angewandt wurde, durch das die IBM MQ.NET-Assembly auf Version `8.0.0.3` aktualisiert wurde.

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
                        publicKeyToken="dd3cb1c9aae9ec97"
                        culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

Zugehörige Konzepte

„Installieren von IBM MQ classes for .NET” auf Seite 584
IBM MQ classes for .NET, einschließlich Beispiele, werden mit IBM MQ unter Windows und Linux installiert.

Weiterverteilbare Clients

.NET-Anwendungslaufzeit - nur Windows

Zugehörige Tasks

„Anwendung 'WMQDotnetXAMonitor' verwenden” auf Seite 604

Der IBM MQ .NET-Client stellt eine XA-Überwachungsanwendung namens 'WmqDotnetXAMonitor' zur Verfügung, mit der Sie alle unvollständigen verteilten Transaktionen wiederherstellen können. Die Anwendung 'WmqDotnetXAMonitor' stellt eine Verbindung zum Warteschlangenmanager mit unbestätigten Transaktionen her und löst anschließend die Transaktion gemäß den von Ihnen festgelegten Parametern auf.

Traceerstellung für IBM MQ .NET-Anwendungen

`OutboundSNI`Eigenschaft

Sie können die Eigenschaft **`OutboundSNI`** in einer Anwendung über eine Eigenschaft oder eine Umgebungsvariable festlegen.

Ab IBM MQ 9.3.0 können Sie die Eigenschaft MQC.OUTBOUND_SNI_PROPERTY in der Anwendung unter Verwendung einer Hashtabelle festlegen, wenn Sie zum Herstellen der Verbindung zum Warteschlangenmanager die Klasse `MQQueueManager` verwenden.

Für die Eigenschaft MQC.OUTBOUND_SNI_PROPERTY können die folgenden Werte angegeben werden:

- MQC.OUTBOUND_SNI_CHANNEL (wird CHANNEL zugeordnet)
- MQC.OUTBOUND_SNI_HOSTNAME (wird HOSTNAME zugeordnet)
- MQC.OUTBOUND_SNI_ASTERISK (wird * zugeordnet)

Darüber hinaus können Sie die Eigenschaft **`OutboundSNI`** mithilfe der Umgebungsvariablen MQOUTBOUND_SNI festlegen, für die die folgenden Werte angegeben werden können:

- CHANNEL
- HOSTNAME
- *

Legen Sie den Wert für **`OutboundSNI`** in der Datei `App.config` wie bei allen anderen `mqclient.ini` -Eigenschaften fest.

Anmerkung: Falls kein bestimmter Wert festgelegt wird, hat die Eigenschaft standardmäßig den Wert MQC.OUTBOUND_SNI_CHANNEL.

Die Vorrangregelung für die Einstellung der Eigenschaft **`OutboundSNI`** im verwalteten Knoten lautet wie folgt:

1. Eigenschaft auf Anwendungsebene
2. Umgebungsvariable

Für die Eigenschaft **`OutboundSNI`** im nicht verwalteten Knoten wird nur `mqclient.ini` unterstützt.

Die Eigenschaften, die in der Datei `App.config` festgelegt sind, gelten nur für .NET Framework -Anwendungen.

Falls Sie auf Anwendungsebene oder in der Datei `App.config` einen Wert angeben, der nicht gültig ist, wird der Rückkehrcode MQRC_OUTBOUND_SNI_NOT_VALID ausgegeben.

Wenn Sie eine ungültige Umgebungsvariable festlegen oder einen ungültigen Wert in der Datei `mqclient.ini` angeben, wird der Standardwert `CHANNEL` verwendet.

`OutboundSNI` und mehrere Zertifikate

IBM MQ verwendet den SNI-Header, um mehrere Zertifikatsfunktionen bereitzustellen. Wenn eine Anwendung eine Verbindung zu einem IBM MQ -Kanal herstellt, der für die Verwendung eines anderen Zertifikats über das CERTLABL-Feld konfiguriert ist, muss die Anwendung eine Verbindung mit der **`OutboundSNI`** -Einstellung CHANNEL herstellen.

Wenn eine Anwendung mit einer anderen **`OutboundSNI`** -Einstellung als CHANNEL eine Verbindung zu einem Kanal mit einer konfigurierten Zertifikatsbezeichnung herstellt, wird die Anwendung mit dem Fehler MQRC_SSL_INITIALIZATION_ERROR zurückgewiesen und eine Nachricht AMQ9673 in den Fehlerprotokollen des Warteschlangenmanagers ausgegeben.

Weitere Informationen dazu, wie IBM MQ mehrere Zertifikatsfunktionen bereitstellt, finden Sie unter Funktionalität für mehrere Zertifikate in IBM MQ .

XMS .NET-Anwendungen entwickeln

IBM MQ Message Service Client (XMS) for .NET (kurz: XMS .NET) stellt eine Anwendungsprogrammierschnittstelle (API) namens XMS bereit, die dieselben Schnittstellen besitzt wie die API für Java Message Service (kurz: JMS). IBM MQ Message Service Client (XMS) for .NET enthält eine vollständig verwaltete Implementierung von XMS, die von jeder mit .NET kompatiblen Sprache verwendet werden kann.

Vorbereitende Schritte

V 9.4.0 Deprecated V 9.4.0 Ab IBM MQ 9.4.0sind in IBM MQ classes for XMS .NETdie Methoden WriteObject(), ReadObject(), CreateObjectMessage () und die Klassen ObjectMessage und XmsObjectMessageImpl , die für die Serialisierung und Deserialisierung von Daten verwendet werden, veraltet.

V 9.4.0 V 9.4.0 Removed Die mit.NET Standard 2.0erstellte XMS .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

Informationen zu diesem Vorgang

XMS unterstützt Folgendes:

- Punkt-zu-Punkt-Messaging
- Publish/Subscribe-Messaging
- Synchrone Nachrichtenübermittlung
- Asynchrone Nachrichtenübermittlung

Eine XMS-Anwendung kann Nachrichten mit folgenden Anwendungstypen austauschen:

- Eine XMS-Anwendung
- Eine IBM MQ classes for JMS-Anwendung
- Eine native IBM MQ-Anwendung
- Eine JMS -Anwendung, die den Standard-Messaging-Provider von IBM MQ verwendet

Eine XMS-Anwendung kann eine Verbindung zu folgenden Messaging-Servern herstellen und deren Ressourcen verwenden:

Warteschlangenmanager der IBM MQ
Die Anwendung kann die Verbindung entweder im Bindungsmodus oder im Clientmodus herstellen.

WebSphere Application Server service integration bus
Die Anwendung kann eine direkte TCP/IP-Verbindung oder HTTP über TCP/IP verwenden.

IBM Integration Bus
Der Transport der Nachrichten zwischen der Anwendung und dem Broker erfolgt mithilfe von WebSphere MQ Real-Time Transport. Die Übermittlung der Nachrichten zur Anwendung kann mit WebSphere MQ Multicast Transport erfolgen.

Wenn eine XMS-Anwendung eine Verbindung zu einem IBM MQ-Warteschlangenmanager herstellt, kann sie WebSphere MQ Enterprise Transport für die Kommunikation mit IBM Integration Bus verwenden. Alternativ dazu kann eine XMS-Anwendung eine Verbindung zu IBM MQ herstellen und das Publish/Subscribe-Messaging nutzen.

V 9.4.0 IBM MQ 9.4.0 stellt eine XMS .NET -Clientbibliothek bereit, die für .NET 6 als Zielframework erstellt wurde. Weitere Informationen finden Sie unter „Installieren von IBM MQ classes for XMS .NET” auf Seite 651.

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0unterstützt IBM MQ .NET 8 -Anwendungen, die IBM MQ classes for XMS .NETverwenden. Weitere Informationen finden Sie unter „Installieren von IBM MQ classes for XMS .NET” auf Seite 651.

Von XMS .NET verwaltete Anwendungen können automatisch Verbindungen zwischen Clusterwarteschlangenmanagern ausgleichen. Sowohl die IBM MQ classes for XMS .NET -als auch die IBM MQ classes for XMS .NET Framework -Bibliotheken werden unterstützt. Weitere Informationen finden Sie unter Informationen zu Uniform-Clustern und Automatische Verteilung der Anwendungslast.

Weitere Informationen zu den Unterschieden zwischen IBM MQ classes for XMS .NET Framework und IBM MQ classes for XMS .NETfinden Sie unter „Installieren von IBM MQ classes for XMS .NET” auf Seite 651.

Zugehörige Tasks

Anfordern von Unterstützung beim IBM Support

Fehlerbehebung bei Problemen mit XMS .NETproblems

Von XMS unterstützte Messaging-Stile

XMS unterstützt das Punkt-zu-Punkt- und das Publish/Subscribe-Messaging.

Messaging-Stile werden auch als Messagingdomänen bezeichnet.

Punkt-zu-Punkt-Messaging

Eine gängige Form des Punkt-zu-Punkt-Messaging nutzt ein Warteschlangensystem. Im einfachsten Fall sendet eine Anwendung eine Nachricht an eine andere Anwendung gibt dabei implizit oder explizit eine Zielwarteschlange an. Das zugrunde liegende Messaging- und Warteschlangensystem empfängt die Nachricht von der sendenden Anwendung und leitet sie an die Zielwarteschlange weiter. Die empfangende Anwendung kann die Nachricht dann aus der Warteschlange abrufen.

Wenn das zugrunde liegende Messaging- und Warteschlangensystem IBM Integration Bus enthält, kann IBM Integration Bus die Nachricht replizieren und Kopien der Nachricht an unterschiedliche Warteschlangen senden. Auf diese Weise kann mehr als eine Anwendung die Nachricht empfangen. IBM Integration Bus kann eine Nachricht auch umwandeln und ihr weitere Daten hinzufügen.

Ein Schlüsselmerkmal des Punkt-zu-Punkt-Messaging ist, dass eine Anwendung eine Nachricht in eine lokale Warteschlange einreiht, wenn sie die Nachricht sendet. Das zugrunde liegende Messaging- und Warteschlangensystem bestimmt dann, an welche Zielwarteschlange die Nachricht gesendet wird. Die empfangende Anwendung ruft schließlich die Nachricht aus der Zielwarteschlange ab.

Publish/Subscribe-Messaging

Beim Publish/Subscribe-Messaging gibt es zwei Anwendungstypen: Publisher (Bereitsteller) und Subskribenten (Abonnent).

Ein *Publisher* stellt Informationen in Form einer Veröffentlichungsnachricht bereit. Bei der Veröffentlichung einer Nachricht gibt der Publisher ein Thema an, auf das sich die Informationen in der Nachricht beziehen.

Ein *Subskribent* konsumiert die bereitgestellten Informationen. Ein Subskribent gibt die Themen an, zu denen er Nachrichten empfangen möchte, indem er Subskriptionen (Abonnements) erstellt.

Das Publish/Subscribe-System erhält Veröffentlichungen von Publishern und Subskriptionen von Subskribenten. Das System leitet die Veröffentlichungen an die Subskribenten weiter. Ein Subskribent erhält nur Veröffentlichungen zu den Themen, die er abonniert hat.

Ein Schlüsselmerkmal des Publish/Subscribe-Messaging ist, dass ein Publisher beim Veröffentlichen einer Nachricht ein Thema angibt. Er gibt jedoch nicht die Subskribenten an. Wenn eine Nachricht zu einem Thema veröffentlicht wird, für das es keine Subskribenten gibt, empfängt keine Anwendung diese Nachricht.

Eine Anwendung kann sowohl als Veröffentlichungsanwendung als auch als Subskribent fungieren.

XMS-Objektmodell

Die XMS-API ist eine objektorientierte Schnittstelle. Das XMS-Objektmodell basiert auf dem Objektmodell von JMS 1.1.

XMS-Hauptklassen

Die wichtigsten XMS-Klassen, oder Objekttypen, sind folgende:

ConnectionFactory
Ein `ConnectionFactory`-Objekt enthält eine Gruppe von Parametern für eine Verbindung. Eine Anwendung verwendet ein `ConnectionFactory`-Objekt zum Erstellen einer Verbindung. Eine Anwendung kann die Parameter während der Laufzeit bereitstellen und basierend darauf ein `ConnectionFactory`-Objekt erstellen. Alternativ dazu können die Verbindungsparameter auch in einem Repository für verwaltete Objekte gespeichert werden. Eine Anwendung kann ein Objekt aus dem Repository abrufen und basierend darauf ein `ConnectionFactory`-Objekt erstellen.

Verbindung
Ein `Connection`-Objekt enthält eine aktive Verbindung von einer Anwendung zu einem Messaging-Server. Eine Anwendung verwendet eine Verbindung, um Sitzungen zu erstellen.

Destination
Eine Anwendung sendet oder empfängt Nachrichten mithilfe eines `Destination`-Objekts. In der Publish/Subscribe-Domäne bindet ein `Destination`-Objekt ein Thema ein; in der Punkt-zu-Punkt-Domäne bindet ein `Destination`-Objekt eine Warteschlange ein. Eine Anwendung kann mit den Parametern während der Laufzeit ein `Destination`-Objekt erstellen. Alternativ dazu kann die Anwendung ein `Destination`-Objekt auch basierend auf einer Objektdefinition erstellen, die in einem Repository für verwaltete Objekte gespeichert ist.

Sitzung
Ein `Session`-Objekt ist ein Einzelthreadkontext zum Senden und Empfangen von Nachrichten. Eine Anwendung verwendet ein `Session`-Objekt zum Erstellen von `Message`-, `MessageProducer`- und `MessageConsumer`-Objekten.

Nachricht
Ein `Message`-Objekt kapselt das `Message`-Objekt, das eine Anwendung mithilfe eines `MessageProducer`-Objekts sendet oder eines `MessageConsumer`-Objekts empfängt.

MessageProducer
Eine Anwendung verwendet ein `MessageProducer`-Objekt zum Senden von Nachrichten an ein Ziel.

MessageConsumer
Eine Anwendung verwendet ein `MessageConsumer`-Objekt zum Empfangen von Nachrichten, die an ein Ziel gesendet wurden.

XMS-Objekte und ihre Beziehungen

In Abbildung 52 auf Seite 649 sind die wichtigsten XMS-Objekttypen dargestellt: ConnectionFactory-, Connection-, Session-, MessageProducer-, MessageConsumer-, Message- und Destination-Objekte. 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.

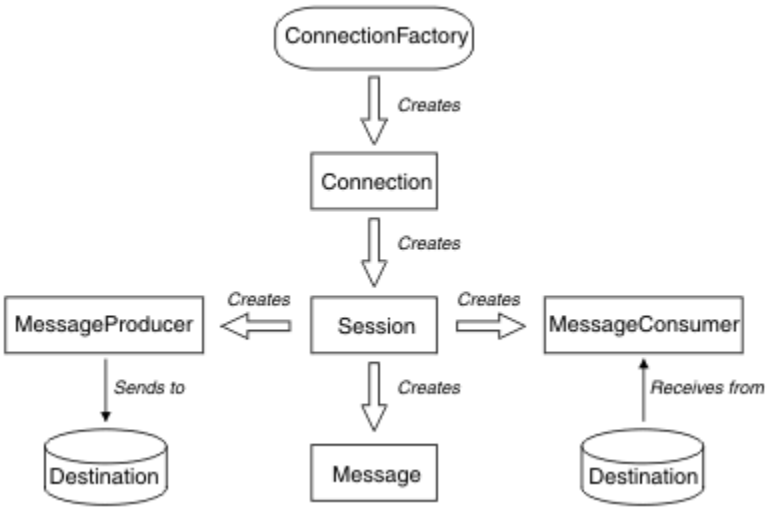


Abbildung 52. XMS-Objekte und ihre Beziehungen

In XMS .NET werden die XMS-Klassen als eine Gruppe von .NET-Schnittstellen definiert. Wenn Sie XMS .NET-Anwendungen codieren, benötigen Sie nur die deklarierten Schnittstellen.

Das XMS-Objektmodell basiert auf den domänenunabhängigen Schnittstellen, die in Java Message Service Specification Version 1.1 beschrieben sind. Domänenspezifische Klassen wie `Topic`, `TopicPublisher` und `TopicSubscriber` werden nicht bereitgestellt.

Attribute und Eigenschaften von XMS-Objekten

Die Merkmale eines XMS-Objekts können in Form von Attributen und Eigenschaften auf unterschiedliche Weisen implementiert werden.

Attribute
Ein Objektmerkmal, das immer vorhanden ist und Speicherplatz belegt, selbst wenn dem Attribut kein Wert zugeordnet ist. In dieser Hinsicht ähnelt ein Attribut einem Feld in einer Datenstruktur mit fester Länge. Das Unterscheidungskriterium von Attributen ist, dass es für jedes Attribut spezifische Methoden gibt, mit denen sein Wert festgelegt und abgerufen wird.

Eigenschaften
Eine Objekteigenschaft ist erst dann vorhanden und belegt Speicherplatz, nachdem ein Wert für sie angegeben wurde. Nachdem ein Wert für eine Eigenschaft festgelegt wurde, kann sie nicht mehr gelöscht oder der ihr zugeordnete Speicher wiederhergestellt werden. Sie können jedoch den Wert einer Eigenschaft ändern. XMS stellte eine Gruppe generischer Methoden zum Festlegen und Abrufen von Eigenschaftswerten bereit.

Verwaltete Objekte

Mithilfe von verwalteten Objekten können Sie die Verbindungseinstellungen von Clientanwendungen verwalten, die über ein zentrales Repository verwaltet werden sollen. Dabei ruft eine Anwendung Objektdefinitionen aus dem zentralen Repsitory ab und erstellt basierend darauf `ConnectionFactory`- und `Destination`-Objekte. Verwaltete Objekte ermöglichen eine Entkopplung der Anwendungen von den Ressourcen, die sie während der Laufzeit verwenden.

XMS-Anwendungen können beispielsweise mit verwalteten Objekten geschrieben und getestet werden, die auf eine Gruppe von Verbindungen und Zielen in einer Testumgebung verweisen. Wenn diese Anwendungen dann in der Produktionsumgebung bereitgestellt werden, kann die Konfiguration der verwalteten Objekte so geändert werden, dass sie nun auf Verbindungen und Ziele in der Produktionsumgebung verweisen.

XMS unterstützt zwei Arten von verwalteten Objekten:

- Ein `ConnectionFactory`-Objekt, mit dem die Anwendungen die einleitende Verbindung zum Server herstellen.
- Ein `Destination`-Objekt, mit dem die Anwendungen das Ziel für gesendete Nachrichten und die Quelle von empfangenen Nachrichten angeben. Ein Ziel kann entweder ein Thema oder eine Warteschlange auf dem Server sein, zu dem die Anwendung eine Verbindung herstellt.

Das Verwaltungstool **`JMSAdmin`** gehört zum Lieferumfang von IBM MQ. Mit diesem Tool werden verwaltete Objekte in einem zentralen Repository für verwaltete Objekte erstellt und verwaltet.

Die verwalteten Objekte im Repository können von IBM MQ classes for JMS- und XMS-Anwendungen verwendet werden. XMS-Anwendungen können mit dem `ConnectionFactory`- und dem `Destination`-Objekt eine Verbindung zu einem IBM MQ-Warteschlangenmanager herstellen. Ein Administrator kann die im Repository befindlichen Objektdefinitionen ohne Auswirkung auf den Anwendungscode ändern.

Das folgende Diagramm veranschaulicht, wie verwaltete Objekte normalerweise von einer XMS-Anwendung verwendet werden. Die linke Seite des Diagramms zeigt ein Repository, das die Definitionen des ConnectionFactory- und des Destination-Objekts enthält, die mithilfe einer Administrationskonsole verwaltet werden. Die rechte Seite des Diagramms zeigt eine XMS-Anwendung, die Objektdefinitionen im Repository abruft und sie zum Herstellen von Verbindungen zu einem Messaging-Server verwendet.

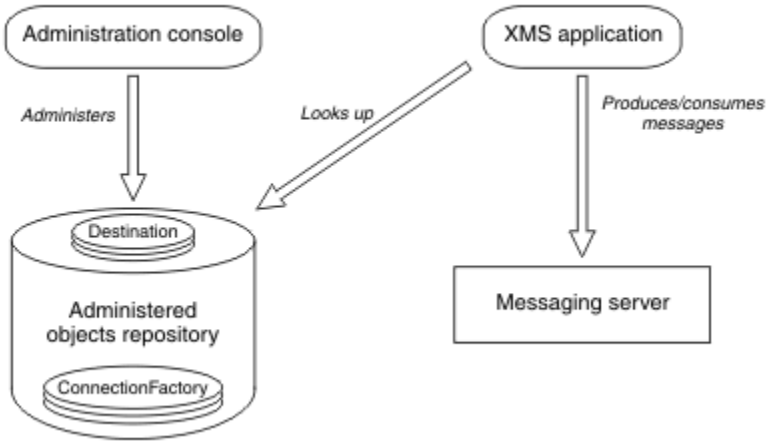


Abbildung 53. Typische Verwendung von verwalteten Objekten durch eine XMS-Anwendung

XMS-Nachrichtenmodell

Das XMS-Nachrichtenmodell ist mit dem IBM MQ classes for JMS-Nachrichtenmodell identisch.

XMS implementiert insbesondere auch dieselben Nachrichtenheaderfelder und Nachrichteneigenschaften wie IBM MQ classes for JMS:

- JMS-Headerfelder. Die Namen dieser Felder beginnen mit dem Präfix 'JMS'.
- Von JMS definierte Eigenschaften. Diese Felder haben Eigenschaften, deren Namen mit dem Präfix 'JMSX' beginnen.
- Von IBM definierte Eigenschaften. Diese Felder haben Eigenschaften, deren Namen mit dem Präfix 'JMS_IBM' beginnen.

Daher können XMS-Anwendungen Nachrichten mit IBM MQ classes for JMS-Anwendungen austauschen. In jeder Nachricht werden einige Headerfelder und Eigenschaften durch die Anwendung und andere durch XMS oder IBM MQ classes for JMS festgelegt. Einige der von XMS oder IBM MQ classes for JMS definierten Felder werden beim Senden der Nachricht und andere bei deren Empfang festgelegt. Headerfelder und Eigenschaften werden gegebenenfalls mithilfe einer Nachricht über einen Messaging-Server weitergegeben. Sie werden dann für Anwendungen, die die Nachricht erhalten, verfügbar gemacht.

Zugehörige Konzepte
IBM MQ classes for JMS

Windows Linux Installieren von IBM MQ classes for XMS .NET

IBM MQ classes for XMS .NET, einschließlich Beispiele, werden mit IBM MQ unter Windows und Linuxinstalliert.

Installation

V 9.4.0 IBM MQ 9.4.0 stellt eine XMS .NET -Clientbibliothek bereit, die für .NET 6 als Zielframework erstellt wurde. Ab IBM MQ 9.4.0ist Microsoft .NET 6.0 die mindestens erforderliche Version für aktive Anwendungen, die IBM MQ -Bibliotheken verwenden, die mit .NET 6 als Zielframework erstellt wurden. Die XMS .NET -Clientbibliothek, die mit .NET 6 als Zielframework erstellt wurde, ist unter *MQ_INSTALLATION_PATH*`/bin` unter Windows und unter *MQ_INSTALLATION_PATH*`/lib64` unter Linuxverfügbar.

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0unterstützt IBM MQ .NET 8 -Anwendungen, die IBM MQ classes for XMS .NETverwenden. Wenn Sie eine .NET 6 -Anwendung verwenden, können Sie diese Anwendung ausführen, ohne dass eine erneute Kompilierung erforderlich ist, indem Sie eine kleine Bearbeitung in der Datei `runtimeconfig` vornehmen, um `targetframeworkversion` auf `"net8.0"`zu setzen.

V 9.4.0 Deprecated V 9.4.0 Ab IBM MQ 9.4.0sind in IBM MQ classes for XMS .NETdie Methoden WriteObject(), ReadObject(), CreateObjectMessage () und die Klassen ObjectMessage und XmsObjectMessageImpl , die für die Serialisierung und Deserialisierung von Daten verwendet werden, veraltet.

V 9.4.0 V 9.4.0 Removed Die mit.NET Standard 2.0erstellte XMS .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

Bibliothek `amqmxmsstd.dll`

V 9.4.0 Ab IBM MQ 9.4.0ist die Bibliothek `amqmxmsstd.dll` , die mit .NET 6 als Zielframework erstellt wurde, an den folgenden Positionen verfügbar:

- Windows Unter Windows: *MQ_INSTALLATION_PATH*`\bin`. Die Beispielanwendungen werden in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`installiert.
- Linux Unter Linux: *MQ_INSTALLATION_PATH*`\lib64`. Die .NET -Beispiele befinden sich in *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

V 9.4.0 V 9.4.0 Removed Die mit .NET Standard 2.0erstellte XMS .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

Achtung: V 9.4.0 V 9.4.0 Removed Ab IBM MQ 9.4.0werden XMS .NET -Clientbibliotheken, die mit .NET Standard 2.0 als Zielframework erstellt wurden, entfernt. Diese Bibliotheken sind in IBM MQ 9.3.1veraltet.

Stabilized Alle Bibliotheken `IBM.XMS.*` werden weiterhin bereitgestellt, sind jedoch stabilisiert, d. h., es werden keine neuen Funktionen mehr hinzugefügt. Um die neuesten Funktionen nutzen zu können, müssen Sie eine Migration auf die Bibliothek `amqmxmsstd.dll` durchführen. In IBM MQ 9.1 Long Term Support- oder Continuous Delivery-Releases können Sie die vorhandenen Bibliotheken jedoch weiterhin verwenden.

V 9.4.0 V 9.4.0 Es folgen zwei Szenarios, die nach dem Entfernen der `netstandard2.0` -Bibliotheken auftreten können:

- Wenn Sie eine IBM MQ classes for XMS .NET Framework -Anwendung verwenden, die mit `netstandard2.0` -Bibliotheken wie `amqmdnetstd.dll`erstellt wurde, müssen Sie Ihre Anwendung mit den Microsoft.NET Framework 4.7.2 -Bibliotheken wie `amqmdnet.dll`erneut erstellen, damit Ihre Anwen-

dung erfolgreich ausgeführt werden kann. Wenn Sie Ihre Anwendung nicht erneut erstellen, erhalten Sie möglicherweise ein System.IO.Unexceptionable -Nachricht ohne Ausnahme:

```
Ausnahmebedingung abgefangen: System.IO.FileLoadException: Could not load file or assembly
'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' or one of its
dependencies. Die Manifestdefinition der loktierten Baugruppe stimmt nicht mit der Assembly-Re-
ferenz überein. (Ausnahme von HRESULT: 0x80131040)
Dateiname: 'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e'
   bei SimplePut.SimplePut.PutMessages()
   bei SimplePut.SimplePut.Main (String [] args) in C:\SampleCode\Program.cs:line 132
```

- Wenn Sie eine .NET 6 -Anwendung verwenden, die mit `netstandard2.0` -Bibliotheken erstellt wurde, müssen Sie diese Bibliotheken nur durch dieselben .NET 6 -Bibliotheken im Ordner `bin` des Anwendungslaufzeitverzeichnisses ersetzen. Es ist keine Neuerstellung erforderlich.

  Anmerkung: Die .NET 6 -Ersatzbibliothek sollte immer dieselbe oder eine höhere Version als die ersetzte `netstandard2.0` -Bibliothek haben.

Die IBM MQ classes for XMS .NET Standard können aus dem NuGet -Repository heruntergeladen werden. Das NuGet-Paket enthält die Bibliothek `amqmxmsstd.dll` und die Bibliothek `amqmdnetstd.dll`. `amqmxmsstd.dll` ist von `amqmdnetstd.dll` abhängig und beim Packen der XMS .NET Core-Anwendung sollten sowohl `amqmxmsstd.dll` als auch `amqmdnetstd.dll` zusammen mit der XMS .NET Core-Anwendung gepackt werden. Weitere Informationen finden Sie im Abschnitt „IBM MQ classes for XMS .NET aus dem NuGet-Repository herunterladen” auf Seite 654.

dspmqver-Befehl

Mit dem Befehl **dspmqver** können Versions- und Buildinformationen für die .NET Core-Komponente angezeigt werden.

Vergleich zwischen IBM MQ classes for XMS .NET Framework -und IBM MQ classes for XMS .NET .NET 6 -Bibliotheken -und .NET 6 -Bibliotheken)

In der folgenden Tabelle sind die Features für IBM MQ classes for XMS .NET Framework im Vergleich zu den Features für IBM MQ classes for XMS .NET und .NET 6) aufgelistet.

Tabelle 80. Unterschiede zwischen IBM MQ classes for XMS .NET Framework und IBM MQ classes for XMS .NET

Funktion	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
Klassennamen (APIs)	Alle Klassen bleiben in jedem Netz gleich.	Alle Klassen bleiben in jedem Netz gleich.
Betriebssystem	Windows	Windows Dockerisierte Container Linux macOS
`app.config`-Datei (Konfigurationsdatei zum Aktivieren von Trace im erneut verteilbaren Client)	Mit der Datei `app.config` wird der Trace für das weiterverteilbare Paket aktiviert.	`app.config` wird nicht unterstützt. Verwenden Sie Umgebungsvariablen.

Tabelle 80. Unterschiede zwischen IBM MQ classes for XMS .NET Framework und IBM MQ classes for XMS .NET (Forts.)

Funktion	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
Trace	Um einen Trace für den XMS .NET -Client zu erstellen, können Sie die vorhandenen Umgebungsvariablen verwenden, z. B. die Umgebungsvariable **XMS_TRACE_ON** , die zum Aktivieren des Trace verwendet wird. Weitere Informationen finden Sie unter XMS .NET mithilfe von XMS-Umgebungsvariablenkonfigurieren. Für weiterverteilbare Clients kann die Datei `app.config` zum Aktivieren des Trace verwendet werden. V 9.4.0 Ab IBM MQ 9.4.0können Sie den Trace aktivieren und inaktivieren, indem Sie die Datei `mqclient.ini` verwenden und die entsprechenden Eigenschaften der Trace-Zeilengruppe festlegen. Sie können die Traceerstellung auch dynamisch mit der Datei `mqclient.ini` aktivieren und inaktivieren. Weitere Informationen hierzu finden Sie im Abschnitt Traceerstellung für IBM MQ .NET -Anwendungen mit mqclient.ini.	Um einen Trace für den XMS .NET -Client zu erstellen, können Sie die vorhandenen Umgebungsvariablen verwenden, z. B. die Umgebungsvariable **XMS_TRACE_ON** , die zum Aktivieren des Trace verwendet wird. Weitere Informationen finden Sie unter XMS .NET mithilfe von XMS-Umgebungsvariablenkonfigurieren. V 9.4.0 Ab IBM MQ 9.4.0können Sie den Trace aktivieren und inaktivieren, indem Sie die Datei `mqclient.ini` verwenden und die entsprechenden Eigenschaften der Trace-Zeilengruppe festlegen. Sie können die Traceerstellung auch dynamisch mit der Datei `mqclient.ini` aktivieren und inaktivieren. Weitere Informationen hierzu finden Sie im Abschnitt Traceerstellung für IBM MQ .NET -Anwendungen mit mqclient.ini.
Transportmodi	Verwaltet, nicht verwaltet und Bindungen	Verwaltet
TLS	Der Windows-Schlüsselspeicher wird für die Speicherung der Zertifikate verwendet.	Unter Windows muss der Schlüsselspeicher für die Speicherung der Zertifikate verwendet werden. Zulässige Werte sind *USER oder *SYSTEM. Auf Basis der Eingabe sucht der IBM MQ .NET-Client im Windows-Keystore des aktuellen Benutzers oder im gesamten System. Unter Linux wird empfohlen, die Klasse X509Store für die Installation von Zertifikaten zu verwenden, und .NET Core installiert Zertifikate an der folgenden Position: `".dotnet/corefx/cryptography/x509stores"`.
CCDT (Client Channel Definition Table)	Unterstützt	Unterstützt und die Einstellungen des CCDT-Pfads sind mit .NET Framework-Klassen identisch.
Automatische Neuverbindung des Clients	Unterstützt	Unterstützt

Tabelle 80. Unterschiede zwischen IBM MQ classes for XMS .NET Framework und IBM MQ classes for XMS .NET (Forts.)		
Funktion	**IBM MQ classes for XMS .NET Framework**	**IBM MQ classes for XMS .NET**
Dezentrale Transaktionsverarbeitung	Unterstützt	Nicht unterstützt
Installation von DLLs (Dynamic Link Library) im Global Assembly Cache (GAC)	Dlls werden als Teil der IBM MQ-Installation im GAC installiert.	Dlls werden nicht als Teil der IBM MQ-Installation im GAC installiert.
Unterstützung für WMQ-, WPM- und RTT-Verbindungstypen	Unterstützt WMQ-, WPM- und RTT-Verbindungstypen	Unterstützt nur WMQ
JNDI-verwaltete Objekte	Unterstützt LDAP und FileSystem	Unterstützt nur das Dateisystem

Ab IBM MQ 9.3.0müssen Sie zum Ausführen von IBM MQ classes for XMS .NET Framework Microsoft.NET Framework V4.7.2 oder höher installieren.

Zugehörige Tasks

„XMS-Beispielanwendungen verwenden” auf Seite 661
Die XMS .NET-Beispielanwendungen bieten einen Überblick über die häufig verwendeten Funktionen der jeweiligen APIs. Mithilfe der Beispielanwendungen können Sie Ihre Installation und Messaging-Server-Konfiguration überprüfen sowie Ihre eigenen Anwendungen erstellen.

Windows Linux IBM MQ classes for XMS .NET aus dem NuGet-Repository herunterladen

Die IBM MQ classes for XMS .NET sind im NuGet-Repository für den Download verfügbar, damit sie auf einfache Weise von .NET Developers verarbeitet werden können.

Informationen zu diesem Vorgang

Bei NuGet handelt es sich um den Paketmanager für die Entwicklungsplattformen von Microsoft, einschließlich .NET. Die NuGet-Client-Tools ermöglichen das Erzeugen und Verarbeiten von Paketen. Ein NuGet-Paket ist eine einzelne komprimierte Datei mit der Erweiterung `.nupkg`, die kompilierten Code (DLLs), weitere zu diesem Code gehörende Dateien sowie eine beschreibende Manifestdatei mit Informationen wie beispielsweise der Versionsnummer des Pakets enthält.

Sie können das Paket `IBMXMSDotnetClient` NuGet , das sowohl die Bibliothek `amqmdnetstd.dll` als auch die Bibliothek `amqmxmsstd.dll` enthält, aus der NuGet -Galerie herunterladen. Dies ist das zentrale Paketrepository, das von allen Paketautoren und -konsumenten verwendet wird.

Anmerkung: V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0enthält das Paket NuGet Bibliotheken, die mit .NET 6 als Zielframework erstellt wurden.

Removed Die mit .NET Standard 2.0erstellte XMS .NET -Clientbibliothek, die ab IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

V 9.4.0 V 9.4.0 Ab IBM MQ 9.4.0unterstützt IBM MQ .NET 8 -Anwendungen, die IBM MQ classes for XMS .NETverwenden. Wenn Sie eine .NET 6 -Anwendung verwenden, können Sie diese Anwendung ausführen, ohne dass eine erneute Kompilierung erforderlich ist, indem Sie eine kleine Bearbeitung in der Datei `runtimeconfig` vornehmen, um `targetframeworkversion` auf `"net8.0"`zu setzen.

Zum Herunterladen des Pakets `IBMXMSDotnetClient` gibt es drei Möglichkeiten:

- Mithilfe von Microsoft Visual Studio. NuGet wird als Erweiterung für Microsoft Visual Studio verteilt. Ab Microsoft Visual Studio 2012 ist NuGet standardmäßig vorinstalliert.
- Aus der Befehlszeile mithilfe des NuGet-Paketmanagers oder der .NET-Befehlszeilenschnittstelle.
- Über einen Web-Browser.

Wie beim weiterverteilbaren Paket aktivieren Sie den Trace mit der Umgebungsvariablen **XMS_TRACE_ON**.

Prozedur

- Führen Sie zum Herunterladen des Pakets `IBMXMSDotnetClient` über die Benutzerschnittstelle des Paketmanagers in Microsoft Visual Studio die folgenden Schritte aus:
  a) Klicken Sie mit der rechten Maustaste auf das .NET-Projekt und klicken Sie anschließend auf **Manage Nuget-Packages** (Nuget-Pakete verwalten).
  b) Klicken Sie auf die Registerkarte **Durchsuchen** und suchen Sie nach `"IBMXMSDotnetClient"`.
  c) Wählen Sie das Paket aus und klicken Sie auf **Install** (Installieren).

  Während der Installation stellt der Paketmanager Informationen zum Fortschritt in Form von Konsolenanweisungen bereit.
- Wählen Sie zum Herunterladen des Pakets `IBMXMSDotnetClient` über die Befehlszeile eine der folgenden Optionen aus:
  - Geben Sie bei Verwendung des NuGet-Paketmanagers den folgenden Befehl ein:

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    Während der Installation stellt der Paketmanager Informationen zum Fortschritt in Form von Konsolenanweisungen bereit. Sie können die Ausgabe an eine Protokolldatei weiterleiten.
  - Geben Sie bei Verwendung der .NET-Befehlszeilenschnittstelle den folgenden Befehl ein:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- Bei der Verwendung eines Web-Browsers laden Sie das Paket `IBMXMSDotnetClient` unter https://www.nuget.org/packages/IBMXMSDotnetClient herunter.

Zugehörige Konzepte

„Installieren von IBM MQ classes for .NET” auf Seite 584
IBM MQ classes for .NET, einschließlich Beispiele, werden mit IBM MQ unter Windows und Linux installiert.

Lizenzinformationen für IBM MQ Client for .NET

Zugehörige Tasks

„IBM MQ classes for .NET aus dem NuGet -Repository herunterladen” auf Seite 589
Die IBM MQ classes for .NET -Werte können aus dem NuGet -Repository heruntergeladen werden, sodass sie einfach von .NET -Entwicklern verarbeitet werden können.

Messaging-Serverumgebung einrichten

In den Themen dieses Abschnitts wird beschrieben, wie die Messaging-Serverumgebung eingerichtet werden muss, damit XMS-Anwendungen eine Verbindung zu einem Server herstellen können.

Informationen zu diesem Vorgang

Für Anwendungen, die eine Verbindung zu einem IBM MQ-Warteschlangenmanager herstellen sollen, ist der IBM MQ-Client (oder -Warteschlangenmanager für den Bindungsmodus) erforderlich.

Aktuell gibt es keine Voraussetzungen für Anwendungen, die eine Echtzeitverbindung zu einem Broker verwenden.

Sie müssen die Messaging-Serverumgebung einrichten, bevor Sie jegliche XMS-Anwendungen verwenden, einschließlich der mit XMS bereitgestellten Beispielanwendungen.

Dieser Abschnitt enthält die folgenden Themen:

- „Warteschlangenmanager und Broker für Anwendungen mit Verbindung zu einem IBM MQ-Warteschlangenmanager konfigurieren" auf Seite 658
- „Installieren von IBM MQ classes for XMS .NET" auf Seite 651
- „Broker für Anwendungen mit Echtzeitverbindung konfigurieren" auf Seite 659
- „Service Integration Bus für Anwendungen mit Verbindung zu WebSphere Application Server konfigurieren" auf Seite 660

Nachrichtenlistener in XMS .NET

Ein Nachrichtenlistener wird verwendet, um Nachrichten asynchron zu empfangen. Im Gegensatz zum `MessageConsumer.receive()` -Aufruf blockiert der Nachrichtenlistener den aufrufenden Thread nicht, sondern er stellt Nachrichten einer von der Anwendung angegebenen Callback-Methode zu, in der Regel der Methode `onMessage` .

Die Nachrichtenzustellung beginnt, sobald die Methode `Connection.Start()` aufgerufen wird. Die Nachrichtenübermittlung kann jederzeit mit den Methoden `Connection.Stop()` und `Connection.Start()` gestoppt bzw. fortgesetzt werden.

Sobald die Methode `Connection.Start()` aufgerufen wird, nachdem ein Nachrichtenlistener auf mindestens einen Konsumenten in einer Sitzung gesetzt wurde, wird diese Sitzung zu einer asynchronen Sitzung. Sobald eine Sitzung asynchron wird, ist es nicht mehr möglich, synchrone XMS .NET -Methoden aufzurufen. Zum Beispiel `MessageProducer.Send()`. Dies führt zu einer Ausnahmebedingung mit dem IBM MQ -Ursachencode MQRC_HCONN_ASYNC_ACTIVE (2500).

Synchrone Aufrufe in einer asynchronen Sitzung

`Session.Close` ist der einzige synchrone Aufruf, der in einer asynchronen Sitzung zulässig ist. Anwendungen können auch synchrone Aufrufe (außer `Session.Close`) mit der Callback-Methode des Nachrichtenlisteners durchführen, d. h. mit der Methode `onMessage` .

Abgesehen von diesen beiden Optionen müssen Sie die Verbindung mit der Methode `Connection.Stop()` stoppen, damit eine Anwendung einen synchronen Aufruf ausführen kann. Nach den Aufrufen müssen Sie die Verbindung mit der Methode `Connection.Start()` wieder aufnehmen. Dadurch wird die Nachrichtenzustellung erneut gestartet.

Wie viele asynchrone Nachrichtenkonsumenten können eine Sitzung haben?

Eine Sitzung kann mehrere asynchrone Nachrichtenkonsumenten haben. Eine Nachricht wird jedoch zu jedem beliebigen Zeitpunkt nur einem Konsumenten zugestellt. Dies bedeutet praktisch Folgendes: Wenn eine zweite Nachricht eintrifft, während XMS .NET die Methode `onMessage()` eines Konsumenten aufgerufen hat, um die erste Nachricht zuzustellen, wird die zweite Nachricht erst an einen Konsumenten in der Sitzung zugestellt, wenn die Methode `onMessage()` zurückgegeben wird.

Die zweite Nachricht wird einem Konsumenten in der Sitzung erst zugestellt, nachdem die Methode `onMessage()` zurückgegeben wurde. Dies liegt daran, dass eine Sitzung die Nachrichtenübermittlung an Konsumenten mit nur einem Thread verwaltet. Dies bedeutet, dass jeweils nur eine Nachricht zugestellt werden kann und der Konsument eine beliebige Nachricht sein kann.

Wenn eine Anwendung die gleichzeitige Nachrichtenzustellung erfordert, d. h., alle Konsumenten gleichzeitig Nachrichten empfangen müssen, muss die Anwendung mehrere Sitzungen erstellen und jeweils einen asynchronen Nachrichtenkonsumenten haben.

Die folgenden Beispiele zeigen diese Funktion deutlicher.

Im ersten Beispiel gibt es mehrere asynchrone Nachrichtenkonsumenten in einer Sitzung. Eine Sitzung S hat drei asynchrone Nachrichtenkonsumenten: AMC1, AMC2 und AMC3 , die Nachrichten von drei verschiedenen Zielen Q1, Q2 und Q3empfangen.

Da es nur eine Sitzung gibt S, gibt es nur einen Nachrichtenübermittlungsthread für die Zustellung von Nachrichten an Konsumenten AMC1, AMC2und AMC3. Wenn die Sitzung Nachrichten an AMC1übermittelt, warten die beiden anderen Konsumenten AMC2 und AMC3 , auch wenn Nachrichten in Q2 und Q3 für die Zustellung bereit sind.

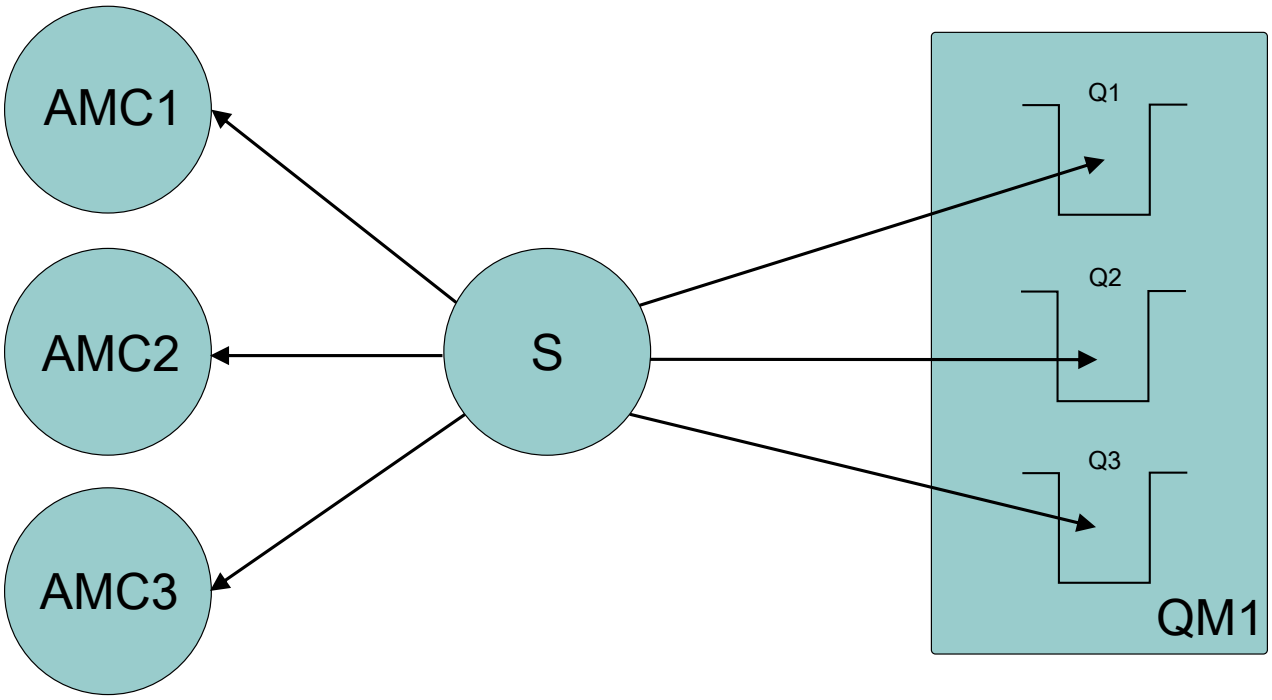


Abbildung 54. Eine Sitzung mit drei asynchronen Nachrichtenkonsumenten

Im zweiten Fall gibt es mehrere Sitzungen S1, S2und S3mit jeweils einem asynchronen Nachrichtenkonsumenten AMC1, AMC2und AMC3 . Da es für jede Sitzung einen Konsumenten gibt, werden Nachrichten gleichzeitig an Konsumenten zugestellt.

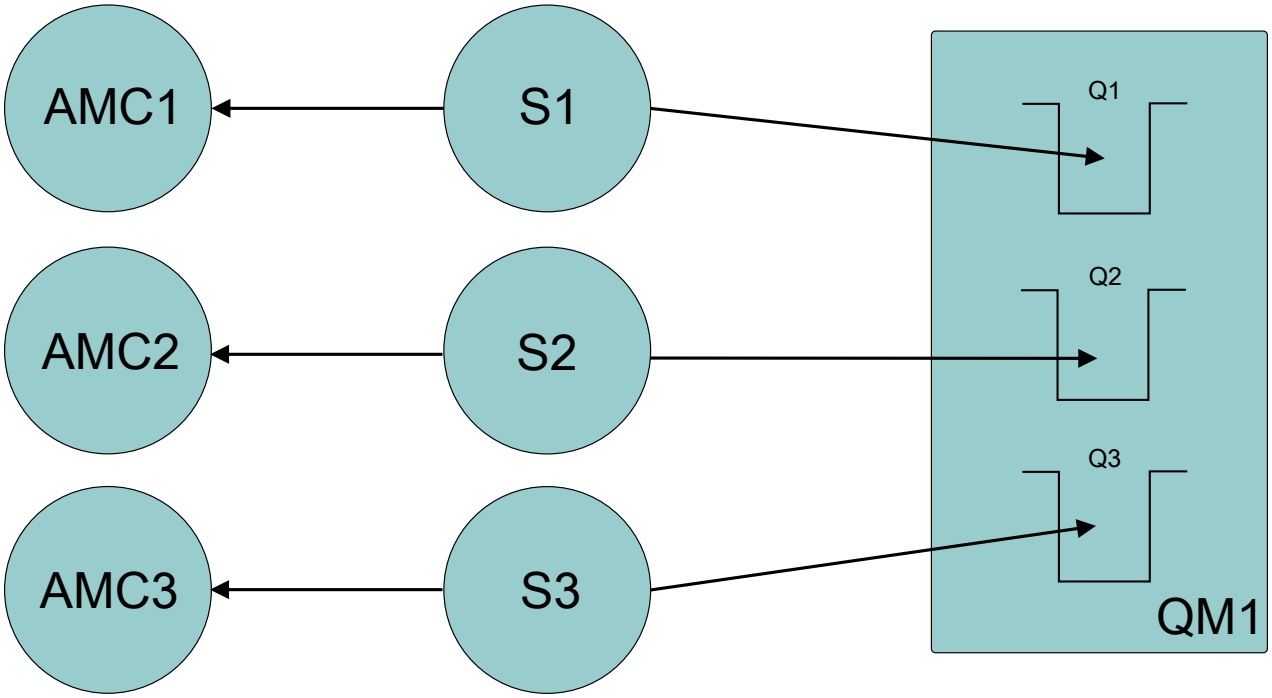


Abbildung 55. Mehrere Sitzungen mit jeweils einem asynchronen Nachrichtenkonsumenten

Dies zeigt, dass Sie mehrere Sitzungen benötigen, wenn die gleichzeitige Nachrichtenzustellung erforderlich ist.

Warteschlangenmanager und Broker für Anwendungen mit Verbindung zu einem IBM MQ-Warteschlangenmanager konfigurieren

Bei diesem Abschnitt wird vorausgesetzt, dass Sie IBM WebSphere MQ 7.0.1 oder höher verwenden. Bevor Sie eine Anwendung ausführen können, die eine Verbindung zu einem IBM MQ-Warteschlangenmanager verwendet, müssen Sie den entsprechenden Warteschlangenmanager konfigurieren. Bei einer Publish/Subscribe-Anwendung, für die Sie die warteschlangengesteuerte Publish/Subscribe-Schnittstelle verwenden, sind einige zusätzliche Konfigurationsschritte erforderlich.

Vorbereitende Schritte

XMS kann zusammen mit IBM Integration Bus oder WebSphere Message Broker 6.1 oder höher eingesetzt werden.

Bevor Sie mit dieser Task beginnen, müssen Sie folgende Schritte ausführen:

- Stellen Sie sicher, dass Ihre Anwendung Zugriff auf einen aktiven Warteschlangenmanager hat.
- Wenn Ihre Anwendung eine Publish/Subscribe-Anwendung ist, die die warteschlangengesteuerte Publish/Subscribe-Schnittstelle verwendet, stellen Sie sicher, dass im Warteschlangenmanager das Attribut **PSMODE** auf den Wert ENABLED festgelegt ist.
- Stellen Sie sicher, dass Ihre Anwendung eine Verbindungsfactory verwendet, deren Eigenschaften korrekt für eine Verbindung zu einem Warteschlangenmanager festgelegt sind. Wenn Ihre Anwendung eine Publish/Subscribe-Anwendung ist, stellen Sie sicher, dass die entsprechenden Eigenschaften der Verbindungsfactory für die Verwendung des Brokers festgelegt sind. Weitere Informationen zu den Eigenschaften einer Verbindungsfactory finden Sie im Abschnitt Eigenschaften eines ConnectionFactory-Objekts.

Informationen zu diesem Vorgang

Bei der Konfiguration eines Warteschlangenmanagers und eines Brokers zum Ausführen von XMS-Anwendungen ist die Vorgehensweise dieselbe wie bei der Konfiguration des Warteschlangenmanagers und der warteschlangengesteuerten Publish/Subscribe-Schnittstelle zum Ausführen von IBM MQ-JMS-Anwendungen. In der folgenden Vorgehensweise sind die erforderlichen Schritte zusammengefasst.

Vorgehensweise

1. Erstellen Sie im Warteschlangenmanager die für Ihre Anwendung erforderlichen Warteschlangen.

   Eine Übersicht über das Erstellen von Warteschlangen finden Sie im Abschnitt Warteschlangen definieren.

   Wenn Ihre Anwendung eine Publish/Subscribe-Anwendung ist und die warteschlangengesteuerte Publish/Subscribe-Schnittstelle verwendet, die Zugriff auf die Systemwarteschlangen der IBM MQ classes for JMS benötigt, erstellen Sie die Warteschlangen erst, nachdem Sie Schritt 4a ausgeführt haben.
2. Erteilen Sie der Benutzer-ID, die Ihrer Anwendung zugeordnet ist, die Berechtigung, eine Verbindung zum Warteschlangenmanager herzustellen und die entsprechende Berechtigung für den Zugriff auf die Warteschlangen.

   Eine Übersicht über die Berechtigungen finden Sie im Abschnitt Sicherheit. Wenn Ihre Anwendung die Verbindung zum Warteschlangenmanager im Clientmodus herstellt, finden Sie weitere Informationen hierzu auch im Abschnitt Clients und Server.
3. Wenn Ihre Anwendung die Verbindung zum Warteschlangenmanager im Clientmodus herstellt, stellen Sie sicher, dass im Warteschlangenmanager ein Serververbindungskanal definiert ist und ein Listener gestartet wird.

   Diesen Schritt brauchen Sie nicht für jede Anwendung auszuführen, die eine Verbindung zum Warteschlangenmanager herstellt. Für die Unterstützung aller Anwendungen mit Verbindungen im Clientmodus ist nur eine Serververbindungskanaldefinition und ein Listener erforderlich.

4. Wenn Ihre Anwendung eine Publish/Subscribe-Anwendung ist und die warteschlangengesteuerte Publish/Subscribe-Schnittstelle verwendet, führen Sie folgende Schritte aus.
   a) Erstellen Sie im Warteschlangenmanager die Systemwarteschlangen für die IBM MQ classes for JMS, indem Sie das Script der MQSC-Befehle ausführen, die mit IBM MQ bereitgestellt werden. Stellen Sie sicher, dass die Benutzer-ID, die IBM Integration Bus oder WebSphere Message Broker zugeordnet ist, die Berechtigung für den Zugriff auf die Warteschlangen besitzt.

      Weitere Informationen zur Speicherposition des Scripts und dessen Verwendung finden Sie im Abschnitt IBM MQ classes for Java verwenden.

      Führen Sie diesen Schritte für jeden Warteschlangenmanager nur einmal aus. Für die Unterstützung aller XMS-Anwendungen und aller Anwendungen für die IBM MQ classes for JMS mit Verbindungen zum Warteschlangenmanager kann dieselbe Gruppe von Systemwarteschlangen für die IBM MQ classes for JMS verwendet werden.
   b) Erteilen Sie der Benutzer-ID, die Ihrer Anwendung zugeordnet ist, die Berechtigung für den Zugriff auf die Systemwarteschlangen für die IBM MQ classes for JMS.

      Weitere Informationen zu den Berechtigungen, die die Benutzer-ID benötigt, finden Sie im Abschnitt IBM MQ classes for JMS verwenden.
   c) Wenn Sie einen Broker für IBM Integration Bus oder WebSphere Message Broker verwenden, erstellen und implementieren Sie einen Nachrichtenfluss für die Warteschlangen, über den die Anwendungen Nachrichten senden können, die veröffentlicht werden sollen.

      Der grundlegende Nachrichtenfluss umfasst den Nachrichtenverarbeitungsknoten 'MQInput' zum Lesen der veröffentlichten Nachrichten sowie den Nachrichtenverarbeitungsknoten 'Publication' zum Veröffentlichen der Nachrichten.

      Weitere Informationen zum Erstellen und Implementieren von Nachrichtenflüssen finden Sie auf der Webseite mit der IBM Integration Bus -Produktdokumentationsbibliothek in der Produktdokumentation zu IBM Integration Bus oder WebSphere Message Broker.

      Wenn bereits zuvor ein entsprechender Nachrichtenfluss für den Broker vorhanden ist, können Sie diesen Schritt überspringen.

Ergebnisse
Sie können Ihre Anwendung nun starten.

Broker für Anwendungen mit Echtzeitverbindung konfigurieren

Bevor Sie eine Anwendung ausführen können, die eine Echtzeitverbindung zu einem Broker verwendet, müssen Sie den entsprechenden Broker konfigurieren.

Vorbereitende Schritte

Bevor Sie mit dieser Task beginnen, müssen Sie folgende Schritte ausführen:

- Stellen Sie sicher, dass Ihre Anwendung Zugriff auf einen aktiven Broker hat.
- Stellen Sie sicher, dass Ihre Anwendung eine Verbindungsfactory verwendet, deren Eigenschaften korrekt für eine Echtzeitverbindung zu einem Broker festgelegt sind. Weitere Informationen zu den Eigenschaften einer Verbindungsfactory finden Sie im Abschnitt Eigenschaften eines ConnectionFactory-Objekts.

Informationen zu diesem Vorgang
Bei der Konfiguration eines Brokers zum Ausführen von XMS-Anwendungen ist die Vorgehensweise dieselbe wie bei der Konfiguration eines Brokers für das Ausführen von Anwendungen für die IBM MQ classes for JMS. In der folgenden Vorgehensweise sind die erforderlichen Schritte zusammengefasst:

Vorgehensweise

1. Erstellen und implementieren Sie einen Nachrichtenfluss, um die Nachrichten an dem TCP/IP-Port, der von einem Broker überwacht wird, zu lesen und zu veröffentlichen.

   Führen Sie dazu eine der folgenden Vorgehensweisen aus:

   - Erstellen Sie einen Nachrichtenfluss, der den Nachrichtenverarbeitungsknoten **Real-timeOptimizedFlow** enthält.
   - Erstellen Sie einen Nachrichtenfluss, der die beiden Nachrichtenverarbeitungsknoten **Real-time Input** und 'Publication' enthält.

   Konfigurieren Sie den Knoten **Real-timeOptimizedFlow** bzw. **Real-timeInput** für die Überwachung des Ports, der für die Echtzeitverbindung verwendet wird. In XMS wird standardmäßig Portnummer 1506 für Echtzeitverbindungen verwendet.

   Wenn bereits zuvor ein entsprechender Nachrichtenfluss für den Broker vorhanden ist, können Sie diesen Schritt überspringen.
2. Wenn es erforderlich ist, dass Ihrer Anwendung Nachrichten mithilfe der IBM MQ classes for JMS übermittelt werden, konfigurieren Sie die Multicastunterstützung für den Broker. Konfigurieren Sie die Themen, die multicastfähig sein müssen, und geben Sie dabei für die Themen, die eine zuverlässige Multicastkommunikation erfordern, eine zuverlässige Servicequalität an.
3. Wenn Ihre Anwendung beim Herstellen der Verbindung zum Broker einen Benutzernamen und ein Kennwort angibt, die der Broker zum Authentifizieren der Anwendung nutzen soll, konfigurieren Sie für den Benutzernamensserver und den Broker eine einfache Kennwortauthentifizierung, wie sie in ähnlicher Weise auch bei Telnet verwendet wird.

Ergebnisse

Sie können Ihre Anwendung nun starten.

Service Integration Bus für Anwendungen mit Verbindung zu WebSphere Application Server konfigurieren

Damit Sie eine Anwendung ausführen können, die eine Verbindung zu einer WebSphere Application Server service integration technologies Service Integration Bus-Instanz herstellt, müssen Sie die Serviceintegration so konfigurieren, dass die Service Integration Bus-Instanz für die Ausführung von JMS-Anwendungen konfiguriert ist, die den Standard-Messaging-Provider verwenden.

Vorbereitende Schritte

Bevor Sie mit dieser Task beginnen, müssen Sie folgende Schritte ausführen:

- Stellen Sie sicher, dass ein Messaging-Bus erstellt und Ihr Server diesem Bus als Busmember hinzugefügt wird.
- Stellen Sie sicher, dass Ihre Anwendung Zugriff auf einen Service Integration Bus hat, der mindestens eine aktive Messaging-Engine enthält.
- Wenn HTTP-Operationen erforderlich sind, muss eine HTTP-Messaging-Engine mit eingehendem Transportkanal definiert werden. Während der Serverinstallation werden standardmäßig SSL- und TCP-Kanäle definiert.
- Stellen Sie sicher, dass Ihre Anwendung eine Verbindungsfactory verwendet, deren Eigenschaften korrekt für eine Verbindung zu einem Service Integration Bus mithilfe eines Bootstrap-Servers festgelegt sind. Folgende Informationen sind mindestens erforderlich:
  - Der Providerendpunkt, der die Position und das Protokoll angibt, das bei der (über den Bootstrap-Server ausgeführten) Verhandlung einer Verbindung zum Messaging-Server verwendet werden soll. Im einfachsten Fall, d. h. bei einem mit Standardeinstellungen installierten Server, kann der Hostname des Servers als Providerendpunkt festgelegt werden.
  - Der Name des Bus, über den die Nachrichten gesendet werden.

Weitere Informationen zu den Eigenschaften einer Verbindungsfactory finden Sie im Abschnitt Eigenschaften eines ConnectionFactory-Objekts.

Informationen zu diesem Vorgang

Sie müssen alle Warteschlangen- oder Themenbereiche definieren, die Sie benötigen. Während der Serverinstallation wird standardmäßig der Themenbereich 'Default.Topic.Space' definiert, doch wenn Sie weitere Themenbereiche benötigen, müssen Sie diese selbst erstellen. Innerhalb eines Themenbereichs brauchen Sie vorab keine individuellen Themen zu definieren, denn der Server instanziiert einzelne Themen dynamisch nach Bedarf.

In der folgenden Vorgehensweise sind die erforderlichen Schritte zusammengefasst.

Vorgehensweise

1. Erstellen Sie die Warteschlangen, die Ihre Anwendung für das Punkt-zu-Punkt-Messaging benötigt.
2. Erstellen Sie alle zusätzlich erforderlichen Themenbereiche, die Ihre Anwendung für das Publish/Subscribe-Messaging benötigt.

Ergebnisse

Sie können Ihre Anwendung nun starten.

XMS-Beispielanwendungen verwenden

Die XMS .NET-Beispielanwendungen bieten einen Überblick über die häufig verwendeten Funktionen der jeweiligen APIs. Mithilfe der Beispielanwendungen können Sie Ihre Installation und Messaging-Server-Konfiguration überprüfen sowie Ihre eigenen Anwendungen erstellen.

Informationen zu diesem Vorgang

Wenn Sie beim Erstellen Ihrer eigenen Anwendungen Unterstützung benötigen, können Sie die Beispielanwendungen als Ausgangspunkt dafür verwenden. Für jede Anwendung wird eine Quellenversion und ein kompilierte Version bereitgestellt. Prüfen Sie den Quellencode einer Beispielanwendung, um die wichtigsten Schritte zum Erstellen aller für Ihre Anwendung erforderlichen Objekte (ein ConnectionFactory-, Connection-, Session-, Destination- und ein Producer- und/oder ein Consumer-Objekt) zu ermitteln, und um spezielle Eigenschaften festzulegen, mit denen die Funktionsweise Ihrer Anwendung gesteuert wird. Weitere Informationen finden Sie unter „Anwendungen der XMS .NET schreiben” auf Seite 664. Die Beispielanwendungen können sich in zukünftigen XMS-Releases ändern.

In der folgenden Tabelle sind die Gruppen der Beispielanwendungen (je eine für jede API) aufgeführt, die mit XMS bereitgestellt werden.

Tabelle 81. Beispielanwendungen für XMS .NET

Name der Beispielanwendung	Beschreibung
SampleConsumerCS	Eine Nachrichtenkonsumentenanwendung, die Nachrichten aus einer Warteschlange abruft oder ein Thema abonniert.
SampleProducerCS	Eine Nachrichtenproduzentenanwendung, die Nachrichten in einer Warteschlange oder in einem Thema erstellt.
SampleConfigCS	Eine Konfigurationsanwendung, mit dem Sie ein dateibasiertes Repository für verwaltete Objekte erstellen können. Die Anwendung enthält eine Verbindungsfactory und ein Ziel für Ihre speziellen Verbindungseinstellungen. Dieses Repository für verwaltete Objekte kann dann zusammen mit jeder als Beispiel bereitgestellten Konsumenten- und Produzentenanwendung verwendet werden.

Die Beispielanwendungen, die dieselben Funktionen in verschiedenen APIs unterstützen, haben eine unterschiedliche Syntax.

- Die Nachrichtenkonsumenten- und Nachrichtenproduzenten-Beispielanwendungen unterstützen beide die folgenden Funktionen:
  - Verbindungen zu IBM MQ, zu IBM Integration Bus (mithilfe einer Echtzeitverbindung zu einem Broker) und zu einem WebSphere Application Server service integration bus
  - Suchvorgänge in einem Repository für verwaltete Objekte mithilfe der Ausgangskontextschnittstelle
  - Verbindungen zu Warteschlangen (IBM MQ und WebSphere Application Server service integration bus) und zu Themen (IBM MQ, Echtzeitverbindung zu einem Broker und WebSphere Application Server service integration bus)
  - Basis-, Byte-, Zuordnungs-, Objekt-, Datenstrom- und Textnachrichten
- Die Nachrichtenkonsumenten-Beispielanwendung unterstützt den synchronen und asynchronen Empfangsmodus sowie SQL-Selektoranweisungen.
- Die Nachrichtenproduzenten-Beispielanwendung unterstützt den persistenten und nicht persistenten Übermittlungsmodus.

Die Beispielanwendungen können in einem der folgenden beiden Modi ausgeführt werden:

Einfacher Modus
Sie können die Beispielanwendungen mit minimalen Benutzereingaben ausführen.

Erweiterter Modus
In diesem Fall können Sie die Funktionsweise der Beispielanwendungen exakter für Ihren Bedarf anpassen.

Alle Beispielanwendungen sind untereinander kompatibel und können deshalb über verschiedene Programmiersprachen hinweg ausgeführt werden.

Windows IBM MQ unterstützt .NET Core für XMS .NET -Anwendungen in Windows -Umgebungen. IBM MQ classes for .NET Standard, einschließlich Beispiele, wird standardmäßig im Rahmen der IBM MQ-Standardinstallation installiert.

Linux IBM MQ unterstützt auch .NET Core für Anwendungen in Linux -Umgebungen.

Die Beispielanwendungen für XMS .NET werden im Verzeichnis `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms` installiert.

Weitere Informationen finden Sie unter „Installieren von IBM MQ classes for XMS .NET” auf Seite 651.

.NET-Beispielanwendungen ausführen

Beim Ausführen der .NET-Beispielanwendungen haben Sie folgende Möglichkeiten: entweder interaktiv im einfachen oder im erweiterten Modus oder nicht interaktiv, indem Sie automatisch generierte oder angepasste Antwortdateien verwenden.

Vorbereitende Schritte

Bevor Sie die bereitgestellten Beispielanwendungen ausführen können, müssen Sie zunächst die Messaging-Serverumgebung einrichten, damit die Anwendungen eine Verbindung zu einem Server herstellen können. Siehe „Messaging-Serverumgebung einrichten” auf Seite 655.

Vorgehensweise

Führen Sie folgende Schritte aus, um eine .NET-Beispielanwendung auszuführen:

Tipp: Wenn Sie eine Beispielanwendung ausführen, geben Sie Folgendes ein: jederzeit, um Hilfe zu erhalten, was als Nächstes zu tun ist.

1. Wählen Sie den Modus aus, in dem Sie die Beispielanwendung ausführen möchten.
   Geben Sie entweder `Advanced` (Erweitert) oder `Simple` (Einfach) ein.

2. Beantworten Sie die Fragen.

   Drücken Sie die Eingabetaste, um den jeweiligen Standardwert auszuwählen, der in eckigen Klammern am Ende der Frage angegeben ist. Wenn Sie einen anderen Wert auswählen möchten, geben Sie den entsprechenden Wert ein und drücken Sie dann die Eingabetaste.

   Es folgt eine Beispielfrage:

   ```
   Enter connection type [wpm]:
   ```

   In diesem Fall lautet der Standardwert `wpm` (die entspricht einer Verbindung zu einem WebSphere Application Server service integration bus).

Ergebnisse

Wenn Sie die Beispielanwendungen ausführen, werden automatisch Antwortdateien im aktuellen Arbeitsverzeichnis generiert. Antwortdateinamen besitzen das Format *`connection_type-sample_type`*`.rsp`; z. B. `wpm-producer.rsp`. Bei Bedarf können Sie die generierte Antwortdatei nutzen, um die Beispielanwendung erneut mit denselben Optionen auszuführen, damit Sie die zuvor gewählten Einstellungen nicht erneut eingeben müssen.

Zugehörige Tasks

.NET-Beispielanwendungen erstellen
Wenn Sie eine .NET-Beispielanwendung erstellen, wird eine ausführbare Version Ihres gewählten Beispiels erstellt.

Eigene Anwendung erstellen
Sie können Ihre eigenen Anwendungen auf die gleiche Weise erstellen, wie Sie die Beispielanwendungen erstellen.

.NET-Beispielanwendungen erstellen

Wenn Sie eine .NET-Beispielanwendung erstellen, wird eine ausführbare Version Ihres gewählten Beispiels erstellt.

Vorbereitende Schritte

Installieren Sie den entsprechenden Compiler. Bei dieser Task wird vorausgesetzt, dass Sie Microsoft Visual Studio 2012 installiert haben und mit dessen Verwendung vertraut sind.

Vorgehensweise

Führen Sie folgende Schritte aus, um eine .NET-Beispielanwendung auszuführen:

1. Klicken Sie auf die Lösungsdatei `Samples.sln`, die mit den .NET-Beispielen bereitgestellt wird.
2. Klicken Sie mit der rechten Maustaste im 'Solution Explorer' auf die `Samples` (Beispiele) und wählen Sie **Build Solution** (Lösung erstellen) aus.

Ergebnisse

Im entsprechenden Unterordner des Beispiels, der je nach der von Ihnen gewählten Konfiguration entweder `bin/Debug` oder `bin/Release` heißt, wird ein ausführbares Programm erstellt. Dieses Programm hat denselben Namen wie der Ordner und das Suffix CS. Wenn Sie beispielsweise die C#-Version der Beispielanwendung für einen Nachrichtenproduzenten erstellen, wird im Ordner `SampleProducer` das ausführbare Programm `SampleProducerCS.exe` erstellt.

Zugehörige Tasks

.NET-Beispielanwendungen ausführen
Beim Ausführen der .NET-Beispielanwendungen haben Sie folgende Möglichkeiten: entweder interaktiv im einfachen oder im erweiterten Modus oder nicht interaktiv, indem Sie automatisch generierte oder angepasste Antwortdateien verwenden.

Eigene Anwendung erstellen

Sie können Ihre eigenen Anwendungen auf die gleiche Weise erstellen, wie Sie die Beispielanwendungen erstellen.

„Eigene Anwendung erstellen” auf Seite 664
Sie können Ihre eigenen Anwendungen auf die gleiche Weise erstellen, wie Sie die Beispielanwendungen erstellen.

Eigene Anwendung erstellen

Sie können Ihre eigenen Anwendungen auf die gleiche Weise erstellen, wie Sie die Beispielanwendungen erstellen.

Vorbereitende Schritte

Installieren Sie den entsprechenden Compiler. Bei dieser Task wird vorausgesetzt, dass Sie Microsoft Visual Studio 2012 installiert haben und mit dessen Verwendung vertraut sind.

Prozedur

- Erstellen Sie Ihre .NET-Anwendung wie im Abschnitt „.NET-Beispielanwendungen erstellen” auf Seite 663 beschrieben.

  Weitere Anleitungen zum Erstellen Ihrer eigenen Anwendungen finden Sie in den Makefiles, die für die jeweilige Beispielanwendung bereitgestellt werden.

  Tipp: Als Unterstützung bei der Problemdiagnose im Falle eines Fehlers kann es hilfreich sein, die Anwendungen mit eingegebenen Symbolen zu kompilieren.

Zugehörige Tasks

.NET-Beispielanwendungen ausführen
Beim Ausführen der .NET-Beispielanwendungen haben Sie folgende Möglichkeiten: entweder interaktiv im einfachen oder im erweiterten Modus oder nicht interaktiv, indem Sie automatisch generierte oder angepasste Antwortdateien verwenden.

.NET-Beispielanwendungen erstellen
Wenn Sie eine .NET-Beispielanwendung erstellen, wird eine ausführbare Version Ihres gewählten Beispiels erstellt.

Anwendungen der XMS .NET schreiben

Dieser Abschnitt enthält Informationen, die Sie beim Schreiben von XMS .NET -Anwendungen unterstützen, einschließlich Informationen zu Eigenschaften, Datentypen und Fehlerbehandlung.

Vorbereitende Schritte

V 9.4.0 Deprecated V 9.4.0 Ab IBM MQ 9.4.0sind in IBM MQ classes for XMS .NETdie Methoden WriteObject(), ReadObject(), CreateObjectMessage () und die Klassen ObjectMessage und XmsObjectMessageImpl , die für die Serialisierung und Deserialisierung von Daten verwendet werden, veraltet.

V 9.4.0 V 9.4.0 Removed Die mit.NET Standard 2.0erstellte XMS .NET -Clientbibliothek, die in IBM MQ 9.3.1veraltet war, wurde aus dem Produkt unter IBM MQ 9.4.0entfernt.

Ab IBM MQ 9.2.0wurde die Anzahl der XMS .NET -Bibliotheken für dynamische Links auf insgesamt fünf deutlich reduziert. Die fünf Dynamic Link Librarys sind:

- `IBM.XMS.dll` - enthält alle landessprachlichen Nachrichten
- `IBM.XMS.Comms.RMM.dll`
- Drei Policy Dynamic Link Librarys:
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`

– `policy.9.1.IBM.XMS.dll`

In XMS .NET werden alle Zeichenfolgen mithilfe der nativen .NET-Zeichenfolge übergeben. Da diese Zeichenfolge eine feste Codierung hat, sind zur Interpretation keine weiteren Informationen erforderlich. Daher ist die Eigenschaft XMSC_CLIENT_CCSID für XMS .NET -Anwendungen nicht erforderlich.

Informationen zu diesem Vorgang

Dieser Abschnitt enthält die folgenden Themen:

Verwaltete und nicht verwaltete Operationen in .NET

Verwalteter Code wird ausschließlich in der Umgebung der .NET Common Language Runtime ausgeführt und ist vollständig von den in dieser Laufzeit bereitgestellten Services abhängig. Eine Anwendung wird als 'nicht verwaltet' klassifiziert, wenn ein Teil der Anwendung oder Services, die von der Anwendung aufgerufen werden, außerhalb der Umgebung der .NET Common Language Runtime ausgeführt werden.

Bestimmte erweiterte Funktionen können in der verwalteten .NET-Umgebung aktuell nicht unterstützt werden.

Wenn Ihre Anwendung Funktionen benötigt, die in der vollständig verwalteten Umgebung aktuell nicht unterstützt werden, können Sie Ihre Anwendung für die Verwendung der nicht verwalteten Umgebung anpassen, ohne wesentliche Änderungen an der Anwendung vornehmen zu müssen. Wenn Sie sich für diese Vorgehensweise entscheiden, müssen Sie jedoch beachten, dass der XMS-Stack nicht verwalteten Code verwendet.

Verbindungen zu einem IBM MQ-Warteschlangenmanager

Verwaltete Verbindungen zu 'WMQ_CM_CLIENT' unterstützten weder die TCP-fremde Kommunikation noch die Kanalkomprimierung. Diese Verbindungen werden jedoch möglicherweise von eine nicht verwalteten Verbindung (WMQ_CM_CLIENT_UNMANAGED) unterstützt. Weitere Informationen finden Sie unter „.NET-Anwendungen entwickeln” auf Seite 582.

Wenn Sie eine Verbindungsfactory basierend auf einem verwalteten Objekt in einer nicht verwalteten Umgebung erstellen, müssen Sie den Wert für den Verbindungsmodus manuell in 'XMSC_WMQ_CM_CLIENT_UNMANAGED' ändern.

Verbindungen zur Messaging-Engine eines WebSphere Application Server Service Integration Bus

Verbindungen zur Messaging-Engine eines WebSphere Application Server Service Integration Bus, für die das SSL-Protokoll (einschließlich HTTPS) verwendet werden muss, werden aktuell nicht als verwalteter Code unterstützt.

Windows Projektvorlage von IBM MQ XMS .NET verwenden

Beim IBM MQ XMS .NET-Client haben Sie die Möglichkeit, sich durch Verwendung einer Projektvorlage beim Entwickeln Ihrer XMS-Anwendungen für .NET Core unterstützen zu lassen.

Vorbereitende Schritte

Sie müssen über Microsoft Visual Studio 2017 oder höher und .NET Core 2.1 auf Ihrem System verfügen.

Sie müssen die XMS .NET-Vorlage aus dem Verzeichnis

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip

in das Verzeichnis

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

kopieren. Dabei gilt Folgendes:

- *&MQ-INSTALLATIONSSTAMMVERZEICHNIS* ist das Stammverzeichnis Ihrer Installation.
- *&BENUTZERAUSGANGSVERZEIHNIS* ist Ihr Ausgangsverzeichnis.

Sie müssen Microsoft Visual Studio stoppen und erneut starten, um die Vorlage zu übernehmen.

Informationen zu diesem Vorgang

Die XMS .NET-Projektvorlage enthält allgemeinen Code, den Sie zur Unterstützung bei der Entwicklung Ihrer Anwendungen verwenden können. Mit dem integrierten Code können Sie eine Verbindung zum IBM MQ-Warteschlangenmanager herstellen und eine Operation zum Einreihen oder Abrufen ausführen, indem Sie einfach die Eigenschaften in dem integrierten Code ändern.

Vorgehensweise

1. Öffnen Sie Microsoft Visual Studio.
2. Klicken Sie auf **File** (Datei), **New** (Neu) und dann **Project** (Projekt).
3. Wählen Sie im Fenster *Neues Projekt erstellen* `IBM XMS .NET Client App (.NET Core)` aus und klicken Sie auf **Weiter**.
4. Ändern Sie im Fenster *Configure your new project* (Neues Projekt konfigurieren) gegebenenfalls den Projektnamen (*Project name*) und klicken Sie auf **Create** (Erstellen), um das XMS .NET-Projekt zu erstellen.

   Die Datei `XMSDotnetApp.cs` wird zusammen mit der Projektdatei erstellt. Diese Datei enthält den Code für die Verbindung zum Warteschlangenmanager und führt eine Sende- und Empfangsoperation aus.

   Die Verbindungseigenschaften sind auf Standardwerte gesetzt:

   - WMQ_CONNECTION_NAME_LIST ist auf *localhost(1414)* gesetzt.

- XMSC.WMQ_CHANNEL ist auf *DOTNET.SVRCONN* gesetzt.

Die Warteschlange ist auf *Q1* gesetzt. Sie können diese Eigenschaften entsprechend ändern.

5. Kompilieren Sie die Anwendung und führen Sie sie aus.

Zugehörige Konzepte
Komponenten und Funktionen von IBM MQ
.NET-Anwendungslaufzeit - nur Windows

Threading-Modell

Für die Verwendung von XMS-Objekten in einer Multithread-Anwendung gibt es allgemeine Regeln.

- Nur folgende Objekte können gleichzeitig in unterschiedlichen Threads verwendet werden:
  - ConnectionFactory
  - Verbindung
  - ConnectionMetaData
  - Destination
- Ein Session-Objekt kann zu jedem Zeitpunkt nur in einem einzigen Thread verwendet werden.

Ausnahmen von diesen Regeln sind durch Einträge mit der Bezeichnung "Threadkontext" in den Schnittstellendefinitionen der Methoden in IBM Message Service Client for .NET Referencegekennzeichnet.

Eigenschaften in XMS.NET

Eine .NET-Anwendung verwendet die Methoden in der Schnittstelle 'PropertyContext', um Eigenschaften von Objekten abzurufen und festzulegen. Die Handhabung nicht vorhandener Eigenschaften in XMS .NET ist im Wesentlichen mit der JMS-Spezifikation sowie mit den C- und C++-Implementierungen für XMS konsistent.

XMS .NET -Eigenschaften und ihre Werte

Die Schnittstelle PropertyContext enthält Methoden zum Abrufen und Festlegen von Eigenschaften. Diese Methoden werden direkt oder indirekt von folgenden Klassen übernommen:

- BytesMessage
- Verbindung
- ConnectionFactory
- ConnectionMetaData
- Destination
- MapMessage
- Nachricht
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- Sitzungen
- StreamMessage
- TextMessage

Wenn eine Anwendung den Wert einer Eigenschaft festlegt, ersetzt der neue Wert immer den vorherigen Wert, der der Eigenschaft zugewiesen war. Weitere Informationen zu XMS-Eigenschaften finden Sie im Abschnitt Eigenschaften von XMS-Objekten.

Um die Verwendung zu vereinfachen, sind XMS-Eigenschaftsnamen und -werte in XMS als öffentliche Konstanten in einem 'XMSC' genannten Konstrukt vordefiniert. Die Namen dieser Konstanten haben folgendes Format: XMSC.*Konstante*. Beispiele: 'XMSC.USERID' (Konstante eines Eigenschaftsnamens) und 'XMSC.DELIVERY_AS_APP' (Konstante eines Eigenschaftswerts).

Außerdem haben Sie mithilfe des Konstrukts 'IBM.XMS.MQC' Zugriff auf IBM MQ-Konstanten. Wenn der Namensbereich 'IBM.XMS' bereits importiert wurde, haben Sie Zugriff auf die Werte für diese Eigenschaften im Format 'MQC.*Konstante*'. Beispiel: 'MQC.MQRO_COA_WITH_FULL_DATA'.

Sie verfügen über eine Hybridanwendung, die sowohl XMS .NET -als auch IBM MQ -Klassen für .NET verwendet und beide IBM.XMS und IBM.WMQ -Namensbereiche müssen Sie den MQC-Strukturnamensbereich vollständig qualifizieren, um sicherzustellen, dass jedes Vorkommen eindeutig ist.

Anmerkung: Einige erweiterte Funktionen werden in der verwalteten .NET-Umgebung aktuell nicht unterstützt. Weitere Informationen finden Sie unter „Verwaltete und nicht verwaltete Operationen in .NET” auf Seite 665.

Handhabung nicht vorhandener Eigenschaften in XMS .NET

In JMS kann der Zugriff auf eine nicht vorhandene Eigenschaft zu einer Java-Systemausnahme führen, wenn eine Methode versucht, den nicht vorhandenen Wert (Null-Wert) in den erforderlichen Datentyp umzuwandeln. Wenn eine Eigenschaft nicht vorhanden ist, treten folgende Ausnahmebedingungen auf:

- 'getStringProperty' und 'getObjectProperty' geben 'null' zurück
- 'getBooleanProperty' gibt 'false' zurück, weil 'Boolean.valueOf(null)' den Wert 'false' zurückgibt
- 'getIntProperty' usw. lösen die Ausnahme 'java.lang.NumberFormatException' aus, weil 'Integer.valueOf(null)' diese Ausnahme auslöst

Wenn eine Eigenschaft in XMS .NETnicht vorhanden ist, treten die folgenden Ausnahmen auf:

- 'GetStringProperty' und 'GetObjectProperty' (sowie 'GetBytesProperty') geben 'null' zurück (dieses Verhalten ist identisch mit Java)
- 'GetBooleanProperty' löst die Ausnahme 'System.NullReferenceException' aus
- 'GetIntProperty' usw. lösen die Ausnahme 'System.NullReferenceException' aus

Diese Implemntierung unterscheidet sich zwar von Java, ist jedoch im Wesentlichen mit der JMS-Spezifikation sowie mit den C- und C++-Schnittstellen für XMS konsitent. Wie die Java-Implementierung gibt XMS .NET alle Ausnahmen von Aufruf System.Convert an das aufrufende Programm weiter. Anders als Java löst XMS jedoch explizit die Ausnahme 'NullReferenceException' aus, anstatt lediglich das native .NET-Framework zu verwenden und den Wert 'null' an die Konvertierungsroutinen zu übergeben. Wenn Ihre Anwendung eine Eigenschaft auf eine Zeichenfolge wie beispielsweise "abc" setzt und die Methode 'GetIntProperty' aufruft, wird die von 'Convert.ToInt32("abc")' ausgelöste Ausnahme 'System.FormatException' an das aufrufende Programm weitergegeben, wobei dieses Verhalten mit Java konsistent ist. Die Ausnahme 'MessageFormatException' wird nur ausgelöst, wenn die für 'setProperty' und 'getProperty' verwendeten Datentypen inkompatibel sind. Auch dieses Verhalten ist mit Java konsistent.

ConnectionFactory- und Connection-Objekte

Ein Verbindungsfactoryobjekt (ConnectionFactory) stellt eine Vorlage bereit, die von einer Anwendung zum Erstellen eines Verbindungsobjekts (Connection) verwendet wird. Das Connection-Objekt verwendet die Anwendung wiederum zum Erstellen eines Sitzungsobjekts (Session).

Für .NET verwendet eine XMS-Anwendung zunächst ein XMSFactoryFactory-Objekt, um einen Verweis auf ein ConnectionFactory-Objekt abzurufen, das für den erforderlichen Protokolltyp geeignet ist. Mit diesem ConnectionFactory-Objekt können dann nur Verbindungen für diesen Protokolltyp erstellt werden.

Eine XMS-Anwendung kann mehrere Verbindungen erstellen, und eine Multithread-Anwendung kann ein einzelnes Connection-Objekt in mehreren Threads gleichzeitig verwenden. Ein Connection-Objekt enthält die Kommunikationsverbindung zwischen einer Anwendung und einem Messaging-Server.

Eine Verbindung dient mehreren Zwecken:

- Wenn eine Anwendung eine Verbindung erstellt, kann die Anwendung authentifiziert werden.
- Eine Anwendung kann einer Verbindung eine eindeutige Client-ID zuordnen. Mithilfe der Client-ID werden permanente Subskriptionen in der Publish/Subscribe-Domäne unterstützt. Die Client-ID kann auf zwei Arten festgelegt werden:

  Die bevorzugte Art, um einer Verbindung eine Client-ID zuzuweisen, besteht darin, mithilfe von Eigenschaften ein clientspezifisches ConnectionFactory-Objekt zu konfigurieren und es der Verbindung, die mit diesem Objekt erstellt wird, transparent zuzuweisen.

  Als Alternative dazu kann beim Zuweisen einer Client-ID auch ein im Connection-Objekt festgelegter providerspezifischer Wert verwendet werden. Die administrativ konfigurierte ID wird durch diesen Wert jedoch nicht überschrieben. Der Wert wird für den Fall bereitgestellt, dass keine adminstrativ erstellte ID vorhanden ist. Bei dem Versuch, eine administrativ erstellte ID durch einen providerspezifischen Wert zu überschreiben, wird eine Ausnahmebedingung ausgelöst. Wenn eine Anwendung explizit eine ID festlegt, muss sie dies unmittelbar nach dem Erstellen der Verbindung tun und bevor eine andere Aktion über die Verbindung ausgeführt wird, denn andernfalls wird eine Ausnahmebedingung ausgelöst.

Eine XMS-Anwendung erstellt normalerweise eine Verbindung, mindestens eine Sitzung und eine Reihe von Nachrichtenproduzenten und -konsumenten.

Das Erstellen einer Kommunikationsverbindung ist ein relativ kostenintensiver Vorgang, denn er benötigt eine große Menge an Systemressourcen und kann darüber hinaus eventuell die Authentifizierung der Anwendung erfordern.

Gestarteter und gestoppter Verbindungsmodus

Eine Verbindung kann sowohl im gestarteten als auch im gestoppten Modus betrieben werden.

Wenn eine Anwendung eine Verbindung erstellt, befindet sich die Verbindung im gestoppten Modus. Im gestoppten Verbindungsmodus kann die Anwendung Sitzungen initialisieren sowie Nachrichten synchron oder asynchron senden, jedoch nicht empfangen.

Zum Starten einer Verbindung ruft die Anwendung die Methode `Start Connection` (Verbindung starten) auf. Im gestarteten Verbindungsmodus kann die Anwendung Nachrichten senden und empfangen. Außerdem kann die Anwendung die Verbindung stoppen und erneut starten, indem sie die Methoden 'Stop Connection' und `Start Connection` aufruft.

Verbindung schließen

Zum Schließen einer Verbindung ruft die Anwendung die Methode 'Close Connection' (Verbindung schließen) auf. Wenn eine Anwendung einer Verbindung schließt, führt XMS folgende Aktionen aus:

- Es schließt alle Sitzungen, die dieser Verbindung zugeordnet sind, und löscht bestimmte Objekte, die diesen Sitzungen zugeordnet sind. Weitere Informationen dazu, welche Objekte gelöscht werden, finden Sie im Abschnitt „Objektlöschung” auf Seite 683. Gleichzeitig macht XMS alle Transaktionen rückgängig, die in der Sitzung aktuell in Bearbeitung sind.
- Es beendet die Kommunikationsverbindung zum Messaging-Server.
- Es gibt den Speicher und andere interne Ressourcen frei, die von der Verbindung verwendet werden.

Für Nachrichten, deren Empfang während einer Sitzung nicht bestätigt wurde, gibt XMS auch vor dem Schließen der Verbindung keine Empfangsbestätigung aus. Weitere Informationen zu Empfangsbestätigungen für Nachrichten finden Sie im Abschnitt „Nachrichtenbestätigung” auf Seite 671.

Handhabung von Ausnahmebedingungen

XMS .NET Ausnahmebedingungen werden alle von System.Exceptionabgeleitet. Weitere Informationen finden Sie unter „Fehlerbehandlung in XMS .NET” auf Seite 688.

Verbindungen zu einem Service Integration Bus

Eine XMS-Anwendung kann zum Herstellen einer Verbindung zu einem Service Integration Bus für WebSphere Application Server eine direkte TCP/IP-Verbindung oder HTTP über TCP/IP verwenden.

Das HTTP-Protokoll kann in Situationen verwendet werden, in denen eine direkte TCP/IP-Verbindung nicht möglich ist. Dies kommt häufig vor, wenn die Kommunikation über eine Firewall ausgeführt wird, beispielsweise beim Nachrichtenaustausch zwischen zwei Unternehmen. Die Verwendung von HTTP für die Kommunikation über eine Firewall wird häufig als *HTTP-Tunneling* bezeichnet. Das HTTP-Tunneling ist jedoch wesentlich langsamer als eine direkte TCP/IP-Verbindung, weil sich die Menge der zu übertragenden Daten durch die HTTP-Header erheblich vergrößert und weil das HTTP-Protokoll mehr Kommunikationsflüsse benötigt als TCP/IP.

Zum Erstellen einer TCP/IP-Verbindung kann eine Anwendung eine Verbindungsfactory verwenden, deren Eigenschaft XMSC_WPM_TARGET_TRANSPORT_CHAIN auf den Wert 'XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC' gesetzt ist. Dies ist der Standardwert der Eigenschaft. Wenn die Verbindung erfolgreich erstellt wurde, wird die Eigenschaft XMSC_WPM_CONNECTION_PROTOCOL auf den Wert 'XMSC_WPM_CP_TCP' gesetzt.

Um eine HTTP-Verbindung zu erstellen, muss eine Anwendung eine Verbindungsfactory verwenden, deren Eigenschaft 'XMSC_WPM_TARGET_TRANSPORT_CHAIN' auf den Namen einer eingehenden Transportkette gesetzt ist, die für die Verwendung eines HTTP-Transportkanals konfiguriert ist. Wenn die Verbindung erfolgreich erstellt wurde, wird die Eigenschaft 'XMSC_WPM_CONNECTION_PROTOCOL' auf den Wert 'XMSC_WPM_CP_HTTP' gesetzt. Informationen zum Konfigurieren von Transportketten finden Sie in der Produktdokumentation zu WebSphere Application Server unter Transportketten konfigurieren .

Beim Herstellen einer Verbindung zu einem Bootstrap-Server steht einer Anwendung eine ähnlich Auswahl an Kommunikationsprotokollen zur Verfügung. Die Eigenschaft XMSC_WPM_PROVIDER_ENDPOINTS einer Verbindungsfactory enthält eine Sequenz von einer oder mehreren Endpunktadressen von Bootstrap-Servern. Die Bootstrap-Transportkettenkomponente einer jeden Endpunktadresse kann einen der beiden folgenden Werte annehmen: 'XMSC_WPM_BOOTSTRAP_TCP' für eine TCP/IP-Verbindung zu einem Bootstrap-Server oder 'XMSC_WPM_BOOTSTRAP_HTTP' für eine HTTP-Verbindung.

Sitzungen

Sitzung - Ein Session-Objekt ist ein Einzelthreadkontext zum Senden und Empfangen von Nachrichten.

Eine Anwendung kann mithilfe einer Sitzung Nachrichten, Nachrichtenproduzenten, Nachrichtenkonsumenten, Warteschlangenbrowser und temporäre Ziele erstellen. Außerdem kann eine Anwendung eine Sitzung zum Ausführen lokaler Transaktionen verwenden.

Eine Anwendung kann mehrere Sitzungen erstellen, wobei jede Sitzung unabhängig von den anderen Sitzungen Nachrichten produziert und konsumiert. Wenn zwei Nachrichtenkonsumenten in verschiedenen Sitzungen (oder auch in derselben Sitzung) dasselbe Thema subskribieren, empfängt jeder eine eigene Kopie aller Nachrichten, die in diesem Thema veröffentlicht werden.

Anders als ein Connection-Objekt kann ein Session-Objekt nicht gleichzeitig in verschiedenen Threads verwendet werden. Nur die Methode 'Close Session' eines Session-Objekts kann von einem anderen Thread aus aufgerufen werden als der Thread, den das Session-Objekt zu diesem Zeitpunkt verwendet. Mit der Methode 'Close Session' wird eine Sitzung beendet und alle der Sitzung zugeordneten Systemressourcen werden freigeben.

Wenn es erforderlich ist, dass eine Anwendung Nachrichten in mehreren Threads gleichzeitig verarbeitet, dann muss die Anwendung in jedem dieser Threads eine Sitzung erstellen und diese Sitzung dann für die Sende- und Empfangsoperationen in dem jeweiligen Thread verwenden.

Durchgeführte Sitzungen

XMS-Anwendungen können lokale Transaktionen ausführen. Eine *lokale Transaktion* ist eine Transaktion, die Änderungen umfasst, welche nur die Ressourcen des Warteschlangenmanagers oder Service Integration Bus betreffen, mit dem die Anwendung verbunden ist.

Die Informationen in diesem Abschnitt sind nur relevant für den Fall, dass eine Anwendung eine Verbindung zu einem IBM MQ-Warteschlangenmanager oder einem WebSphere Application Server Service

Integration Bus herstellt. Für Echtzeitverbindungen zu einem Broker sind die Informationen in diesem Abschnitt irrelevant.

Um lokale Transaktionen ausführen zu können, muss eine Anwendung zunächst eine transaktionsbasierte Sitzung erstellen, indem sie die Methode 'Create Session' eines Connection-Objekts aufruft und als Parameter angibt, dass die Sitzung transaktionsbasiert ist. Danach werden alle innerhalb der Sitzung gesendeten und empfangenen Nachrichten in einer Folge von Transaktionen gruppiert. Eine Transaktion endet, wenn die Anwendung die Nachrichten, die sie seit Beginn der Transaktion gesendet oder empfangen hat, festschreibt oder rückgängig macht.

Um eine Transaktion festzuschreiben, ruft eine Anwendung die Methode 'Commit' des Session-Objekts auf. Wenn eine Transaktion festgeschrieben wird, werden alle Nachrichten, die innerhalb der Transaktion gesendet wurden, für die Zustellung an andere Anwendungen verfügbar. Außerdem werden sämtliche Nachrichten, die innerhalb der Transaktion empfangen wurden, bestätigt, damit der Messaging-Server nicht erneut versucht, diese an die Anwendung zuzustellen. In der Punkt-zu-Punkt-Domäne entfernt der Messaging-Server außerdem die empfangenen Nachrichten aus ihren Warteschlangen.

Um eine Transaktion rückgängig zu machen, ruft eine Anwendung die Methode 'Rollback' des Session-Objekts auf. Wenn eine Transaktion rückgängig gemacht wird, werden alle Nachrichten, die innerhalb der Transaktion gesendet wurden, vom Messaging-Server gelöscht und alle Nachrichten, die innerhalb der Transaktion empfangen wurden, sind erneut für die Zustellung verfügbar. In der Punkt-zu-Punkt-Domäne werden die empfangenen Nachrichten zurück in ihre Warteschlangen gestellt und sind somit wieder für andere Anwendungen sichtbar.

Wenn eine Anwendung eine transaktionsbasierte Sitzung erstellt oder die Methode 'Commit' bzw. 'Rollback' aufruft, wird automatisch eine neue Transaktion gestartet. Daher verfügt eine transaktionsbasierte Sitzung immer über eine aktive Transaktion.

Wenn eine Anwendung eine transaktionsbasierte Sitzung schließt, erfolgt ein impliziter Rollback. Sobald eine Anwendung eine Verbindung schließt, wird für alle transaktionsbasierten Sitzungen der Verbindung ein impliziter Rollback durchgeführt.

Eine Transaktion ist voll und ganz in einer transaktionsbasierten Sitzung enthalten. Eine Transaktion kann sich nicht über verschiedene Sitzungen erstrecken. Dies bedeutet, dass es einer Anwendung nicht möglich ist, Nachrichten in zwei oder mehr transaktionsbasierten Sitzungen zu senden und zu empfangen und alle diese Aktionen danach als einzelne Transaktion festzuschreiben oder rückgängig zu machen.

Zugehörige Konzepte

Nachrichtenbestätigung
Jede Sitzung, die nicht transaktionsbasiert ist, verfügt über einen Bestätigungsmodus, der festlegt, wie die von der Anwendung empfangenen Nachrichten bestätigt werden. Drei Bestätigungsmodi sind verfügbar und die Auswahl des Bestätigungsmodus wirkt sich auf das Design der Anwendung aus.

Nachrichtenübermittlung
XMS unterstützt persistente und nicht persistente Nachrichtenübermittlungsmodi sowie die asynchrone und synchrone Nachrichtenübermittlung.

Verwaltete IBM MQ XA-Transaktionen über XMS
Verwaltete IBM MQ XA-Transaktionen können über XMS verwendet werden.

Nachrichtenbestätigung

Jede Sitzung, die nicht transaktionsbasiert ist, verfügt über einen Bestätigungsmodus, der festlegt, wie die von der Anwendung empfangenen Nachrichten bestätigt werden. Drei Bestätigungsmodi sind verfügbar und die Auswahl des Bestätigungsmodus wirkt sich auf das Design der Anwendung aus.

Anmerkung: Dieser Abschnitt ist nur relevant, wenn eine Anwendung eine Verbindung zu einem IBM MQ -Warteschlangenmanager oder einem WebSphere Application Server Service Integration Bus herstellt. Für Echtzeitverbindungen zu einem Broker sind die Informationen in diesem Abschnitt irrelevant.

XMS verwendet zur Bestätigung des Empfangs von Nachrichten denselben Mechanismus wie JMS.

Wenn eine Sitzung nicht transaktionsbasiert ist, wird die Art des Nachrichtenempfangs durch eine Anwendung durch den Bestätigungsmodus der Sitzung bestimmt. Es gibt drei Bestätigungsmodi:

XMSC_AUTO_ACKNOWLEDGE
Die Sitzung bestätigt automatisch jede Nachricht, die von der Anwendung empfangen wird.

Wenn Nachrichten synchron an die Anwendung übermittelt werden, bestätigt die Sitzung den Empfang einer Nachricht jedes Mal, wenn ein Receive-Aufruf erfolgreich abgeschlossen wurde. Wenn die Anwendung eine Nachricht erfolgreich empfängt, ein Fehler jedoch die Bestätigung verhindert, wird die Nachricht wieder für die Übermittlung verfügbar. Daher muss die Anwendung in der Lage sein, eine erneut übermittelte Nachricht zu verarbeiten.

XMSC_DUPS_OK_ACKNOWLEDGE
Die Sitzung bestätigt den Empfang von Nachrichten durch die Anwendung zu bestimmten Zeiten, die sie auswählt.

Die Verwendung dieses Bestätigungsmodus verringert den Arbeitsaufwand für die Sitzung. Allerdings kann ein Fehler, der die Nachrichtenbestätigung verhindert, dazu führen, dass mehr als eine Nachricht wieder für die Übermittlung verfügbar wird. Daher muss die Anwendung in der Lage sein, eine erneut übermittelte Nachricht zu verarbeiten.

XMSC_CLIENT_ACKNOWLEDGE
Die Anwendung bestätigt die empfangenen Nachrichten, indem sie die Acknowledge-Methode (Bestätigungsmethode) der Message-Klasse (Nachrichtenklasse) aufruft.

Die Anwendung kann den Empfang jeder Nachricht einzeln bestätigen oder einen Nachrichtenstapel empfangen und die Acknowledge-Methode nur für die zuletzt empfangene Nachricht aufrufen. Wenn die Acknowledge-Methode aufgerufen wird, werden alle Nachrichten bestätigt, die seit dem letzten Aufruf der Methode empfangen wurden.

Eine Anwendung kann in Verbindung mit einer dieser Bestätigungsmodi die Nachrichtenübermittlung in einer Sitzung stoppen und erneut starten, indem sie die Recover-Methode der Session-Klasse aufruft. Nachrichten, deren Empfang zuvor nicht bestätigt wurde, werden erneut übermittelt. Sie werden jedoch möglicherweise nicht in der Reihenfolge übermittelt, in der sie zuvor übermittelt wurden. Es kann vorkommen, dass zwischenzeitlich Nachrichten mit einer höheren Priorität eingetroffen und manche der ursprünglichen Nachrichten abgelaufen sind. In der Punkt-zu-Punkt-Domäne wurden einige der ursprünglichen Nachrichten möglicherweise bereits von einer anderen Anwendung verarbeitet.

Eine Anwendung kann feststellen, ob eine Nachricht erneut übermittelt wird, indem sie den Inhalt des Headerfelds 'JMSRedelivered' der Nachricht überprüft. Hierfür ruft die Anwendung die Methode 'Get JMSRedelivered' der Message-Klasse auf.

Zugehörige Konzepte
Durchgeführte Sitzungen
XMS-Anwendungen können lokale Transaktionen ausführen. Eine *lokale Transaktion* ist eine Transaktion, die Änderungen umfasst, welche nur die Ressourcen des Warteschlangenmanagers oder Service Integration Bus betreffen, mit dem die Anwendung verbunden ist.

Nachrichtenübermittlung
XMS unterstützt persistente und nicht persistente Nachrichtenübermittlungsmodi sowie die asynchrone und synchrone Nachrichtenübermittlung.

Verwaltete IBM MQ XA-Transaktionen über XMS
Verwaltete IBM MQ XA-Transaktionen können über XMS verwendet werden.

Nachrichtenübermittlung
XMS unterstützt persistente und nicht persistente Nachrichtenübermittlungsmodi sowie die asynchrone und synchrone Nachrichtenübermittlung.

Nachrichtenübermittlungsmodi

XMS unterstützt zwei Nachrichtenübermittlungsmodi:

- Persistent

Persistente Nachrichten werden sofort übermittelt. Dabei übernimmt ein Messaging-Server spezielle Vorsichtsmaßnahmen wie beispielsweise die Protokollierung der Nachrichten, um sicherzustellen, dass persistente Nachrichten, selbst im Falle eines Fehlers, während der Übermittlung, nicht verloren gehen.

- Nicht persistent

  Nicht persistente Nachrichten werden nur ein einziges Mal übermittelt. Da nicht persistente Nachrichten im Falle eines Fehlers während der Übermittlung verloren gehen können, sind sie nicht so zuverlässig wie persistente Nachrichten.

Die Wahl des geeigneten Übermittlungsmodus ist ein Kompromiss zwischen Zuverlässigkeit und Leistung. Nicht persistente Nachrichten werden normalerweise schneller übermittelt als persistente Nachrichten.

Asynchrone Nachrichtenübermittlung

XMS führt alle asynchronen Nachrichtenübermittlungen für eine Sitzung über einen einzigen Thread aus. Dies bedeutet, dass immer nur eine Nachrichtenlistenerfunktion oder eine onMessage()-Methode nach der anderen ausgeführt werden kann.

Wenn in einer Sitzung mehr als ein Nachrichtenkonsument Nachrichten asynchron empfängt und eine Nachrichtenlistenerfunktion oder onMessage()-Methode gerade eine Nachricht an einen Nachrichtenkonsumenten übermittelt, dann müssen alle anderen Nachrichtenkonsumenten, denen dieselbe Nachricht übermittelt werden soll, warten, bis dieser eine Übermittlungsvorgang abgeschlossen ist. Auch andere Nachrichten, die an diese Sitzung übermittelt werden sollen, müssen ebenso lange warten.

Wenn für eine Anwendung eine gleichzeitige Nachrichtenübermittlung erforderlich ist, sollten Sie mehr als eine Sitzung erstellen, damit XMS mehrere Threads gleichzeitig für die asynchrone Nachrichtenübermittlung nutzen kann. Dies bedeutet, dass dann auch mehrere Nachrichtenlistenerfunktionen oder onMessage()-Methoden gleichzeitig ausgeführt werden können.

Eine Sitzung wird nicht in dem Moment asynchron, in dem einem Nachrichtenlistener ein Konsument zugeordnet wird. Eine Sitzung wird erst dann asynchron, wenn die Methode Connection.Start aufgerufen wird. Bis zum Aufruf der Methode Connection.Start sind alle synchronen Aufrufe zulässig. Die Nachrichtenübermittlung an Konsumenten beginnt, wenn Connection.Start aufgerufen wird.

Wenn in einer asynchronen Sitzung synchrone Aufrufe erforderlich sind, z. B. zum Erstellen eines Nachrichtenkonsumenten oder -produzenten, muss zunächst die Methode Connection.Stop aufgerufen werden. Durch erneuten Aufruf von Connection.Start wird die Sitzung fortgesetzt und die Nachrichtenübermittlung erneut gestartet. Die einzige Ausnahme dazu ist der Thread zur Übermittlung von Sitzungsnachrichten, also der Thread, der Nachrichten an die Callback-Funktion übermittelt. Dieser Thread kann in der Callback-Funktion für Nachrichten jeden beliebigen Aufruf für eine Sitzung ausführen (mit Ausnahme des Aufrufs 'Close').

Anmerkung: Im nicht verwalteten Modus wird der MQDISC-Aufruf innerhalb einer Callback-Funktion vom IBM MQ .NET-Client nicht unterstützt. Eine Clientanwendung kann also innerhalb der Callback-Funktion 'MessageListener' im asynchronen Empfangsmodus keine Sitzungen erstellen (Create) oder schließen (Close). Sie müssen die Sitzung außerhalb der MessageListener-Methode erstellen und verwerfen.

Synchrone Nachrichtenübermittlung

Nachrichten werden synchron an eine Anwendung übermittelt, wenn die Anwendung für den Empfang die Receive-Methoden der MessageConsumer-Objekte verwendet.

Bei Verwendung der Receive-Methoden kann für eine Anwendung festgelegt werden, dass sie eine bestimmte Zeit lang oder ohne zeitliche Begrenzung auf eine Nachricht warten soll. Wenn die Anwendung hingegen überhaupt nicht auf eine Nachricht warten soll, kann die Receive-Methode ohne jegliche Wartezeit ('No Wait') festgelegt werden.

Zugehörige Konzepte

Durchgeführte Sitzungen

XMS-Anwendungen können lokale Transaktionen ausführen. Eine *lokale Transaktion* ist eine Transaktion, die Änderungen umfasst, welche nur die Ressourcen des Warteschlangenmanagers oder Service Integration Bus betreffen, mit dem die Anwendung verbunden ist.

Nachrichtenbestätigung
Jede Sitzung, die nicht transaktionsbasiert ist, verfügt über einen Bestätigungsmodus, der festlegt, wie die von der Anwendung empfangenen Nachrichten bestätigt werden. Drei Bestätigungsmodi sind verfügbar und die Auswahl des Bestätigungsmodus wirkt sich auf das Design der Anwendung aus.

Verwaltete IBM MQ XA-Transaktionen über XMS
Verwaltete IBM MQ XA-Transaktionen können über XMS verwendet werden.

Verwaltete IBM MQ XA-Transaktionen über XMS

Verwaltete IBM MQ XA-Transaktionen können über XMS verwendet werden.

Um XA-Transaktionen über XMS verwenden zu können, muss eine Sitzung mit Transaktionsunterstützung, eine so genannte transaktionsbasierte Sitzung erstellt werden. Bei der Verwendung von XA-Transaktionen erfolgt die Transaktionssteuerung durch globale DTC-Transaktionen (Distributed Transaction Coordinator) und nicht durch XMS-Sitzungen. Wenn XA-Transaktionen verwendet werden, können die Methoden `Session.commit` oder `Session.rollback` nicht in der XMS-Sitzung ausgegeben werden. Verwenden Sie stattdessen die DTC-Methoden `Transscope.Commit` oder `Transscope.Rollback`, um die Transaktionen festzuschreiben oder rückgängig zu machen. Wenn eine Sitzung für XA-Transaktionen verwendet wird, müssen der Nachrichtenproduzent und -konsument, die mithilfe der Sitzung erstellt werden, Teil der XA-Transaktion sein. Ihre Verwendung außerhalb des Geltungsbereichs der XA-Transaktion ist nicht möglich. Das heißt, sie können für Operationen wie `Producer.send` oder `Consumer.receive` außerhalb der XA-Transaktion nicht verwendet werden.

In folgenden Fällen wird ein `IllegalStateException`-Ausnahmeobjekt ausgelöst:

- Eine Sitzung mit XA-Transaktionsunterstützung wird für `Session.commit` oder `Session.rollback` verwendet.
- Produzenten- oder Konsumentenobjekte, die einmal in einer XA-transaktionsbasierten Sitzung verwendet wurden, werden danach außerhalb des Geltungsbereichs der XA-Transaktion verwendet.

XA-Transaktionen werden für asynchrone Nachrichtenkonsumenten nicht unterstützt.

Anmerkung:

1. Das `Producer`-, `Consumer`-, `Session`- oder `Connection`-Objekt wird möglicherweise vor der Festschreibung der XA-Transaktion geschlossen. In diesem Fall werden die Nachrichten in der Transaktion durch ein Rollback rückgängig gemacht. Auch bei einer Unterbrechung der Verbindung vor der Festschreibung der XA-Transaktion werden alle Nachrichten in der Transaktion rückgängig gemacht. Bei einem `Producer`-Objekt bedeutet ein Rollback, dass die Nachrichten nicht in die Warteschlange eingereiht werden. Bei einem `Consumer`-Objekt bedeutet ein Rollback, dass die Nachrichten in der Warteschlange verbleiben, weil sie nicht abgerufen werden.
2. Wenn ein `Producer`-Objekt eine Nachricht mit der Option `TimeToLive` in `TransactionScope` einreiht und ein `commit`-Befehl ausgegeben wird, nachdem das Zeitlimit abgelaufen ist, kann die Nachricht ablaufen, bevor der `commit`-Befehl ausgegeben wird. In diesem wird die Nachricht nicht für die `Consumer`-Objekte verfügbar gemacht.
3. `Session`-Objekte werden nicht über mehrere Threads hinweg unterstützt. Das bedeutet, dass Transaktionen mit `Session`-Objekten, die von mehreren Threads gemeinsam genutzt werden, nicht unterstützt werden.

Zugehörige Konzepte

Durchgeführte Sitzungen
XMS-Anwendungen können lokale Transaktionen ausführen. Eine *lokale Transaktion* ist eine Transaktion, die Änderungen umfasst, welche nur die Ressourcen des Warteschlangenmanagers oder Service Integration Bus betreffen, mit dem die Anwendung verbunden ist.

Nachrichtenbestätigung
Jede Sitzung, die nicht transaktionsbasiert ist, verfügt über einen Bestätigungsmodus, der festlegt, wie die von der Anwendung empfangenen Nachrichten bestätigt werden. Drei Bestätigungsmodi sind verfügbar und die Auswahl des Bestätigungsmodus wirkt sich auf das Design der Anwendung aus.

Nachrichtenübermittlung

XMS unterstützt persistente und nicht persistente Nachrichtenübermittlungsmodi sowie die asynchrone und synchrone Nachrichtenübermittlung.

Ziele

XMS-Anwendungen geben mit einem Destination-Objekt das Ziel für gesendete Nachrichten und die Quelle von empfangenen Nachrichten an.

Eine XMS-Anwendung kann entweder ein Destination-Objekt während der Laufzeit erstellen oder ein vordefiniertes Ziel aus dem Repository für verwaltete Objekte abrufen.

Als flexibelste Möglichkeit zum Angeben eines Ziels kann eine XMS-Anwendung das Ziel, ähnlich wie bei einem ConnectionFactory-Objekt, als verwaltetes Objekt definieren. Bei dieser Vorgehensweise können Anwendungen, die in C, C++, .NET-Sprachen und Java geschrieben sind, die Zieldefinitionen gemeinsam nutzen. Die Eigenschaften von verwalteten Destination-Objekten können geändert werden, ohne Änderungen am Code vorzunehmen.

Ziele in .NET

In .NET werden Ziele entsprechend dem jeweiligen Protokolltyp erstellt und können dann auch nur mit dem Protokolltyp verwendet werden, mit dem sie erstellt wurden. Es gibt zwei Methoden zum Erstellen von Zielen, eine für Themen und eine für Warteschlangen:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

Diese Methoden sind für die folgenden beiden Objekte in der .NET -API verfügbar:

- ISession
- XMSFactoryFactory

In beiden Fällen können diese Methoden eine Zeichenfolge im URI-Stil akzeptieren, die Parameter im folgenden Format enthalten kann:

```
"topic://some/topic/name?priority=5"
```

Alternativ dazu kann in diesen Methoden auch ein reiner Zielname angegeben werden, d. h. ein Name ohne den Präfix 'topic://' oder 'queue://' und ohne Parameter. Beispiel für eine Zeichenfolge im URI-Stil:

```
CreateTopic("topic://some/topic/name");
```

Mit dieser Zeichenfolge im URI-Stil wird dasselbe Ergebnis erzielt wie mit dem folgenden Zielnamen. Beispiel für einen reinen Zielnamen:

```
CreateTopic("some/topic/name");
```

Weitere Informationen finden Sie unter IDestination.

Beim WebSphere Application Server Service Integration Bus für JMS können Themen auch in einer Kurzform angegeben werden, die die beiden Variablen *topicname* (Themenname) und *topicspace* (Themenbereich) umfasst, aber keine Parameter enthalten darf:

```
CreateTopic("topicspace:topicname");
```

Themen-URIs

Der Uniform Resource Identifier (URI) eines Themas gibt den Namen des Themas und ggf. auch eine oder mehrere seiner Eigenschaften an.

Ein Themen-URI beginnt mit der Sequenz 'topic://' gefolgt von dem Namen des Themas. Darüber hinaus kann er optional eine Liste der Name/Wert-Paare enthalten, mit denen die anderen Themeneigenschaften festgelegt werden. Ein Themenname darf nicht leer sein.

Es folgte ein Beispiel für ein Fragment aus einem .NET-Code:

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

Weitere Informationen zu den Eigenschaften eines Themas, einschließlich des Namens und der gültigen Werte, die in einem URI verwendet werden können, finden Sie im Abschnitt Eigenschaften von Destination-Objekten.

Beim Angeben eines Themen-URI, der in einer Subskription verwendet werden soll, können auch Platzhalter verwendet werden. Die Syntax für diese Platzhalterzeichen hängt vom Verbindungstyp und der Brokerversion ab. Die folgende Option ist verfügbar:

- WebSphere Application Server Service Integration Bus

WebSphere Application Server Service Integration Bus

Ein WebSphere Application Server Service Integration Bus verwendet folgende Platzhalterzeichen:

* für eine beliebige Anzahl von Zeichen auf einer Gliederungsebene
// für 0 oder mehr Ebenen
//. für 0 oder mehr Ebenen am Ende eines Themenausdrucks

In Tabelle 82 auf Seite 676 sind einige Beispiele für die Verwendung dieses Platzhalterformats aufgeführt.

Tabelle 82. Beispiele für URIs mit dem Platzhalterformat für einen WebSphere Application Server Service Integration Bus

Uniform Resource Identifier (URI)	Entsprechungen	Beispiele
"topic://Sport/*ball/Results"	Alle Themen mit einem einzigen Gliederungsebenennamen, der mit "ball" endet, zwischen den Ebenen "Sport" und "Results"	"topic://Sport/Football/Results" und "topic://Sport/Netball/Results"
"topic://Sport//Results"	Alle Themen die mit "Sport/" beginnen und mit "/Results" enden	"topic://Sport/Football/Results" und "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football//."	Alle Themen, die mit "Sport/Football/" beginnen	"topic://Sport/Football/Results" und "topic://Sport/Football/TeamNews/Signings/Managerial"
"topic://Sport/*ball//Results//."	Themen	"topic://Sport/Football/Results" und "topic://Sport/Netball/National/Div3/Results/2002/November"

Zugehörige Konzepte

Warteschlangen-URIs
Der Uniform Resource Identifier (URI) einer Warteschlange gibt den Namen der Warteschlange und ggf. auch eine oder mehrere ihrer Eigenschaften an.

Temporäre Ziele
XMS-Anwendungen können temporäre Ziele erstellen und verwenden.

Warteschlangen-URIs

Der Uniform Resource Identifier (URI) einer Warteschlange gibt den Namen der Warteschlange und ggf. auch eine oder mehrere ihrer Eigenschaften an.

Ein Warteschlangen-URI beginnt mit der Folge `queue://`, auf die der Name der Warteschlange folgt. Darüber hinaus kann er eine Liste der Name/Wert-Paare enthalten, mit denen die anderen Warteschlangeneigenschaften festgelegt werden.

Bei IBM MQ-Warteschlangen (jedoch nicht bei Warteschlangen des Standard-Messaging-Providers für WebSphere Application Server) kann der Warteschlangenmanager, in dem sich die Warteschlange befindet, vor der Warteschlange angegeben werden. In diesem Fall wird der Warteschlangenmanagername durch einen Schrägstrich (/) vom Warteschlangennamen getrennt.

Falls ein Warteschlangenmanager angegeben wird, muss es derjenige sein, mit dem XMS für die Verbindung, die diese Warteschlange verwendet, direkt verbunden ist, oder er muss über diese Warteschlange zugänglich sein. Ferne Warteschlangenmanager werden nur für das Abrufen von Nachrichten aus Warteschlangen, nicht jedoch für das Einreihen von Nachrichten in Warteschlangen unterstützt. Weitere Informationen finden Sie in der Dokumentation zu IBM MQ-Warteschlangenmanagern.

Falls kein Warteschlangenmanager angegeben wird, ist es irrelevant, ob das zusätzliche Trennzeichen '/' eingefügt oder weggelassen wird, da es keinen Einfluss auf die Definition der Warteschlange hat.

Alle folgenden Warteschlangendefinitionen geben auf unterschiedliche Weise dieselbe Warteschlange an: eine IBM MQ-Warteschlange mit dem Namen 'QB' in einem Warteschlangenmanager mit dem Namen 'QM_A', mit dem XMS direkt verbunden ist:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

Zugehörige Konzepte

Themen-URIs
Der Uniform Resource Identifier (URI) eines Themas gibt den Namen des Themas und ggf. auch eine oder mehrere seiner Eigenschaften an.

Temporäre Ziele
XMS-Anwendungen können temporäre Ziele erstellen und verwenden.

Temporäre Ziele

XMS-Anwendungen können temporäre Ziele erstellen und verwenden.

Eine Anwendung verwendet ein temporäres Ziel normalerweise, um Antworten auf Anforderungsnachrichten zu empfangen. Um das Ziel anzugeben, an das die Antwort auf eine Anforderungsnachricht gesendet werden soll, ruft eine Anwendung die Methode 'Set JMSReplyTo' des Nachrichtenobjekts (Message) auf, das die Anforderungsnachricht darstellt. Das in dem Aufruf angegebene Ziel kann ein temporäres Ziel sein.

Obwohl zum Erstellen eines temporären Ziels eine Sitzung verwendet wird, ist der Geltungsbereich eines temporären Ziels eigentlich die Verbindung, die zum Erstellen der Sitzung verwendet wurde. Alle Sitzungen der Verbindung können Nachrichtenproduzenten und Nachrichtenkonsumenten für das temporäre Ziel erstellen. Das temporäre Ziel bleibt bestehen, bis es explizit gelöscht wird oder die Verbindung endet, je nachdem, welches Ereignis zuerst eintritt.

Wenn eine Anwendung eine temporäre Warteschlange erstellt, wird die Warteschlange auf dem Messaging-Server erstellt, mit dem die Anwendung verbunden ist. Wenn die Anwendung mit einem Warteschlangenmanager verbunden ist, wird eine dynamische Warteschlange aus der Modellwarteschlange erstellt, deren Name durch die Eigenschaft XMSC_WMQ_TEMPORARY_MODEL angegeben wird. Das Präfix, das zur Bildung des Namens der dynamischen Warteschlange verwendet wird, wird durch die Eigenschaft XMSC_WMQ_TEMP_Q_PREFIX angegeben. Wenn die Anwendung mit einem Service Integration Bus verbunden ist, wird eine temporäre Warteschlange im Bus erstellt und das Präfix, das zur Bildung des Namens der temporären Warteschlange verwendet wird, wird durch die Eigenschaft XMSC_WPM_TEMP_Q_PREFIX angegeben.

Wenn eine Anwendung, die mit einem Service Integration Bus verbunden ist, ein temporäres Topic erstellt, wird das Präfix, das zur Bildung des Namens des temporären Topic verwendet wird, durch die Eigenschaft XMSC_WPM_TEMP_TOPIC_PREFIX angegeben.

Zugehörige Konzepte

Themen-URIs

Der Uniform Resource Identifier (URI) eines Themas gibt den Namen des Themas und ggf. auch eine oder mehrere seiner Eigenschaften an.

Warteschlangen-URIs

Der Uniform Resource Identifier (URI) einer Warteschlange gibt den Namen der Warteschlange und ggf. auch eine oder mehrere ihrer Eigenschaften an.

Nachrichtenproduzenten

In XMS kann einem Nachrichtenproduzenten bei dessen Erstellung entweder genau ein gültiges Ziel oder gar kein Ziel zugeordnet werden. Wenn ein Nachrichtenproduzent erstellt wird, ohne dass ihm ein Ziel zugeordnet wird, muss stattdessen beim Senden einer Nachricht ein gültiges Ziel angegeben werden.

Nachrichtenproduzenten mit zugeordnetem Ziel

In diesem Szenario wird ein Nachrichtenproduzent erstellt, dem ein gültiges Ziel zugeordnet wird. In diesem Fall muss bei der Sendeoperation kein Ziel angegeben werden.

Nachrichtenproduzent ohne zugeordnetes Ziel

In XMS .NET kann ein Nachrichtenproduzent erstellt werden, dem kein Ziel zugeordnet wird.

Um einen Nachrichtenproduzenten ohne zugeordnetes Ziel bei Verwendung der API .NET zu erstellen, muss NULL als Parameter an die Methode `CreateProducer()` des ISession-Objekts übergeben werden (z. B. `session.CreateProducer(null)`). Beim Senden der Nachricht muss jedoch ein gültiges Ziel angegeben werden.

Nachrichtenkonsumenten

Bei den Nachrichtenkonsumenten wird zwischen permanenten und nicht permanenten Subskribenten sowie zwischen synchronen und asynchronen Nachrichtenkonsumenten unterschieden.

Permanente Subskribenten

Ein permanenter Subskribent ist ein Nachrichtenkonsument, der alle in einem Thema veröffentlichten Nachrichten empfängt, einschließlich der Nachrichten, die veröffentlicht werden, während der Subskribent inaktiv ist.

Achtung: Diese Informationen sind nur relevant, wenn eine Anwendung eine Verbindung zu einem IBM MQ -Warteschlangenmanager oder einem WebSphere Application Server Service Integration Bus herstellt. Für Echtzeitverbindungen zu einem Broker sind die Informationen in diesem Abschnitt irrelevant.

Um einen permanenten Subskribenten für ein Thema zu erstellen, ruft eine Anwendung die Methode 'Create Durable Subscriber' eines Session-Objekts auf und gibt als Parameter einen Namen zum Angeben der permanenten Subskription an sowie ein Destination-Objekt, das das Thema angibt. Die Anwendung kann einen permanenten Subskribenten mit oder ohne Nachrichtenselektor erstellen und angeben, ob der permanente Subskribent Nachrichten empfangen soll, die über seine eigene Verbindung veröffentlicht werden.

Der Sitzung, die zum Erstellen eines permanenten Subskribenten verwendet wird, muss eine Client-ID zugeordnet sein. Die Client-ID der Sitzung ist mit der Client-ID der Verbindung identisch, die zum Erstellen der Sitzung verwendet wird. Weitere Informationen zum Angeben der Client-ID finden Sie im Abschnitt „ConnectionFactory- und Connection-Objekte” auf Seite 668.

Der Name, mit dem die permanente Subskription angegeben wird, muss innerhalb der Client-ID eindeutig sein. Daher ist die Client-ID Bestandteil der vollständigen, eindeutigen ID der permanenten Subskription. Der Messaging-Server zeichnet die permanente Subskription in einem Datensatz auf und stellt auf diese

Weise sicher, dass alle in einem Thema veröffentlichten Nachrichten aufbewahrt werden, bis sie vom permanenten Subskribenten bestätigt wurden oder bis sie ablaufen.

Der Messaging-Server setzt die Aufzeichnung der permanenten Subskription selbst dann fort, nachdem der permanente Subskribenten geschlossen wurde. Damit eine zuvor erstellte permanente Subskription weiterverwendet werden kann, muss eine Anwendung einen permanenten Subskribenten erstellen und dabei denselben Subskriptionsnamen und eine Sitzung mit derselben Client-ID verwenden, die der ursprünglichen permanenten Sitzung zugeordnet waren. Zu jedem Zeitpunkt kann immer nur eine Sitzung einen permanenten Subskribenten für eine bestimmte permanente Subskription haben.

Der Geltungsbereich einer permanenten Subskription ist der Messaging-Server, der die Subskription aufzeichnet. Wenn zwei Anwendungen, die mit unterschiedlichen Messaging-Servern verbunden sind, jeweils einen permanenten Subskribenten mit demselben Subskriptionsnamen und derselben Client-ID erstellen, entstehen dabei zwei völlig voneinander unabhängige permanente Subskriptionen.

Um eine permanente Subskription zu löschen, ruft eine Anwendung die Methode 'Unsubscribe' eines Session-Objekts auf und gibt als Parameter den Namen der permanenten Subskription an. Die Client-ID, die der Sitzung zugeordnet ist, muss dabei dieselbe sein, die auch der permanenten Subskription zugeordnet ist. Der Messaging-Server löscht den aufgezeichneten Datensatz der permanenten Subskription und sendet danach keine weiteren Nachrichten an den permanenten Subskribenten.

Um eine vorhandene Subskription zu ändern, kann eine Anwendung einen permanenten Subskribenten erstellen und dabei denselben Subskriptionsnamen und dieselbe Client-ID, jedoch ein anderes Thema und/oder einen anderen Nachrichtenselektor angeben. Eine permanente Subskription zu ändern hat dieselben Auswirkungen wie die vorhandene Subskription zu löschen und eine neue zu erstellen.

Für eine Anwendung, die eine Verbindung zu einem IBM MQ -Warteschlangenmanager herstellt, verwaltet XMS die Subskribentenwarteschlangen. Daher ist es nicht erforderlich, dass die Anwendung einen Subskribentennamen angibt. Falls dennoch eine Subskribentenwarteschlange angegeben wird, ignoriert XMS dies.

Beachten Sie, dass Sie die Subskribentenwarteschlange einer permanenten Subskription nicht ändern können. Die einzige Möglichkeit, die Subskribentenwarteschlange zu ändern, ist die Subskription zu löschen und eine neue zu erstellen.

Wenn eine Anwendung eine Verbindung zu einem Service Integration Bus herstellt, muss jeder permanente Subskribent eine festgelegte Ausgangsposition für permanente Subskriptionen haben. Um die Ausgangsposition für permanente Subskriptionen für alle permanenten Subskribenten anzugeben, die dieselbe Verbindung verwenden, legen Sie die Eigenschaft XMSC_WPM_DUR_SUB_HOME des ConnectionFactory-Objekts, das zum Erstellen der Verbindung verwendet wird, entsprechend fest. Um die Ausgangsposition für permanente Subskriptionen für ein bestimmtes Thema anzugeben, legen Sie die Eigenschaft 'XMSC_WPM_DUR_SUB_HOME' des Destination-Objekts, das das Thema angibt, entsprechend fest. Die Ausgangsposition für permanente Subskriptionen muss für eine Verbindung angegeben werden, bevor eine Anwendung einen permanenten Subskribenten erstellen kann, der diese Verbindung verwendet. Ein Wert, der für ein Ziel angegeben wird, überschreibt in jedem Fall den für eine Verbindung angegebenen Wert.

Synchrone Nachrichtenkonsumenten

Der synchrone Nachrichtenkonsument empfängt die Nachrichten synchron aus einer Warteschlange und empfängt jeweils eine Nachricht. Wenn die Methode `Receive(wait interval)` verwendet wird, wartet der Aufruf nur eine in Millisekunden angegebene Zeit lang auf eine Nachricht oder bis der Nachrichtenkonsument geschlossen wird.

Wenn die Methode 'ReceiveNoWait()' verwendet wird, empfängt der synchrone Nachrichtenkonsument Nachrichten ohne jede Verzögerung. Sobald die nächste Nachricht verfügbar ist, wird sie sofort empfangen, andernfalls wird ein Verweis auf ein Message-Objekt mit dem Wert 'null' zurückgegeben.

Asynchrone Nachrichtenkonsumenten

Der asynchrone Nachrichtenkonsument empfängt Nachrichten aus einer Warteschlange asynchron. Der von der Anwendung registrierte Nachrichtenlistener wird aufgerufen, sobald eine neue Nachricht in der Warteschlange verfügbar ist.

Nicht verarbeitbare Nachrichten in XMS

Eine nicht verarbeitbare Nachricht ist eine Nachricht, die nicht von einer empfangenden MDB-Anwendung verarbeitet werden kann. Wenn eine nicht verarbeitete Nachricht auftritt, kann das XMS -Objekt MessageConsumer sie gemäß den beiden Warteschlangeneigenschaften **BOQUEUE**und **BOTHRESH**erneut in die Warteschlange stellen.

Gelegentlich kann es vorkommen, dass eine Nachricht, die an eine MDB zugestellt wurde, mit einem Rollback in eine IBM MQ-Warteschlange zurückgesetzt wird. Dies kann beispielsweise der Fall sein, wenn eine Nachricht innerhalb einer Arbeitseinheit zugestellt wird, die dann rückgängig gemacht wird. Eine Nachricht, für die ein Rollback erfolgt ist, wird normalerweise erneut zugestellt, allerdings kann eine falsch formatierte Nachricht dazu führen, dass eine MDB wiederholt fehlschlägt und daher keine Zustellung möglich ist. Eine solche Nachricht wird als nicht verarbeitbare Nachricht bezeichnet. Sie können in der Konfiguration von IBM MQ festlegen, dass eine nicht verarbeitbare Nachricht zur weiteren Untersuchung automatisch an eine andere Warteschlange übertragen oder gelöscht wird. Informationen zum Konfigurieren von IBM MQ auf diese Weise finden Sie unter „Behandlung nicht verarbeitbarer Nachrichten in ASF” auf Seite 682.

Manchmal kommt eine fehlerhaft formatierte Nachricht in einer Warteschlange an. In diesem Kontext bedeutet 'fehlerhaft formatiert', dass die empfangende Anwendung die Nachricht nicht ordnungsgemäß verarbeiten kann. Eine solche Nachricht kann dazu führen, dass die empfangende Anwendung fehlschlägt und diese fehlerhaft formatierte Nachricht zurücksetzt. Die Nachricht kann dann mehrfach an die Eingabewarteschlange übermittelt und mehrfach von der Anwendung zurückgesetzt werden. Diese Nachrichten werden als nicht verarbeitbare Nachrichten bezeichnet. Das XMS-Objekt 'MessageConsumer' erkennt nicht verarbeitbare Nachrichten und leitet diese an ein alternatives Ziel um.

Der IBM MQ-Warteschlangenmanager zeichnet die Häufigkeit auf, mit der die einzelnen Nachrichten zurückgesetzt wurden. Sobald diese Anzahl einen konfigurierbaren Schwellenwert erreicht, reiht der Nachrichtenkonsument die Nachricht in eine namentlich genannte Rücksetzwarteschlange ein. Wenn diese Neueinreihung aus irgendeinem Grund fehlschlägt, wird die Nachricht aus der Eingabewarteschlange entfernt und entweder neu in die Warteschlange für nicht zustellbare Nachrichten eingereiht oder gelöscht.

XMS-Objekte des Typs 'ConnectionConsumer' handhaben nicht verarbeitbare Nachrichten auf dieselbe Weise und mit denselben Warteschlangeneigenschaften. Wenn mehrere Verbindungskonsumenten dieselbe Warteschlange überwachen, kann es vorkommen, dass die nicht verarbeitbare Nachricht häufiger an eine Anwendung übermittelt wird, als durch den Schwellenwert festgelegt, bevor die Neueinreihung durchgeführt wird. Dieses Verhalten ist auf die Art und Weise zurückzuführen, in der einzelne Verbindungskonsumenten Warteschlangen überwachen und nicht verarbeitbare Nachrichten erneut in die Warteschlange stellen.

Der Schwellenwert und der Name der Rücksetzwarteschlange sind Attribute einer IBM MQ-Warteschlange. Die Namen der Attribute lauten **BackoutThreshold** und **BackoutRequeueQName**. Sie werden auf folgende Warteschlangen angewendet:

- Beim Punkt-zu-Punkt-Messaging handelt es sich um die zugrunde liegende lokale Warteschlange. Dies ist wichtig, wenn Nachrichtenkonsumenten und Verbindungskonsumenten Aliasnamen einer Warteschlange verwenden.
- Beim Publish/Subscribe-Messaging im normalen Modus des IBM MQ-Messaging-Providers wird die verwaltete Warteschlange des Themas auf Basis der Modellwarteschlange erstellt.
- Beim Publish/Subscribe-Messaging im Migrationsmodus des IBM MQ-Messaging-Providers gelten sie für die CCSUB-Warteschlange, die im TopicConnectionFactory-Objekt definiert ist, oder für die CCDSUB-Warteschlange, die im Topic-Objekt definiert ist.

Um die Attribute **BackoutThreshold** und **BackoutRequeueQName** festzulegen, geben Sie den folgenden MQSC-Befehl aus:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Wenn Ihr System beim Publish/Subscribe-Messaging eine dynamische Warteschlange für jede Subskription erstellt, werden diese Attributwerte aus der IBM MQ classes for JMS -Modellwarteschlange SYSTEM.JMS.MODEL.QUEUE. Um diese Einstellungen zu ändern, verwenden Sie Folgendes:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Wenn der Rücksetzschwellenwert null ist, ist die Behandlung nicht verarbeitbarer Nachrichten inaktiviert und nicht verarbeitbare Nachrichten bleiben in der Eingabewarteschlange. Andernfalls wird die Nachricht an die namentlich genannte Rücksetzwarteschlange gesendet, sobald der Rücksetzungszähler den Schwellenwert erreicht.

Wenn der Rücksetzungszähler den Schwellenwert erreicht, aber die Nachricht nicht in die Rücksetzwarteschlange eingereiht werden kann, wird die Nachricht stattdessen an die Warteschlange für nicht zustellbare Nachrichten gesendet oder, wenn es eine nicht persistente Nachricht ist, gelöscht.

Diese Situation tritt ein, wenn die Rücksetzwarteschlange nicht definiert ist oder wenn das MessageConsumer-Objekt die Nachricht nicht an die Rücksetzwarteschlange senden kann.

System für Behandlung nicht verarbeitbarer Nachrichten konfigurieren

Welche Warteschlange XMS .NET beim Abfragen der Attribute **BOTHRESH** und **BOQNAME** verwenden, hängt von der Art der durchgeführten Nachrichtenübermittlung (Messaging) ab:

- Beim Punkt-zu-Punkt-Messaging handelt es sich um die zugrunde liegende lokale Warteschlange. Dies ist wichtig, wenn eine XMS .NET-Anwendung Nachrichten aus Aliaswarteschlangen oder Clusterwarteschlangen verarbeitet.
- Beim Publish/Subscribe-Messaging wird eine verwaltete Warteschlange erstellt, in der die Nachrichten für eine Anwendung aufbewahrt werden. XMS .NET fragt die verwaltete Warteschlange ab, um die Werte der Attribute **BOTHRESH** und **BOQNAME** zu ermitteln.

  Die verwaltete Warteschlange wird aus einer Modellwarteschlange erstellt, die dem Topic-Objekt zugeordnet ist, das von der Anwendung subskribiert wurde, und sie übernimmt die Werte der Attribute **BOTHRESH** und **BOQNAME** aus der Modellwarteschlange. Welche Modellwarteschlange verwendet wird, hängt davon ab, ob die empfangende Anwendung eine permanente oder nicht permanente Subskription eingerichtet hat:

  – Die für permanente Subskriptionen verwendete Modellwarteschlange wird durch das Attribut **MDURMDL** des Topics angegeben. Der Standardwert dieses Attributs ist SYSTEM.DURABLE.MODEL.QUEUE.
  – Für nicht permanente Subskriptionen wird die verwendete Modellwarteschlange durch das Attribut **MNDURMDL** angegeben. Der Standardwert des Attributs **MNDURMDL** ist SYSTEM.NDURABLE.MODEL.QUEUE.

Beim Abfragen der Attribute **BOTHRESH** und **BOQNAME** geht XMS .NET wie folgt vor:

- Es wird die lokale Warteschlange oder die Zielwarteschlange für eine Aliaswarteschlange geöffnet.
- Die Attribute **BOTHRESH** und **BOQNAME** werden abgefragt.
- Die lokale Warteschlange oder die Zielwarteschlange für eine Aliaswarteschlange wird geschlossen.

Welche Optionen beim Öffnen einer lokalen Warteschlange oder der Zielwarteschlange für eine Aliaswarteschlange verwendet werden, hängt von der verwendeten Version von IBM MQ ab:

- Für IBM MQ 9.1.0 Fix Pack 4 Long Term Support und früher und für IBM MQ 9.1.4 Continuous Delivery und früher gilt: Wenn die lokale Warteschlange oder die Zielwarteschlange für eine Aliaswarteschlan-

ge eine Clusterwarteschlange ist, öffnet XMS .NET die Warteschlange mit den Optionen MQOO_INPUT_AS_Q_DEF, MQOO_INQUIRE und MQOO_FAIL_IF_QUIESCING. Dies bedeutet, dass der Benutzer, der die empfangende Anwendung ausführt, über Abfrage- und Abrufzugriff auf die lokale Instanz der Clusterwarteschlange verfügen muss.

XMS .NET öffnet alle anderen Arten von lokalen Warteschlangen mit den Optionen MQOO_INQUIRE und MQOO_FAIL_IF_QUIESCING. Damit XMS .NET die Werte der Attribute abfragen kann, muss der Benutzer, der die empfangende Anwendung ausführt, über Abfragezugriff auf die lokale Warteschlange verfügen.

Wenn Sie nicht verarbeitbare Nachrichten in eine Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten oder in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers verschieben möchten, müssen Sie dem Benutzer, der die Anwendung ausführt, die Berechtigungen put und passall erteilen.

Behandlung nicht verarbeitbarer Nachrichten in ASF

Wenn Sie Anwendungsserverfunktionen (Application Server Facilities, ASF) verwenden, werden nicht verarbeitbare Nachrichten nicht vom MessageConsumer, sondern vom ConnectionConsumer verarbeitet. Der ConnectionConsumer stellt Nachrichten entsprechend den **BackoutThreshold** -und **BackoutRequeueQName** -Eigenschaften der Warteschlange erneut in die Warteschlange.

Wenn eine Anwendung ConnectionConsumers verwendet, hängen die Umstände, unter denen eine Nachricht zurückgesetzt wird, von der Sitzung ab, die der Anwendungsserver bereitstellt:

- Wenn die Sitzung nicht transaktionsbasiert und AUTO_ACKNOWLEDGE oder DUPS_OK_ACKNOWLEDGE festgelegt ist, wird eine Nachricht nur nach einem Systemfehler oder einer unerwarteten Beendigung der Anwendung zurückgesetzt.
- Wenn die Sitzung nicht transaktionsbasiert und CLIENT_ACKNOWLEDGE festgelegt ist, können unbestätigte Nachrichten vom Anwendungsserver mit dem Aufruf von Session.recover() zurückgesetzt werden.

  Normalerweise ruft die Clientimplementierung von MessageListener oder der Anwendungsserver Message.acknowledge() auf. Message.acknowledge() bestätigt alle Nachrichten, die bisher in der Sitzung übermittelt wurden.
- Wenn die Sitzung transaktionsbasiert ist, können unbestätigte Nachrichten vom Anwendungsserver mit dem Aufruf von Session.rollback() zurückgesetzt werden.

Warteschlangenbrowser

Eine Anwendung verwendet einen Warteschlangenbrowser, um die Nachrichten in einer Warteschlange zu durchsuchen, ohne sie zu entfernen.

Um einen Warteschlangenbrowser zu erstellen, ruft eine Anwendung die Methode 'Create Queue Browser' eines ISession-Objekts auf und gibt als Parameter ein Destination-Objekt an, das die zu durchsuchende Warteschlange angibt. Die Anwendung kann einen Warteschlangenbrowser mit oder ohne Nachrichtenselektor erstellen.

Nach dem Erstellen eines Warteschlangenbrowsers kann die Anwendung die Methode 'GetEnumerator' des IQueueBrowser-Objekts aufrufen, um eine Liste der Nachrichten in der Warteschlange abzurufen. Die Methode gibt einen Aufzählungsausdruck zurück, der eine Liste der Nachrichtenobjekte ('Message') enthält. Die Reihenfolge der Message-Objekte in der Liste ist identisch mit der Reihenfolge, in der die Nachrichten aus der Warteschlange abgerufen würden. Die Anwendung kann dann den Aufzählungsausdruck dazu verwenden, alle Nachrichten einzeln nacheinander zu durchsuchen.

Der Aufzählungsausdruck wird dynamisch aktualisiert, wenn Nachrichten in die Warteschlange eingereiht bzw. daraus entfernt werden. Jedes Mal, wenn die Anwendung 'IEnumerator.MoveNext()' aufruft, um die nächste Nachricht in der Warteschlange zu durchsuchen, gibt die Nachricht den aktuellen Inhalt der Warteschlange wieder.

Eine Anwendung kann die GetEnumerator-Methode für einen bestimmten Warteschlangenbrowser mehrmals aufrufen. Bei jedem Aufruf wird ein neuer Aufzählungsausdruck zurückgegeben. Daher kann eine

Anwendung mehrere Aufzählungsausdrücke zum Durchsuchen einer Warteschlange verwenden und dadurch mehrere Positionen in der Warteschlange kennzeichnen.

Mithilfe eines Warteschlangenbrowsers kann eine Anwendung nach einer bestimmten Nachricht suchen, die aus einer Warteschlange entfernt werden soll, und dann einen Nachrichtenkonsumenten mit einem Nachrichtenselektor dazu verwenden, die Nachricht zu löschen. Der Nachrichtenselektor kann die Nachricht mithilfe des Werts im Headerfeld 'JMSMessageID' auswählen. Weitere Informationen zu diesem anderen JMS-Nachrichtenheaderfeldern finden Sie im Abschnitt „Headerfelder in einer XMS-Nachricht" auf Seite 702.

Anforderer

Eine Anwendung verwendet einen Anforderer, um eine Anforderungsnachricht zu senden und dann auf eine Antwort zu warten und diese zu empfangen.

Viele Messaging-Anwendungen basieren auf Algorithmen, die eine Anforderungsnachricht senden und dann auf eine Antwort warten. Zur Unterstützung der Entwicklung dieser Art von Anwendung stellt XMS die Klasse 'Requestor' bereit.

Um einen Anforderer zu erstellen, ruft eine Anwendung den Konstruktor 'Create Requestor' oder die Klasse 'Requestor' auf und gibt als Parameter ein Session-Objekt an sowie ein Destination-Objekt, das angibt, wohin die Anforderungsnachricht gesendet werden soll. Die Sitzung darf nicht transaktionsbasiert sein und nicht den Bestätigungsmodus 'XMSC_CLIENT_ACKNOWLEDGE' haben. Der Konstruktor erstellt automatisch eine temporäre Warteschlange oder ein temporäres Thema, an die oder das die Antwortnachrichten gesendet werden sollen.

Nach dem Erstellen eines Anforderers kann die Anwendung die Request-Methode des Requestor-Objekts aufrufen, um eine Anforderungsnachricht zu senden und dann eine Antwort von der Anwendung, die die Anforderungsnachricht empfangen hat, zu erwarten und zu empfangen. Der Aufruf wartet, bis die Antwort empfangen wurde oder bis die Sitzung endet, je nachdem, welches Ereignis zuerst eintritt. Für den Anforderer ist nur eine Antwort für jede Anforderungsnachricht erforderlich.

Wenn die Anwendung den Anforderer schließt, wird die temporäre Warteschlange bzw. das temporäre Thema gelöscht. Die zugeordnete Sitzung wird jedoch nicht geschlossen.

Objektlöschung

Wenn eine Anwendung ein von ihr erstelltes XMS-Objekt löscht, gibt XMS die internen Ressourcen frei, die dem Objekt zugeordnet waren.

Wenn eine Anwendung ein XMS-Objekt erstellt, ordnet XMS dem Objekt Speicherplatz und andere interne Ressourcen zu. XMS behält diese internen Ressourcen so lange bei, bis die Anwendung das Objekt explizit löscht, indem sie die Methode zum Schließen oder Löschen des Objekts aufruft. Erst dann gibt XMS die internen Ressourcen frei. Wenn eine Anwendung versucht, ein Objekt zu löschen, das bereits gelöscht ist, wird der entsprechende Aufruf ignoriert.

Wenn ein Connection- oder Session-Objekt durch eine Anwendung gelöscht wird, löscht XMS bestimmte zugeordnete Objekte automatisch und gibt deren interne Ressourcen frei. Diese Objekte wurden vom jeweiligen Connection- oder Session-Objekt erstellt und haben keine von diesem Objekt unabhängige Funktion. Diese Objekte sind in Tabelle 83 auf Seite 683 aufgeführt.

Anmerkung: Wenn eine Anwendung eine Verbindung mit abhängigen Sitzungen schließt, werden auch alle von diesen Sitzungen abhängenden Objekte gelöscht. Nur Connection- oder Session-Objekte können abhängige Objekte haben.

Tabelle 83. Automatisch gelöschte Objekte

Gelöschtes Objekt	Methode	Abhängige Objekte, die automatisch gelöscht werden
Verbindung	Close Connection (Verbindung schließen)	ConnectionMetaData- und Session-Objekte

Tabelle 83. Automatisch gelöschte Objekte (Forts.)		
Gelöschtes Objekt	**Methode**	**Abhängige Objekte, die automatisch gelöscht werden**
Sitzung	Close Session (Sitzung schließen)	MessageConsumer-, MessageProducer-, QueueBrowser- und Requestor-Objekte

Datentypen für XMS.NET

XMS .NET unterstützt die Datentypen 'System.Boolean', 'System.Byte', 'System.SByte', 'System.Char', 'System.String', 'System.Single', 'System.Double', 'System.Decimal', 'System.Int16', 'System.Int32', 'System.Int64', 'System.UInt16', 'System.UInt32', 'System.UInt64' und 'System.Object'. Datentypen für XMS .NET unterscheiden sich von den Datentypen für XMS C/C + +. Dieser Abschnitt informiert Sie über die entsprechenden Datentypen.

Die folgende Tabelle enthält die einander entsprechenden Datentypen für XMS .NET und für XMS C/C++ sowie eine kurze Beschreibung der Datentypen.

Tabelle 84. Datentypen für XMS .NET und für XMS C/C++		
XMS .NET-Typ	**XMS C/C++-Typ**	**Beschreibung**
System.SByte	xmsSBYTE xmsINT8	8-Bit-Wert mit Vorzeichen
System.Byte	xmsBYTE xmsUINT8	8-Bit-Wert ohne Vorzeichen
System.Int16	xmsINT16 xmsSHORT	16-Bit-Wert mit Vorzeichen
System.UInt16	xmsUINT16 xmsUSHORT	16-Bit-Wert ohne Vorzeichen
System.Int32	xmsINT32 xmsINT	32-Bit-Wert mit Vorzeichen
System.UInt32	xmsUINT32 xmsUINT	32-Bit-Wert ohne Vorzeichen
System.Int64	xmsLONG xmsINT64	64-Bit-Wert mit Vorzeichen
System.UInt64	xmsULONG xmsUINT64	64-Bit-Wert ohne Vorzeichen
System.Char	xmsCHAR16	16-Bit-Zeichen ohne Vorzeichen (Unicode für .NET)
System.Single	xmsFLOAT	32-Bit-IEEE-Gleitkommazahl
System.Double	xmsDOUBLE	64-Bit-IEEE-Gleitkommazahl
System.Boolean	xmsBOOL	True/False-Wert (Wahr/Falsch)
Nicht zutreffend	xmsCHAR	8-Bit-Wert mit oder ohne Vorzeichen (mit/ohne Vorzeichen hängt von der Plattform ab)

Tabelle 84. Datentypen für XMS .NET und für XMS C/C++ (Forts.)		
XMS .NET-Typ	**XMS C/C++-Typ**	**Beschreibung**
System.Decimal	Nicht zutreffend	96-Bit-Ganzzahl multipliziert mit 10^0 bis 10^{28}
System.Object	Nicht zutreffend	Basis aller Typen
System.String	Nicht zutreffend	Zeichenfolgedatentyp

Primitive XMS-Datentypen

XMS stellt Datentypen bereit, die den acht primitiven Java-Datentypen entsprechen: 'byte', 'short', 'int', 'long', 'float', 'double', 'char' und 'boolean'. Dies ermöglicht den Austausch von Nachrichten zwischen XMS und JMS, ohne dass Daten verloren gehen oder beschädigt werden.

In Tabelle 85 auf Seite 685 sind die den Java-Datentypen entsprechenden primitiven XMS-Datentypen einschließlich Größe, Mindest- und Höchstwert aufgelistet.

Tabelle 85. XMS-Datentypen und deren Java-Entsprechungen				
XMS-Datentyp	**Kompatibler Java-Datentyp**	**Größe**	**Mindestwert**	**Höchstwert**
System.Boolean	boolean	32 Bit	false	true
System.SBYTE	Byte	8 Bit	-2^7 (-128)	2^7-1 (127)
System.BYTE	Byte	8 Bit	-2^7 (-128)	2^7-1 (127)
System.CHAR	Byte	8 Bit	-2^7 (-128)	2^7-1 (127)
System.Int16	short	16 Bit	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	int	32 Bit	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	lang	64 Bit	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	float	32 Bit	-3.402823E-38 (mit 7-stelliger Genauigkeit)	3.402823E+38 (mit 7-stelliger Genauigkeit)
System.Double	double	64 Bit	-1.79769313486231E-308 (mit 15-stelliger Genauigkeit)	1.79769313486231E+308 (mit 15-stelliger Genauigkeit)

Implizite Konvertierung eines Eigenschaftswerts von einem Datentyp in einen anderen Datentyp

Wenn eine Anwendung den Wert einer Eigenschaft abruft, kann der Wert von XMS in einen anderen Datentyp konvertiert werden. Es gibt viele Regeln, mit denen gesteuert wird, welche Konvertierungen unterstützt werden und wie XMS die Konvertierungen ausführt.

Eine Eigenschaft eines Objekts hat einen Namen und einen Wert. Dem Wert ist ein bestimmter Datentyp zugeordnet. Der Wert einer Eigenschaft wird auch als *Eigenschaftstyp* bezeichnet.

Eine Anwendung verwendet die Methoden der Klasse 'PropertyContext', um Eigenschaften von Objekten abzurufen und festzulegen. Um den Wert einer Eigenschaft abzurufen, ruft eine Anwendung die für den Eigenschaftstyp geeignete Methode auf. Um beispielsweise den Wert einer ganzzahligen Eigenschaft (Integer) abzurufen, ruft eine Anwendung normalerweise die Methode 'GetIntProperty' auf.

Wenn eine Anwendung den Wert einer Eigenschaft abruft, kann der Wert jedoch von XMS in einen anderen Datentyp konvertiert werden. Wenn beispielsweise der Wert einer ganzzahligen Eigenschaft (Integer) abgerufen werden soll, kann eine Anwendung die Methode 'GetStringProperty' aufrufen, sodass

der Eigenschaftswert als Zeichenfolge zurückgegeben wird. Die von XMS unterstützten Konvertierungen sind in Tabelle 86 auf Seite 686 dargestellt.

Tabelle 86. Unterstützte Konvertierungen von einem Eigenschaftstyp in andere Datentypen

Eigenschaftstyp	**Unterstützte Zieldatentypen**
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte array	System.String
System.Int16	System.String, System.Int32, System.Int64

Folgende allgemeine Regeln steuern die unterstützten Konvertierungen:

- Numerische Eigenschaftswerte können von einem Datentyp in einen anderen Datentyp konvertiert werden, vorausgesetzt, bei der Konvertierung gehen keine Daten verloren. So kann beispielsweise ein Eigenschaftswert mit dem Datentyp 'System.Int32' in einen Wert mit dem Datentyp 'System.Int64' konvertiert werden, es ist jedoch nicht möglich, ihn in einen Wert mit dem Datentyp 'System.Int16' zu konvertieren.
- Ein Eigenschaftswert eines Datentyps kann in eine Zeichenfolge umgewandelt werden.
- Ein Zeichenfolgen-Eigenschaftswert kann in jeden anderen Datentyp konvertiert werden, vorausgesetzt, die Zeichenfolge wird für die Konvertierung korrekt formatiert. Wenn eine Anwendung versucht, eine Zeichenfolge zu konvertieren, die nicht das richtige Format aufweist, gibt XMS möglicherweise Fehler zurück.
- Wenn eine Anwendung eine nicht unterstützte Konvertierung versucht, gibt XMS möglicherweise einen Fehler zurück.

Folgende Regeln gelten für die Konvertierung eines Eigenschaftswerts von einem Datentyp in einen anderen:

- Bei der Konvertierung eines booleschen Eigenschaftswerts in eine Zeichenfolge wird der Wert 'true' in die Zeichenfolge "true" und der Wert 'false' in die Zeichenfolge "false" konvertiert.
- Beim Konvertieren eines booleschen Eigenschaftswerts in einen numerischen Datentyp einschließlich 'System.SByte' wird der Wert 'true' in den Wert '1' und der Wert 'false' in den Wert '0' konvertiert.
- Beim Konvertieren eines Zeichenfolgeeigenschaftswerts in einen booleschen Wert wird die Zeichenfolge "true" (ohne Beachtung der Groß-/Kleinschreibung) oder der Wert "1" in den booleschen Wert 'true' und die Zeichenfolge "false" (ohne Beachtung der Groß-/Kleinschreibung) oder der Wert "0" in den booleschen Wert 'false' konvertiert. Alle anderen Zeichenfolgen können nicht konvertiert werden.
- Beim Konvertieren eines Zeichenfolgeeigenschaftswerts in einen Wert des Datentyps 'System.Int32', 'System.Int64', 'System.SByte' oder 'System.Int16' muss die Zeichenfolge das folgende Format haben:

  [*blanks*][*sign*] *digits*

  Die Zeichenfolgenkomponenten sind wie folgt definiert:

  blanks
  Optionale führende Leerzeichen.

sign
Ein optionales Pluszeichen (+) oder Minuszeichen (-).

Ziffern
Eine zusammenhängende Folge von Ziffern (0-9). Es muss mindestens ein Ziffernzeichen vorhanden sein.

Auf die Ziffernfolge können weitere Zeichen folgen, bei denen es sich nicht um Ziffern handelt, doch wird die Konvertierung beendet, sowie das erste dieser Zeichen erreicht wird. Es wird vorausgesetzt, dass die Zeichenfolge eine Ganzzahl im Dezimalformat darstellt.

Wenn die Zeichenfolge nicht korrekt formatiert ist, gibt XMS möglicherweise einen Fehler zurück.

- Beim Konvertieren eines Zeichenfolgeeigenschaftswerts in einen Wert des Datentyps 'System.Double' oder 'System.Float' muss die Zeichenfolge das folgende Format haben:

  [*blanks*][*sign*][*digits*][*point*[*d_digits*]][*e_char*[*e_sign*]*e_digits*]

  Die Zeichenfolgenkomponenten sind wie folgt definiert:

  blanks
  Optionale führende Leerzeichen.

  sign
  Optionales Pluszeichen (+) oder Minuszeichen (-).

  Ziffern
  Eine zusammenhängende Folge von Ziffern (0-9). In einer der Komponenten *digits* oder *d_digits* muss mindestens ein Ziffernzeichen vorhanden sein.

  Point
  (Optional) Dezimalzeichen (.).

  d_digits
  Eine zusammenhängende Folge von Ziffern (0-9). In einer der Komponenten *digits* oder *d_digits* muss mindestens ein Ziffernzeichen vorhanden sein.

  e_Zeich
  Ein Exponentenzeichen, entweder *E* oder *e*.

  e_Vorz
  Optionales Pluszeichen (+) oder Minuszeichen (-) für den Exponenten.

  e_Ziffern
  Eine zusammenhängende Folge von Ziffern (0-9) für den Exponenten. Mindestens eine Ziffer muss vorhanden sein, wenn die Zeichenfolge ein Exponentenzeichen enthält.

  Auf die Ziffernfolge bzw. die optionalen, einen Exponenten darstellenden Zeichen, können weitere Zeichen folgen, bei denen es sich nicht um Ziffern handelt, doch wird die Konvertierung beendet, sowie das erste dieser Zeichen erreicht wird. Es wird davon ausgegangen, dass die Zeichenfolge eine Gleitkommazahl mit einem Exponenten der Potenz 10 darstellt.

  Wenn die Zeichenfolge nicht korrekt formatiert ist, gibt XMS möglicherweise einen Fehler zurück.

- Beim Konvertieren eines numerischen Eigenschaftswerts einschließlich Werte des Datentyps 'System.SByte' in eine Zeichenfolge wird der Wert in die Zeichenfolgedarstellung des Werts als Dezimalzahl konvertiert, nicht in die Zeichenfolge, die den ASCII-Zeichen für diesen Wert enthält. Die Ganzzahl 65 wird beispielsweise in die Zeichenfolge "65" konvertiert, nicht in die Zeichenfolge "A".
- Beim Konvertieren eines Byte-Array-Eigenschaftswerts in eine Zeichenfolge wird jedes Byte in die zwei Hexadezimalzeichen konvertiert, die das Byte darstellen. Beispiel: Das Byte-Array {0xF1, 0x12, 0x00, 0xFF} wird in die Zeichenfolge "F11200FF" konvertiert.

Konvertierungen von einem Eigenschaftstyp in andere Datentypen werden von den Methoden sowohl der Property- als auch der PropertyContext-Klassen unterstützt.

Iteratoren

Ein Iterator enthält eine Liste mit Objekten sowie einen Cursor, der die aktuelle Position in der Liste kennzeichnet. Das Konzept eines Iterators, wie er in IBM MQ Message Service Client (XMS) for C/C++ verfügbar ist, wird durch die Verwendung der Schnittstelle 'IEnumerator' in IBM MQ Message Service Client (XMS) for .NET implementiert.

Beim Erstellen eines Iterators wird der Cursor vor dem ersten Objekt positioniert. Eine Anwendung verwendet einen Iterator, um alle Objekte einzeln der Reihe nach abzurufen.

Die Iterator-Klasse von IBM MQ Message Service Client (XMS) for C/C++ entspricht funktional der Enumerator-Klasse in Java. IBM MQ Message Service Client (XMS) for .NET ist Java ähnlich und verwendet eine IEnumerator-Schnittstelle.

Eine Anwendung kann mit einem IEnumerator-Element folgende Tasks ausführen:

- Eigenschaften einer Nachricht abrufen
- Name/Wert-Paare im Hauptteil einer Zuordnungsnachricht abrufen
- Nachrichten in einer Warteschlange durchsuchen
- Namen der in JMS definierten und von einer Verbindung unterstützten Nachrichteneigenschaften abrufen

Fehlerbehandlung in XMS .NET

XMS .NET Ausnahmebedingungen werden alle von System.Exceptionabgeleitet.

XMS .NETAusnahmebedingungen

XMS-Methodenaufrufe können spezielle XMS-Ausnahmebedingungen auslösen, wie z. B. 'MessageFormatException', allgemeine Ausnahmen des Typs 'XMSException' oder Systemausnahmen wie 'NullReferenceException'.

Schreiben Sie Anwendungen, um diese Fehler abzufangen, entweder in bestimmten Catch-Blöcken oder in allgemeinen `System.Exception` -Catch-Blöcken, je nach den Anforderungen Ihrer Anwendung.

XMS-Fehler- und -Ausnahmecodes

XMS umfasst zahlreiche Fehlercodes, um auf Fehler hinzuweisen. Die Fehlercodes werden in dieser Dokumentation nicht explizit aufgeführt, weil sie je nach Release unterschiedlich sein können. Nur die XMS-Ausnahmecodes (im Format 'XMS_X_...') sind dokumentiert, weil in allen XMS-Releases identisch sind.

Zugehörige Informationen

MQException.NET -Klasse

Allgemeine SSL-Fehlercodes, die von XMS .NET -Clientbibliotheken ausgelöst werden

FFDC-Konfiguration für XMS .NET-Anwendungen

Tracing von XMS .NET-Anwendungen

Listener für Nachrichten und Ausnahmen in .NET verwenden

Eine .NET-Anwendung verwendet einen Nachrichtenlistener, um Nachrichten asynchron zu empfangen, und einen Ausnahmelistener, um asynchron über Probleme mit der Verbindung benachrichtigt zu werden.

Informationen zu diesem Vorgang

Die Nachrichten- und Ausnahmelistener für .NET haben dieselbe Funktionalität wie die Listener für C++. Es gibt jedoch einige geringfügige Implementierungsunterschiede.

Prozedur

- Führen Sie folgende Schritte aus, um einen Nachrichtenlistener für den asynchronen Empfang zu konfigurieren:
  a) Definieren Sie eine Methode, die mit der Signatur des Nachrichtenlistenerdelegierten übereinstimmt.

     Die von Ihnen definierte Methode kann statisch oder eine Instanzdefinitionsmethode sein und in jeder beliebigen, zugänglichen Klasse definiert werden. Die Delegiertensignatur lautet wie folgt:

     ```
     public delegate void MessageListener(IMessage msg);
     ```

     Daher können Sie die Methode wie folgt definieren:

     ```
     void SomeMethodName(IMessage msg);
     ```

  b) Instanziieren Sie diese Methode als Delegierten mithilfe einer Anweisung ähnlich dem folgenden Beispiel:

     ```
     MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
     ```

  c) Registrieren Sie den Delegierten bei mindestens einem Konsumenten, indem Sie ihn in der Eigenschaft 'MessageListener' des Konsumenten angeben:

     ```
     consumer.MessageListener = OnMsgMethod;
     ```

     Sie können den Delegierten entfernen, indem Sie die Eigenschaft 'MessageListener' wieder auf den Wert 'null' setzen:

     ```
     consumer.MessageListener = null;
     ```

- Führen Sie die folgenden Schritte aus, um einen Ausnahmelistener zu konfigurieren.

  Der Ausnahmelistener funktioniert in ähnlicher Weise wie der Nachrichtenlistener, hat jedoch eine andere Delegiertendefinition und wird nicht dem Nachrichtenkonsumenten, sondern der Verbindung zugeordnet. Dies ist mit C++ identisch.
  a) Definieren Sie die Methode.

     Die Delegiertensignatur lautet wie folgt:

     ```
     public delegate void ExceptionListener(Exception ex);
     ```

     Daher können Sie die Methode wie folgt definieren:

     ```
     void SomeMethodName(Exception ex);
     ```

  b) Instanziieren Sie diese Methode als Delegierten mithilfe einer Anweisung ähnlich dem folgenden Beispiel:

     ```
     ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
     ```

  c) Registrieren Sie den Delegierten bei der Verbindung, indem Sie deren Eigenschaft 'ExceptionListener' wie folgt festlegen:

     ```
     connection.ExceptionListener = OnExMethod ;
     ```

     Sie können den Delegierten entfernen, indem Sie die Eigenschaft 'ExceptionListener' wieder auf den Wert 'null' setzen:

```
null:  connection.ExceptionListener = null;
```

Automatische IBM MQ-Client-Verbindungswiederholung mit XMS

Konfigurieren Sie Ihren XMS-Client so, dass er die Verbindung nach einem Netz-, Warteschlangenmanager- oder Serverfehler automatisch wiederherstellt und dabei ein Client mit IBM WebSphere MQ 7.1 oder höher als Messaging-Provider verwendet wird.

Mit den Eigenschaften `WMQ_CONNECTION_NAME_LIST` und `WMQ_CLIENT_RECONNECT_OPTIONS` der Klasse `MQConnectionFactory` können Sie die automatische Wiederherstellung einer Clientverbindung konfigurieren. Bei Aktivierung der automatischen Clientverbindungswiederholung wird eine Clientverbindung nach einem Verbindungsfehler oder optional nach dem Stoppen des Warteschlangenmanagers automatisch wiederhergestellt. Einige Clientanwendungen sind aufgrund ihres Designs für die automatische Verbindungswiederherstellung ungeeignet.

Automatisch wiederverbindungsfähige Clientverbindungen werden unmittelbar nach dem Einrichten der Verbindung wiederverbindungsfähig.

Anmerkung: Die Eigenschaften `Client Reconnect Options` (Clientwiederverbindungsoptionen), `Client Reconnect Timeout` (Zeitlimit für Clientverbindungswiederholung) und `Connection Name-list` (Verbindungsnamensliste) können auch über eine CCDT (Definitionstabelle für Clientkanal) oder durch Aktivieren der Clientverbindungswiederholung in der Datei `mqclient.ini` festgelegt werden.

Anmerkung: Wenn sowohl im Objekt `ConnectionFactory` als auch in der CCDT Verbindungswiederholungseigenschaften festgelegt sind, gilt folgende Vorrangregel. Wenn die Eigenschaft der Verbindungsnamensliste im Objekt `ConnectionFactory` auf den Standardwert festgelegt ist, hat die CCDT Vorrang. Wenn die Verbindungsnamensliste hingegen nicht auf den Standardwert festgelegt ist, haben die im Objekt `ConnectionFactory` festgelegten Eigenschaftswerte Vorrang. Der Standardwert der Verbindungsnamensliste lautet `localhost(1414)`.

Mit von XMS .NET verwalteten Objekten arbeiten

Die Themen in diesem Abschnitt enthalten Informationen zu verwalteten Objekten. XMS-Anwendungen können Objektdefinitionen aus einem zentralen Repository für verwaltete Objekte abrufen und zum Erstellen von Verbindungsfactorys und Zielen verwenden.

Informationen zu diesem Vorgang

Dieser Abschnitt enthält Informationen, die Sie bei der Erstellung und Handhabung von verwalteten Objekten unterstützen, unter anderem eine Beschreibung der verschiedenen Repositorytypen für verwaltete Objekte, die XMS unterstützt. Außerdem wird in diesem Abschnitt erläutert, wie eine XMS-Anwendung eine Verbindung zu einem Repository für verwaltete Objekte herstellt, um die erforderlichen verwalteten Objekte abzurufen.

Dieser Abschnitt enthält die folgenden Themen:

Von XMS .NET unterstützte Repositorytypen für verwaltete Objekte

Mit verwalteten Dateisystemobjekten und LDAP-Objekten können Verbindungen zu IBM MQ und WebSphere Application Server hergestellt werden, während COS-Benennungen nur Verbindungen zu WebSphere Application Server ermöglichen.

Verzeichnisse für Dateisystemobjekte haben das Format serialisierter Objekte von Java Naming Directory Interface (JNDI). LDAP-Objektverzeichnisse enthalten JNDI-Objekte. Dateisystem-und LDAP-Objektverzeichnisse können von der IBM MQ Explorerverwaltet werden, die mit IBM MQ und höher bereitgestellt wird. Das Dateisystem und die LDAP-Objektverzeichnisse können verwendet werden, um Clientverbindungen zu verwalten, indem IBM MQ -Verbindungsfactorys und -Ziele zentralisiert werden. Der Netzadministrator kann mehrere Anwendungen bereitstellen, die auf dasselbe zentrale Repository verweisen und automatisch aktualisiert werden, wenn im zentralen Repository Änderungen an den Verbindungseinstellungen vorgenommen werden.

Ein COS-Benennungsverzeichnis (CORBA Object Services Naming Directory) enthält WebSphere Application Server service integration bus-Verbindungsfactorys und -Ziele und kann über die Administrationskonsole für WebSphere Application Server verwaltet werden. Damit eine XMS-Anwendung Objekte aus dem COS-Benennungsverzeichnis abrufen kann, muss ein Web-Service für die JNDI-Suche bereitgestellt werden. Dieser Web-Service ist nicht in allen WebSphere Application Server service integration technologies verfügbar. Detaillierte Informationen finden Sie in der entsprechenden Produktdokumentation.

Anmerkung: Damit Änderungen am Objektverzeichnis wirksam werden, müssen die Anwendungsverbindungen anschließend neu gestartet werden.

XMS .NET-Eigenschaftszuordnung für verwaltete Objekte

Damit XMS .NET-Anwendungen die Objektdefinitionen von Verbindungsfactorys und Zielen für IBM MQ JMS und WebSphere Application Server verwenden können, müssen die aus diesen Definitionen abgerufenen Eigenschaften den entsprechenden XMS-Eigenschaften zugeordnet werden, die in den XMS-Verbindungsfactorys und -Zielen festgelegt werden können.

Um beispielsweise eine XMS-Verbindungsfactory mit den aus einer IBM MQ JMS-Verbindungsfactory abgerufenen Eigenschaften zu erstellen, müssen die entsprechenden Eigenschaften aus beiden Quellen einander zugeordnet werden.

Alle Eigenschaftszuordnungen werden automatisch ausgeführt.

Tabelle 87 auf Seite 691 veranschaulicht die Zuordnungen zwischen einigen der gängigsten Eigenschaften von Verbindungsfactorys und Zielen. Die in dieser Tabelle angezeigten Eigenschaften sind nur eine kleine Auswahl von Beispielen und nicht alle der aufgeführten Eigenschaften sind für alle Verbindungstypen und Server relevant.

Tabelle 87. Beispiele für die Namenszuordnung von Eigenschaften für Verbindungsfactorys und Ziele

IBM MQ JMS-Eigenschaftsname	**XMS-Eigenschaftsname**	**WebSphere Application Server service integration bus-Eigenschaftsname**
PERSISTENCE (PER)	XMSC_DELIVERY_MODE	
EXPIRY (EXP)	XMSC_TIME_TO_LIVE	
PRIORITY (PRI)	XMSC_PRIORITY	
	XMSC_WPM_HOST_NAME	serverName
	XMSC_WPM_BUS_NAME	busName
	XMSC_WPM_TOPIC_SPACE	topicName

Anmerkung: Die in Tabelle 88 auf Seite 692 gezeigten Beispiele gelten auch für JMS und XMS .NET.

Tabelle 88. XMS .NETEigenschaften

Eigenschaft	Objekttyp				
	CF	**QCF**	**TCF**	**Warteschlange**	**Thema**
APPLICATIONNAME	Y	Y	Y	nicht zutreffend	nicht zutreffend
ASYNCEXCEPTION	Y	Y	Y	nicht zutreffend	nicht zutreffend
CCDTURL	Y	Y	Y	nicht zutreffend	nicht zutreffend
CHANNEL	Y	Y	Y	nicht zutreffend	nicht zutreffend
CONNECTIONNAMELIST	Y	Y	Y	nicht zutreffend	nicht zutreffend
CLIENTRECONNECTOPTIONS	Y	Y	Y	nicht zutreffend	nicht zutreffend
CLIENTRECONNECTTIMEOUT	Y	Y	Y	nicht zutreffend	nicht zutreffend
CLIENTID	nicht zutreffend	Y	nicht zutreffend	nicht zutreffend	nicht zutreffend
COMPHDR „1" auf Seite 693	Y	nicht zutreffend	Y	nicht zutreffend	nicht zutreffend
COMPMSG „1" auf Seite 693	Y	Y	Y	nicht zutreffend	nicht zutreffend
KONNOPT „1" auf Seite 693	Y	Y	Y	nicht zutreffend	nicht zutreffend
CONNTAG „1" auf Seite 693	Y	Y	Y	nicht zutreffend	nicht zutreffend
BESCHREIBUNG „1" auf Seite 693	nicht zutreffend	Y	nicht zutreffend	Y	Y
ABLAUF „1" auf Seite 693	nicht zutreffend	nicht zutreffend	nicht zutreffend	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
HOSTNAME	nicht zutreffend	Y	nicht zutreffend	nicht zutreffend	nicht zutreffend
LOCALADDRESS	nicht zutreffend	Y	nicht zutreffend	nicht zutreffend	nicht zutreffend
PERSISTENCE	nicht zutreffend	nicht zutreffend	nicht zutreffend	Y	Y
PORT	nicht zutreffend	Y	nicht zutreffend	nicht zutreffend	nicht zutreffend
Priorität „1" auf Seite 693	nicht zutreffend	nicht zutreffend	nicht zutreffend	Y	Y
PROVIDERVERSION „1" auf Seite 693	nicht zutreffend	Y	nicht zutreffend	nicht zutreffend	nicht zutreffend
QMANAGER	Y	Y	Y	Y	nicht zutreffend

Tabelle 88. XMS .NETEigenschaften (Forts.)					
Eigenschaft	**Objekttyp**				
	CF	**QCF**	**TCF**	**Warteschlange**	**Thema**
Warteschlange „1" auf Seite 693	nicht zutreffend	nicht zutreffend	nicht zutreffend	Y	nicht zutreffend
SHARECONVALLOWED	Y	Y	Y	nicht zutreffend	nicht zutreffend
THEMA „1" auf Seite 693	nicht zutreffend	nicht zutreffend	nicht zutreffend	nicht zutreffend	Y
Transport „1" auf Seite 693	nicht zutreffend	Y	nicht zutreffend	nicht zutreffend	nicht zutreffend

Anmerkung:

1. Diese Eigenschaften haben keine Eigenschaften auf Anwendungsebene, können aber optional mit verwalteten Eigenschaften definiert werden.

`OutboundSNI`Eigenschaft

Ab IBM MQ 9.3.0 können Sie die Eigenschaft XMSC_WMQ_OUTBOUND_SNI festlegen, die die **`OutboundSNI`**-Eigenschaft in einer Anwendung festlegt.

Für die Eigenschaft XMSC_WMQ_OUTBOUND_SNI können die folgenden Werte angegeben werden:

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL (wird CHANNEL zugeordnet)
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME (wird HOSTNAME zugeordnet)
- XMSC_WMQ_OUTBOUND_SNI_ASTERISK (wird * zugeordnet)

Darüber hinaus können Sie die Eigenschaft **`OutboundSNI`** mit der Umgebungsvariablen MQOUTBOUND_SNI festlegen, die folgende Werte hat:

- CHANNEL
- HOSTNAME
- *

Anmerkung: Falls kein bestimmter Wert festgelegt wird, hat die Eigenschaft standardmäßig den Wert XMSC_WMQ_OUTBOUND_SNI_CHANNEL.

Die Vorrangregelung für die Einstellung der Eigenschaft **`OutboundSNI`** im verwalteten Knoten lautet wie folgt:

1. Eigenschaft auf Anwendungsebene
2. Umgebungsvariable

Für die Eigenschaft **`OutboundSNI`** im nicht verwalteten Knoten wird nur `mqclient.ini` unterstützt.

Von XMS .NET benötigte Eigenschaften für verwaltete ConnectionFactory-Objekte

Wenn eine Anwendung eine Verbindungsfactory erstellt, müssen einige Eigenschaften zum Erstellen einer Verbindung zu einem Messaging-Server definiert werden.

Die in der folgenden Tabelle aufgeführten Eigenschaften müssen von einer Anwendung mindestens festgelegt werden, um eine Verbindung zu einem Messaging-Server zu erstellen. Wenn Sie anpassen möchten, wie eine Verbindung erstellt wird, kann Ihre Anwendung beliebige Eigenschaften des Connecti-

onFactory-Objekts zusätzlich festlegen. Weitere Informationen finden Sie im Abschnitt Eigenschaften von ConnectionFactory-Objekten. Dort finden Sie auch eine vollständige Liste der verfügbaren Eigenschaften.

Verbindung zu einem IBM MQ-Warteschlangenmanager

Tabelle 89. Eigenschafteneinstellungen für verwaltete ConnectionFactory-Objekte für Verbindungen zu einem IBM MQ-Warteschlangenmanager

Erforderliche XMS-Eigenschaft	Entsprechende erforderliche IBM MQ JMS-Eigenschaft
XMSC_CONNECTION_TYPE	XMS leitet dies anhand des Klassennamens der Verbindungsfactory und der Eigenschaft 'TRANSPORT' (TRAN) ab.
XMSC_WMQ_HOST_NAME	HOSTNAME (HOST)
XMSC_WMQ_PORT	PORT
XMSC_WMQ_QUEUE_MANAGER	Name des Warteschlangenmanagers

Echtzeitverbindung zu einem Broker

Tabelle 90. Eigenschafteneinstellungen für verwaltete ConnectionFactory-Objekte für Echtzeitverbindungen zu einem Broker

Erforderliche XMS-Eigenschaft	Entsprechende erforderliche IBM MQ JMS-Eigenschaft
XMSC_CONNECTION_TYPE	XMS leitet dies anhand des Klassennamens der Verbindungsfactory und der Eigenschaft 'TRANSPORT' (TRAN) ab.
XMSC_RTT_HOST_NAME	HOSTNAME (HOST)
XMSC_RTT_PORT	PORT

Verbindung zu einem WebSphere Application Server service integration bus

Tabelle 91. Eigenschafteneinstellungen für verwaltete ConnectionFactory-Objekte für Verbindungen zu einem WebSphere Application Server service integration bus

XMSEigenschaft	Beschreibung
XMSC_CONNECTION_TYPE	Der Typ des Messaging-Servers, zu dem eine Anwendung eine Verbindung herstellt.. Dieser Wert wird anhand des Klassennamens für die Verbindungsfactory bestimmt.
XMSC_WPM_BUS_NAME	Für eine Verbindungsfactory der Name des Service Integration Bus, zu dem die Anwendung eine Verbindung herstellt, bzw. für ein Ziel der Name des Service Integration Bus, in dem sich das Ziel befindet.

Von XMS .NET benötigte Eigenschaften für verwaltete Zielobjekte

Eine Anwendung, die ein Ziel erstellt, muss verschiedene Eigenschaften festlegen, die die Anwendung für ein verwaltetes Destination-Objekt verwendet.

Tabelle 92. Eigenschafteneinstellungen für verwaltete Destination-Objekte

Verbindungstyp	Eigenschaft	Beschreibung
Warteschlangenmanager der IBM MQ	QUEUE (QU) TOPIC (TOP)	Die Warteschlange, zu der Sie eine Verbindung herstellen möchten Das Thema, das die Anwendung als Ziel verwendet

Tabelle 92. Eigenschafteneinstellungen für verwaltete Destination-Objekte (Forts.)		
Verbindungstyp	**Eigenschaft**	**Beschreibung**
Echtzeitverbindung zu einem Broker	TOPIC (TOP)	Das Thema, das die Anwendung als Ziel verwendet
WebSphere Application Server service integration bus	topicName queueName	Falls Ihre Anwendung eine Verbindung zu einem Thema herstellen soll Falls Ihre Anwendung eine Verbindung zu einer Warteschlange herstellen soll

XMS .NET - Erstellung von verwalteten Objekten

Die Definitionen der ConnectionFactory- und Destination-Objekte, die XMS-Anwendungen zum Herstellen einer Verbindung zu einem Messaging-Server benötigen, müssen mit den geeigneten Verwaltungstools erstellt werden.

Vorbereitende Schritte

Weitere Informationen zu den verschiedenen Repositorytypen für verwaltete Objekte, die XMS unterstützt, finden Sie im Abschnitt „Von XMS .NET unterstützte Repositorytypen für verwaltete Objekte” auf Seite 691.

Informationen zu diesem Vorgang

Verwenden zum Erstellen von verwalteten Objekten für IBM MQ entweder IBM MQ Explorer oder das IBM MQ JMS-Verwaltungstool (JMSAdmin).

Verwenden Sie zum Erstellen von verwalteten Objekten für IBM MQ oder IBM Integration Bus das IBM MQ JMS-Verwaltungstool (JMSAdmin).

Verwenden Sie zum Erstellen von verwalteten Objekten für den WebSphere Application Server service integration bus die Administrationskonsole für WebSphere Application Server.

In den Verwaltungstools wird die Eigenschaft als **APPLICATIONNAME** oder kurz **APPNAME** bezeichnet.

Anmerkung: 'JMSAdmin' kann nicht zum Festlegen von 'TRANSPORT(UNMANAGED)' verwendet werden. Um einen nicht verwalteten XMS-Client mithilfe eines administrativ ausgewählten Anwendungsnamens abzurufen, müssen Sie daher folgenden Befehl eingeben:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

In der folgenden Vorgehensweise sind die Schritte zusammengefasst, die Sie zum Erstellen von verwalteten Objekten ausgeführen müssen.

Vorgehensweise

1. Erstellen Sie eine Verbindungsfactory und definieren Sie die erforderlichen Eigenschaften, damit eine Verbindung von Ihrer Anwendung zu Ihrem ausgewählten Server hergestellt werden kann.

   Welche Eigenschaften XMS mindestens zum Herstellen einer Verbindung benötigt, ist im Abschnitt „Von XMS .NET benötigte Eigenschaften für verwaltete ConnectionFactory-Objekte” auf Seite 693 beschrieben.
2. Erstellen Sie das erforderliche Ziel auf dem Messaging-Server, zu dem Ihre Anwendung eine Verbindung herstellen soll:
   - Für eine Verbindung zu einem IBM MQ-Warteschlangenmanager erstellen Sie eine Warteschlange oder ein Thema.
   - Für eine Echtzeitverbindung zu einem Broker, erstellen Sie ein Thema.

- Für eine Verbindung zu einem WebSphere Application Server service integration bus erstellen Sie eine Warteschlange oder ein Thema.

Welche Eigenschaften XMS mindestens zum Herstellen einer Verbindung benötigt, ist im Abschnitt „Von XMS .NET benötigte Eigenschaften für verwaltete Zielobjekte” auf Seite 694 beschrieben.

XMS .NET - Erstellung von InitialContext-Objekten

Eine Anwendung muss einen Ausgangskontext erstellen, damit eine Verbindung zum Repository für verwaltete Objekte hergestellt und die erforderlichen verwalteten Objekte von dort abgerufen werden können.

Informationen zu diesem Vorgang

Ein InitialContext-Objekt enthält eine Verbindung zum Repository. Die XMS-API stellt Methoden zum Ausführen folgender Tasks bereit:

- InitialContext-Objekt erstellen
- Verwaltetes Objekt im Repository für verwaltete Objekte suchen

Prozedur

- Weitere Informationen zum Erstellen eines InitialContext-Objekts finden Sie in den Abschnitten Initial-Context-Objekte für .NET und Eigenschaften von InitialContext-Objekten.

InitialContext-Eigenschaften von XMS .NET

Zu den Parametern des Konstruktors 'InitialContext' gehört die Position des Repositorys für verwaltete Objekte, die als Uniform Resource Indicator (URI) angegeben wird. Damit eine Anwendung eine Verbindung zum Repository herstellen kann, sind zusätzlich zu den im URI enthaltenen Informationen möglicherweise weitere Angaben erforderlich.

In JNDI und in der .NET-Implementierung von XMS werden die zusätzlichen Informationen in einer Umgebungs-Hashtabelle für den Konstruktor bereitgestellt.

Die Position des Repositorys für verwaltete Objekte ist in der Eigenschaft XMSC_IC_URL definiert. Diese Eigenschaft wird normalerweise im Aufruf 'Create' übergeben, kann jedoch so geändert werden, dass vor der Suche eine Verbindung zu einem anderen Benennungsverzeichnis hergestellt wird. Im Dateisystem- oder LDAP-Kontext gibt diese Eigenschaft die Verzeichnisadresse an. Bei COS-Benennungen ist dies die Adresse des Web-Services, der mit diesen Eigenschaften eine Verbindung zum JNDI-Verzeichnis herstellt.

Die folgenden Eigenschaften werden unverändert an den Web-Service übergeben, der damit die Verbindung zum JNDI-Verzeichnis herstellt.

- XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

URI-Format für XMS-Ausgangskontexte

Die Position des Repositorys für verwaltete Objekte wird als URI-Wert (Uniform Resource Indicator) bereitgestellt. Das URI-Format hängt dabei vom jeweiligen Kontexttyp ab.

Dateisystemkontext

Im Dateisystemkontext gibt die URL die Position eines Verzeichnisses im Dateisystem an. Die Struktur der URL ist in RFC 1738, *Uniform Resource Locators (URL)*, wie folgt definiert: Die URL hat den Präfix `file://`

gefolgt von der gültigen Definition einer Datei, die auf dem System geöffnet werden kann, auf dem XMS ausgeführt wird.

Diese Syntax ist plattformspezifisch und als Trennzeichen kann entweder '/' oder '\' verwendet werden. Wenn Sie '\' als Trennzeichen verwenden möchten, muss ihm jedes Mal ein zusätzliches '\' als Excapezeichen vorangestellt werden. So wird verhindert, dass das .NET Framework das ursprüngliche Trennzeichen als Excapezeichen für den nachfolgenden Text interpretiert.

Die folgenden Beispiele veranschaulichen diese Syntax:

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

LDAP-Kontext

Für den LDAP-Kontext ist die Grundstruktur der URL im RFC 2255, *The LDAP URL Format* definiert, und die Syntax beginnt mit dem von Groß-/Kleinschreibung unabhängigen Präfix `ldap://`.

Die exakte Syntax ist in folgendem Beispiel dargestellt:

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

Diese Syntax ist zwar in dem genannten RFC definiert, jedoch ohne Unterstützung für jegliche Attribute, Gültigkeitsbereiche, Filter oder Erweiterungen.

Folgende Beispiele veranschaulichen diese Syntax:

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

WSS-Kontext

Im WSS-Kontext hat die URL das Format eines Web-Service-Endpunkts mit dem Präfix `http://`.

Alternativ können Sie das Präfix `cosnaming://` oder `wsvc://` verwenden.

Die Bedeutung dieser beiden Präfixe wird so interpretiert, dass Sie einen WSS-Kontext verwenden, wobei der Zugriff auf die URL über das Hypertext Transfer Protocol (HTTP) erfolgt, sodass der ursprüngliche Kontexttyp einfach direkt aus der URL abgeleitet werden kann.

Folgende Beispiele veranschaulichen diese Syntax:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

Web-Service für die JNDI-Suche für XMS .NET

Damit von XMS aus auf ein COS-Namensverzeichnis zugegriffen werden kann, muss ein Web-Service für die JNDI-Suche auf einem WebSphere Application Server service integration bus-Server implementiert sein. Dieser Web-Service setzt die Java-Informationen aus dem COS-Namensservice in ein von der XMS-Anwendung lesbares Format um.

Der Web-Service wird im Installationsverzeichnis in der EAR-Datei 'SIBXJndiLookupEAR.ear' bereitgestellt. Für das aktuelle Release von IBM MQ Message Service Client (XMS) for .NETbefindet sich SIBXJndiLookupEAR.ear im Verzeichnis *install_dir*`\java\lib` . Die Installation auf einem WebSphere Appli-

cation Server service integration bus-Server kann entweder mit der Administrationskonsole oder mit dem Scripting-Tool 'wsaadmin' ausgeführt werden. Weitere Informationen zum Bereitstellen von Web-Service-Anwendungen finden Sie in der entsprechenden Produktdokumentation.

Um den Web-Service in XMS-Anwendungen zu definieren, geben Sie im InitialContext-Objekt für die Eigenschaft XMSC_IC_URL die URL des Web-Service-Endpunkts an. Wenn der Web-Service beispielsweise auf einem Serverhost mit dem Namen 'MyServer' bereitgestellt wird, lautet die Web-Service-Endpunkt-URL wie folgt:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

Wenn die Eigenschaft 'XMSC_IC_URL' festgelegt wurde, kann der Web-Service auf dem definierten Endpunkt durch InitialContext-Suchaufrufe aufgerufen werden und dann seinerseits das erforderliche verwaltete Objekt im COS-Namensservice suchen.

.NET-Anwendungen können den Web-Service verwenden. Die serverseitige Bereitstellung ist XMS C, /C++ und XMS .NET identisch.XMS .NET ruft den Web-Service direkt über Microsoft .NET Frameworkauf.

XMS .NET-Abruf von verwalteten Objekten

Zum Abrufen eines verwalteten Objekts aus dem Repository verwendet XMS die Adresse, die beim Erstellen des InitialContext-Objekts oder in den InitialContext-Eigenschaften bereitgestellt wurde.

Objekte, die abgerufen werden sollen, können folgende Arten von Namen haben:

- Einen einfachen Namen, der das Destination-Objekt beschreibt, beispielsweise das Warteschlangenziel 'SalesOrders'
- Einen zusammengesetzten Namen, der aus mehreren durch das Trennzeichen '/' getrennten Subkontexten bestehen kann und mit dem Objektnamen enden muss. Ein Beispiel für einen zusammengesetzten Namen ist 'Warehouse/PickLists/DispatchQueue2', wobei 'Warehouse' und 'Picklists' Subkontexte im Benennungsverzeichnis sind und 'DispatchQueue2' der Name eines Destination-Objekts ist.

Verwendung einer neueren XMS-Version durch eine Anwendung verhindern

Wenn eine neuere XMS-Version installiert wird, wechseln Anwendungen, die die vorherige Version verwendet haben, standardmäßig automatisch zur neueren Version, ohne dass sie erneut kompiliert werden müssen. Sie können dies jedoch verhindern, indem Sie ein Attribut in der Anwendungskonfigurationsdatei entsprechend festlegen.

Informationen zu diesem Vorgang

Die Funktion für die Koexistenz mehrerer Versionen stellt sicher, dass die vorherige XMS-Version bei der Installation einer neueren XMS-Version nicht überschrieben wird. Stattdessen können mehrere Instanzen ähnlicher XMS .NET-Assemblys (jedoch mit unterschiedlichen Versionsnummern) gleichzeitig im Global Assembly Cache (GAC) koexistieren. Intern verwendet der GAC eine Richtliniendatei, um die Anwendungsaufrufe an die neuste Version von XMS weiterzuleiten. In diesem Fall können die Anwendung ohne Neukompilierung ausgeführt werden und die neuen Funktionen verwenden, die in der neueren XMS .NET-Version verfügbar sind.

Prozedur

- Wenn eine Anwendung die ältere XMS .NET-Version verwenden muss, setzen Sie das Attribut `publisherpolicy` in der Anwendungskonfigurationsdatei auf `no`.

  Anmerkung: Der Name einer Anwendungskonfigurationsdatei besteht aus dem Namen des ausführbaren Programms, auf das sich die Datei bezieht, und dem Suffix '.config'. Demnach hätte beispielsweise die Anwendungskonfigurationsdatei für das ausführbare Programm 'text.exe' den Namen 'text.exe.config'.

  Alle Anwendungen eines Systems verwenden jedoch stets dieselbe Version von XMS .NET.

Kommunikation für XMS-Anwendungen sichern

Dieser Abschnitt enthält Informationen zum Einrichten einer sicheren Kommunikation, damit XMS-Anwendungen SSL-Verbindungen (Secure Sockets Layer) zur Messaging-Engine eines WebSphere Application Server service integration bus oder zu einem IBM MQ-Warteschlangenmanager herstellen können.

Informationen zu diesem Vorgang

Dieser Abschnitt enthält folgende Themen:

Sichere Verbindungen zu einem IBM MQ-Warteschlangenmanager

Damit eine XMS .NET-Anwendung sichere Verbindungen zu einem IBM MQ-Warteschlangenmanager herstellen kann, müssen die relevanten Eigenschaften im ConnectionFactory-Objekt definiert sein.

Ob bei der Verschlüsselungsverhandlung das SSL-Protokoll (Secure Sockets Layer) oder TLS-Protokoll (Transport Layer Security) verwendet wird, hängt davon ab, welche Cipher-Suite Sie im ConnectionFactory-Objekt angeben.

Die ConnectionFactory -Eigenschaften für Verbindungen über SSL zu einem IBM MQ -Warteschlangenmanager werden in der folgenden Tabelle kurz beschrieben:

Tabelle 93. Eigenschaften von 'ConnectionFactory' für SSL-Verbindungen zu einem IBM MQ-Warteschlangenmanager

Name der Eigenschaft	Beschreibung
XMSC_WMQ_SSL_CERT_STORES	Die Positionen der Server, auf denen sich die Zertifikatswiderrufslisten (CRLs) befinden, die für eine SSL-Verbindung zu einem Warteschlangenmanager verwendet werden sollen.
XMSC_WMQ_SSL_CIPHER_SPEC	Der Name der CipherSpec, die für eine sichere Verbindung zu einem Warteschlangenmanager verwendet werden soll.
XMSC_WMQ_SSL_CIPHER_SUITE	Der Name der Cipher-Suite, die für eine TLS-Verbindung zu einem Warteschlangenmanager verwendet werden soll. Welches Protokoll bei der Vereinbarung der sicheren Verbindung verwendet wird, ist von der angegebenen Cipher-Suite abhängig.
XMSC_WMQ_SSL_CRYPTO_HW	Konfigurationsdetails für die Verschlüsselungshardware, die mit dem Clientsystem verbunden ist.
XMSC_WMQ_SSL_FIPS_REQUIRED	Der Wert dieser Eigenschaft bestimmt, ob eine Anwendung nicht FIPS-konforme Cipher-Suites verwenden kann oder nicht. Wenn diese Eigenschaft auf 'true' gesetzt ist, werden nur FIPS-Algorithmen für die Client-Server-Verbindung verwendet.
XMSC_WMQ_SSL_KEY_REPOSITORY	Gibt die Position der Schlüsseldatenbankdatei an, in der Schlüssel und Zertifikate gespeichert werden.

Tabelle 93. Eigenschaften von 'ConnectionFactory' für SSL-Verbindungen zu einem IBM MQ-Warteschlangenmanager (Forts.)	
Name der Eigenschaft	**Beschreibung**
XMSC_WMQ_SSL_KEY_RESETCOUNT	Der Wert von KeyResetCount gibt die Gesamtzahl unverschlüsselter Bytes an, die vor einer Neuvereinbarung des geheimen Schlüssels in einem SSL-Dialog gesendet oder empfangen werden.
XMSC_WMQ_SSL_PEER_NAME	Der Peername, der für eine SSL-Verbindung zu einem Warteschlangenmanager verwendet werden soll.

Namenszuordnung zwischen Cipher-Suites und CipherSpecs für XMS-Verbindungen zu einem IBM MQ-Warteschlangenmanager

Das InitialContext-Objekt übernimmt die Umsetzung zwischen der Eigenschaft 'SSLCIPHERSUITE' der JMSAdmin-Verbindungsfactory und der fast äquivalenten XMS-Eigenschaft 'XMSC_WMQ_SSL_CIPHER_SPEC'. Eine ähnliche Umsetzung ist erforderlich, wenn Sie einen Wert für 'XMSC_WMQ_SSL_CIPHER_SUITE' angeben, aber einen Wert für 'XMSC_WMQ_SSL_CIPHER_SPEC' weglassen.

In Tabelle 94 auf Seite 700 sind die verfügbaren CipherSpecs und deren entsprechende JSSE-Cipher-Suites aufgeführt.

Tabelle 94. Verfügbare CipherSpecs und deren entsprechende JSSE-Cipher-Suites	
CipherSpec	**Entsprechende JSSE-Cipher-Suite**
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

Anmerkung: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA ist veraltet. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.

Sichere Verbindungen zu einer Messaging-Engine für einen WebSphere Application Server service integration bus

Damit eine XMS .NET-Anwendung sichere Verbindungen zu einer Messaging-Engine für einen WebSphere Application Server service integration bus herstellen kann, müssen die relevanten Eigenschaften im ConnectionFactory-Objekt definiert sein.

XMS stellt die SSL- und HTTPS-Unterstützung für Verbindungen zu einem WebSphere Application Server service integration bus bereit. Die Protokolle SSL und HTTPS ermöglichen sichere Verbindungen zur Authentifizierung und für Vertraulichkeit.

Die XMS-Sicherheit wird ebenso wie die WebSphere-Sicherheit unter Beachtung der JSSE-Sicherheitsstandards und -Namenskonventionen konfiguriert, was auch die Verwendung von Cipher-Suites zur Angabe der Algorithmen einschließt, die beim Vereinbaren einer sicheren Verbindung zum Einsatz kommen. Ob bei der Verschlüsselungsverhandlung das SSL-Protokoll oder TLS-Protokoll verwendet wird, hängt davon ab, welche Cipher-Suite Sie im ConnectionFactory-Objekt angeben.

In Tabelle 95 auf Seite 701 sind die Eigenschaften aufgeführt, die im ConnectionFactory-Objekt definiert sein müssen.

Tabelle 95. Eigenschaften von 'ConnectionFactory' für sichere Verbindungen zu einer Messaging-Engine für einen WebSphere Application Server service integration bus.	
Name der Eigenschaft	**Beschreibung**
XMSC_WPM_SSL_CIPHER_SUITE	Der Name desCipherSuite zur Verwendung bei einer TLS-Verbindung zu einemWebSphere Application Server service integration bus Messaging-Engine. Welches Protokoll bei der Vereinbarung der sicheren Verbindung verwendet wird, ist von der angegebenen Cipher-Suite abhängig.
XMSC_WPM_SSL_KEYRING_LABEL	Das Zertifikat, das bei der Authentifizierung beim Server verwendet werden soll.

Die folgenden Angaben sind ein Beispiel für ConnectionFactory-Eigenschaften für sichere Verbindungen zu einer Messaging-Engine für einen WebSphere Application Server service integration bus:

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

Dabei sollte die Variable 'chain_name' (Kettenname) entweder auf 'BootstrapTunneledSecureMessaging' oder 'BootstrapSecureMessaging' und die Variable 'port_number' (Portnummer) auf die Nummer des Ports festgelegt sein, den der Bootstrap-Server auf eingehende Anforderungen überwacht.

Die folgenden Angaben sind ein Beispiel für ConnectionFactory-Eigenschaften für sichere Verbindungen zu einer Messaging-Engine für einen WebSphere Application Server service integration bus mit eingesetzten Beispielwerten:

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

Namenszuordnung zwischen Cipher-Suites und CipherSpecs für Verbindungen zu einem WebSphere Application Server service integration bus

Da IBM Global Security Kit (GSKit) CipherSpecs anstelle von CipherSuitesverwendet, müssen die in der Eigenschaft XMSC_WPM_SSL_CIPHER_SUITE angegebenen JSSE- CipherSuite -Namen den GSKit-style CipherSpec -Namen zugeordnet werden.

In Tabelle 96 auf Seite 701 sind die entsprechenden CipherSpecs für die jeweiligen erkannten Cipher-Suites aufgeführt.

Tabelle 96. Verfügbare Cipher-Suites und deren entsprechende CipherSpecs	
CipherSuite	**Entsprechende CipherSpec**
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

Anmerkung: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA ist veraltet. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten

oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.

XMS-Nachrichten

In diesem Abschnitt werden die Struktur und der Inhalt von XMS-Nachrichten beschrieben und erläutert, wie Anwendungen XMS-Nachrichten verarbeiten.

Dieser Abschnitt enthält die folgenden Themen:

Bestandteile einer XMS-Nachricht

Eine XMS-Nachricht besteht aus einem Header, einer Gruppe von Eigenschaften und einem Hauptteil.

Header
Der Header einer Nachricht enthält Felder, wobei die Gruppe der Headerfelder für alle Nachrichten identisch ist. Mithilfe der Werte der Headerfelder können XMS-Anwendungen Nachrichten ermitteln und weiterleiten. Weitere Informationen zu Headerfeldern finden Sie im Abschnitt „Headerfelder in einer XMS-Nachricht" auf Seite 702.

Gruppe der Eigenschaften
Die Eigenschaften einer Nachricht enthalten zusätzliche Informationen zu dieser Nachricht. Obwohl die Gruppe der Headerfelder für alle Nachrichten identisch ist, können alle Nachrichten unterschiedliche Gruppen von Eigenschaften haben. Weitere Informationen finden Sie unter „Eigenschaften einer XMS-Nachricht" auf Seite 703.

Hauptteil
Der Hauptteil einer Nachricht enthält Anwendungsdaten. Weitere Informationen finden Sie unter „Hauptteil einer XMS-Nachricht" auf Seite 706.

Eine Anwendung kann auswählen, welche Nachrichten sie empfangen möchte. Dies geschieht mithilfe von Nachrichtenselektoren, die die Auswahlkriterien angeben. Die Kriterien können auf den Werten bestimmter Headerfelder und den Werten aller Eigenschaften einer Nachricht basieren. Weitere Informationen zu Nachrichtenselektoren finden Sie unter „Nachrichtenselektoren" auf Seite 710.

Headerfelder in einer XMS-Nachricht

Damit eine XMS-Anwendung Nachrichten mit einer WebSphere-JMS-Anwendung austauschen kann, muss der Header einer XMS-Nachricht die JMS-Nachrichtenheaderfelder enthalten.

Die Namen dieser Headerfelder beginnen mit dem Präfix JMS. Eine Beschreibung der JMS-Nachrichtenheaderfelder finden Sie im Abschnitt *Spezifikation für Java Message Service*.

XMS implementiert die JMS-Nachrichtenheaderfelder als Attribute eines Message-Objekts. Für jedes Headerfeld gibt es spezifische Methoden, mit denen sein Wert festgelegt und abgerufen wird. Eine Beschreibung dieser Methoden finden Sie im Abschnitt IMessage. Ein Headerfeld ist immer lesbar und änderbar (ohne Schreib- oder Leseschutz).

In Tabelle 97 auf Seite 703 ist für alle JMS-Nachrichtenheaderfelder angegeben, wie deren jeweilige Werte für eine übertragene Nachricht festgelegt werden. Für einige Felder wird der Wert automatisch durch XMS festgelegt, wenn eine Anwendung eine Nachricht sendet oder, im Fall von 'JMSRedelivered', wenn eine Anwendung eine Nachricht empfängt.

Tabelle 97. JMS-Nachrichtenheaderfelder.]	
Name des JMS-Nachrichten-headerfeldes	**Vorgang zum Festlegen des Werts für eine übertragene Nachricht (Format: *Methode* [*Klasse*])**
JMSCorrelationID	Set JMSCorrelationID [Message]
JMSDeliveryMode	Send [MessageProducer]
JMSDestination	Send [MessageProducer]
JMSExpiration	Send [MessageProducer]
JMSMessageID	Send [MessageProducer]
JMSPriority	Send [MessageProducer]
JMSRedelivered	Receive [MessageConsumer]
JMSReplyTo	Set JMSReplyTo [Message]
JMSTimestamp	Send [MessageProducer]
JMSType	Set JMSType [Message]

Eigenschaften einer XMS-Nachricht

XMS unterstützt drei Arten von Eigenschaften: JMS-definierte Eigenschaften, IBM-definierte Eigenschaften und anwendungsdefinierte Eigenschaften.

Eine XMS-Anwendung kann Nachrichten mit einer WebSphere JMS-Anwendung austauschen, weil XMS die folgenden vordefinierten Eigenschaften eines Message-Objekts unterstützt:

- Dieselben JMS-definierten Eigenschaften, die auch WebSphere JMS unterstützt. Die Namen dieser Eigenschaften beginnen mit dem Präfix 'JMSX'.
- Dieselben IBM-definierten Eigenschaften, die auch WebSphere JMS unterstützt. Die Namen dieser Eigenschaften beginnen mit dem Präfix 'JMS_IBM_'.

Jede vordefinierte Eigenschaft hat zwei Namen:

- Zum einen haben JMS-definierte Eigenschaften einen JMS-Namen und IBM-definierte Eigenschaften einen WebSphere JMS-Namen.

  Dieser Name dient in JMS bzw. WebSphere JMS als Bezeichnung für die jeweilige Eigenschaft und wird zusammen mit einer Nachricht, die diese Eigenschaft hat, auch übertragen. Eine XMS-Anwendung verwendet diesen Namen, um die Eigenschaft in einem Nachrichtenselektorausdruck zu ermitteln.
- Zum anderen haben Eigenschaften einen XMS-Namen, der als Bezeichnung der Eigenschaft in allen Situationen außer in Nachrichtenselektorausdrücken dient. Jeder XMS-Name ist als benannte Konstante in der Klasse `IBM.XMS.XMSC` definiert. Der Wert der benannten Konstante ist der entsprechende JMS- oder WebSphere JMS-Name.

Neben den vordefinierten Eigenschaften kann eine XMS-Anwendung ihre eigenen Nachrichteneigenschaften erstellen und verwenden. Dies sind die so genannten *anwendungsdefinierten Eigenschaften*.

Nachdem eine Anwendung eine Nachricht erstellt hat, sind die Eigenschaften der Nachricht sowohl lesbar als auch änderbar, also weder lese- noch schreibgeschützt. Auch wenn die Anwendung die Nachricht gesendet hat, bleiben die Eigenschaften lesbar und änderbar. Wenn eine Anwendung eine Nachricht empfängt, sind die Eigenschaften der Nachricht zunächst schreibgeschützt. Wenn eine Anwendung die Methode `Clear Properties` der Nachrichtenklasse aufruft, wenn die Eigenschaften einer Nachricht schreibgeschützt sind, werden die Eigenschaften lesbar und beschreibbar. Außerdem löscht diese Methode die Eigenschaften.

Wenn die empfangene Nachricht nach dem Löschen der Nachrichteneigenschaften weitergeleitet wird, ist das weitere Verhalten der Nachricht konsistent mit dem Weiterleitungsverhalten eines standardmäßigen BytesMessage-Objekts für WMQ XMS for .NET, dessen Eigenschaften gelöscht wurden.

Dies wird jedoch nicht empfohlen, weil dabei folgende Eigenschaften verloren gehen:

- Der Eigenschaftswert 'JMS_IBM_Encoding', der angibt, dass die Nachrichtendaten nicht aussagekräftig entschlüsselt werden können.
- Der Eigenschaftswert 'JMS_IBM_Format', der angibt, dass die Headerverkettung zwischen dem MQMD- bzw. dem neuen MQRFH2-Nachrichtenheader und den vorhandenen Headern unterbrochen wird.

Um die Werte aller Eigenschaften einer Nachricht abzurufen, kann eine Anwendung die Methode 'Get Properties' der Message-Klasse aufrufen. Mit der Methode wird ein Iterator erstellt, der eine Liste der Eigenschaftsobjekte (Property) enthält, in der jedes Property-Objekt eine Eigenschaft der Nachricht darstellt. Dann kann die Anwendung mit den Methoden der Iterator-Klasse zunächst jedes Property-Objekt einzeln abrufen und anschließend mit den Methoden der Property-Klasse den Namen, Datentyp und Wert jeder einzelnen Eigenschaft abrufen.

JMS-definierte Eigenschaften einer Nachricht

Verschiedene JMS-definierte Eigenschaften einer Nachricht werden sowohl von XMS als auch von WebSphere JMS unterstützt.

In Tabelle 98 auf Seite 704 sind die JMS-definierten Eigenschaften einer Nachricht aufgeführt, die sowohl von XMS als auch von WebSphere JMS unterstützt werden. Eine Beschreibung der JMS-definierten Eigenschaften finden Sie in der *Spezifikation für Java Message Service*. Die JMS-definierten Eigenschaften sind für eine Echtzeitverbindung zu einem Broker nicht zulässig.

In der Tabelle ist für alle Eigenschaften der Datentyp angegeben und wie der jeweilige Eigenschaftswert für eine übertragene Nachricht festgelegt wird. Für einige Eigenschaften wird der Wert automatisch durch XMS festgelegt, wenn eine Anwendung eine Nachricht sendet oder, im Fall von 'JMSXDeliveryCount', wenn eine Anwendung eine Nachricht empfängt.

Tabelle 98. JMS-definierte Eigenschaften einer Nachricht

XMS-Name der JMS-definierten Eigenschaft	**JMS-Name**	**Datentyp**	**Vorgang zum Festlegen des Werts für eine übertragene Nachricht (Format: *Methode* [*Klasse*])**
JMSX_APPID	JMSXAppID	System.String	Send [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Receive [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	Set String Property [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Set Integer Property [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	Send [MessageProducer]

IBM-definierte Eigenschaften einer Nachricht

Verschiedene IBM-definierte Eigenschaften einer Nachricht werden sowohl von XMS als auch von WebSphere JMS unterstützt.

In Tabelle 99 auf Seite 705 sind die von IBM definierten Eigenschaften einer Nachricht aufgeführt, die sowohl bei XMS als auch bei WebSphere JMS unterstützt werden. Weitere Informationen zu den von IBM definierten Eigenschaften finden Sie unter IBM MQ oder in der Produktdokumentation von WebSphere Application Server.

In der Tabelle ist für alle Eigenschaften der Datentyp angegeben und wie der jeweilige Eigenschaftswert für eine übertragene Nachricht festgelegt wird. Für einige Eigenschaften wird der Wert automatisch durch XMS festgelegt, wenn eine Anwendung eine Nachricht sendet.

Tabelle 99. IBM-definierte Eigenschaften einer Nachricht

XMS-Name der IBM-definierten Eigenschaft	WebSphere JMS-Name	Datentyp	Vorgang zum Festlegen des Werts für eine übertragene Nachricht (Format: *Methode* [*Klasse*])
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_EXCEPTION-MESSAGE	JMS_IBM_ExceptionMessage	System.String	Receive [MessageConsumer]
JMS_IBM_EXCEPTION-REASON	JMS_IBM_ExceptionReason	System.Int32	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Receive [MessageConsumer]
JMS_IBM_EXCEPTION-PROBLEM DESTINATION	JMS_IBM_ExceptionProblemDestination	System.String	Receive [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	Set String Property [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Set Integer Property [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	Send [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Send [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Send [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	Set Integer Property [PropertyContext]

Tabelle 99. IBM-definierte Eigenschaften einer Nachricht (Forts.)			
XMS-Name der IBM-definierten Eigenschaft	**WebSphere JMS-Name**	**Datentyp**	**Vorgang zum Festlegen des Werts für eine übertragene Nachricht (Format: *Methode* [*Klasse*])**
JMS_IBM_REPORT_PASS_CORREL_ID	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Send [MessageProducer]

Anwendungsdefinierte Eigenschaften einer Nachricht

Eine XMS-Anwendung kann ihre eigenen Nachrichteneigenschaften erstellen und verwenden. Wenn eine Anwendung einer Nachricht sendet, werden diese Eigenschaften ebenfalls mit der Nachricht übertragen. Eine empfangende Anwendung kann dann mithilfe von Nachrichtenselektoren anhand der Werte dieser Eigenschaften die Nachrichten auswählen, die sie empfangen möchte.

Damit eine WebSphere JMS-Anwendung die von einer XMS-Anwendung gesendeten Nachrichten auswählen und verarbeiten kann, muss der Name einer anwendungsdefinierten Eigenschaft mit den Regeln zum Erstellen einer ID in einem Nachrichtenselektorausdruck erfüllen. Weitere Informationen finden Sie unter „Nachrichtenselektoren in JMS" auf Seite 152. Der Werte einer anwendungsdefinierten Eigenschaft muss zu einem der folgenden Datentypen gehören: 'System.Boolean', 'System.SByte', 'System.Int16', 'System.Int32', 'System.Int64', 'System.Float', 'System.Double' oder 'System.String'.

Hauptteil einer XMS-Nachricht

Der Hauptteil einer Nachricht enthält Anwendungsdaten. Eine Nachricht kann jedoch durchaus keinen Rumpf bzw. Hauptteil haben und nur die Headerfelder und Eigenschaften enthalten.

XMS unterstützt fünf Nachrichtenhauptteiltypen:

Bytes
Der Hauptteil enthält einen Bytestrom. Eine Nachricht mit diesem Hauptteiltyp wird als *Bytenachricht* bezeichnet. Die Schnittstelle 'IBytesMessage' enthält die Methoden zur Verarbeitung des Hauptteils einer Bytenachricht.

Zuordnung
Der Hauptteil enthält eine Gruppe von Name/Wert-Paaren, wobei jedem Wert ein bestimmter Datentyp zugeordnet ist. Eine Nachricht mit diesem Hauptteiltyp wird als *Zuordnungsnachricht* bezeichnet. Die Schnittstelle 'IMapMessage' enthält die Methoden zur Verarbeitung des Hauptteils einer Zuordnungsnachricht.

Objekt
Der Hauptteil enthält ein serialisiertes Java- oder .NET-Objekt. Eine Nachricht mit diesem Hauptteiltyp wird als *Objektnachricht* bezeichnet. Die Schnittstelle 'IObjectMessage' enthält die Methoden zur Verarbeitung des Hauptteils einer Objektnachricht.

Datenstrom
Der Hauptteil enthält einen Datenstrom von Werten, wobei jedem Wert ein bestimmter Datentyp zugeordnet ist. Eine Nachricht mit diesem Hauptteiltyp wird als *Datenstromnachricht* bezeichnet. Die Schnittstelle 'IStreamMessage' enthält die Methoden zur Verarbeitung des Hauptteils einer Datenstromnachricht.

Text
Der Hauptteil enthält eine Zeichenfolge. Eine Nachricht mit diesem Hauptteiltyp wird als *Textnachricht* bezeichnet. Die Schnittstelle 'ITextMessage' enthält die Methoden zur Verarbeitung des Hauptteils einer Textnachricht.

Die Schnittstelle 'IMessage' ist das übergeordnete Element aller Nachrichtenobjekte und kann in Messaging-Funktionen zur Darstellung eines beliebigen XMS-Nachrichtentyps verwendet werden.

Informationen zur Größe sowie zu den Höchst- und Mindestwerten eines jeden Datentyps finden Sie im Abschnitt Tabelle 85 auf Seite 685.

Bytenachrichten

Der Hauptteil einer Bytenachricht enthält einen Bytestrom. Der Hauptteil enthält nur die tatsächlichen Daten, wobei die sendende und empfangende Anwendung dafür zuständig sind, diese Daten zu interpretieren.

Bytenachrichten sind nützlich, wenn eine XMS-Anwendung Nachrichten mit Anwendungen austauschen muss, die nicht die XMS- oder JMS-Anwendungsprogrammierschnittstelle verwenden.

Nachdem eine Anwendung eine Bytenachricht erstellt hat, ist der Hauptteil der Nachricht zunächst lesegeschützt. Die Anwendung stellt die Anwendungsdaten im Hauptteil zusammen, indem Sie die entsprechenden Schreibmethoden der IBytesMessage-Schnittstelle für .NET aufruft. Jedes Mal, wenn die Anwendung einen weiteren Wert in den Datenstrom der Bytenachricht schreibt, wird der Wert unmittelbar nach dem zuletzt von der Anwendung geschriebenen Wert angefügt. XMS verwaltet einen internen Cursor, mit dessen Hilfe die Position des zuletzt geschriebenen Bytewerts gespeichert wird.

Wenn die Anwendung die Nachricht sendet, wird der Hauptteil der Nachricht nun schreibgeschützt. In diesem Modus kann die Anwendung die Nachricht wiederholt senden.

Wenn eine Anwendung eine Bytenachricht empfängt, ist der Hauptteil der Nachricht schreibgeschützt. Mit den entsprechenden Lesemethoden der IBytesMessage-Schnittstelle kann die Anwendung den Inhalt des Datentstroms in der Bytenachricht lesen. Die Anwendung liest die Bytewerte der Reihenfolge entsprechend nacheinander, und XMS verwaltet einen internen Cursor, mit dessen Hilfe die Position des zuletzt gelesenen Bytewerts gespeichert wird.

Wenn eine Anwendung die Methode 'Reset' der IBytesMessage-Schnittstelle aufruft, während der Hauptteil einer Bytenachricht änderbar ist, wird der Hauptteil dadurch schreibgeschützt. Außerdem wird der Cursor durch diese Methode zurück an den Anfang des Bytenachrichten-Datenstroms gesetzt.

Wenn eine Anwendung die Methode `Clear Body` der IMessage-Schnittstelle für .NET aufruft, während der Hauptteil einer Bytenachricht schreibgeschützt ist, wird der Hauptteil dadurch änderbar. Außerdem löscht diese Methode den Inhalt des Hauptteils.

Zuordnungsnachrichten

Der Hauptteil einer Zuordnungsnachricht enthält eine Gruppe von Name/Wert-Paaren, wobei jedem Wert ein bestimmter Datentyp zugeordnet ist.

In jedem Name/Wert-Paar ist der Name eine Zeichenfolge, die angibt, um welchen Wert es sich handelt, und der Wert ist ein Element der Anwendungsdaten, dem einer der XMS-Datentypen zugeordnet ist, die in Tabelle 100 auf Seite 709 aufgelistet sind. Die Reihenfolge der Name/Wert-Paare ist nicht definiert. Die Klasse 'MapMessage' enthält die Methoden zum Festlegen und Abrufen von Name/Wert-Paaren.

Eine Anwendung kann durch Angabe des Namens beliebig auf ein Name/Wert-Paar zugreifen.

Eine .NET-Anwendung kann mithilfe der Eigenschaft 'MapNames' eine Auflistung der Namen im Hauptteil einer Zuordnungsnachricht abrufen.

Wenn eine Anwendung den Wert eines Name/Wert-Paars abruft, kann der Wert durch XMS in einen anderen Datentyp konvertiert werden. Wenn beispielsweise ein Ganzzahl-Wert (Integer) aus dem Hauptteil einer Zuordnungsnachricht abgerufen werden soll, kann eine Anwendung die GetString-Methode der MapMessage-Klasse aufrufen, sodass die Ganzzahl als Zeichenfolge zurückgegeben wird. Dabei unterstützt XMS dieselben Konvertierungen wie bei der Konvertierung eines Eigenschaftswerts von einem Datentyp

in einen anderen. Weitere Informationen zu den unterstützen Konvertierungen finden Sie im Abschnitt „Implizite Konvertierung eines Eigenschaftswerts von einem Datentyp in einen anderen Datentyp” auf Seite 685.

Nachdem eine Anwendung eine Zuordnungsnachricht erstellt hat, ist der Hauptteil sowohl lesbar als auch änderbar, also weder lese- noch schreibgeschützt. Auch wenn die Anwendung die Nachricht gesendet hat, bleibt der Hauptteil lesbar und änderbar. Wenn eine Anwendung eine Zuordnungsnachricht empfängt, ist der Hauptteil der Nachricht zunächst schreibgeschützt. Wenn eine Anwendung die Methode 'Clear Body' aufruft, während der Hauptteil einer Zuordnungsnachricht schreibgeschützt ist, wird der Hauptteil dadurch wieder lesbar und änderbar. Außerdem löscht diese Methode den Inhalt des Hauptteils.

Objektnachrichten

Der Hauptteil einer Objektnachricht enthält ein serialisiertes Java- oder .NET-Objekt.

Eine XMS-Anwendung kann eine Objektnachricht empfangen, deren Headerfelder und Eigenschaften ändern und sie anschließend an ein anderes Ziel senden. Außerdem kann eine Anwendung den Hauptteil einer Objektnachricht kopieren und zum Erstellen einer anderen Objektnachricht verwenden. XMS behandelt den Hauptteil einer Objektnachricht wie ein Byte-Array.

Nachdem eine Anwendung eine Objektnachricht erstellt hat, ist der Hauptteil der Nachricht sowohl lesbar als auch änderbar, also weder lese- noch schreibgeschützt. Auch wenn die Anwendung die Nachricht gesendet hat, bleibt der Hauptteil lesbar und änderbar. Wenn eine Anwendung eine Objektnachricht empfängt, ist der Hauptteil der Nachricht zunächst schreibgeschützt. Wenn eine Anwendung die Methode `Clear Body` der IMessage-Schnittstelle für .NET aufruft, während der Hauptteil einer Objektnachricht schreibgeschützt ist, wird der Hauptteil dadurch wieder lesbar und änderbar. Außerdem löscht diese Methode den Inhalt des Hauptteils.

Datenstromnachrichten

Der Hauptteil einer Datenstromnachricht enthält einen Datenstrom von Werten, wobei jedem Wert ein bestimmter Datentyp zugeordnet ist.

Der Datentyp eines Werts ist einer der XMS-Datentypen, die in Tabelle 100 auf Seite 709 aufgelistet sind.

Nachdem eine Anwendung eine Datenstromnachricht erstellt hat, ist der Hauptteil der Nachricht zunächst änderbar. Die Anwendung stellt die Anwendungsdaten im Hauptteil zusammen, indem Sie die entsprechenden Schreibmethoden der IStreamMessage-Schnittstelle für .NET aufruft. Jedes Mal, wenn die Anwendung einen weiteren Wert in den Nachrichtendatenstrom schreibt, wird der Wert und dessen Datentyp unmittelbar nach dem zuletzt von der Anwendung geschriebenen Wert angefügt. XMS verwaltet einen internen Cursor, mit dessen Hilfe die Position des zuletzt geschriebenen Werts gespeichert wird.

Wenn die Anwendung die Nachricht sendet, wird der Hauptteil der Nachricht nun schreibgeschützt. In diesem Modus kann die Anwendung die Nachricht mehrmals senden.

Wenn eine Anwendung eine Datenstromnachricht empfängt, ist der Hauptteil der Nachricht zunächst schreibgeschützt. Mit den entsprechenden Lesemethoden der IStreamMessage-Schnittstelle für .NET kann die Anwendung den Inhalt Nachrichtendatenstroms lesen. Die Anwendung liest die Werte der Reihenfolge entsprechend nacheinander, und XMS verwaltet einen internen Cursor, mit dessen Hilfe die Position des zuletzt gelesenen Werts gespeichert wird.

Wenn eine Anwendung den Wert eines Nachrichtendatenstroms abruft, kann der Wert durch XMS in einen anderen Datentyp konvertiert werden. Wenn beispielsweise ein Ganzzahl-Wert (Integer) aus dem Nachrichtendatenstrom abgerufen werden soll, kann eine Anwendung die ReadString-Methode aufrufen, sodass die Ganzzahl als Zeichenfolge zurückgegeben wird. Dabei unterstützt XMS dieselben Konvertierungen wie bei der Konvertierung eines Eigenschaftswerts von einem Datentyp in einen anderen. Weitere Informationen zu den unterstützen Konvertierungen finden Sie im Abschnitt „Implizite Konvertierung eines Eigenschaftswerts von einem Datentyp in einen anderen Datentyp” auf Seite 685.

Wenn ein Fehler auftritt, während eine Anwendung einen Wert im Nachrichtendatenstrom zu lesen versucht, wird die Cursorposition nicht aktualisiert. Die Anwendung kann einen erneuten Fehler vermeiden, indem sie versucht, den Wert als einen anderen Datentyp zu lesen.

Wenn eine Anwendung die Methode 'Reset' der IStreamMessage-Schnittstelle für XMS aufruft, während der Hauptteil einer Datenstromnachricht lesegeschützt ist, wird der Hauptteil dadurch schreibgeschützt. Außerdem wird der Cursor durch diese Methode zurück an den Anfang des Nachrichtendatenstroms gesetzt.

Wenn eine Anwendung die Methode `Clear Body` der IMessage-Schnittstelle für XMS aufruft, während der Hauptteil einer Datenstromnachricht schreibgeschützt ist, wird der Hauptteil dadurch lesegeschützt. Außerdem löscht diese Methode den Inhalt des Hauptteils.

Textnachrichten

Der Hauptteil einer Textnachricht enthält eine Zeichenfolge.

Nachdem eine Anwendung eine Textnachricht erstellt hat, ist der Hauptteil der Nachricht sowohl lesbar als auch änderbar, also weder lese- noch schreibgeschützt. Auch wenn die Anwendung die Nachricht gesendet hat, bleibt der Hauptteil lesbar und änderbar. Wenn eine Anwendung eine Textnachricht empfängt, ist der Hauptteil der Nachricht zunächst schreibgeschützt. Wenn eine Anwendung die Methode 'Clear Body' der IMessage-Schnittstelle für .NET aufruft, während der Hauptteil einer Textnachricht schreibgeschützt ist, wird der Hauptteil dadurch wieder lesbar und änderbar. Außerdem löscht diese Methode den Inhalt des Hauptteils.

Datentypen für Anwendungsdatenelemente

Um sicherzustellen, dass eine XMS-Anwendung Nachrichten mit einer Anwendung für die IBM MQ classes for JMS austauschen kann, müssen beide Anwendungen die Anwendungsdaten im Hauptteil einer Nachricht auf gleiche Weise interpretieren können.

Daher muss jedes Anwendungsdatenelement, das von einer XMS-Anwendung in den Hauptteil einer Nachricht geschrieben wird, zu einem der in Tabelle 100 auf Seite 709 aufgelisteten Datentypen gehören. Für jeden Datentyp ist in der Tabelle der kompatible Java-Datentyp angegeben. Mit den von XMS bereitgestellten Methoden können ausschließlich Anwendungsdatenelemente mit diesen Datentypen geschrieben werden.

Tabelle 100. XMS-Datentypen, die mit Java-Datentypen kompatibel sind

XMS Datentyp	Beschreibung	Kompatibler Java-Datentyp
System.Boolean	Boolescher Wert 'true' oder 'false' (Wahr oder Falsch)	boolean
System.Char16	Doppelbyte-Zeichen	char
System.SByte	8-Bit-Ganzzahl mit Vorzeichen	Byte
System.Int16	16-Bit-Ganzzahl mit Vorzeichen	short
System.Int32	32-Bit-Ganzzahl mit Vorzeichen	int
System.Int64	64-Bit-Ganzzahl mit Vorzeichen	lang
System.Float	Gleitkommazahl mit Vorzeichen	float
System.Double	Gleitkommazahl mit doppelter Genauigkeit und Vorzeichen	double
System.String	Zeichenfolge	Zeichenfolge

Informationen zur Größe sowie zu den Höchst- und Mindestwerten eines jeden Datentyps finden Sie im Abschnitt „Primitive XMS-Datentypen” auf Seite 685.

Nachrichtenselektoren

Mithilfe von Nachrichtenselektoren kann eine XMS-Anwendung festlegen, welche Nachrichten sie empfangen möchte.

Nachdem eine Anwendung einen Nachrichtenkonsumenten erstellt hat, kann sie ihm einen Nachrichtenselektorausdruck zuordnen. Der Nachrichtenselektorausdruck gibt die Auswahlkriterien an.

Wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager für IBM WebSphere MQ 7.0 herstellt, erfolgt die Nachrichtenauswahl auf der Seite des Warteschlangenmanagers. In diesem Fall nimmt XMS keine Auswahl vor und übermittelt lediglich die vom Warteschlangenmanager empfangene Nachricht. Auf diese Weise wird die Leistung verbessert.

Eine Anwendung kann mehrere Nachrichtenkonsumenten erstellen und jedem einen eigenen Nachrichtenselektorausdruck zuordnen. Wenn eine eingehende Nachricht die Auswahlkriterien mehrerer Nachrichtenkonsumenten erfüllt, übermittelt XMS die Nachricht an jeden dieser Konsumenten.

Ein Nachrichtenselektorausdruck kann folgende Nachrichteneigenschaften referenzieren:

- JMS-definierte Eigenschaften
- IBM-definierte Eigenschaften
- Anwendungsdefinierte Eigenschaften

Darüber hinaus kann er folgende Nachrichtenheaderfelder referenzieren:

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

Ein Nachrichtenselektorausdruck kann jedoch keine Daten im Hauptteil einer Nachricht referenzieren.

Es folgt ein Beispiel für einen Nachrichtenselektorausdruck:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

XMS stellt eine Nachricht einem Nachrichtenkonsumenten mit diesem Nachrichtenselektorausdruck nur dann zu, wenn die Nachricht eine höhere Priorität als 3hat; eine anwendungsdefinierte Eigenschaft (Hersteller) mit dem Wert `Jaguar;` und eine andere anwendungsdefinierte Eigenschaft (Modell) mit dem Wert `xj6` oder `xj12.`

Für die Erstellung eines Nachrichtenselektorausdrucks in XMS gelten dieselben Syntaxregeln wie in den IBM MQ classes for JMS. Weitere Informationen zum Erstellen eines Nachrichtenselektorausdrucks finden Sie in der Produktdokumentation zu IBM MQ. Es ist zu beachten, dass in einem Nachrichtenselektorausdruck für die JMS-definierten Eigenschaften die JMS-Namen und für die IBM-definierten Eigenschaften die für die IBM MQ classes for JMS geltenden Namen verwendet werden müssen. Hingegen ist die Verwendung der XMS-Namen der Eigenschaften in einem Nachrichtenselektorausdruck nicht zulässig.

Zuordnung von XMS-Nachrichten zu IBM MQ-Nachrichten

Die JMS-Headerfelder und -Eigenschaften einer XMS-Nachricht werden den Feldern in den Headerstrukturen einer IBM MQ-Nachricht zugeordnet.

Wenn eine XMS-Anwendung mit einem IBM MQ-Warteschlangenmanager verbunden ist, werden die an den Warteschlangenmanager gesendeten Nachrichten den IBM MQ-Nachrichten in gleicher Weise zugeordnet, wie die Nachrichten für die IBM MQ classes for JMS unter ähnlichen Umständen den IBM MQ-Nachrichten zugeordnet werden.

Wenn die Eigenschaft XMSC_WMQ_TARGET_CLIENT eines Destination-Objekts auf den Wert 'XMSC_WMQ_TARGET_DEST_JMS' festgelegt wird, werden die JMS-Headerfelder und -Eigenschaften einer an dieses Ziel gesendeten Nachricht den Feldern in den MQMD- und MQRFH2-Headerstrukturen der IBM MQ-Nachricht zugeordnet. Bei dieser Einstellung der Eigenschaft 'XMSC_WMQ_TARGET_CLIENT' wird davon ausgegangen, dass die Anwendung, die die Nachricht empfängt, einen MQRFH2-Header korrekt verarbeiten kann. Die empfangende Anwendung kann demnach eine andere XMS-Anwendung, eine Anwendung für die IBM MQ classes for JMS oder eine native IBM MQ-Anwendung sein, die für die Verarbeitung eines MQRFH2-Headers ausgelegt ist.

Wenn die Eigenschaft 'XMSC_WMQ_TARGET_CLIENT' eines Destination-Objekts stattdessen auf 'XMSC_WMQ_TARGET_DEST_MQ' festgelegt wird, werden die JMS-Headerfelder und -Eigenschaften einer an dieses Ziel gesendeten Nachricht den Feldern in der MQMD-Headerstruktur der IBM MQ-Nachricht zugeordnet. Die Nachricht enthält in diesem Fall keinen MQRFH2-Header, und alle JMS-Headerfelder und -Eigenschaften, die den Feldern in der MQMD-Headerstruktur nicht zugeordnet werden können, werden ignoriert. Die Anwendung, die die Nachricht empfängt, kann demnach eine native IBM MQ-Nachricht sein, die nicht für die Verarbeitung eines MQRFH2-Headers ausgelegt ist.

Die von einem Warteschlangenmanager empfangenen IBM MQ-Nachrichten werden XMS-Nachrichten in derselben Weise zugeordnet wie IBM MQ-Nachrichten unter ähnlichen Umständen den Nachrichten für die IBM MQ classes for JMS zugeordnet werden.

Wenn eine eingehende IBM MQ-Nachricht einen MQRFH2-Header aufweist, verfügt die sich ergebende XMS-Nachricht über einen Hauptteil, dessen Typ durch den Wert der Eigenschaft **Msd** bestimmt wird, die im Ordner `mcd` des MQRFH2-Headers enthalten ist. Wenn die Eigenschaft **Msd** nicht im MQRFH2-Header enthalten ist oder die IBM MQ-Nachricht keinen MQRFH2-Header aufweist, verfügt die sich ergebende XMS-Nachricht über einen Hauptteil, dessen Typ durch den Wert des Feldes *`Format`* im MQMD-Header bestimmt wird. Wenn das Feld *`Format`* den Wert 'MQFMT_STRING' hat, ist die XMS-Nachricht eine Textnachricht. Andernfalls ist die XMS-Nachricht eine Byte-Nachricht. Wenn die IBM MQ-Nachricht keinen MQRFH2-Header hat, werden nur die JMS-Headerfelder und -Eigenschaften festgelegt, die aus den Feldern im MQMD-Header abgeleitet werden können.

Weiter Informationen zur Zuordnung zwischen den Nachrichten für die IBM MQ classes for JMS und den IBM MQ-Nachrichten finden Sie im Abschnitt „Zuordnung von JMS-Nachrichten zu IBM MQ-Nachrichten” auf Seite 155.

Über eine Anwendung für einen IBM MQ Message Service Client (XMS) for .NET einen Nachrichtendeskriptor lesen und schreiben

Sie haben Zugriff auf alle Nachrichtendeskriptorfelder (MQMD) einer IBM MQ-Nachricht, mit Ausnahme der Felder 'StrucId' und 'Version'. Außerdem ist das Feld 'BackoutCount' schreibgeschützt, kann also nur gelesen werden.

Mithilfe der vom IBM MQ Message Service Client (XMS) for .NET bereitgestellten Nachrichtenattribute können XMS-Anwendungen MQMD-Felder festlegen und auch IBM WebSphere MQ-Anwendungen steuern.

Für das Publish/Subscribe-Messaging gelten einige Einschränkungen. Beispielsweise werden MQMD-Felder wie 'MsgID' und 'CorrelId' ignoriert, wenn für sie ein Wert festgelegt wurde.

Die Funktion ist auch nicht verfügbar, wenn die Eigenschaft **PROVIDERVERSION** auf 6gesetzt ist.

Aus einer IBM MQ Message Service Client (XMS) for .NET-Anwendung auf IBM MQ-Nachrichtendaten zugreifen

Von einer IBM MQ Message Service Client (XMS) for .NET-Anwendung aus können Sie als Hauptteil eines Objekt vom Typ 'JMSBytesMessage' auf die gesamten IBM MQ-Nachrichtendaten zugreifen, inklusive des Headers MQRFH2 (sofern vorhanden) und aller anderen IBM MQ-Header (sofern vorhanden).

Die in diesem Abschnitt beschriebene Funktionalität ist nur verfügbar, wenn eine Verbindung zu einem Warteschlangenmanager von IBM WebSphere MQ 7.0 oder höher besteht wird und der IBM MQ-Messaging-Provider im normalen Modus ausgeführt wird.

Eigenschaften des Zielobjekts legen fest, wie die XMS-Anwendung auf die vollständige IBM MQ-Nachricht (einschließlich des gegebenenfalls vorhandenen MQRFH2-Headers) als Hauptteil eines Objekts vom Typ 'JMSBytesMessage' zugreift.

ALW AMQP-Clientanwendungen entwickeln

Die IBM MQ-Unterstützung für AMQP-APIs ermöglicht es einem IBM MQ-Administrator, einen AMQP-Kanal zu erstellen. Wenn dieser Kanal gestartet wird, definiert dieser Kanal eine Portnummer, die Verbindungen von AMQP-Clientanwendungen akzeptiert.

Sie können einen AMQP-Kanal auf AIX, Linux, and Windows -Systemen installieren; er ist unter IBM i oder z/OSnicht verfügbar.

Eine AMQP 1.0-Clientanwendung kann über einen AMQP-Kanal eine Verbindung zum WS-Manager herstellen.

Anwendungen mithilfe der Apache Qpid JMS-Bibliothek entwickeln

Einführung

DieApache Qpid JMS-Bibliothek stellt unter Verwendung des AMQP 1.0-Protokolls eine Implementierung der JMS 2-Spezifikation bereit.

Apache Qpid JMS nutzt einige Aspekte des AMQP 1.0-Protokolls auf andere Weise als die MQ Light-Messaging-APIs. IBM MQ 9.2 wurde Unterstützung für IBM MQ AMQP-Kanäle hinzugefügt, sodass Apache Qpid JMS -Anwendungen eine Verbindung zu IBM MQ herstellen und Publish/Subscribe-Messaging durchführen können, einschließlich der Verwendung gemeinsam genutzter Subskriptionen.

IBM MQ 9.3 hat weitere Unterstützung für IBM MQ -AMQP-Kanäle hinzugefügt, sodass Apache Qpid JMS -Anwendungen eine Verbindung zu IBM MQ herstellen und Punkt-zu-Punkt-Messaging durchführen können. Weitere Informationen finden Sie unter „Punkt-zu-Punkt-Unterstützung auf AMQP-Kanälen” auf Seite 717.

In IBM MQ 9.3.0 wurde weitere Unterstützung zum Durchsuchen von Warteschlangen für IBM MQ -AMQP-Kanäle hinzugefügt, sodass Apache Qpid JMS-Anwendungen eine Verbindung zu IBM MQ herstellen und das Durchsuchen von Nachrichten aus einer Warteschlange ausführen können. Weitere Informationen finden Sie unter „Punkt-zu-Punkt-Unterstützung auf AMQP-Kanälen” auf Seite 717.

IBM MQ 9.3.0 hat zwei zusätzliche Kanalattribute für AMQP-Kanäle hinzugefügt: TMPMoDEL und TMPQPRFX. Diese Attribute sind für Modellwarteschlange und dienen als vorläufiges Warteschlangenpräfix bei der Erstellung einer temporären Warteschlange.

Übergreifende Kommunikation mit anderen IBM MQ-Anwendungen

Es ist möglich, Nachrichten zwischen Apache Qpid JMS-Anwendungen und anderen IBM MQ-Anwendungen auszutauschen. Beispielsweise kann eine Apache QPid-Anwendung Nachrichten zu einem Thema veröffentlichen, und MQ Light-Anwendungen können diese empfangen, indem sie eine Subskription erstellen.

Eine Apache Qpid JMS-Anwendung kann auch Nachrichten veröffentlichen, die von traditionellen IBM MQ-Anwendungen verarbeitet werden, z. B. indem sie mit dem API-Aufruf MQSUB das betreffende Thema subskribieren.

Entsprechend können Apache Qpid JMS-Anwendungen IBM MQ-Themen subskribieren, zu denen traditionelle IBM MQ-Anwendungen Nachrichten veröffentlichen.

Es ist auch möglich, dass eine Apache Qpid JMS-Anwendung und eine MQ Light-Anwendung eine Subskription gemeinsam nutzen, sofern beide Clients denselben Share-Namen und dasselbe Themenmuster angeben.

Dies setzt allerdings voraus, dass die Apache Qpid JMS-Anwendung die Verbindung ohne eine Client-ID herstellt. Dadurch wird sichergestellt, dass der von beiden Anwendungen verwendete IBM MQ-Subskriptionsname identisch ist.

Achtung: Eine JMS-Anwendung von Apache Qpid kann eine Subskription nicht gemeinsam mit einer IBM MQ-JMS-Anwendung nutzen.

Einschränkungen für Apache Qpid JMS

Es werden folgende JMS-Leistungsmerkmale unterstützt:

- Clientbestätigung, automatische Bestätigung und verzögerte Bestätigung (DUPS_OK_ACKNOWLEDGE)
  - Herstellen einer Verbindung mit oder ohne Berechtigungsnachweise
  - Erstellen eines Konsumenten für ein Themenziel
  - Erstellen eines permanenten Konsumenten für ein Themenziel
  - Erstellen eines gemeinsam genutzten Konsumenten für ein Themenziel
  - Erstellen eines gemeinsam genutzten, permanenten Konsumenten für ein Themenziel
  - Clientbestätigungsmodus und Modus für automatische Bestätigung
  - Nachrichtenbestätigung und Sitzungsbestätigung
  - Abmelden von einer permanenten Subskription
  - Erstellen einer temporären Warteschlange
  - Erstellen eines Konsumenten für eine Warteschlange oder temporäre Warteschlange als Ziel
  - JMS MessageListeners
  - JMS-Konsument für den Empfang des Hauptteils; die Methode JMS 2.0 hat `Consumer.receiveBody()` aufgerufen.
  - Folgende JMS-Nachrichtentypen werden unterstützt:
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - Nachrichten aus einer Warteschlange anzeigen

Folgende JMS-Leistungsmerkmale werden von AMQP-Clients nicht unterstützt:

- Verwendung von Sitzungen mit Transaktionsunterstützung und JMSContexts mit Transaktionsunterstützung
  - Verwendung von Nachrichtenselektoren
  - Verwendung des Attributs **`nolocal`**
  - Verwendung von Sitzungen mit Transaktionsunterstützung
  - Verwendung von Zustellungsverzögerungen
  - In IBM MQ 9.3.0 Nachrichten aus einer Warteschlange anzeigen
  - Mehrere permanente Subskriptionen oder Konsumenten mit der gleichen Client-ID und demselben Thema erstellen
  - Temporäre JMS-Themen
  - AMQP-Filter werden nicht unterstützt.

AMQP-Beispielclients herunterladen

IBM MQ stellt keine AMQP-Clients bereit, aber Sie können MQ Light-Clients oder Open-Source-AMQP-Clients basierend auf Apache Qpid-Bibliotheken herunterladen. Weitere Informationen finden Sie unter IBM MQ Light und Apache-Qpid.

Sie können auch andere Open-Source-AMQP-Clients auf Basis von Apache Qpid-Bibliotheken herunterladen. Weitere Informationen finden Sie unter https://qpid.apache.org/index.html.

Achtung: IBM Support kann keine Konfigurations-oder Fehlerunterstützung für diese Clientpakete bereitstellen und alle Verwendungsfragen oder Codefehlerberichte sollten an die entsprechenden Projekte weitergeleitet werden.

AMQP-Clients in IBM MQ implementieren

Eine Anwendung, die bereit zur Implementierung ist, erfordert alle Überwachungs-, Zuverlässigkeits- und Sicherheitsfunktionen anderer Unternehmensanwendungen. Sie kann auch Daten mit anderen Unternehmensanwendungen austauschen.

Nachdem Sie einen AMQP-Client implementiert haben, können Sie Nachrichten mit IBM MQ-Anwendungen austauschen. Wenn Sie beispielsweise den AMQP-Client zum Senden einer JavaScript-Zeichenfolgenachricht verwenden, empfängt die IBM MQ-Anwendung eine MQ-Nachricht, bei der das Formatfeld des MQMD auf MQSTR gesetzt ist.

AMQP-Kanal verwalten

Der AMQP-Kanal kann auf die gleiche Weise wie andere MQ-Kanäle verwaltet werden. Sie können die Kanäle mit MQSC-Befehlen, PCF-Befehlsnachrichten oder IBM MQ Explorer definieren, starten, stoppen und verwalten. Im Abschnitt AMQP-Kanäle erstellen und verwenden wird anhand von Beispielbefehlen beschrieben, wie eine Verbindung zu einem Warteschlangenmanager definiert und gestartet wird.

Wenn ein AMQP-Kanal gestartet wird, können Sie ihn testen, indem Sie eine Verbindung zu einem AMQP 1.0-Client herstellen. Beispiel: MQ Light, Apache Qpid Proton oder Apache Qpid JMS.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW MQ Light, Apache Qpid JMSund AMQP (Advanced Message Queuing Protocol)

Der MQ Light -Client, Apache Qpid-Clients wie Apache Proton und Apache Qpid JMS -APIs basieren auf dem Verbindungsprotokoll OASIS Standard AMQP 1.0 . AMQP gibt an, wie Nachrichten zwischen Sendern und Empfängern gesendet werden. Eine Anwendung fungiert als Sender, wenn die Anwendung eine Nachricht an den Nachrichtenbroker sendet, wie z. B. IBM MQ. IBM MQ fungiert als Sender, wenn eine Nachricht an eine AMQP-Anwendung gesendet wird.

Einige der Vorteile von AMQP lauten wie folgt:

- Offenes standardisiertes Protokoll
- Kompatibilität mit anderen AMQP 1.0-Open-Source-Clients
- Viele Open-Source-Client-Implementierungen verfügbar

Obwohl jeder AMQP 1.0-Client eine Verbindung zu einem AMQP-Kanal herstellen kann, werden einige AMQP-Funktionen nicht unterstützt, z. B. Transaktionen oder mehrere Sitzungen.

Weitere Informationen finden Sie auf der AMQP.org-Website und in der PDF OASIS Standard AMQP 1.0.

Die MQ Light and Apache Qpid JMS-APIs verfügen über folgende Messaging-Funktionen:

- At-most-once-Nachrichtenübermittlung (höchstens einmal)
- At-least-once-Nachrichtenübermittlung (mindestens einmal)
- Zieladressierung für Themenzeichenfolge
- Nachrichten- und Zielpermanenz
- Gemeinsam genutzte Ziele, damit mehrere Subskribenten die Workload gemeinsam nutzen können
- Client-Übernahme für eine einfache Auflösung blockierter Clients
- Konfigurierbares Vorauslesen von Nachrichten

- Konfigurierbares Bestätigen von Nachrichten

Die vollständige Dokumentation der Apache Qpid JMS-API finden Sie unter Qpid JMS.

Zugehörige Tasks
AMQP-Kanäle erstellen und verwenden
AMQP-Clients schützen

ALW AMQP 1.0-Unterstützung

AMQP-Kanäle bieten eine Unterstützungsstufe für AMQP 1.0-konforme Anwendungen.

AMQP-Kanäle unterstützen eine Untergruppe des AMQP 1.0-Protokolls. Sie können AMQP 1.0-kompatible Clients mit einem IBM MQ AMQP-Kanal verbinden. Damit alle von AMQP-Kanälen unterstützten Messaging-Funktionen verwendet werden können, müssen Sie den Wert bestimmter AMQP 1.0-Felder ordnungsgemäß festlegen.

Diese Informationen zeigen, wie AMQP-Felder formatiert werden müssen, und enthalten die Funktionen der AMQP 1.0-Spezifikation, die nicht von AMQP-Kanälen unterstützt werden.

Die folgenden Funktionen der AMQP 1.0-Spezifikation werden entweder nicht unterstützt oder sind in ihrer Verwendung begrenzt:

ATTACH-Frame

AMQP-Kanäle erwarten, dass die Funktionen im ATTACH-Frame eine der folgenden Bedingungen enthalten:

```
topic
temporary queue
queue
shared
```

Die Funktionen implizieren den Objekttyp und im Fall der Multifunktionalität ist die Prioritätsreihenfolge bei der Auswahl der Funktion topic, temporary-queue, queue.

Wenn eine Funktion keinen erwarteten Wert enthält, ist die Standardfunktion topic. Alle anderen Funktionen werden ignoriert.

Anmerkung: Einige AMQP-Clients legen diese Funktionalität nicht fest und erhalten das IBM MQ -Standardverhalten Publish/Subscribe. Der Connector Quarkus Reactive Messaging AMQP 1.0 setzt beispielsweise nur Funktionen ab Version 2.8.0CR1 .

AMQP-Kanäle erwarten, dass die `distribution-Mode` im ATTACH-Rahmen eine der folgenden Angaben für eine Quelle oder ein Ziel enthält:

- `move`
- `copy`

Dabei schließt `move` einen Abruf mit Löschen und `copy` einen Browser ein.

Anmerkung: Wenn `distribution-Mode` nicht oder auf einen anderen Wert als `Kopie` gesetzt ist, wird `Verschieben` angenommen.

Linknamen

AMQP-Kanäle erwarten, dass der Name eines AMQP-Links einem der folgenden Formate folgt:

- Einfaches Thema (für Veröffentlichung und Subskription)
  - Nachrichten veröffentlichen: Eine einfache Themenzeichenfolge (beispielsweise der Linkname "`/sports/football`") bewirkt, dass eine Nachricht im Thema `/sports/football` veröffentlicht wird.

  - Subskribieren eines Themas, um Nachrichten zu empfangen: Eine einfache Themenzeichenfolge (z. B. der Linkname "`/sports/football`") bewirkt, dass eine Subskription für das Thema `/sports/football` definiert wird.
- Privates ausführliches Thema (für Subskription)
  - Eine ausführliche Topiczeichenfolge, die eine private Subskription im Format "`private:`*`topic string`*" beschreibt (z. B. "`private:/sports/football`"). Das Verhalten ist mit einer einfachen Themenzeichenfolge identisch. Die Deklaration `private` unterscheidet eine Subskription, die für einen bestimmten AMQP-Client spezifisch ist, von einer Subskription, die von Clients gemeinsam genutzt wird.
- Gemeinsam genutztes ausführliches Thema (für Subskription)
  - Eine ausführliche Themenzeichenfolge, die eine gemeinsam genutzte Subskription im Format "`share:`*`share name`*`:`*`topic string`*" beschreibt (z. B. "`share:bbc:/sports/football`").
- Eine Warteschlange (für Punkt-zu-Punkt-Messaging für Produzenten und Konsumenten)
  - Produzent für das Senden von Nachrichten; eine Zeichenfolge mit Warteschlangennamen bewirkt, dass ein Produzent eine Nachricht in einer Warteschlange sendet.
  - Konsumenten für den Empfang von Nachrichten; eine Zeichenfolge mit Warteschlangennamen bewirkt, dass ein Konsument Nachrichten aus einer Warteschlange empfängt.
- Leer (für Punkt-zu-Punkt-Messaging in einer temporären Warteschlange)
  - Produzent für das Senden von Nachrichten in einer temporären Warteschlange; Leer bewirkt, dass ein Produzent eine Nachricht in einer temporären Warteschlange sendet.
  - Konsument für den Empfang von Nachrichten in einer temporären Warteschlange; Leer bewirkt, dass ein Konsument Nachrichten aus einer temporären Warteschlange empfängt.

Weitere Informationen zur Zuordnung von AMQP-Nachrichten zu und von IBM MQ -Nachrichten finden Sie unter „Zuordnung von AMQP-Feldern zu IBM MQ-Feldern (eingehende Nachrichten)” auf Seite 721.

Maximale Längen für Themenzeichenfolgen, gemeinsam genutzte Namen und Client-IDs

Die Themenzeichenfolge, der gemeinsam genutzte Name und die Client-ID müssen in 10237 Bytes enthalten sein. Darüber hinaus beträgt die maximale Länge einer Client-ID 256 Zeichen.

Diese maximalen Längen bedeuten, dass Sie einen der folgenden Werte haben können:

- eine sehr lange Themenzeichenfolge, sofern der gemeinsam genutzte Name kurz ist
- einen langen gemeinsam genutzten Namen, jedoch eine kurze Themenzeichenfolge

Container-IDs

AMQP-Kanäle erwarten, dass die Container-ID eines AMQP Open-Performativs eine eindeutige AMQP-Client-ID enthält. Die maximale Länge einer AMQP-Client-ID beträgt 256 Zeichen und die ID kann alphanumerische Zeichen, Prozentzeichen (%), Schrägstrich (/), Punkt (.) und Unterstreichungszeichen (_) enthalten.

Sitzungen

AMQP-Kanäle unterstützen nur eine einzige AMQP-Sitzung. Ein AMQP-Client, der versucht, mehr als eine AMQP-Sitzung zu erstellen, erhält eine Fehlernachricht und wird vom Kanal getrennt.

Transaktionen

AMQP-Kanäle unterstützen keine AMQP-Transaktionen. Ein AMQP-Verbindungsrahmen, der versucht, eine neue Transaktion zu koordinieren, oder ein AMQP-Übertragungsrahmen, der versucht, eine neue Transaktion zu deklarieren, wird mit einer Fehlernachricht abgelehnt.

Übermittlungsstatus

AMQP-Kanäle unterstützen nur einen Übermittlungsstatus für Dispositionsrahmen 'Accepted', 'Released' oder 'Modified'. Es ist zu beachten, dass bei Verwendung des Status 'Modified' die AMQP-Kanäle die Option 'undeliverable-here' nicht unterstützten.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW Punkt-zu-Punkt-Unterstützung auf AMQP-Kanälen

Der IBM MQ AMQP-Kanal bietet Unterstützung für das Senden von Nachrichten an Warteschlangen und Empfangen von Nachrichten aus Warteschlangen.

AMQP-Clients wie eine Apache Qpid™ JMS-Bibliothek fordern beim Senden eines AMQP-Attach-Frames eine `queue`- oder `temporary-queue`-Funktionalität an. Funktionen ermöglichen es dem AMQP-Kanal, das Objekt als eine Warteschlange, eine temporäre Warteschlange oder ein Thema zu identifizieren. Wenn weder eine 'queue'- noch eine 'temporary-queue'-Funktion oder wenn gar keine der Funktionen vorhanden ist, wird angenommen, dass sich die Anforderung auf ein Thema bezieht.

IBM MQ AMQP-Kanäle stellen Warteschlangentypunterstützung für Folgendes bereit:

An Warteschlange senden und aus Warteschlange empfangen
Es können Nachrichten an eine Warteschlange gesendet und aus einer Warteschlange abgerufen werden. Für das Abrufen von Nachrichten werden sowohl der synchrone als auch der asynchrone Modus unterstützt.

Nachricht aus Warteschlange anzeigen
Nachrichten können nicht nur in eine Warteschlange eingereiht und aus einer Warteschlange abgerufen werden, es können auch Nachrichten aus einer Warteschlange angezeigt werden.

Unterstützung für temporäre Warteschlangen
Es können Nachrichten an eine temporäre Warteschlange gesendet und aus einer temporären Warteschlange abgerufen werden. Beachten Sie, dass die Löschung einer temporären Warteschlange unterstützt wird, wenn dasselbe temporäre Warteschlangenobjekt, das zum Erstellen der temporären Warteschlange verwendet wurde, auch zum Löschen der temporären Warteschlange verwendet wird.

Eine temporäre Warteschlange wird mithilfe der SYSTEM.DEFAULT.MODEL.QUEUE erstellt und die temporäre Warteschlange erhält das Präfix `AMQP.*`.

Die SYSTEM.DEFAULT.MODEL.QUEUE ist standardmäßig eine temporäre dynamische Warteschlange, aber Sie können die Warteschlange über die Eigenschaft **`Definition type`** in der Warteschlange SYSTEM.DEFAULT.MODEL.QUEUE zu einer permanenten dynamischen Warteschlange machen.

Permanente dynamische Warteschlange
Eine permanente dynamische Warteschlange wird gelöscht, wenn ein AMQP-Client, z. B. eine Apache Qpid JMS-Bibliothek, eine Anforderung mit einem `detach`-Frame sendet, wobei das Attribut **`closed`** auf *true* gesetzt ist.

Wichtig:

Qpid-JMS-Verhalten:
Sie müssen einen Qpid-JMS-API-Befehl aufrufen, z. B. die Methode `javax.jms.TemporaryQueue.delete()`, um die Warteschlange nach der Nutzung zu löschen. Bei diesem Prozess werden auch die Nachrichten, die in der Warteschlange enthalten sind, gelöscht.

Wird ein solcher Befehl nicht ausgegeben, bleibt die Warteschlange mit allen enthaltenen Nachrichten erhalten, wenn die Verbindung geschlossen wird.

-

Temporäre dynamische Warteschlange
Eine temporäre dynamische Warteschlange wird gelöscht, wenn der AMQP-Client die Verbindung schließt.

Wichtig:

Qpid-JMS-Verhalten:

Wenn Sie einen Qpid-JMS-API-Befehl aufrufen, z. B. die Methode `javax.jms.TemporaryQueue.delete()`, die JMS-Verbindung schließen oder die Verbindung unterbrochen wird, wird die Warteschlange gelöscht und es gehen alle Nachrichten verloren.

Das Schließen einer JMS-Sitzung führt nicht dazu, dass die temporäre Warteschlange gelöscht wird, selbst wenn die temporäre Warteschlange mit der Methode `javax.jms.Session.createTemporaryQueue()` erstellt worden wäre.

-

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW AMQP- und IBM MQ-Nachrichtenfelder zuordnen

AMQP-Nachrichten bestehen aus einem Header, Zustellungsanmerkungen, Nachrichtenanmerkungen, Eigenschaften, Anwendungseigenschaften, Hauptteil und Fußzeile.

AMQP-Nachrichten sind aus folgenden Teilen zusammengesetzt:

Header

Der optionale Header enthält fünf feste Attribute der Nachricht:

- **durable** - gibt Permanenzanforderungen an
- **priority** - relative Nachrichtenpriorität
- **ttl** - Lebensdauer in Millisekunden
- **first-acquirer** - wenn 'true', wurde die Nachricht von keinem anderen Link angefordert
- **delivery-count** - Anzahl vorheriger, nicht erfolgreicher Zustellversuche

Zustellungsanmerkungen (delivery-annotations)

Optional. Geben vom Standard abweichende Headerattribute der Nachricht für andere Zielgruppen an. Zustellungsanmerkungen transportieren Informationen vom sendenden Peer an den empfangenden Peer.

Nachrichtenanmerkungen (message-annotations)

Optional. Geben vom Standard abweichende Headerattribute der Nachricht für andere Zielgruppen an. Der Abschnitt mit Nachrichtenanmerkungen enthält Eigenschaften der Nachricht, die sich auf die Infrastruktur beziehen und über alle Zustellschritte hinweg weitergegeben werden sollten.

Eigenschaften

Optional. Dieser Teil entspricht dem MQ-Nachrichtendeskriptor. Er enthält die folgenden festen Felder:

- **message-id** - Anwendungsnachrichten-ID
- **user-id** - ID des erstellenden Benutzers
- **to** - Adresse des Knotens, an den die Nachricht gerichtet ist
- **subject** - Betreff der Nachricht
- **reply-to** - Knoten, an den die Antworten gehen
- **correlation-id** - Korrelations-ID der Anwendung
- **content-type** - MIME-Inhaltstyp
- **content-encoding** - MIME-Inhaltstyp. Wird als Modifikator für den Inhaltstyp verwendet.
- **absolute-expiry-time** - Zeitpunkt, an dem die Nachricht als abgelaufen gilt
- **creation-time** - Zeitpunkt der Erstellung der Nachricht
- **group-id** - Gruppe, zu der die Nachricht gehört

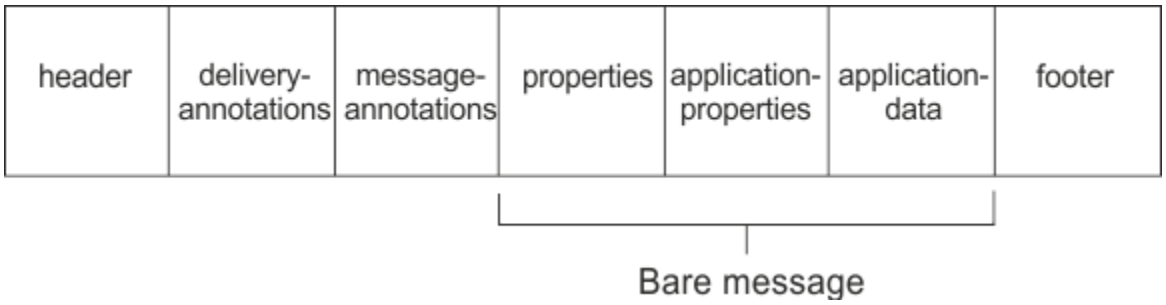
- **group-sequence** - Folgenummer der Nachricht innerhalb der Gruppe
- **reply-to-group-id** - Gruppe, zu der die Antwortnachricht gehört

Anwendungseigenschaften (application-properties)
Entsprechen den MQ-Nachrichteneigenschaften.

Hauptteil
Entspricht den MQ-Benutzernutzdaten.

Fußzeile (footer)
Optional. Die Fußzeile enthält Details zur Nachricht oder Zustellung, die nur berechnet oder ausgewertet werden können, nachdem die vollständige Bare-Nachricht erstellt oder angezeigt wurde (z. B. Nachrichtenhashes, HMACs, Signaturen und Verschlüsselungsdetails).

Das AMQP-Nachrichtenformat wird in der folgenden Abbildung dargestellt:



Die Eigenschaften, Anwendungseigenschaften und Anwendungsdaten werden als "Bare-Nachricht" bezeichnet. Dies ist die Nachricht, wie sie vom Sender gesendet wurde; sie ist nicht veränderbar. Der Empfänger sieht die vollständige Nachricht, einschließlich Header, Fußzeile, Zustellungsanmerkungen und Nachrichtenanmerkungen.

Eine vollständige Beschreibung des AMQP 1.0-Nachrichtenformats finden Sie im OASIS-Standard unter https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf.

Zugehörige Tasks
AMQP-Kanäle erstellen und verwenden
AMQP-Clients schützen

ALW Zuordnung von IBM MQ-Feldern zu AMQP-Feldern (abgehende Nachrichten)

Wenn eine IBM MQ-Nachricht veröffentlicht wird und IBM MQ sie an einen AMQP-Konsumenten sendet, werden einige der Attribute der IBM MQ-Nachricht an entsprechende AMQP-Nachrichtenattribute weitergegeben.

Header

Ein Header ist nur enthalten, wenn eines der fünf Felder im Header einen vom Standard abweichenden Wert enthält. Nur die Felder mit einem anderen Wert als dem Standardwert sind im Header eingeschlossen. Die fünf Headerfelder werden ursprünglich aus der entsprechenden `mq_amqp.Hdr`-Eigenschaft, sofern festgelegt, abgeleitet und dann wie in der folgenden Tabelle gezeigt modifiziert:

Tabelle 101. Headerfeldzuordnungen

Feld	Standardwert	Wert
durable	false	'true', wenn `MQMD.Persistence` auf `MQPER_PERSISTENT` gesetzt ist, andernfalls 'false'.

Tabelle 101. Headerfeldzuordnungen (Forts.)		
Feld	**Standardwert**	**Wert**
priority	4	Aus `mq_amqp.Hdr.Pri`, wenn festgelegt, oder andernfalls aus `MQMD.Priority`, wenn festgelegt. Ist keine der beiden festgelegt, wird 4 eingestellt.
ttl	nicht zutreffend	`MQMD.Expiry` in Millisekunden. Wenn `MQMD.Expiry` den Wert MQEI_UNLIMITED hat, wird das AMQP-Feld ttl auf den maximalen Wert gesetzt.
first-acquirer	false	Aus `mq_amqp.Hdr.Fac`, wenn festgelegt, oder andernfalls 'false'.
delivery-count	0	Aus `mq_amqp.Hdr.Dct`, wenn festgelegt, oder andernfalls '0'.

delivery-annotations

Wird je nach Bedarf vom AMQP-Kanal festgelegt.

message-annotations

Nicht enthalten.

Eigenschaften

Die **properties** (Eigenschaften) kommen unverändert aus den entsprechenden `mq_amqp.Prp`-Eigenschaften, wenn diese festgelegt sind. Wenn die Nachricht ursprünglich keine AMQP-Nachricht war (d. h. PutApplType ist nicht MQAT_AMQP), wird ein 'properties'-Abschnitt generiert, so wie in der folgenden Tabelle beschrieben:

Tabelle 102. Eigenschaftsfeldzuordnungen	
Name	**Wert**
message-id	Die `MQMD.MsgId` wird als binär festgelegt.
Benutzer-ID	Das UTF-8-Format der `MQMD.UserIdentifier` wird in Netzbyte-Reihenfolge als binär festgelegt.
in	Die Warteschlange, aus der die Nachricht abgerufen wurde, bzw. die Themenzeichenfolge bei einer Veröffentlichung.
subject	Nicht festgelegt.
reply-to	Die `MQMD.ReplyToQ`, falls angegeben, andernfalls nicht festgelegt.
correlation-id	Die `MQMD.CorrelId` wird als binär festgelegt, falls angegeben; andernfalls nicht festgelegt.
content-type	Nicht festgelegt.
content-encoding	Nicht festgelegt.
absolute-expiry-time	Nicht festgelegt.
creation-time	Aus den Feldern `MQMD.PutDate` und `MQMD.PutTime` wird eine Zeitmarke generiert.
group-id	Nicht festgelegt.

Tabelle 102. Eigenschaftsfeldzuordnungen (Forts.)	
Name	**Wert**
group-sequence	Nicht festgelegt.
reply-to-group-id	Nicht festgelegt.

application-properties

Alle IBM MQ-Eigenschaften in der Gruppe "usr" werden als die **application-properties** (Anwendungseigenschaften) hinzugefügt.

Hauptteil

Der AMQP-Kanal führt einen GET mit Konvertierung aus, um die IBM MQ-Nutzdaten in UTF-8 zu konvertieren.

Wenn die IBM MQ-Nutzdaten keine AMQP-Nachricht enthalten, werden die IBM MQ-Nutzdaten im Hauptteil (body) als ein einzelner Zeichenfolgedatenabschnitt für das Format MQFMT_STRING (sofern die Konvertierung in UTF-8 erfolgreich war) oder andernfalls als ein einzelner binärer Datenabschnitt festgelegt.

Wenn eine AMQP-Formatnachricht eingeschlossen ist, wird diese als Hauptteil festgelegt. Alle IBM MQ-Header (ohne die Nachrichteneigenschaften, die in einem Nachrichtenhandle zurückgegeben werden), die der AMQP-Nachricht vorangehen, werden als Binärwert vorangestellt, wenn der Hauptteil eine AMQP-Sequenz ist. Andernfalls werden die IBM MQ-Header verworfen.

footer

Es ist keine Fußzeile enthalten.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

Zugehörige Verweise

MQMD - Nachrichtendeskriptor

ALW Zuordnung von AMQP-Feldern zu IBM MQ-Feldern (eingehende Nachrichten)

Wenn der AMQP-Kanal eine Nachricht empfängt und an IBM MQ übergibt, werden dabei einige der Attribute der AMQP-Nachricht an entsprechende IBM MQ-Nachrichtenattribute weitergegeben.

Bei der Zuordnung einer eingehenden AMQP-Nachricht gelten folgende Einschränkungen:

- Wenn das Feld `message-id` oder `correlation-id` im 'properties'-Teil eine 'uuid' oder eine 'ulong' ist, wird die Nachricht zurückgewiesen.
- Ein `message-annotations`-Abschnitt führt dazu, dass die Nachricht zurückgewiesen wird.
- Die Abschnitte `delivery-annotations` und `footer` sind zulässig, werden aber nicht an die IBM MQ-Nachricht weitergegeben.

In den folgenden Unterabschnitten wird der IBM MQ-Ausdruck einer AMQP-Nachricht gezeigt.

Nachrichtendeskriptor

Tabelle 103. Nachrichtendeskriptor für AMQP-Nachricht	
Feld	**Wert**
StrucId	MQMD_STRUC_ID

Tabelle 103. Nachrichtendeskriptor für AMQP-Nachricht (Forts.)	
Feld	**Wert**
Version	MQMD_VERSION_1
Bericht	MQRO_NONE
MsgType	MQMT_DATAGRAM
Verfall	Wert, der aus dem Feld `ttl` im AMQP-Nachrichtenheader übernommen wird.
Feedback	MQFB_NONE
Encoding	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
Format	Siehe Nutzdaten
Priority	Wert, der aus dem Feld `priority` im AMQP-Nachrichtenheader übernommen wird. Wenn festgelegt, ist der Wert auf maximal 9 begrenzt. Wenn nicht festgelegt, wird der Standardwert 4 verwendet.
Permanenz	Ist das Feld `durable` im AMQP-Nachrichtenheader auf 'true' gesetzt, wird dieses Feld auf MQPER_PERSISTENT gesetzt. Andernfalls wird es auf MQPER_NOT_PERSISTENT gesetzt.
MagId	Der Warteschlangenmanager ordnet eine eindeutige 24-Byte-Nachrichten-ID zu.
Correlld	Wert, der aus dem Feld `correlation-id` in den AMQP-Eigenschaften übernommen wird, wenn festgelegt. Wird auf einen binären 24-Byte-Wert gesetzt. Andernfalls wird es auf MQCI_NONE/ gesetzt.
BackoutCount	0
ReplyToQ	Wert, der aus dem Feld `reply-to` in den AMQP-Eigenschaften übernommen wird, wenn festgelegt. Andernfalls auf "" gesetzt.
ReplyToQMgr	""
Bericht	Wert, der von JMS IBM Berichtseigenschaften abgeleitet wird, die in den AMQP-Anwendungseigenschaften festgelegt sind.
UserIdentifier	Wird auf die ID des authentifizierten Benutzers gesetzt, der eine Verbindung zum AMQP-Kanal hergestellt hat.
AccountingToken	MQACT_NONE
ApplIdentityData	Hexadezimale Zeichenfolge. Wird auf die letzten 8 Bytes der MQ-Verbindungs-ID des AMQP-Kanals gesetzt.
PutApplType	MQAT_AMQP
PutApplName	
PutDate	Wert, der aus dem Feld `creation-time` der AMQP-Eigenschaften übernommen wird, wenn festgelegt. Andernfalls wird das aktuelle Datum verwendet.
PutTime	Wert, der aus dem Feld `creation-time` der AMQP-Eigenschaften übernommen wird, wenn festgelegt. Andernfalls wird die aktuelle Uhrzeit verwendet.
ApplOriginData	""

Nachrichteneigenschaften

Es gibt zwei Gründe zum Festlegen von Nachrichteneigenschaften:

- Teile der AMQP-Nachricht sollen den Warteschlangenmanager ohne Auswirkung auf die Nutzdaten der Nachricht durchlaufen können.
- Es soll eine Auswahl der `application-properties` ermöglicht werden.

Die folgende Tabelle zeigt die Eigenschaften, die aus der AMQP-Nachricht festgelegt werden:

Tabelle 104. AMQP-Nachrichteneigenschaften

Eigenschaftsname	MQRFH2-Name	Typ	Beschreibung
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	Eine identifizierende Zeichenfolge für den AMQP-Kanal. Sie wird zum Generieren der Nachricht verwendet, damit interessierte Parteien erkennen können, von welcher Version die Nachricht kommt (z. B. das Service-Team bei der Diagnose von Problemen). Der Wert wird nicht vom Warteschlangenmanager überprüft und darf nicht extern dokumentiert werden.
AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	Die Version der AMQP-Nachricht. Wenn nicht vorhanden, wird "1.0" angenommen. Der Wert wird nicht vom Warteschlangenmanager überprüft.
AMQPClient	mq_amqp.Cli	MQTYPE_STRING	Eine identifizierende Zeichenfolge für die API. Sie wird zum Senden der AMQP-Nachricht an den Kanal verwendet, damit interessierte Parteien erkennen können, von welcher Version die Nachricht kommt (z. B. das Service-Team bei der Diagnose von Problemen). Der Wert wird nicht vom Warteschlangenmanager überprüft und darf nicht extern dokumentiert werden.
AMQPDurable	mq_amqp.Hdr.Dur	MQTYPE_BOOLEAN	Der Wert des Felds `durable` im AMQP-Nachrichtenheader, wenn festgelegt.
AMQPPriority	mq_amqp.Hdr.Pri	MQTYPE_INT32	Der Wert des Felds `priority` im AMQP-Nachrichtenheader, wenn festgelegt.
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	Der Wert des Felds `ttl` im AMQP-Nachrichtenheader, wenn festgelegt.
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQTYPE_BOOLEAN	Der Wert des Felds `first-acquirer` im AMQP-Nachrichtenheader, wenn festgelegt.

Tabelle 104. AMQP-Nachrichteneigenschaften (Forts.)

Eigenschaftsname	MQRFH2-Name	Typ	Beschreibung
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	Der Wert des Felds `delivery-count` im AMQP-Nachrichtenheader, wenn festgelegt.
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	Der Wert des Felds `message-id` in den AMQP-Eigenschaften, wenn als Zeichenfolge festgelegt.
		MQTYPE_BYTE_STRING	Der Wert des Felds `message-id` in den AMQP-Eigenschaften, wenn als Bytefolge festgelegt.
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	Der Wert des Felds `user-id` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	Der Wert des Felds `to` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	Der Wert des Felds `subject` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	Der Wert des Felds `reply-to` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	Der Wert des Felds `correlation-id` in den AMQP-Eigenschaften, wenn als Zeichenfolge festgelegt.
		MQTYPE_BYTE_STRING	Der Wert des Felds `correlation-id` in den AMQP-Eigenschaften, wenn als Bytefolge festgelegt.
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	Der Wert des Felds `content-type` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPContentEncoding	mq_amqp.Prp.Cne	MQTYPE_STRING	Der Wert des Felds `content-encoding` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPAbsoluteExpiryTime	mq_amqp.Prp.Aet	MQTYPE_STRING	Der Wert des Felds `absolute-expiry-time` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	Der Wert des Felds `creation-time` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	Der Wert des Felds `group-id` in den AMQP-Eigenschaften, wenn festgelegt.
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	Der Wert des Felds `group-sequence` in den AMQP-Eigenschaften, wenn festgelegt.

Tabelle 104. AMQP-Nachrichteneigenschaften (Forts.)			
Eigenschaftsname	**MQRFH2-Name**	**Typ**	**Beschreibung**
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	Der Wert des Felds `reply-to-group-id` in den AMQP-Eigenschaften, wenn festgelegt.

Jede der Anwendungseigenschaften (application-properties) aus der AMQP-Nachricht wird als eine IBM MQ-Nachrichteneigenschaft festgelegt. Der Abschnitt `application-properties` muss Byte für Byte identisch rekonstruiert werden. Deshalb gelten folgende Einschränkungen:

- Wird eine Anwendungseigenschaft vom MQSETMP-Validierungscode abgelehnt, wird die Nachricht zurückgewiesen. For example:
  - Die Länge des Eigenschaftsnamens ist auf MQ_MAX_PROPERTY_NAME_LENGTH begrenzt.
  - Der Eigenschaftsname muss den Regeln entsprechen, die in der Java Sprachspezifikation für Java -IDs definiert sind.
  - Der Eigenschaftsname darf nicht mit `JMS` oder `usr.JMS` beginnen, mit Ausnahme der dokumentierten JMS-Eigenschaften, die festgelegt werden können.
  - Der Eigenschaftsname darf kein SQL-Schlüsselwort sein.
- Eine Anwendungseigenschaft, die das Unicode-Zeichen U+002E (".") enthält, bewirkt, dass die Nachricht zurückgewiesen wird. Die Eigenschaft muss in der Eigenschaftengruppe "usr", die von JMS verwendet wird, ausdrückbar sein.
- Es werden nur folgende Eigenschaftstypen unterstützt: null, boolean, byte, short, int, long, float, double, binary und string. Hat eine Anwendungseigenschaft einen anderen Typ, wird die Nachricht zurückgewiesen.

Sie können die folgenden JMS-Eigenschaften mit `application-properties` festlegen:

- JMS_IBM_REPORT_EXCEPTION
- JMS_IBM_REPORT_EXPIRATION
- JMS_IBM_REPORT_COA
- JMS_IBM_REPORT_COD
- JMS_IBM_REPORT_PAN
- JMS_IBM_REPORT_NAN
- JMS_IBM_REPORT_PASS_MSG_ID
- JMS_IBM_REPORT_PASS_CORREL_ID
- JMS_IBM_REPORT_DISCARD_MSG

Beachten Sie, dass Eigenschaftsnamen und -werte mit den entsprechenden „JMS-Provider-spezifische Felder zuordnen” auf Seite 167-Details konsistent sind und dass ungültige Werte ignoriert werden.

Nutzdaten

- Besteht ein AMQP-Hauptteil (body) nur aus einem Binärdatenabschnitt, werden die Binärdaten (ohne die AMQP-Bits) als die IBM MQ-Nutzdaten mit dem Format MQFMT_NONE übergeben.
- Besteht ein AMQP-Hauptteil (body) nur aus einem Zeichenfolgedatenabschnitt, werden die Zeichenfolgedaten (ohne die AMQP-Bits) als die IBM MQ-Nutzdaten mit dem Format MQFMT_STRING übergeben.
- Andernfalls werden aus dem AMQP-Hauptteil (body) die Nutzdaten unverändert mit dem Format MQFMT_AMQP gebildet.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW Zuverlässigkeit der Nachrichtenübermittlung

In diesem Abschnitt werden die Zuverlässigkeitsfunktionen für die MQ Light -API und Apache Qpid JMSverglichen.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW MQ Light-Nachrichtenzuverlässigkeit

Es gibt vier Features der MQ Light-API, mit denen Sie die Zuverlässigkeit der Nachrichtenübermittlung an und von AMQP-Anwendungen steuern können.

Diese sind:

- „Nachrichtenservicequalität (QOS)” auf Seite 726
- „Automatische Bestätigung des Subskribenten” auf Seite 727
- „Lebensdauer einer Subskription” auf Seite 727
- „Nachrichtenpersistenz” auf Seite 727

Nachrichtenservicequalität (QOS)

Die MQ Light API bietet zwei Servicequalitäten:

- At most once (Höchstens einmal)
- At least once (Mindestens einmal)

Sie können wählen, welche Servicequalität Publisher und Subskribenten verwenden sollen.

Wenn Sie einen MQ Light-Client verwenden, setzen Sie die Option client oder subscribe **qos** auf *QOS_AT_MOST_ONZ* oder *QOS_AT_LEAST_ONCE*.

Wenn Sie einen anderen AMQP-Client verwenden, setzen Sie das Attribut **`settled`** des Übertragungsrahmens (für Publisher) oder den Dispositionsrahmen (für Subskribenten) auf *Wahr* oder *Falsch*, je nachdem, welche Servicequalität Sie erreichen möchten.

Die Servicequalität bestimmt, wann eine Nachricht von der `sending` -Seite eines Dialogs gelöscht wird:

Veröffentlichen

- Wenn ein Publisher **QOS** *0* (höchstens einmal) auswählt, wartet der Publisher nicht auf eine Bestätigung vom Warteschlangenmanager, bevor er seine Kopie der Nachricht löscht. Wenn die Verbindung zum Warteschlangenmanager fehlschlägt, bevor der Sendevorgang abgeschlossen ist, wird die Nachricht möglicherweise nicht von Subskribenten empfangen.
- Wenn ein Publisher **QOS** *1* (mindestens einmal) auswählt, wartet der Publisher darauf, dass der Warteschlangenmanager bestätigt, dass die Nachricht in Subskribentenwarteschlangen geschrieben wurde, bevor er seine Kopie der Nachricht verwirft. Wenn die Verbindung zum Warteschlangenmanager während des Sendens fehlschlägt, sendet der Publisher die Nachricht erneut, sobald er die Verbindung mit dem Warteschlangenmanager wiederhergestellt hat.

Subskribieren

- Wenn ein Subskribent **QOS** *0* auswählt, wartet der Warteschlangenmanager nicht auf eine Bestätigung des Subskribenten, bevor er seine Kopie der Nachricht verwirft. Wenn die Verbindung zum Subskribenten fehlschlägt, bevor der Subskribent die Nachricht empfangen hat, kann diese Nachricht verloren gehen.
- Wenn ein Subskribent **QOS** *1* auswählt, wartet der Warteschlangenmanager auf eine Bestätigung des Subskribenten, bevor er seine Kopie der Nachricht verwirft. V 9.4.0 Ab IBM MQ 9.3.3werden bestätigte Nachrichten in Stapeln entfernt, um die Leistung zu verbessern. Weitere Informationen

finden Sie unter „Bestätigte AMQP-Nachrichten in Stapeln aus der Warteschlange entfernen” auf Seite 729.

Wenn die Verbindung zum Subskribenten fehlschlägt, bevor der Subskribent die Nachricht empfangen hat, bewahrt der Warteschlangenmanager die Nachricht auf. Der Warteschlangenmanager sendet die Nachricht erneut an den Subskribenten, wenn er die Verbindung wiederherstellt, oder an einen anderen Subskribenten, wenn es sich um eine gemeinsam genutzte Subskription handelt.

Automatische Bestätigung des Subskribenten

Wenn ein Subskribent **QOS** *1* (mindestens einmal) auswählt, muss er den Empfang jeder Nachricht bestätigen, bevor der WS-Manager seine Kopie löscht. Der Subskribent kann entscheiden, wann er Nachrichten bestätigt.

Wenn **auto-confirm** auf *Wahr*gesetzt ist, bestätigt der MQ Light-Client automatisch die Zustellung jeder Nachricht, sobald der Client die Nachricht erfolgreich über das Netz empfangen hat.

Auf diese Weise wird sichergestellt, dass die Nachricht bei einem Netzausfall erneut an die Anwendung zugestellt wird. Es ist jedoch immer noch möglich, dass die Anwendung die Nachricht verliert, falls die Anwendung fehlschlägt, nachdem der MQ Light-Client die Nachricht bestätigt und bevor die Anwendung sie verarbeitet hat.

Wenn **auto-confirm** auf *Falsch*gesetzt ist, bestätigt der MQ Light-Client die Zustellung der Nachricht nicht automatisch, sondern überlässt es der Anwendung, zu entscheiden, wann sie bestätigt werden soll.

Dies ermöglicht es einer Anwendung, eine externe Ressource, z. B. eine Datenbank oder eine Datei, zu aktualisieren, bevor sie dem Warteschlangenmanager bestätigt, dass die Nachricht verarbeitet wurde und verworfen werden kann.

Lebensdauer einer Subskription

Wenn eine Anwendung eine Subskription einrichtet, legt sie fest, ob die Subskription und das Ziel, an dem Nachrichten für die Subskription gespeichert werden, bestehen bleiben sollen, nachdem die Anwendung die Verbindung getrennt hat.

Die MQ Light Subskriptionsoption **ttl** wird verwendet, um die Zeit (in Millisekunden) anzugeben, die eine Subskription nach dem Trennen der Anwendungsverbindung bestehen bleibt. Wenn die Anwendung die Verbindung vor Ablauf dieser Zeit wiederherstellt, wird die Subskription wiederaufgenommen und die Anwendung kann weiter Nachrichten von dieser Subskription konsumieren.

Wenn die angegebene Lebensdauer abläuft, ohne dass die Anwendung die Verbindung wiederherstellt, wird die Subskription entfernt und alle an ihrem Ziel gespeicherten Nachrichten gehen verloren, selbst wenn es sich um persistente Nachrichten handelt.

Wenn es wichtig ist, dass keine Nachrichten verloren gehen, müssen Sie einen Lebensdauerwert für die Anwendung angeben, der hoch genug ist, um sicherzustellen, dass es bei einer Betriebsunterbrechung nicht zu Nachrichtenverlusten kommt.

Nachrichtenpersistenz

Die Persistenz von Nachrichten wird durch die Veröffentlichungs- und Subskribierungsanwendungen und durch die Konfiguration von IBM MQ-Themenobjekten gesteuert.

Wenn der AMQP-Subskribent höchstens einmal **QOS** *0* verwendet und eine nicht permanente Subskription erstellt, reiht der AMQP-Kanal unabhängig von den anderen im folgenden Text beschriebenen Optionen immer nicht persistente Nachrichten in die Subskribentenwarteschlange ein.

Beachten Sie, dass bei einem Stopp des Warteschlangenmanagers sowohl die Subskription als auch die Nachrichten verloren gehen.

Wenn ein AMQP-Publisher den AMQP-Header **durable** auf *Wahr*setzt, reiht der AMQP-Kanal persistente Nachrichten in Subskribentenwarteschlangen ein.

Wenn der Warteschlangenmanager aus irgendeinem Grund gestoppt wird, stehen die Nachrichten den Subskribenten nach einem Neustart des Warteschlangenmanager weiterhin zur Verfügung.

Wenn der Header **durable** nicht festgelegt ist, wählt der AMQP-Kanal die Persistenz veröffentlichter Nachrichten basierend auf dem Attribut **DEFPSIST** des relevanten IBM MQ -Themenobjekts aus.

Dies ist standardmäßig das SYSTEM.BASE.TOPIC, das das Attribut **DEFPSIST** von *NEIN* (nicht persistent) verwendet.

Achtung: Spätere Versionen des MQ Light-Clients unterstützen die Festlegung des permanenten AMQP-Headers nicht.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW Apache Qpid JMS-Nachrichtenzuverlässigkeit

Es gibt vier Features der Apache Qpid™ JMS-Bibliothek, mit denen Sie die Zuverlässigkeit der Nachrichtenübermittlung an und von AMQP-Anwendungen steuern können.

Dabei handelt es sich um die Features für:

- „Veröffentlichen” auf Seite 728/Produzent für Punkt-zu-Punkt-Messaging
  - Nachrichtenablaufzeit
  - Nachrichtenpersistenz
- „Subskribieren” auf Seite 728
  - Subskriptionspermanenz
  - Sitzungsbestätigungsmodus (gilt auch für Punkt-zu-Punkt-Messaging für Konsumenten)

Veröffentlichen

Nachrichtenablaufzeit

Die Festlegung des Lebensdauerwerts des JMS-Produzenten wirkt sich auf die Ablaufzeit für Nachrichten aus, die von diesem Nachrichtenproduzenten veröffentlicht werden.

Stellen Sie sicher, dass der Lebensdauerwert für einen JMS-Producer groß genug ist, damit Nachrichten verarbeitet werden, bevor sie ablaufen.

Alternativ verhindert eine Nichtfestlegung des Lebensdauerwerts, dass die Nachricht aus der Subskriptionswarteschlange abläuft.

Nachrichtenpersistenz

Durch die Einstellung des Übermittlungsmodus des JMS-Nachrichtenproduzenten wird die Persistenz der IBM MQ-Nachricht, die für das angegebene Thema veröffentlicht wird, festgelegt.

Stellen Sie sicher, dass Sie **DeliveryMode**.PERSISTENT für Nachrichten verwenden, die aufbewahrt werden müssen, wenn ein Warteschlangenmanager beendet wird oder ausfällt.

Subskribieren

Subskriptionspermanenz

AMQP-Kanäle unterstützten die Erstellung permanenter Subskriptionen unter Verwendung der permanenten Versionen der JMS-Methoden zum Erstellen von Konsumenten:

- **createDurableConsumer()**
- **createSharedDurableConsumer()**

Sitzungsbestätigungsmodus

Um sicherzustellen, dass eine konsumierte Nachricht vollständig verarbeitet wurde, bevor sie aus der IBM MQ -Subskriptionswarteschlange entfernt wird, erstellen Sie mit **Session**eine JMS -Sitzung.Verwenden Sie den Modus CLIENT_ACKNOWLEDGE und die Methode **message.acknowledge()** , um diese Nachricht und alle anderen Nachrichten, die zuvor in dieser Sitzung empfangen wurden, zu bestätigen.

Zugehörige Konzepte

AMQP-Clientanwendungen entwickeln

Die IBM MQ-Unterstützung für AMQP-APIs ermöglicht es einem IBM MQ-Administrator, einen AMQP-Kanal zu erstellen. Wenn dieser Kanal gestartet wird, definiert dieser Kanal eine Portnummer, die Verbindungen von AMQP-Clientanwendungen akzeptiert.

V 9.4.0 Bestätigte AMQP-Nachrichten in Stapeln aus der Warteschlange entfernen

Wenn eine AMQP-Anwendung die Nachrichtenübermittlung QOS_AT_LEAST_ONCE (1) verwendet, wartet der AMQP-Service auf eine Bestätigung von der Anwendung, bevor er die Kopie einer Nachricht löscht, die er nach dem Senden der Nachricht an die Anwendung aufbewahrt. Ab IBM MQ 9.3.3werden bestätigte Nachrichten in Stapeln und nicht einzeln aus der Warteschlange entfernt, was zu einer verbesserten Leistung führt.

Informationen zu diesem Vorgang

Vor IBM MQ 9.4.0wird jede Nachricht einzeln aus der Warteschlange entfernt.

Ab IBM MQ 9.4.0können Sie die beiden Systemeigenschaften **com.ibm.mq.AMQP.BATCHSZ** und **com.ibm.mq.AMQP.BATCHINT** verwenden, um die Verarbeitung von Bestätigungen in Batches zur Verbesserung der Leistung zu optimieren:

com.ibm.mq.AMQP.BATCHSZ
: Dieses Attribut definiert die maximale Anzahl der Empfangsbestätigungen, bevor der AMQP-Service Nachrichten entfernt. Der Wert kann im Bereich 1 bis 9999 liegen. Wird eine ungültige Zahl festgelegt oder liegt die angegebene Zahl außerhalb des gültigen Bereichs, wird der Standardwert 50 verwendet.

  Die Stapelgröße wirkt sich nicht auf die Art und Weise aus, wie die Nachrichten übertragen werden. Nachrichten werden immer einzeln übertragen, aber in einem Stapel entfernt, nachdem der AMQP-Service die Bestätigungen empfangen hat. Die tatsächliche Größe eines Batch kann kleiner sein als der durch **com.ibm.mq.AMQP.BATCHINT**angegebene Wert. Beispiel: Ein Stapel wird abgeschlossen, wenn der durch das Attribut **com.ibm.mq.AMQP.BATCHINT** festgelegte Zeitraum abläuft.

com.ibm.mq.AMQP.BATCHINT
: Dieses Attribut definiert, wie lange (in Millisekunden) der AMQP-Service bestätigte Nachrichten in der Warteschlange aufbewahrt. Wenn der Stapel nicht voll ist, wird er nach dieser Dauer gelöscht. Sie können eine beliebige Anzahl von Millisekunden zwischen 1 und 999 999 999 angeben. Der Standardwert ist "50". Wenn Sie für dieses Attribut keinen Wert angeben, wird der Standardwert 50 verwendet.

Anmerkungen:

1. Ob der AMQP-Service auf eine Bestätigung wartet, bevor er eine Nachricht löscht, hängt davon ab, welche der beiden folgenden Servicequalitäten eine Anwendung für die Nachrichtenübermittlung verwendet:
   - QOS für QOS_AT_MOST_ONCE (0)

     Wenn eine AMQP-Anwendung diese Servicequalität verwendet, bestätigt sie keine Nachrichten, sodass der AMQP-Service Nachrichten löscht, nachdem er sie an die Anwendung gesendet hat, ohne auf eine Bestätigung zu warten.
   - QOS_AT_LEAST_ONCE (1)

Wenn eine AMQP-Anwendung diese Servicequalität verwendet, bestätigt sie Nachrichten, sodass der AMQP-Service eine Kopie jeder Nachricht aufbewahrt, nachdem er sie an die Anwendung sendet, bis er eine Bestätigung von der Anwendung erhält. Wenn die Anwendung die Verbindung trennt oder verliert, bevor sie die Nachricht bestätigt, steht die Nachricht anderen Anwendungen zur Verfügung. Der AMQP-Service entfernt eine Nachricht erst aus der Warteschlange, wenn sie bestätigt wurde.

2. MQ Appliance Die Systemeigenschaften **com.ibm.mq.AMQP.BATCHSZ** und **com.ibm.mq.AMQP.BATCHINT** gelten nicht unter IBM MQ Appliance. Unter IBM MQ Appliancewird der Standardwert 50 verwendet.

.

Vorgehensweise

Mit den Systemeigenschaften **com.ibm.mq.AMQP.BATCHSZ** und **com.ibm.mq.AMQP.BATCHINT** können Sie die Verarbeitung von Bestätigungen in Stapeln optimieren.

Ab IBM MQ 9.3.3enthält die Datei `amqp_java.properties` bei der Erstellung des Warteschlangenmanagers die folgenden Standardwerte für die Systemeigenschaften:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

Je nach konsumierter Nachrichtenrate können Sie die Verarbeitung von Bestätigungen in Stapeln optimieren, um die Leistung zu verbessern. Ein migrierter Warteschlangenmanager hat diese Eigenschaften nicht in der Datei `amqp_java.properties` . Für einen migrierten Warteschlangenmanager oder wenn die Eigenschaften nicht festgelegt sind, werden die Standardwerte verwendet. Sie können diese Eigenschaften hinzufügen, um die Werte für eine optimierte Leistung zu optimieren.

Bestätigte Nachrichten werden in Stapeln entfernt, wenn eine der folgenden Bedingungen zutrifft:

- Die Anzahl der bestätigten Nachrichten erreicht **com.ibm.mq.AMQP.BATCHSZ**.
- **com.ibm.mq.AMQP.BATCHINT** wird nach dem Start des Batch überschritten.
- Die Anwendung trennt oder schließt die Warteschlange oder das Thema, bevor die beiden vorherigen Bedingungen erfüllt sind.

ALW Topologien für AMQP-Clients mit IBM MQ

Beispieltopologien, die Sie bei der Entwicklung Ihrer AMQP-Clients für die Arbeit mit IBM MQ unterstützen.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW AMQP-Clients, die über IBM MQ kommunizieren

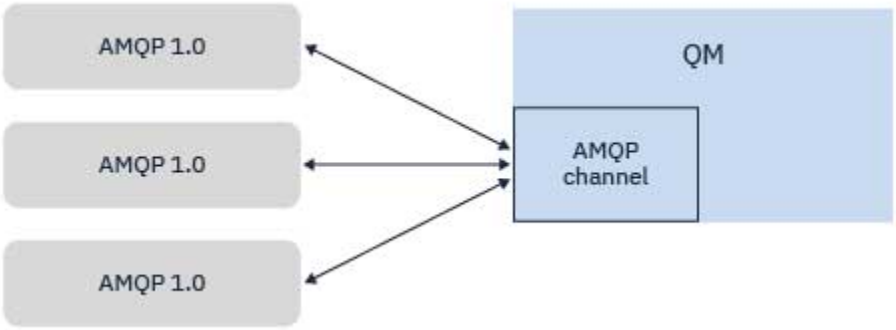
Sie können IBM MQ als Messaging-Provider für jede Anwendung verwenden, die mit AMQP 1.0 kompatibel ist. Obwohl jeder AMQP 1.0-Client eine Verbindung zu einem AMQP-Kanal herstellen kann, werden einige AMQP-Funktionen nicht unterstützt, z. B. Transaktionen oder mehrere Sitzungen.

Durch Definition eines oder mehrerer AMQP-Kanäle können AMQP 1.0-Clients eine Verbindung zum Warteschlangenmanager herstellen und Nachrichten an eine Themenzeichenfolge senden. Clients können auch ein Themenmuster subskribieren, um Nachrichten zu empfangen, die mit dem Muster übereinstimmen.

Im folgenden Szenario sind AMQP 1.0-Anwendungen die einzigen Anwendungen, die Nachrichten senden und empfangen.

Anwendungen können wählen, ob die durch das Subskribieren einer Themenzeichenfolge erstellten Ziele persistent sind, sodass Nachrichten nicht verloren gehen, wenn die Anwendung vorübergehend ihre Verbindung zum Warteschlangenmanager verliert.

Anwendungen können auch festlegen, wie lange Nachrichten aufbewahrt werden, bevor sie aus dem Ziel gelöscht werden.



Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW AMQP-Clients, die mit IBM MQ-Anwendungen Nachrichten austauschen

Durch das Definieren und Starten eines AMQP-Kanals können AMQP 1.0-Anwendungen Nachrichten veröffentlichen, die von vorhandenen MQ-Anwendungen empfangen werden. Nachrichten, die über einen AMQP-Kanal veröffentlicht werden, werden alle an MQ-Themen gesendet, nicht an MQ-Warteschlangen. Eine MQ-Anwendung, die eine Subskription mit dem API-Aufruf MQSUB erstellt hat, empfängt Nachrichten, die von AMQP 1.0-Anwendungen veröffentlicht werden, vorausgesetzt, dass die von der MQ-Anwendung verwendete Themenzeichenfolge oder das Themenobjekt mit der vom AMQP-Client veröffentlichten Themenzeichenfolge übereinstimmt.

AMQP-Nachrichtendaten, Attribute und Eigenschaften werden in der MQ-Nachricht, die von der MQ-Anwendung empfangen wird, festgelegt. Weitere Informationen zu AMQP- MQ -Nachrichtenzuordnungen finden Sie unter „Zuordnung von AMQP-Feldern zu IBM MQ-Feldern (eingehende Nachrichten)” auf Seite 721.

Wenn die MQ-Anwendung eine dauerhafte Subskription erstellt hat, werden Nachrichten, die von der AMQP-Anwendung veröffentlicht werden, in der Warteschlange gespeichert, die die Subskription sichert. Die Nachrichten werden dann von der MQ-Anwendung empfangen, wenn die Anwendung ihre Subskription wieder aufnimmt. Wenn die AMQP-Anwendung eine Nachrichtenlebensdauer angibt und die MQ-Anwendung die Verbindung nicht innerhalb dieses Zeitraums wiederherstellt, wird die Nachricht aus der Warteschlange entfernt.

AMQP 1.0-Anwendungen können auch Nachrichten konsumieren, die von vorhandenen MQ-Anwendungen veröffentlicht werden. Nachrichten, die von MQ-Anwendungen zu einem MQ-Thema oder einer Themenzeichenfolge veröffentlicht werden, werden von einer AMQP 1.0-Anwendung empfangen, sofern die Anwendung ein Themenmuster subskribiert hat, das mit der veröffentlichten Themenzeichenfolge übereinstimmt.

Wenn die AMQP 1.0-Anwendung einen Lebensdauerwert für die Subskription angibt und die AMQP-Anwendung die Verbindung für mehr als diesen Zeitraum trennt, wird die Subskription aus dem Warteschlangenmanager entfernt und alle in der Subskriptionswarteschlange gespeicherten Nachrichten gehen verloren.

MQMD-Felder, Nachrichteneigenschaften und Anwendungsdaten werden in der AMQP-Nachricht festgelegt, die die AMQP-Anwendung empfängt. Weitere Informationen zu den Zuordnungen von MQ zu AMQP-Nachrichten finden Sie unter „Zuordnung von IBM MQ-Feldern zu AMQP-Feldern (abgehende Nachrichten)” auf Seite 719.

Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

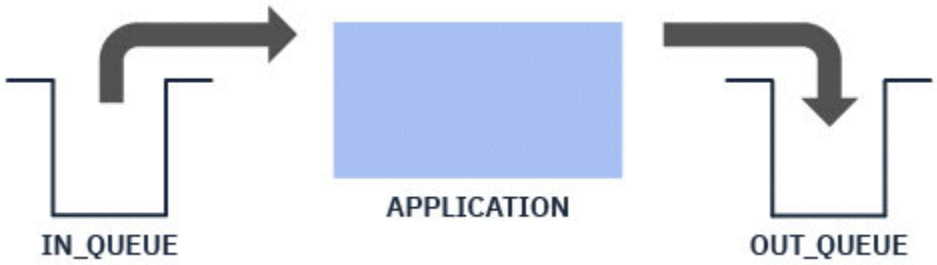
ALW AMQP-Clients für die direkte Interaktion mit Anwendungen in IBM MQ-Warteschlangen konfigurieren

Die AMQP-Implementierung von IBM MQ unterstützt Publish/Subscribe und Punkt zu Punkt. Verwenden Sie für jeden AMQP-Client, der Punkt-zu-Punkt nicht unterstützt, die folgenden Schritte, um Nachrichten an eine Warteschlange zu senden oder Nachrichten aus einer Warteschlange zu empfangen.

Übersicht

Angenommen, eine Anwendung ruft Nachrichten aus einer Eingabewarteschlange ab `IN_QUEUE` und reiht diese Nachrichten in eine Ausgabewarteschlange ein `OUT_QUEUE`. AMQP-Clients können Nachrichten in `IN_QUEUE` einreihen und aus `OUT_QUEUE` abrufen.

Anmerkung: Es sind keine Änderungen an der Anwendung selbst notwendig.



Damit ein AMQP-Publisher eine Nachricht in eine Warteschlange einreihen kann, müssen Sie eine administrative Subskription für die Topic-Zeichenfolge erstellen, in der der AMQP-Client veröffentlicht, mit der beabsichtigten Warteschlange als Ziel. Weitere Informationen hierzu finden Sie unter „Nachrichten an die Anwendung senden:" auf Seite 732.

Damit ein AMQP-Subskribent Nachrichten aus einer Warteschlange abrufen kann, müssen Sie die Warteschlange durch eine gleichnamige Aliaswarteschlange ersetzen, wobei als Ziel ein Topic-Objekt fungiert, das die Topic-Zeichenfolge darstellt, für die der AMQP-Client subskribiert ist. Weitere Informationen hierzu finden Sie unter „Nachrichten aus der Anwendung abrufen:" auf Seite 733.

Nachrichten an die Anwendung senden:

Die Anwendung nimmt bereits Nachrichten von `IN_QUEUE` ab und Sie möchten, dass ein AMQP-Client Nachrichten veröffentlichen kann, sodass sie zur Verarbeitung durch die Anwendung in diese Warteschlange eingereiht werden.

Zu diesem Zweck erstellen Sie eine neue administrative Subskription, wobei die Topic-Zeichenfolge, aus der diese Subskription Nachrichten erhält, die Topic-Zeichenfolge ist, in der der AMQP-Client veröffentlicht. Die Zielwarteschlange dieser Subskription ist die Eingabewarteschlange für die Anwendung `IN_QUEUE`.

Alle Nachrichten, die in der definierten Topiczeichenfolge für diese Verwaltungssubskription veröffentlicht werden, werden an das definierte Ziel weitergeleitet, in diesem Fall `IN_QUEUE`.

Angenommen, der AMQP-Client veröffentlicht Nachrichten in einer Themenzeichenfolge `/application/in`, können Sie mit dem folgenden MQSC-Befehl eine Verwaltungssubskription `APP_IN` erstellen:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

Wenn Sie dieses Objekt definiert haben, werden alle Nachrichten, die in `/application/in` veröffentlicht werden, an das Ziel `IN_QUEUE` weitergeleitet, wo sie von der Anwendung auf dieselbe Weise aufgenommen werden wie alle anderen Nachrichten, die von anderen Anwendungen in diese Warteschlange eingereiht werden.

Nachrichten aus der Anwendung abrufen:

Die Anwendung reiht Nachrichten in OUT_QUEUE ein, wo sie von anderen Clients aufgenommen und verarbeitet werden können.

In diesem Fall sollen die Nachrichten jedoch stattdessen an einen AMQP-Client übergeben werden. AMQP-Clients verwenden aber nur Publish/Subscribe und können Nachrichten nicht direkt aus einer Warteschlange abholen.

Wenn Sie die Clients, die zuvor Nachrichten erhalten haben, durch den subskribierenden AMQP-Client ersetzen möchten, müssen Sie ein Topic-Objekt für die Topic-Zeichenfolge, für die der AMQP-Client subskribiert ist, sowie eine Aliaswarteschlange erstellen.

Achtung: Wenn Sie die Aliaswarteschlange definieren und dann die produzierende Anwendung starten, bevor die AMQP-Clients die Möglichkeit zur Subskription hatten, gehen Nachrichten, die die produzierende Anwendung an die "Warteschlange" (jetzt ein Topic) sendet, verloren, da keine Subskribenten vorhanden sind.

Bei den in diesem Text erläuterten Änderungen werden lediglich die Clients, die zuvor Nachrichten erhalten haben, durch den subskribierenden AMQP-Client ersetzt. Um eine Kombination von AMQP- und anderen Clients für den Abruf von Nachrichten zu verwenden, sind umfangreichere Änderungen notwendig.

Angenommen, der AMQP-Client subskribiert eine Themenzeichenfolge `/application/out`, können Sie das Themenobjekt APP_OUT mit dem folgenden MQSC-Befehl definieren:

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

Alle Nachrichten, die an dieses Topic-Objekt übermittelt werden, werden an den AMQP-Client zugestellt, der dieselbe Topic-Zeichenfolge subskribiert.

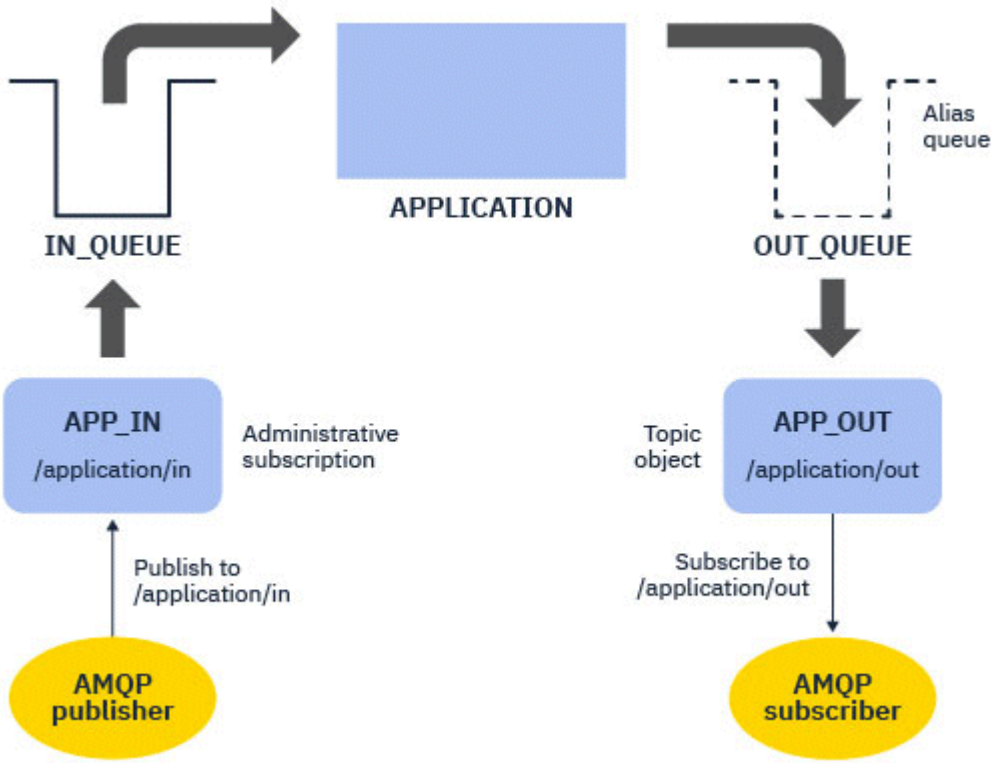
Anschließend müssen Sie sicherstellen, dass Nachrichten, die von der Anwendung in OUT_QUEUE eingereiht werden, diesem neuen Themenobjekt zugestellt werden, damit sie an den subskribierenden Client gesendet werden.

Ersetzen Sie dazu die vorhandene Warteschlange OUT_QUEUE mit dem folgenden MQSC-Befehl durch eine gleichnamige Aliaswarteschlange mit einem Zieltyp des soeben erstellten Themenobjekts:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

Jetzt warten Nachrichten, die von der Anwendung in OUT_QUEUE eingereiht werden, nicht darauf, dass die Warteschlange aufgenommen wird. Stattdessen werden sie an das Ziel dieser Aliaswarteschlange zugestellt, d. h. an das neue Themenobjekt APP_OUT.

Der AMQP-Client, der die Themenzeichenfolge subskribiert hat, die durch dieses Themenobjekt `/application/out`dargestellt wird, empfängt dann alle Nachrichten, die an dieses Themenobjekt gesendet werden, aus der Aliaswarteschlange.



Zugehörige Tasks

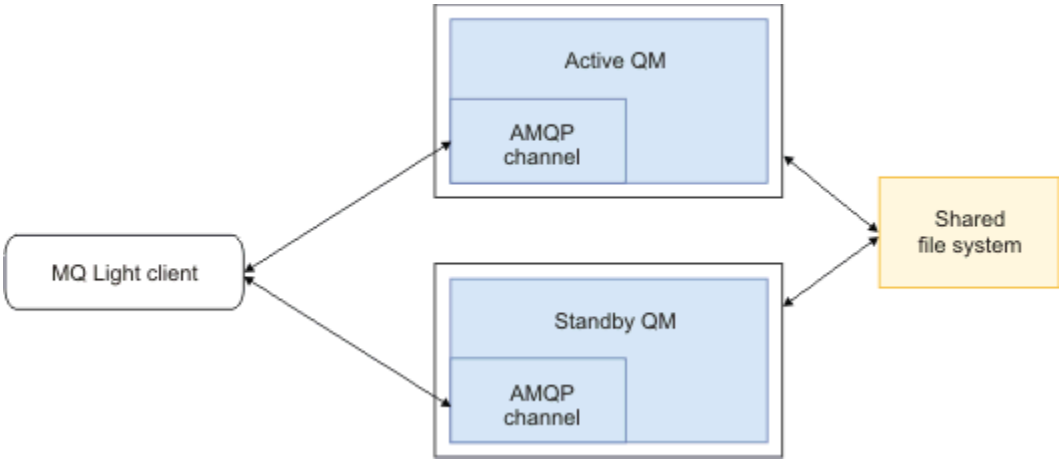
AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW AMQP-Client für hohe Verfügbarkeit konfigurieren

Sie können AMQP 1.0-Anwendungen so konfigurieren, dass sie eine Verbindung zur aktiven Instanz eines IBM MQ-Multi-Instanz-Warteschlangenmanagers und eine Übernahme durch die Standby-Instanz des Multi-Instanz-Warteschlangenmanagers in einem Hochverfügbarkeitspaar herstellen. Zu diesem Zweck konfigurieren Sie die AMQP-Anwendung mit zwei IP-Adressen und Portpaaren.

Sie können die AMQP-Client-API mit einer angepassten Funktion konfigurieren, die aufgerufen wird, wenn der Client seine Verbindung zum Server verliert. Die Funktion kann eine Verbindung zu einer alternativen IP-Adresse herstellen, z. B. zu einem Standby-IBM MQ-Warteschlangenmanager oder zur ursprünglichen IP-Adresse. Für andere AMQP-Clients gilt Folgendes: Wenn der Client die Konfiguration mehrerer Verbindungsendpunkte unterstützt, konfigurieren Sie die Anwendung mit zwei Host-Port-Paaren und verwenden Sie die von der AMQP-Bibliothek bereitgestellten Funktionen für die Wiederherstellung der Verbindung, um auf den Standby-Warteschlangenmanager zu wechseln.



Zugehörige Tasks

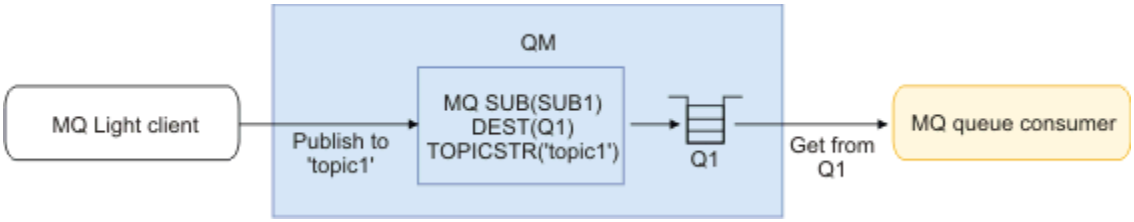
AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW Publish/Subscribe für AMQP-Clients konfigurieren

AMQP-Clients können mit einer IBM MQ-Subskription Veröffentlichungen in einem Thema vornehmen, die Nachrichten an eine IBM MQ-Warteschlange weiterleitet, die von einer vorhandenen Anwendung gelesen wird. Wenn eine AMQP 1.0-Anwendung Nachrichten an eine vorhandene IBM MQ-Anwendung senden soll, die für das Lesen aus einer Warteschlange konfiguriert ist, müssen Sie eine verwaltete IBM MQ-Subskription auf dem Warteschlangenmanager definieren.

Konfigurieren Sie die Subskription so, dass sie ein Themenmuster verwendet, das mit der von der AMQP-Anwendung verwendeten Themenzeichenfolge übereinstimmt. Legen Sie das Subskriptionsziel auf den Namen der Warteschlange fest, von der die IBM MQ-Anwendung Nachrichten abruft oder durchsucht.



Zugehörige Tasks

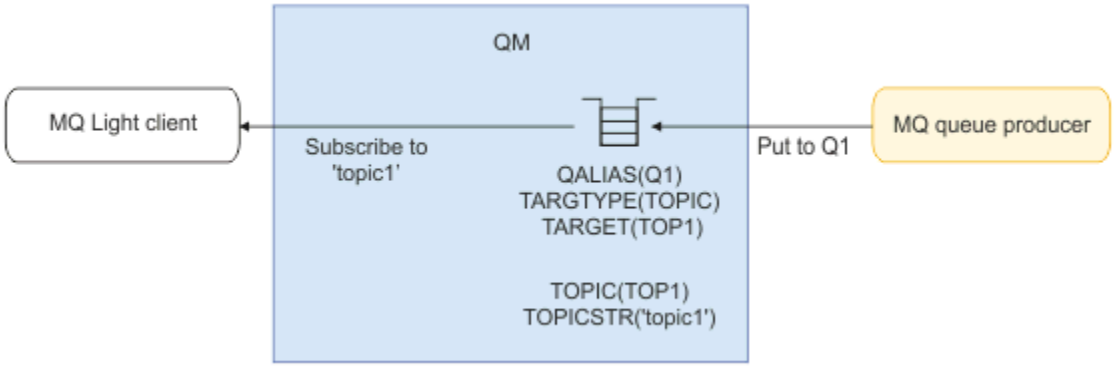
AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW AMQP-Client, der einen Warteschlangenaliasnamen verwendet, um Nachrichten von einer IBM MQ-Anwendung zu empfangen

Ein AMQP-Client kann ein Thema subskribieren und Nachrichten empfangen, die von einer IBM MQ-Anwendung in eine Aliaswarteschlange eingereiht werden. Wenn eine AMQP 1.0-Anwendung Nachrichten von einer vorhandenen IBM MQ-Anwendung empfangen soll, die für das Einreihen von Nachrichten in eine Warteschlange konfiguriert ist, müssen Sie einen Warteschlangenalias (QALIAS) auf dem Warteschlangenmanager definieren.

Der Warteschlangenaliasname muss denselben Namen wie die Warteschlange haben, die die IBM MQ-Anwendung für das Einreihen öffnet. Der Warteschlangenaliasname muss einen TOPIC-Basistyp und ein Basisobjekt eines IBM MQ-Themenobjekts angeben, das eine Themenzeichenfolge aufweist, die mit dem Themenmuster übereinstimmt, das von der AMQP-Anwendung subskribiert wird.



Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW AMQP-Client, der Anforderungen an einen Anwendungsserver übergibt und Antworten von einem Anwendungsserver verarbeitet

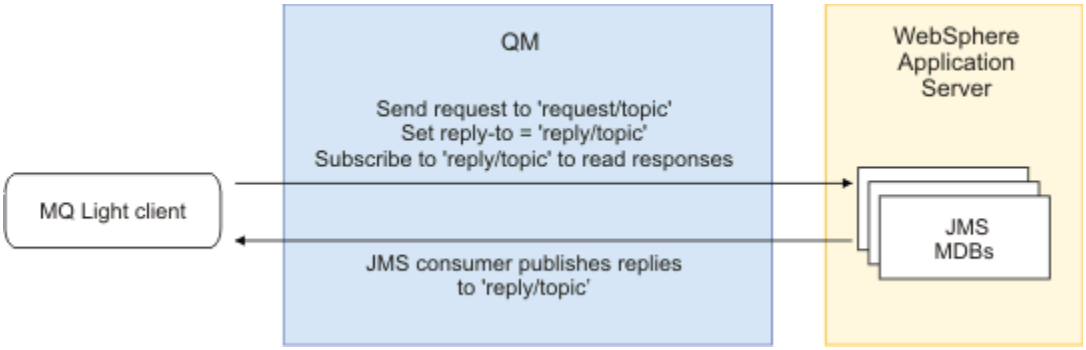
Ein AMQP-Client kann Anforderungen an eine Message-driven Bean übergeben, die in einem Anwendungsserver ausgeführt wird, und Antworten aus einem Antwortthema konsumieren. IBM MQ unterstützt AMQP 1.0-Anwendungen, indem in den Nachrichten, die IBM MQ veröffentlicht, ein Antwortthema festgelegt wird. Wenn eine AMQP-Nachricht mit festgelegtem Antwortattribut veröffentlicht wird, wird der Wert des Antwortfelds als JMS-Eigenschaft für JMS-Konsumenten für den Empfang festgelegt. Diese Einstellung ermöglicht es JMS-Konsumenten, das Antwortthema aus der Nachricht zu lesen und eine Antwortnachricht zurück an den AMQP-Client zu senden.

Die JMS-Eigenschaft ist **JMSReplyTo**. Bei der AMQP-Antwortzeichenfolge muss es sich um einen der folgenden Typen handeln:

- Eine Themenzeichenfolge. Beispiel: `'reply/topic'`
- Eine AMQP-Adress-URL im Format `amqp://host:port/[topic-string]` Zum Beispiel,`amqp://localhost:5672/reply/topic`

Wenn Sie eine AMQP-Adress-URL als Antwortfeld angeben, wird alles außer der Themenzeichenfolge am Ende der URL entfernt, bevor die Eigenschaft **JMSReplyTo** festgelegt wird.

Weitere Informationen zu den Zuordnungen einer AMQP-Antwortadresse zu einer **JMSReplyTo** -Eigenschaft finden Sie unter „Zuordnung von AMQP-Feldern zu IBM MQ-Feldern (eingehende Nachrichten)” auf Seite 721



Zugehörige Tasks

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ALW Interoperabilität zwischen MQ Light und Apache Qpid JMS-Anwendungen

MQ Light- und Apache Qpid JMS-Anwendungen funktionieren auf ähnliche Weise und erstellen beim Subskribieren eines Themas IBM MQ-Subskriptionen, die derselben Namenskonvention entsprechen.

Private, nicht gemeinsam genutzte Subskription

Der Name der von der Anwendung erstellten IBM MQ-Subskription lautet `:private:<clientid>:<topicstring>`.

Eine Anwendung, die eine andere Client-ID verwendet, kann auf die Subskriptionen, die von anderen Anwendungen erstellt wurden, nicht zugreifen, da der Subskriptionsname automatisch generiert wird und die AMQP-Client-ID einschließt.

Sowohl Apache Qpid JMS- als auch MQ Light-Anwendungen verwenden diese Namenskonvention für private Subskriptionen.

Global gemeinsam genutzte Subskriptionen

Der Name einer von einem AMQP-Client erstellten global gemeinsam genutzten IBM MQ-Subskription lautet `:share:<sharename>:<topicstring>`.

Wenn mehrere Anwendungen mit verschiedenen AMQP-Client-IDs denselben Share-Namen und dieselbe Topic-Zeichenfolge angeben, nutzen sie eine einzelne Subskription gemeinsam und können zusammenarbeiten, um die Nachrichten für diese Subskription zu verarbeiten. Sie können dieses Muster verwenden, wenn Sie die Anzahl der Worker-Anwendungen, die Nachrichten aus einer Subskription abholen, skalieren möchten.

Sowohl Apache QPid JMS- als auch MQ Light-Anwendungen verwenden diese Namenskonvention für global gemeinsam genutzte Subskriptionen. Im Fall von Apache Qpid JMS erfordert dies, dass für die JMS-Verbindung keine Client-ID angegeben ist.

Die Apache Qpid JMS-Bibliothek generiert automatisch eine AMQP-Client-ID, die aber nicht zu Zwecken der IBM MQ-Subskriptionsbenennung verwendet wird.

Anmerkung: Global gemeinsam genutzte Subskriptionen gelten weiterhin für einen einzelnen Warteschlangenmanager.

Private, gemeinsam genutzte Subskriptionen

Der Name einer von einem AMQP-Client erstellten privat gemeinsam genutzten IBM MQ-Subskription lautet `:privateshare:<clientid>:<sharename>:<topicstring>`.

Wenn mehrere Threads einer einzelnen Apache Qpid JMS-Anwendung denselben Share-Namen und dieselbe Topic-Zeichenfolge verwenden und eine Client-ID für die JMS-Verbindung konfiguriert wurde, nutzen diese Threads dasselbe IBM MQ-Subskriptionsobjekt gemeinsam.

Andere Apache Qpid JMS-Verbindungen können die Subskription jedoch nicht gemeinsam nutzen, weil sie eine andere Client-ID verwenden müssen.

MQ Light-Clients unterstützen das Konzept von privaten, gemeinsam genutzten Subskriptionen nicht und können keine Nachrichten aus einer privaten, gemeinsam genutzten Subskription konsumieren, die von einer Apache Qpid JMS-Anwendung erstellt wurde.

IBM MQ JMS-Subskriptionen

Für IBM MQ JMS-Subskriptionen wird ein anderes Benennungsschema als für AMQP-Kanäle verwendet. Es ist nicht möglich, dass MQ Light- oder Apache Qpid JMS-Anwendungen Subskriptionen gemeinsam mit IBM MQ JMS-Anwendungen nutzen.

Zugehörige Konzepte

AMQP-Clientanwendungen entwickeln

Die IBM MQ-Unterstützung für AMQP-APIs ermöglicht es einem IBM MQ-Administrator, einen AMQP-Kanal zu erstellen. Wenn dieser Kanal gestartet wird, definiert dieser Kanal eine Portnummer, die Verbindungen von AMQP-Clientanwendungen akzeptiert.

ALW Steuereigenschaften des AMQP-Listeners von IBM MQ

Für eine bessere Leistung in einer Multithread-Anwendung können Sie die Anzahl der Worker-Threads optimieren, die der AMQP-Service verwenden soll, indem Sie eine Eigenschaft in der AMQP-Eigenschaftendatei konfigurieren.

Sie können die Eigenschaften des AMQP-Listener-Service in den folgenden Eigenschaftendateien konfigurieren:

- Windows Auf Windows -Systemen: `amqp_win.properties` .
- Linux AIX Auf AIX and Linux -Systemen: `amqp_unix.properties` .

Sie können die folgenden Eigenschaften konfigurieren:

Tabelle 105. Eigenschaften des AMQP-Listener-Service

Eigenschaft	Beschreibung
com.ibm.mq.MQXR.Workers	Die Anzahl der Server-Worker-Threads, die der AMQP-Listener-Service erstellt. Wenn dieser Wert nicht angegeben wird, entspricht er standardmäßig der Anzahl der logischen Prozessoren im System.
MQIBindType	Der Bindungstyp für den AMQP-Service: FASTPATH, SHAREDoder ISOLIERT. Der Standardwert ist FASTPATH.

Der AMQP-Listener-Service verteilt die Clientverbindungsworkload auf eine Reihe von Worker-Threads. Die Anzahl der Worker-Threads, die der AMQP-Service verwenden sollte, kann mithilfe der Eigenschaft **com.ibm.mq.MQXR.Workers** angegeben werden.

Der IBM MQ -Warteschlangenmanageradministrator kann die Anzahl der Worker-Threads optimieren, um die Leistung in einer Multithread-Anwendung zu verbessern. Normalerweise wird die beste Leistung erzielt, wenn die Anzahl der Worker-Threads mit der Anzahl der logischen Prozessoren auf dem System übereinstimmt. Dies ist jedoch möglicherweise nicht immer der Fall für bestimmte Maschinenkonfigurationen und Clientlastmerkmale, sodass möglicherweise ein Optimierungselement erforderlich ist, um den optimalen Wert für die Anzahl der Worker-Threads zu finden.

Stellen Sie vor der Optimierung sicher, dass Sie die Art Ihrer Clientanwendungen und deren Workloads gründlich verstehen. Die Messung der Leistung Ihrer Anwendung mit unterschiedlichen Threadzahlen und Benchmarking-Werten sollte helfen, den optimalen Wert für die Anzahl der Worker-Threads zu ermitteln.

Anmerkung: MQ Appliance Diese Eigenschaften gelten nicht unter IBM MQ Appliance. Unter IBM MQ Appliancewerden Standardwerte verwendet.

REST-Anwendungen mit IBM MQ entwickeln

Sie können REST-Anwendungen entwickeln, um Nachrichten zu senden und zu empfangen. IBM MQ unterstützt abhängig von Plattform und Funktionalität verschiedene REST-APIs.

Für das Senden von Nachrichten an und Empfangen von Nachrichten von IBM MQ können Sie aus den folgenden, von IBM MQ unterstützten Optionen auswählen:

- IBM MQ messaging REST API
- IBM z/OS Connect EE

- IBM Integration Bus
- DataPower

IBM MQ messaging REST API

Sie können mithilfe der messaging REST API IBM MQ-Nachrichten im einfachen Textformat senden, empfangen und durchsuchen. Die messaging REST API ist standardmäßig aktiviert.

Es werden verschiedene HTTP-Header unterstützt, mit deren Hilfe allgemeine Nachrichteneigenschaften festgelegt werden können.

Die messaging REST API ist vollständig in die IBM MQ-Sicherheit integriert. Um die messaging REST API verwenden zu können, müssen Benutzer beim mqweb-Server authentifiziert und Mitglied der Rolle MQWebUser sein.

Weitere Informationen finden Sie unter „Messaging mit der REST API” auf Seite 740. Siehe auch Tutorial: Get started with the IBM MQ messaging REST API auf IBM Developer. Dort werden Go- und Node.js-Beispiele für die Verwendung der Messaging-REST-API gezeigt.

IBM z/OS Connect EE

IBM z/OS Connect EE ist ein z/OS-Produkt, mit dem Sie REST-APIs für vorhandene z/OS-Assets, z. B. CICS- oder IMS-Transaktionen, und für IBM MQ-Warteschlangen und -Themen erstellen können. Das vorhandene z/OS-Asset bleibt dem Benutzer verborgen. Auf diese Weise können Sie Assets REST-fähig machen, ohne sie selbst oder die vorhandenen Anwendungen, von denen sie verwendet werden, ändern zu müssen.

IBM z/OS Connect EE stellt eine automatische Datenkonvertierung zwischen den von den REST-APIs verwendeten JSON-Daten und den eher traditionellen Sprachstrukturen, z. B. COBOL, bereit, die von vielen Mainframeanwendungen erwartet werden.

Mit dem Eclipse-basierten IBM z/OS Connect EE-API-Toolkit können Sie unter Verwendung von Abfrageparametern und URL-Pfadsegmenten eine komplexe REST-konforme API erstellen, um das JSON-Format zu bearbeiten, während es die IBM z/OS Connect EE-Runtime durchläuft.

Mithilfe von IBM z/OS Connect EE können IBM MQ-Warteschlangen und -Themen als REST-konforme APIs über den IBM MQ-Service-Provider zugänglich gemacht werden. Es werden zwei verschiedene Servicetypen unterstützt:

- Unidirektionale Services: Diese stellen eine REST-API bereit, die es ermöglicht, eine einzelne IBM MQ-Operation für eine Warteschlange oder ein Thema auszuführen. Abhängig von der genauen Konfiguration kann eine HTTP-Anforderung dazu führen, dass eine Nachricht an eine Warteschlange gesendet oder zu einem Thema veröffentlicht wird, oder sie kann dazu führen, dass eine Nachricht aus einer Warteschlange abgerufen und gelöscht wird.
- Bidirektionale Services: Diese stellen eine REST-API für ein Warteschlangenpaar bereit, die von einer Back-End-Anwendung mit Anforderung/Antwort-Funktion verwendet wird. Aufrufende geben eine HTTP-Anforderung an den bidirektionalen Service aus. Die HTTP-Anforderungsnutzdaten werden von JSON in eine traditionelle Sprachstruktur transformiert und in eine Anforderungswarteschlange eingereiht. Dort werden sie von der Back-End-Anwendung verarbeitet und es wird eine Antwort in die Antwortwarteschlange eingereiht. Diese Antwort wird vom Service abgerufen, von der traditionellen Sprachstruktur in JSON konvertiert und als POST-Antworthauptteil an den Aufrufenden zurückgesendet.

Weitere Informationen zu IBM z/OS Connect EE finden Sie unter z/OS Connect EE.

Weitere Informationen zum IBM MQ-Service-Provider finden Sie im Abschnitt IBM MQ-Service-Provider verwenden.

IBM Integration Bus

IBM Integration Bus ist die führende Integrationstechnologie von IBM, mit der Anwendungen und Systeme unabhängig von den Nachrichtenformaten und Protokollen, die sie unterstützen, miteinander verbunden werden können.

IBM Integration Bus hat IBM MQ immer unterstützt und stellt die Knoten *HTTPInput* und *HTTPRequest* bereit, mit deren Hilfe eine REST-konforme Schnittstelle für IBM MQ und viele andere Systemen, z. B. Datenbanken, erstellt werden kann.

Mit IBM Integration Bus ist jedoch noch viel mehr möglich, als eine einfache REST-Schnittstelle für IBM MQ bereitzustellen. Es bietet Funktionen, mit denen eine erweiterte Nutzdatenbearbeitung, eine Nutzdatenanreicherung und viele andere Erweiterungen als Teil einer REST API bereitgestellt werden können.

Weitere Informationen finden Sie unter diesem Technologiebeispiel, das eine JSON-über-REST-Schnittstelle für eine IBM MQ-Anwendung, die XML-Nutzdaten erwartet, bereitstellt.

DataPower

Das DataPower-Gateway ist ein einzelnes Mehrkanal-Gateway, das die Bereitstellung von Sicherheit, Steuerung, Integration und optimiertem Zugriff auf eine Reihe von Systemen, einschließlich IBM MQ, unterstützt. Es ist sowohl in physischen als auch in virtuellen Formfaktoren verfügbar.

Einer der von DataPower bereitgestellten Services ist ein Multiprotokoll-Gateway, das in einem Protokoll eine Eingabe annehmen und in einem anderen Protokoll eine Ausgabe generieren kann. Insbesondere kann DataPower so konfiguriert werden, dass es HTTP(S)-Daten akzeptiert und über eine Clientverbindung an IBM MQ weiterleitet. Auf diese Weise kann eine REST-Schnittstelle für IBM MQ erstellt werden. Andere DataPower-Services, z. B. die Umsetzung, können ebenfalls zur Erweiterung der REST-Schnittstelle verwendet werden.

Weitere Informationen finden Sie unter Multi-Protocol Gateway.

Messaging mit der REST API

Sie können messaging REST API verwenden, um einfaches Punkt-zu-Punkt-und Publish-Messaging durchzuführen. Sie können Nachrichten zu einem Thema veröffentlichen, Nachrichten an eine Warteschlange senden, Nachrichten in einer Warteschlange durchsuchen und Nachrichten aus der Warteschlange abrufen. Informationen werden in einem einfachen Textformat an die messaging REST API gesendet und von dort empfangen.

Vorbereitende Schritte

Anmerkung:

- Die messaging REST API ist standardmäßig aktiviert. Sie können die messaging REST API inaktivieren, um die Messaging-Funktion auszuschalten. Weitere Informationen zum Aktivieren oder Inaktivieren der messaging REST API finden Sie im Abschnitt messaging REST API konfigurieren.
- Die messaging REST API ist in die IBM MQ-Sicherheit integriert. Um die messaging REST API verwenden zu können, müssen Benutzer beim mqweb-Server authentifiziert und Mitglied der Rolle MQWebUser sein. Der Benutzer muss auch für den Zugriff auf die angegebene Warteschlange oder das angegebene Thema berechtigt sein. Weitere Informationen zur Sicherheit für die REST API finden Sie unter Sicherheit von IBM MQ Console und REST API.
- Wenn Sie Advanced Message Security (AMS) mit der messaging REST API verwenden, ist zu beachten, dass alle Nachrichten mit dem Kontext des MQWeb-Servers verschlüsselt werden und nicht mit dem Kontext des Benutzers, der die Nachricht postet.
- Beim Empfangen oder Durchsuchen einer Nachricht werden nur IBM MQ `MQSTR` oder `JMS TextMessage` formatierte Nachrichten unterstützt. Nachfolgend werden alle Nachrichten unter dem Synchronisationspunkt als 'Abruf mit Löschen' empfangen, und alle nicht verarbeiteten Nachrichten bleiben in der Warteschlange. Die IBM MQ-Warteschlange kann so konfiguriert werden, dass diese nicht verarbeit-

baren Nachrichten an ein alternatives Ziel verschoben werden. Weitere Informationen finden Sie unter „Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln” auf Seite 244.

- Die messaging REST API ermöglicht keine einmalige und zuverlässige Nachrichtenübermittlung mit Transaktionsunterstützung. Wenn eine HTTP POST ausgegeben wird und die Verbindung fehlschlägt, bevor eine HTTP-Antwort vom Client empfangen wird, kann der Client nicht sofort feststellen, ob die Nachricht an die angegebene Warteschlange gesendet oder für das angegebene Topic veröffentlicht wurde. Wenn eine HTTP-Anfrage des Typs DELETE ausgegeben wird und die Verbindung fehlschlägt, bevor eine HTTP-Antwort vom Client empfangen wird, wurde eine Nachricht möglicherweise aus der Warteschlange abgerufen und gelöscht, da keine Möglichkeit besteht, den Abruf mit Löschen rückgängig zu machen.
- Ab IBM MQ 9.3.0werden Zeilenumbrüche in eingehenden Zeichenfolgen nicht mehr durch die HTTP-POST-Operation entfernt. REST-Anwendungen, die frühere Versionen verwenden, sollten keine Zeilenumbrüche in Nachrichten verwenden, die über die REST API gesendet oder veröffentlicht werden, da sie verloren gehen.

Prozedur

- „Erste Schritte mit der messaging REST API” auf Seite 741
- „messaging REST API verwenden” auf Seite 744
- REST API Fehlerbehandlung
- REST API Erkennung
- REST API Unterstützung in der Landessprache

Zugehörige Verweise

Referenz zur Messaging-REST API

Zugehörige Informationen

Lernprogramm: Get started with the IBM MQ messaging REST API

Erste Schritte mit der messaging REST API

Unternehmen Sie erste Schritte mit der messaging REST API und probieren Sie einige Beispielbefehle mithilfe von cURL aus.

Vorbereitende Schritte

Um Ihnen den Einstieg in die Verwendung der messaging REST API zu erleichtern, gelten für die Beispiele in dieser Aufgabe folgende Voraussetzungen:

- Die Beispiele verwenden cURL zum Senden von REST-Anforderungen, um Nachrichten in eine Warteschlange einzureihen und aus einer Warteschlange abzurufen. Deshalb muss cURL auf Ihrem System installiert sein, damit Sie diese Aufgabe ausführen können.
- In den Beispielen wird ein WS-Manager `QM1`verwendet. Erstellen Sie entweder einen Warteschlangenmanager mit diesem Namen oder ersetzen Sie einen vorhandenen Warteschlangenmanager auf Ihrem System. Der Warteschlangenmanager muss sich auf demselben System wie der mqweb-Server befinden.
- Für die Ausführung dieser Task müssen Sie ein Benutzer mit bestimmten Berechtigungen sein, damit Sie den Befehl **dspmqweb** verwenden können:
  - z/OS Unter z/OS benötigen Sie die Berechtigung zur Ausführung des Befehls **dspmqweb** sowie Schreibzugriff für die Datei `mqwebuser.xml`.
  - Multi Auf allen anderen Betriebssystemen müssen Sie ein privilegierter Benutzer sein.

  IBM i Unter IBM isollten die Befehle in QSHELL ausgeführt werden.

Vorgehensweise

1. Stellen Sie sicher, dass der mqweb-Server für die messaging REST API konfiguriert ist:
   - Stellen Sie sicher, dass Sie den mqweb-Server für die Verwendung durch die administrative REST API, die administrative REST API für MFT, die messaging REST APIoder IBM MQ Consolekonfiguriert haben.Weitere Informationen zur Konfiguration des mqweb-Servers mit einer Basisregistry enthält der Artikel Basiskonfiguration für den mqweb-Server.
   - Wenn der mqweb-Server bereits konfiguriert ist, müssen Sie sicherstellen, dass Sie in Schritt 5 unter Basiskonfiguration für den mqweb-Serverdie entsprechenden Benutzer hinzugefügt haben, um das Messaging zu aktivieren.
     - Wenn **mqRestMessagingAdoptWebUserContext** in der mqweb-Serverkonfiguration auf `true` gesetzt ist, müssen Benutzer von messaging REST API ein Mitglied der Rolle `MQWebUser` sein. Die Rollen `MQWebAdmin` und `MQWebAdminRO` gelten nicht für die messaging REST API. Die Benutzer müssen außerdem berechtigt sein, auf Warteschlangen und Topics zuzugreifen, die für das Messaging über OAM oder RACFverwendet werden.
     - Wenn **mqRestMessagingAdoptWebUserContext** in der mqweb-Serverkonfiguration auf `false` gesetzt ist, muss die Benutzer-ID, die zum Starten des mqweb-Servers verwendet wird, berechtigt sein, auf Warteschlangen zuzugreifen, die für die Nachrichtenübertragung über OAM oder RACFverwendet werden.
2. z/OS
   Setzen Sie unter z/OSdie Umgebungsvariable WLP_USER_DIR so, dass Sie den Befehl **dspmqweb** verwenden können. Geben Sie folgenden Befehl ein, um die Variable so zu setzen, dass sie auf Ihre mqweb-Serverkonfiguration zeigt:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   , wobei *WLP_user_directory* der Name des Verzeichnisses ist, das an `crtmqweb`übergeben wird. Beispiel:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   Weitere Informationen finden Sie im Abschnitt mqweb-Server erstellen.
3. Bestimmen Sie dieREST API URL, indem Sie den folgenden Befehl eingeben:

   ```
   dspmqweb status
   ```

   Bei den Beispielen in den folgenden Schritten wird davon ausgegangen, dass Ihre REST API-URL die Standard-URL `https://localhost:9443/ibmmq/rest/v2/` ist. Sollte Ihre URL von der Standard-URL abweichen, ersetzen Sie die URL in den folgenden Schritten durch Ihre URL.
4. Erstellen Sie eine Warteschlange `MSGQ` auf dem Warteschlangenmanager `QM1`. Diese Warteschlange wird für das Messaging verwendet. Verwenden Sie eine der folgenden Methoden:
   - Verwenden Sie eine POST-Anforderung für die Ressource `mqsc` der administrative REST API, die sich als `mqadmin`-Benutzer authentifiziert:

     ```
     curl -k https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QM1/mqsc -X POST -u mqad
     min:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json"
     --data "{\"type\": \"runCommandJSON\",\"command\": \"define\", \"qualifier\": \"qlo
     cal\",\"name\": \"MSGQ\"}"
     ```

   - Verwenden Sie MQSC-Befehle:

     z/OS Verwenden Sie unter z/OS eine 2CR-Quelle anstelle des Befehls **runmqsc**. Weitere Informationen hierzu finden Sie im Abschnitt Quellen, aus denen Sie MQSC-und PCF-Befehle unter IBM MQ for z/OSausgeben können.

a. Starten Sie **runmqsc** für den Warteschlangenmanager, indem Sie folgenden Befehl eingeben:

```
runmqsc QM1
```

b. Erstellen Sie die Warteschlange mit dem MQSC-Befehl **DEFINE QLOCAL**:

```
DEFINE QLOCAL(MSGQ)
```

c. Beenden Sie **runmqsc**, indem Sie folgenden Befehl eingeben:

```
end
```

5. Erteilen Sie dem Benutzer, den Sie in Schritt 5 von Basiskonfiguration für den mqweb-Server zum `mqwebuser.xml` hinzugefügt haben, die Berechtigung für den Zugriff auf die Warteschlange MSGQ. Ersetzen Sie Ihren Benutzer, wenn `myuser` verwendet wird:

- z/OS Unter z/OS:

a. Erteilen Sie Ihrem Benutzer Zugriffsberechtigung für die Warteschlange:

```
RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
```

b. Erteilen Sie der mqweb-Benutzer-ID der gestarteten Task Zugriffsberechtigung zum Einstellen des gesamten Kontexts für die Warteschlange.

```
RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
```

- Multi Unter allen anderen Betriebssystemen ist die Berechtigung bereits erteilt, falls Ihr Benutzer Mitglied der Gruppe mqm ist. Geben Sie andernfalls folgende Befehle ein:

a. Starten Sie **runmqsc** für den Warteschlangenmanager, indem Sie folgenden Befehl eingeben:

```
runmqsc QM1
```

b. Erteilen Sie Ihrem Benutzer mit dem MQSC-Befehl **SET AUTHREC** Anzeige-, Abfrage-, Abruf- und Einreihungsberechtigungen für die Warteschlange:

```
SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
```

c. Beenden Sie **runmqsc**, indem Sie folgenden Befehl eingeben:

```
end
```

6. Reihen Sie eine Nachricht mit dem Inhalt `Hello World!` in die Warteschlange MSGQ im Warteschlangenmanager QM1ein, indem Sie eine POST-Anforderung für die Ressource `message` verwenden. Ersetzen Sie `myuser` und `mypassword` durch Ihre Benutzer-ID und Ihr Kennwort aus dem `mqwebuser.xml`:

   Die Basisauthentifizierung wird verwendet und ein HTTP-Header `ibm-mq-rest-csrf-token` mit einem beliebigen Wert wird in der cURL-REST-Anforderung festgelegt. Dieser zusätzliche Header ist für POST-, PATCH- und DELETE-Anforderungen erforderlich.

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X POST
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/plain;char
set=utf-8" --data "Hello World!"
```

7. Rufen Sie die Nachricht mit Löschen aus der Warteschlange `Hello World!` in der Warteschlange MSGQ auf dem Warteschlangenmanager QM1 ab, indem Sie eine DELETE-Anforderung für die Ressource `message` verwenden. Ersetzen Sie `myuser` und `mypassword` durch Ihre Benutzer-ID und Ihr Kennwort aus dem `mqwebuser.xml`:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
```

Die Nachricht `Hello World!` wird zurückgegeben.

Nächste Schritte

- In den Beispielen wird die Anforderung durch die Basisauthentifizierung geschützt. Stattdessen können Sie auch die tokenbasierte Authentifizierung oder die clientbasierte Authentifizierung verwenden. Weitere Informationen finden Sie im Abschnitt Clientzertifikatsauthentifizierung mit der REST API und IBM MQ Console verwenden und Tokenbasierte Authentifizierung mit der REST API verwenden.
- Weitere Informationen zur Verwendung der messaging REST API und zum Erstellen von URLs mit Abfrageparametern finden Sie im Abschnitt „messaging REST API verwenden" auf Seite 744.
- Bei Verwendung der messaging REST API werden Verbindungen zum Warteschlangenmanager in Pools zusammengefasst, um die Leistung zu optimieren. Sie können die maximale Poolgröße und die Aktion konfigurieren, die ausgeführt wird, wenn alle Verbindungen im Pool im Gebrauch sind: messaging REST API konfigurieren.
- Durchsuchen Sie die Referenzinformationen nach den verfügbaren messaging REST API-Ressourcen und allen verfügbaren Abfrageparametern: messaging REST API-Referenz.
- Machen Sie sich mit der administrative REST API vertraut, einer REST-konformen Schnittstelle für die IBM MQ-Verwaltung: Verwaltung mithilfe der REST API.
- Machen Sie sich mit der IBM MQ Console vertraut, einer browserbasierten grafischen Benutzerschnittstelle (GUI): Verwaltung mithilfe der IBM MQ Console.

messaging REST API verwenden

Wenn Sie die messaging REST API verwenden, rufen Sie HTTP-Methoden für URLs auf, um IBM MQ-Nachrichten zu senden und zu empfangen. Die HTTP-Methode, z. B. POST, stellt den Typ der Aktion dar, die für das Objekt ausgeführt werden soll, das durch die URL dargestellt wird. Weitere Informationen zur Aktion können in Abfrageparametern codiert werden. Informationen zum Ergebnis der Ausführung der Aktion können als Hauptteil der HTTP-Antwort zurückgegeben werden.

Vorbereitende Schritte

Beachten Sie folgende Punkte, bevor Sie die messaging REST API verwenden:

- Sie müssen sich beim MQWeb-Server authentifizieren, um die messaging REST API verwenden zu können. Die Authentifizierung kann über die HTTP-Basisauthentifizierung, die Clientzertifikatsauthentifizierung oder die tokenbasierte Authentifizierung erfolgen. Weitere Informationen zur Verwendung dieser Authentifizierungsmethoden finden Sie unter IBM MQ Console und REST API Sicherheit.
- Bei der REST API muss die Groß-/Kleinschreibung beachtet werden. Ein HTTP POST für die folgende URL führt beispielsweise zu einem Fehler, wenn der Warteschlangenmanager den Namen `qmgr1` hat.

```
/ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
```

- V 9.4.0 Wenn Sie mit messaging REST APIeine Verbindung zu einem fernen Warteschlangenmanager herstellen, müssen Sie anstelle des Namens des Warteschlangenmanagers den eindeutigen Namen für die Warteschlangenmanagerverbindung verwenden.
- Nicht alle Zeichen, die in IBM MQ-Objektnamen verwendet werden können, können direkt in einer URL codiert werden. Um diese Zeichen richtig zu codieren, muss die entsprechende URL-Codierung verwendet werden:
  - Ein Schrägstrich muss als %2F codiert werden.
  - Ein Prozentzeichen muss als %25 codiert werden.
  - Ein Punkt muss als %2E codiert werden.
  - Ein Fragezeichen muss als %3F codiert werden.

- Beim Empfangen oder Durchsuchen einer Nachricht werden nur IBM MQ MQSTR und JMS TextMessage formatierte Nachrichten unterstützt. Nachfolgend werden alle Nachrichten unter dem Synchronisationspunkt als 'Abruf mit Löschen' empfangen, und alle nicht verarbeiteten Nachrichten bleiben in der Warteschlange. Die IBM MQ-Warteschlange kann so konfiguriert werden, dass diese nicht verarbeitbaren Nachrichten an ein alternatives Ziel verschoben werden. Weitere Informationen finden Sie unter „Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln” auf Seite 244.

Informationen zu diesem Vorgang

Wenn Sie die REST API zum Ausführen einer Messaging-Aktion für ein IBM MQ-Warteschlangenobjekt verwenden, müssen Sie zuerst eine URL erstellen, die dieses Objekt darstellt. Jede URL beginnt mit einem Präfix, das den Hostnamen und den Port beschreibt, an die die Anforderung gesendet werden soll. Der Rest der URL beschreibt ein bestimmtes Objekt oder die Route zu diesem Objekt, bekannt als eine Ressource.

Die Messaging-Aktion, die für die Ressource ausgeführt werden soll, legt fest, ob die URL Abfrageparameter benötigt oder nicht. Sie legt auch die verwendete HTTP-Methode fest und ob zusätzliche Informationen an die URL gesendet oder von ihr zurückgegeben werden. Die zusätzlichen Informationen können Teil der HTTP-Anforderung sein oder als Teil der HTTP-Antwort zurückgegeben werden.

Nachdem Sie die URL erstellt haben, können Sie die HTTP-Anforderung an IBM MQ senden. Sie können die Anforderung mithilfe der HTTP-Implementierung senden, die in die Programmiersprache Ihrer Wahl integriert ist. Sie können die Anforderung auch mithilfe von Befehlszeilentools, z. B. cURL, oder eines Web-Browsers oder Web-Browser-Add-ons senden.

Wichtig: Sie müssen als Minimum die Schritte „1.a” auf Seite 745 und „1.b” auf Seite 745ausführen.

Vorgehensweise

1. Konstrukt die URL:
   a) Ermitteln Sie die Präfix-URL, indem Sie den folgenden Befehl eingeben:

      ```
      dspmqweb status
      ```

      Die URL, die Sie verwenden möchten, enthält die Phrase `/ibmmq/rest/`.

   b) Fügen Sie die Warteschlange und die zugehörigen Warteschlangenmanagerressourcen, die für das Messaging verwendet werden sollen, zum URL-Pfad hinzu.

      In der Messaging-Referenz können die variablen Segmente in der URL daran erkannt werden, dass sie in geschweiften Klammern (*{ }*) stehen. Weitere Informationen finden Sie unter /messaging/qmgr/{qmgrName}/queue/{queueName}/message.

      Um beispielsweise mit der Warteschlange *Q1* zu interagieren, die dem Warteschlangenmanager *QM1* zugeordnet ist, fügen Sie `/qmgr` und `/queue` zur Präfix-URL hinzu, um die folgende URL zu erstellen:

      ```
      https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
      ```

      Tipp: V 9.4.0 Wenn der Warteschlangenmanager ein ferner Warteschlangenmanager ist, müssen Sie den eindeutigen Namen für den Warteschlangenmanager anstelle des Warteschlangenmanagernamens verwenden. Der ferne Warteschlangenmanager muss konfiguriert werden, bevor er mit messaging REST APIverwendet werden kann. Weitere Informationen finden Sie unter „Fernen Warteschlangenmanager für die Verwendung mit messaging REST API einrichten” auf Seite 746.

   c) Optional: Fügen Sie der URL einen optionalen Abfrageparameter hinzu.

      Fragezeichen hinzufügen,?, Abfrageparameter, Gleichheitszeichen = und ein Wert für die URL.

Erstellen Sie beispielsweise folgende URL, wenn maximal 30 Sekunden darauf gewartet werden soll, dass die nächste Nachricht verfügbar wird:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

d) Optional: Fügen Sie weitere optionale Abfrageparameter zur URL hinzu.

Fügen Sie der URL ein Et-Zeichen (&) hinzu und wiederholen Sie anschließend Schritt 1c.

2. Rufen Sie die relevante HTTP-Methode in der URL auf. Geben Sie optionale Nachrichtennutzdaten an und stellen Sie die benötigten Sicherheitsberechtigungsnachweise für die Authentifizierung bereit. For example:
   - Verwenden Sie die HTTP/REST-Implementierung der von Ihnen gewählten Programmiersprache.
   - Verwenden Sie ein Tool wie ein REST-Client-Browser-Add-on oder cURL.

V 9.4.0 Fernen Warteschlangenmanager für die Verwendung mit messaging REST API einrichten

Sie können die messaging REST API verwenden, um eine Verbindung zu fernen Warteschlangenmanagern für Messaging herzustellen. Bevor Sie eine Verbindung zu einem fernen Warteschlangenmanager herstellen können, müssen Sie die Konfiguration des fernen Warteschlangenmanagers einrichten. Anschließend können Sie unter Verwendung des eindeutigen Namens, der in den Konfigurationsdaten definiert ist, eine Verbindung zum fernen Warteschlangenmanager herstellen.

Vorbereitende Schritte

- Stellen Sie sicher, dass Sie den mqweb-Server für die Verwendung durch die administrative REST API, die administrative REST API für MFT, die messaging REST APIoder IBM MQ Consolekonfiguriert haben.Weitere Informationen zur Konfiguration des mqweb-Servers mit einer Basisregistry enthält der Artikel Basiskonfiguration für den mqweb-Server.
- Wenn der mqweb-Server bereits konfiguriert ist, müssen Sie sicherstellen, dass Sie die entsprechenden Benutzer hinzugefügt haben, um das Messaging in Schritt 5 unter Basiskonfiguration für den mqweb-Serverzu aktivieren. Benutzer der messaging REST API müssen Mitglied der Rolle `MQWebUser` sein. Die Rollen `MQWebAdmin` und `MQWebAdminRO` gelten nicht für die messaging REST API.
  - Wenn **`mqRestMessagingAdoptWebUserContext`** in der mqweb-Serverkonfiguration auf `true` gesetzt ist, müssen die Benutzer in der Rolle `MQWebUser` für den Zugriff auf Warteschlangen und Topics berechtigt sein, die für das Messaging über OAM oder RACFverwendet werden.
  - Wenn **`mqRestMessagingAdoptWebUserContext`** in der mqweb-Serverkonfiguration auf `false` gesetzt ist, muss die Benutzer-ID, die zum Starten des mqweb-Servers verwendet wird, berechtigt sein, auf Warteschlangen und Themen zuzugreifen, die für das Messaging über OAM oder RACFverwendet werden.
- Stellen Sie sicher, dass messaging REST API für die Verbindung zu fernen Warteschlangenmanagern konfiguriert ist. Weitere Informationen finden Sie unter Verbindungsmodus für messaging REST APIkonfigurieren.

Informationen zu diesem Vorgang

Sie können über die messaging REST APIeine Verbindung zu fernen Warteschlangenmanagern herstellen. Ein ferner Warteschlangenmanager kann ein Warteschlangenmanager auf einem anderen System, ein Warteschlangenmanager in einer anderen Installation oder ein Warteschlangenmanager in derselben Installation wie der mqweb-Server sein.

Um eine Verbindung zu einem fernen Warteschlangenmanager herzustellen, müssen Sie die folgenden Konfigurationsschritte ausführen:

- Konfigurieren Sie einen Serververbindungskanal und einen Listener.

- Erteilen Sie einem entsprechenden Benutzer die Berechtigung für den Zugriff auf den Warteschlangenmanager.
- Erstellen Sie eine CCDT-Datei mit den Verbindungsinformationen für den Warteschlangenmanager.
- Fügen Sie die Verbindungsinformationen mit dem Befehl **setmqweb remote** zur messaging REST API hinzu.

Anschließend können Sie den fernen Warteschlangenmanager verwenden, indem Sie den eindeutigen Namen in der Ressourcen-URL anstelle des Warteschlangenmanagernamens angeben.

Sie können Ihre fernen Warteschlangenmanager auch als Teil einer Warteschlangenmanagergruppe konfigurieren. Weitere Informationen finden Sie unter „Warteschlangenmanagergruppe für die Verwendung mit der Messaging-REST-API einrichten” auf Seite 749.

Vorgehensweise

1. Erstellen Sie auf dem fernen Warteschlangenmanager einen Serververbindungskanal, um ferne Verbindungen zum Warteschlangenmanager zuzulassen. Sie können Serververbindungskanäle mit dem MQSC-Befehl **DEFINE CHANNEL** in der Befehlszeile erstellen.
   Geben Sie beispielsweise den folgenden Befehl ein, um einen Serververbindungskanal `QM1.SVRCONN` für den Warteschlangenmanager QM1zu erstellen:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Weitere Informationen über **DEFINE CHANNEL** und die verfügbaren Optionen finden Sie unter DEFINE CHANNEL.
2. Stellen Sie sicher, dass ein entsprechender Benutzer berechtigt ist, auf den WS-Manager zuzugreifen. Dieser Benutzer muss außerdem berechtigt sein, auf alle Warteschlangen oder Topics zuzugreifen, die Sie für das Messaging verwenden. Der Benutzer benötigt die Berechtigungen `connect`, `inquire`, `alternate user`und `set context` für den Warteschlangenmanager. Unter UNIX, Linux, and Windows verwenden Sie den Steuerbefehl **setmqaut** in der Befehlszeile. Definieren Sie unter z/OS RACF -Profile, um dem berechtigten Benutzer Zugriff auf den Warteschlangenmanager zu erteilen.
   Geben Sie unter UNIX, Linux, and Windowsbeispielsweise den folgenden Befehl ein, um einen Benutzer `exampleUser`für den Zugriff auf den WS-Manager QM1zu berechtigen:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   Weitere Informationen dazu, welcher Benutzer berechtigt werden muss, finden Sie unter „Von messaging REST API verwendeten Sicherheitsprinzipal ermitteln” auf Seite 752.
3. ALW
   Wenn auf dem fernen Warteschlangenmanager kein Listener vorhanden ist, erstellen Sie mithilfe des MQSC-Befehls **DEFINE LISTENER** in der Befehlszeile einen Listener, der eingehende Netzverbindungen akzeptiert.
   Geben Sie beispielsweise den folgenden Befehl ein, um einen Listener `REMOTE.LISTENER` an Port 1414 für den fernen WS-Manager QM1zu erstellen:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. Stellen Sie sicher, dass das Empfangsprogramm aktiv ist, indem Sie den MQSC-Befehl **START LISTENER** in der Befehlszeile verwenden:

   ALW Geben Sie beispielsweise unter AIX, Linux, and Windows zum Starten des Listeners `REMOTE.LISTENER` für Warteschlangenmanager QM1den folgenden Befehl ein:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

z/OS Um beispielsweise den Listener auf z/OS zu starten, geben Sie den folgenden Befehl ein:

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

Der Adressraum des Kanalinitiators muss gestartet werden, bevor Sie einen Listener unter z/OSstarten.

5. Erstellen oder aktualisieren Sie auf dem System, auf dem der mqweb-Server, auf dem messaging REST API ausgeführt wird, eine JSON-CCDT-Datei, die die Verbindungsinformationen des Warteschlangenmanagers enthält.

   Die CCDT-Datei muss die Informationen zu `name`, `clientConnection`und `type` enthalten. Optional können Sie zusätzliche Informationen wie z. B. zu `transmissionSecurity` hinzufügen. Weitere Informationen über alle CCDT-Kanalattributdefinitionen finden Sie unter Vollständige Liste der CCDT-Kanalattributdefinitionen.

   Das folgende Beispiel zeigt eine einfache JSON-CCDT-Datei für die Verbindung eines fernen Warteschlangenmanagers. Der Name des Kanals wird auf den Namen des in Schritt „1” auf Seite 747erstellten Beispielserververbindungskanals gesetzt. Der Verbindungsport wird auf denselben Wert gesetzt wie der vom Listener verwendete Port. Der Verbindungshost wird auf den Hostnamen des Systems gesetzt, auf dem der ferne Warteschlangenmanager QM1ausgeführt wird.

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

6. Verwenden Sie in der Installation, in der der mqweb-Server ausgeführt wird, auf dem sich messaging REST APIbefindet, den Befehl **setmqweb remote** , um die Informationen zum fernen Warteschlangenmanager zur mqweb-Serverkonfiguration hinzuzufügen.

   Sie müssen mindestens die folgenden Parameter angeben:

   - **-qmgrName**, wobei Sie den Namen des Warteschlangenmanagers angeben.
   - **-ccdtURL**, wobei Sie die CCDT-URL für den Warteschlangenmanager angeben
   - **-uniqueName**, wobei Sie einen eindeutigen Namen für den Warteschlangenmanager angeben. Der eindeutige Name wird verwendet, um ferne Warteschlangenmanager zu unterscheiden, die möglicherweise denselben Namen haben und daher nicht vorhanden sein dürfen, um einen anderen Warteschlangenmanager zu identifizieren.

   Sie können mehrere weitere Optionen angeben, z. B. den Benutzernamen und das Kennwort, die für die Verbindung zum fernen Warteschlangenmanager verwendet werden sollen, oder Details zum Truststore und Keystore. Eine vollständige Liste der Parameter, die mit dem Befehl **setmqweb remote** angegeben werden können, finden Sie unter setmqweb remote.

   Geben Sie beispielsweise folgenden Befehl ein, um den fernen Warteschlangenmanager QM1mit der CCDT-Beispieldatei hinzuzufügen:

```
setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
```

Ergebnisse

Der ferne Warteschlangenmanager kann mit dem messaging REST API verwendet werden, indem der eindeutige Name in der Ressourcen-URL anstelle des Warteschlangenmanagernamens verwendet wird.

Beispiel

Im folgenden Beispiel wird die ferne Warteschlangenmanagerverbindung für einen Warteschlangenmanager QM1eingerichtet. Der IBM MQ Console , der berechtigt ist, den Warteschlangenmanager basierend auf der Berechtigung zu verwalten, die dem Benutzer exampleUsererteilt wird Die Berechtigungsnachweise dieses Benutzers werden dem IBM MQ Console bereitgestellt, wenn die **setmqweb remote** zum Konfigurieren der Warteschlangenmanagerverbindung verwendet wird.

1. Auf dem System, auf dem sich der ferne Warteschlangenmanager QM1 befindet, werden ein Serververbindungskanal und ein Listener erstellt. Der Listener wird gestartet und der Benutzer `exampleUser` erhält die Berechtigung, eine Verbindung zum WS-Manager herzustellen und auf eine Warteschlange zuzugreifen, die für das Messaging verwendet wird:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq
```

2. Auf dem System, auf dem der mqweb-Server ausgeführt wird, wird eine Datei `QM1_ccdt.json` mit den folgenden Verbindungsinformationen erstellt:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. Auf dem System, auf dem der mqweb-Server ausgeführt wird, werden die Verbindungsinformationen für Warteschlangenmanager QM1 zum mqweb-Server hinzugefügt. Die Berechtigungsnachweise für `exampleUser` sind in den Verbindungsinformationen enthalten:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. Der messaging REST API kann eine Verbindung zum fernen Warteschlangenmanager QM1 herstellen, indem er den eindeutigen Namen der Warteschlangenmanagerverbindung anstelle des Warteschlangenmanagernamens in der Ressourcen-URL verwendet:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/mes
sage -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
plain;charset=utf-8" --data "Hello World!"
```

V 9.4.0 Warteschlangenmanagergruppe für die Verwendung mit der Messaging-REST-API einrichten

Sie können die messaging REST API verwenden, um eine Verbindung zu Warteschlangenmanagergruppen für das Messaging herzustellen. Bevor Sie eine Verbindung zu einer Warteschlangenmanagergruppe herstellen können, müssen Sie die Konfiguration des fernen Warteschlangenmanagers für die Gruppe einrichten. Anschließend können Sie unter Verwendung des eindeutigen Namens, der in den Konfigurationsdaten definiert ist, eine Verbindung zur Warteschlangenmanagergruppe herstellen.

Vorbereitende Schritte

- Stellen Sie sicher, dass Sie den mqweb-Server für die Verwendung durch die administrative REST API, die administrative REST API für MFT, die messaging REST APIoder IBM MQ Consolekonfiguriert haben.Weitere Informationen zur Konfiguration des mqweb-Servers mit einer Basisregistry enthält der Artikel Basiskonfiguration für den mqweb-Server.
- Wenn der mqweb-Server bereits konfiguriert ist, müssen Sie sicherstellen, dass Sie die entsprechenden Benutzer hinzugefügt haben, um das Messaging in Schritt 5 unter Basiskonfiguration für den mqweb-Serverzu aktivieren. Benutzer der messaging REST API müssen Mitglied der Rolle `MQWebUser` sein. Die Rollen `MQWebAdmin` und `MQWebAdminRO` gelten nicht für die messaging REST API.
  - Wenn **`mqRestMessagingAdoptWebUserContext`** in der mqweb-Serverkonfiguration auf `true` gesetzt ist, müssen die Benutzer in der Rolle `MQWebUser` berechtigt sein, auf Warteschlangen und Topics zuzugreifen, die für Messaging verwendet werden. Sie können diese Benutzer über OAM oder RACFberechtigen.
  - Wenn **`mqRestMessagingAdoptWebUserContext`** in der mqweb-Serverkonfiguration auf `false` gesetzt ist, muss die Benutzer-ID, die den mqweb-Server startet, berechtigt sein, auf Warteschlangen und Themen zuzugreifen, die für Messaging verwendet werden. Sie können diesen Benutzer über OAM oder RACFberechtigen.
- Stellen Sie sicher, dass messaging REST API für die Verbindung zu fernen Warteschlangenmanagern konfiguriert ist. Weitere Informationen finden Sie unter Verbindungsmodus für messaging REST APIkonfigurieren .

Informationen zu diesem Vorgang

Eine Warteschlangenmanagergruppe ermöglicht Ihnen, Anwendungen mit einem beliebigen Warteschlangenmanager innerhalb der Gruppe zu verbinden. Die Gruppe ist als Gruppe von Verbindungen in einer Clientkanaldefinitionstabelle (CCDT) definiert. Wenn Sie einen MQCONN-oder MQCONNX-Aufruf verwenden, verweisen Sie auf die Gruppe, indem Sie dem Namen des Warteschlangenmanagers einen Stern voranstellen. Mit messaging REST APIverweisen Sie auf die Gruppe, indem Sie den eindeutigen Namen verwenden, der der Warteschlangenmanagergruppe zugeordnet ist. Der eindeutige Name wird anstelle des Namens des Warteschlangenmanagers in die Ressourcen-URL eingeschlossen. Weitere Informationen zu Warteschlangenmanagergruppen finden Sie im Abschnitt „Warteschlangenmanagergruppen in der CCDT” auf Seite 972.

Sie können Ihre fernen Warteschlangenmanager auch einzeln konfigurieren. Weitere Informationen finden Sie unter „Fernen Warteschlangenmanager für die Verwendung mit messaging REST API einrichten” auf Seite 746.

Vorgehensweise

1. Erstellen Sie auf jedem fernen Warteschlangenmanager in der Gruppe einen Serververbindungskanal, um ferne Verbindungen zum Warteschlangenmanager zuzulassen. Sie können den MQSC-Befehl **`DEFINE CHANNEL`** in der Befehlszeile verwenden, um Serververbindungskanäle zu erstellen.
   Geben Sie beispielsweise den folgenden Befehl ein, um einen Serververbindungskanal `QM1.SVRCONN` für den Warteschlangenmanager QM1zu erstellen:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Weitere Informationen über **`DEFINE CHANNEL`** und die verfügbaren Optionen finden Sie unter DEFINE CHANNEL.
2. Stellen Sie auf jedem fernen Warteschlangenmanager in der Gruppe sicher, dass ein entsprechender Benutzer berechtigt ist, auf den Warteschlangenmanager zuzugreifen. Dieser Benutzer muss außerdem berechtigt sein, auf alle Warteschlangen oder Topics zuzugreifen, die Sie für das Messaging verwenden. Der Benutzer benötigt die Berechtigungen `connect`, `inquire`, `alternate user`und `set context` für den Warteschlangenmanager. Unter UNIX, Linux, and Windows verwenden Sie den Steuerbefehl **`setmqaut`** in der Befehlszeile. Definieren Sie unter z/OS RACF -Profile, um dem berechtigten Benutzer Zugriff auf den Warteschlangenmanager zu erteilen.

Geben Sie beispielsweise unter UNIX, Linux, and Windowsden folgenden Befehl ein, um einen Benutzer (`exampleUser`) für den Zugriff auf den Warteschlangenmanager QM1: zu berechtigen:

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

Weitere Informationen dazu, welcher Benutzer berechtigt werden muss, finden Sie unter „Von messaging REST API verwendeten Sicherheitsprinzipal ermitteln” auf Seite 752.

3. ALW
   Wenn auf jedem der fernen Warteschlangenmanager in der Gruppe kein Listener vorhanden ist, erstellen Sie Listener, um eingehende Netzverbindungen zu akzeptieren. Mit dem MQSC-Befehl **DEFINE LISTENER** in der Befehlszeile können Sie Empfangsprogramme erstellen.
   Geben Sie beispielsweise den folgenden Befehl ein, um einen Listener `REMOTE.LISTENER` an Port 1414 für den fernen WS-Manager QM1zu erstellen:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```

4. Stellen Sie auf jedem fernen Warteschlangenmanager in der Gruppe sicher, dass der Listener ausgeführt wird, indem Sie den MQSC-Befehl **START LISTENER** in der Befehlszeile verwenden.

   ALW Geben Sie beispielsweise unter AIX, Linux, and Windows zum Starten des Listeners `REMOTE.LISTENER` für Warteschlangenmanager QM1den folgenden Befehl ein:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS Um beispielsweise den Listener auf z/OS zu starten, geben Sie den folgenden Befehl ein:

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

   Der Adressraum des Kanalinitiators muss gestartet werden, bevor Sie einen Listener unter z/OSstarten.

5. Erstellen Sie auf dem System, auf dem der mqweb-Server, der als Host für messaging REST API dient, ausgeführt wird, eine JSON-CCDT-Datei. Diese JSON-Datei enthält Verbindungsinformationen für jeden Warteschlangenmanager in der Gruppe.

   Die CCDT-Datei muss die `name`-, `clientConnection`-und `type` -Informationen für jede Warteschlangenmanagerverbindung enthalten. Optional können Sie zusätzliche Informationen wie z. B. zu `transmissionSecurity` hinzufügen. Weitere Informationen über alle CCDT-Kanalattributdefinitionen finden Sie unter Vollständige Liste der CCDT-Kanalattributdefinitionen.

   Das folgende Beispiel zeigt eine grundlegende JSON-CCDT-Datei für zwei Warteschlangenmanagerverbindungen. Die erste Verbindung gilt für Warteschlangenmanager QM1. Er verfügt über den Serververbindungskanal `QM1.SVRCONN`, einen Listener am Port 1414und wird auf dem Host `QM1.example.com`ausgeführt. Die zweite Verbindung gilt für Warteschlangenmanager QM2. Er verfügt über den Serververbindungskanal `QM2.SVRCONN`, einen Listener am Port 1415und wird auf dem Host `QM2.example.com`ausgeführt. Da die Verbindungen jedoch zur Warteschlangenmanagergruppe QMGRP gehören, wird das Feld **queueManager** für beide Verbindungen auf den Namen der Warteschlangenmanagergruppe gesetzt.

   ```
   {
     "channel": [{
         "name": "QM1.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "QM1.example.com",
               "port": 1414
             }],
           "queueManager": "QMGRP"
         },
   ```

```
      "type": "clientConnection"
   }],
   "channel": [{
      "name": "QM2.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM2.example.com",
            "port": 1415
          }],
        "queueManager": "QMGRP"
      },
      "type": "clientConnection"
    }]
}
```

6. Verwenden Sie in der Installation, in der der mqweb-Server ausgeführt wird, auf dem sich messaging REST APIbefindet, den Befehl **setmqweb remote** , um die Warteschlangenmanagergruppe zur mqweb-Serverkonfiguration hinzuzufügen.

   Sie müssen mindestens die folgenden Parameter angeben:

   - **-qmgrName**, wobei Sie den Gruppennamen für die Warteschlangenmanagergruppe angeben.
   - **-ccdtURL**, wobei Sie die CCDT-URL für die Warteschlangenmanager angeben
   - **-uniqueName**: Hier geben Sie einen eindeutigen Namen zur Identifizierung der Warteschlangenmanagergruppe an.
   - **-group**, um die Warteschlangenmanagerinformationen als für eine Gruppe zu definieren.

   Sie können mehrere weitere Optionen angeben, z. B. den Benutzernamen und das Kennwort, die für die Verbindung verwendet werden sollen, oder Details zum Truststore und Keystore. Eine vollständige Liste der Parameter, die mit dem Befehl **setmqweb remote** angegeben werden können, finden Sie unter setmqweb remote.

   Geben Sie beispielsweise den folgenden Befehl ein, um die Warteschlangenmanagergruppe QMGRP mit der Beispieldatei CCDT hinzuzufügen:

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

Ergebnisse

Die ferne Warteschlangenmanagergruppe kann mit messaging REST API verwendet werden, indem der eindeutige Name in der Ressourcen-URL verwendet wird. Ein Warteschlangenmanager aus der Gruppe wird ausgewählt, um die Anforderung auszuführen, und Informationen darüber, welcher Warteschlangenmanager die Anforderung abgeschlossen hat, werden im Antwortheader `ibm-mq-resolved-qmgr`zurückgegeben.

V 9.4.0 Von messaging REST API verwendeten Sicherheitsprinzipal ermitteln

Wenn Sie messaging REST APIverwenden, muss ein entsprechender Benutzer berechtigt sein, auf die Warteschlangenmanager, Warteschlangen und Themen zuzugreifen, zu denen Sie eine Verbindung für Messaging herstellen möchten. Der Benutzer, der berechtigt werden muss, hängt davon ab, wie Ihr mqweb-Server konfiguriert ist und ob Sie ferne Warteschlangenmanager mit messaging REST APIverwenden.

Standardmäßig wird der Sicherheitsprinzipal, der für die Berechtigung des Zugriffs auf den Warteschlangenmanager verwendet wird, als Benutzer verwendet, der den mqweb-Server startet, auf dem messaging REST APIausgeführt wird. Der Sicherheitsprinzipal, der zum Autorisieren des Zugriffs auf die Warteschlangen und Themen verwendet wird, ist der Benutzer, der bei messaging REST APIangemeldet ist. Ihre Verbindung zum mqweb-Server oder zum fernen Warteschlangenmanager kann jedoch so konfiguriert sein, dass ein anderer Sicherheitsprinzipal verwendet wird.

Sicherheitsprinzipal bestimmen, der für die Verbindung zum Warteschlangenmanager verwendet wird

Bei lokalen Warteschlangenmanagerverbindungen ist der Sicherheitsprinzipal, der für die Verbindung zum Warteschlangenmanager verwendet wird, der Benutzer, der den Mqweb-Server startet, auf dem messaging REST APIausgeführt wird. Für Verbindungen zu fernen Warteschlangenmanagern werden die folgenden Sicherheitsprinzipien von messaging REST API verwendet, um den Zugriff auf den Warteschlangenmanager nach Priorität zu berechtigen. Das heißt, wenn Benutzer auf mehrere Arten in der Konfiguration des fernen Warteschlangenmanagers angegeben werden, wird die erste in der Liste für die Autorisierung verwendet.

1. Der Sicherheitsprincipal ist ein übernommener Benutzerkontext aus einem Sicherheitsexit.
2. Der Sicherheitsprinzipal ist ein übernommener Benutzerkontext in einer CHLAUTH-Regel auf dem Serververbindungskanal, der für die Verbindung zum fernen Warteschlangenmanager verwendet wird.
3. Der Sicherheitsprinzipal ist die Benutzer-ID, die in der Konfiguration des fernen Warteschlangenmanagers für messaging REST APIenthalten ist. Diese Benutzer-ID ist optional in den Verbindungsinformationen des Warteschlangenmanagers enthalten, wenn Sie den Warteschlangenmanager mit dem Befehl **setmqweb remote** hinzufügen.
4. Der Sicherheitsprinzipal ist der Benutzer, der den mqweb-Server startet, auf dem messaging REST APIausgeführt wird.

Weitere Informationen zum Konfigurieren von fernen Warteschlangenmanagern für die Verwendung mit messaging REST APIfinden Sie im Abschnitt „Fernen Warteschlangenmanager für die Verwendung mit messaging REST API einrichten” auf Seite 746.

Sicherheitsprinzipal bestimmen, der für die Verbindung zu Warteschlangen und Themen verwendet wird

Sie können eine Eigenschaft in der mqweb-Serverkonfiguration festlegen, um festzulegen, welcher Sicherheitsprinzipal verwendet wird, um Verbindungen zu Warteschlangen und Themen zu berechtigen, wenn Sie messaging REST APIverwenden. Diese Eigenschaft ist die Eigenschaft **mqRestMessagingAdoptWebUserContext** . Sie können die Einstellung dieser Eigenschaft mit dem Befehl **dspmqweb properties** anzeigen.

- Wenn **mqRestMessagingAdoptWebUserContext** auf 'true' gesetzt ist, verwendet messaging REST API die Benutzer-ID des Benutzers, der bei messaging REST API für die Autorisierung angemeldet ist. Daher sind die Benutzer-IDs, die in der mqweb-Serverkonfiguration für die Verwendung mit messaging REST API vorhanden sind, die Sicherheitsprinzipien, die für den Zugriff auf die Warteschlangen und Themen berechtigt sein müssen.
- Wenn **mqRestMessagingAdoptWebUserContext** auf ' false' gesetzt ist, verwendet messaging REST API die Benutzer-ID des Benutzers, der den mqweb-Server gestartet hat, der die messaging REST API -Instanz für die Berechtigung hostet. Daher muss eine Benutzer-ID, die mit der Benutzer-ID identisch ist, die den mqweb-Server startet, der den messaging REST API hostet, berechtigt sein, auf die Warteschlangen und Themen zuzugreifen.

  Wenn sich Ihre Warteschlangen und Themen auf einem fernen Warteschlangenmanager befinden, kann der Sicherheitsprinzipal, der für die Autorisierung verwendet wird, durch Einstellungen in der Warteschlangenmanagerkonfiguration bestimmt werden. Die folgenden Sicherheitsprincipals können in der Reihenfolge ihrer Priorität verwendet werden:

  1. Der Sicherheitsprincipal ist ein übernommener Benutzerkontext aus einem Sicherheitsexit.
  2. Der Sicherheitsprinzipal ist ein übernommener Benutzerkontext in einer CHLAUTH-Regel auf dem Serververbindungskanal, der für die Verbindung zum fernen Warteschlangenmanager verwendet wird. Sie können beispielsweise eine CHLAUTH-Regel auf dem Serververbindungskanal für die Verwendung des Parameters MCAUSER konfigurieren. Anschließend werden alle Verbindungen einer Benutzer-ID zugeordnet, die zur Verwendung des Warteschlangenmanagers berechtigt ist.
  3. Der Sicherheitsprinzipal ist ein übernommener Benutzerkontext aus AUTHINFO des Warteschlangenmanagers. Wenn das AUTHINFO-Objekt, auf das das CONNAUTH-Attribut des Warteschlangen-

managers verweist, für die Verwendung von **ADOPTCTX(yes)** konfiguriert ist, wird der Sicherheitsprinzipal, der für die Berechtigung von Verbindungen zum Warteschlangenmanager verwendet wird, auch für die Berechtigung der Warteschlangen und Themen verwendet. Dieser Sicherheitsprinzipal kann beispielsweise die Benutzer-ID sein, die im Befehl **setmqweb remote** in den Verbindungsinformationen des fernen Warteschlangenmanagers enthalten ist.

Zugehörige Informationen

CHLAUTH

VERBINDUNG

dspmqweb-Eigenschaften

MQI-Anwendungen mit IBM MQ entwickeln

IBM MQ unterstützt die Programmiersprachen C, Visual Basic, COBOL, Assembler, RPG, pTAL und PL/I. Diese prozeduralen Programmiersprachen rufen Message-Queuing-Services über die Schnittstelle für Nachrichtenwarteschlangen (MQI) auf.

Ausführliche Informationen zum Schreiben von Anwendungen in der gewünschten Programmiersprache finden Sie in den folgenden Unterabschnitten.

Eine Übersicht der Call-Schnittstelle für prozedurale Programmiersprachen finden Sie im Abschnitt Beschreibung der Aufrufe. Dieser Abschnitt enthält eine Liste der MQI-Aufrufe. Zu jedem Aufruf wird erläutert, wie Sie ihn in den einzelnen Sprachen codieren.

IBM MQ stellt Datendefinitionsdateien bereit, die Sie zum Schreiben Ihrer Anwendungen verwenden können. Eine ausführliche Beschreibung finden Sie unter „IBM MQ-Datendefinitionsdateien” auf Seite 755.

Die maximale Länge der Nachrichten, die Ihr Programmen verarbeiten wird, ist ein hilfreiches Kriterium bei der Auswahl der geeigneten prozeduralen Programmiersprache. Wenn Ihre Programme nur Nachrichten mit einer bekannten maximalen Länge verarbeiten, können Sie sie in jeder unterstützten Sprache codieren. Falls Sie die maximale Länge der Nachrichten, die die Programme verarbeiten müssen, nicht kennen, richtet sich die Auswahl der Programmiersprache danach, ob Sie eine CICS-, eine IMS- oder eine Stapelanwendung schreiben:

IMS- und Stapelanwendung
: Codieren Sie die Programme in C, PL/I oder in Assemblersprache, um die jeweiligen Funktionen dieser Sprachen zum Anfordern und Freigeben beliebiger Speichermengen zu nutzen. Alternativ könnten Sie Ihre Programme in COBOL codieren. Verwenden Sie aber zum Anfordern und Freigeben von Speicher Subroutinen in Assemblersprache, PL/I oder C.

CICS
: Codieren Sie die Programme in einer von CICS unterstützten Sprache. Falls erforderlich, stellt die EXEC-Schnittstelle von CICS die Aufrufe zur Speicherverwaltung bereit.

Zugehörige Konzepte

„Objektorientierte Anwendungen” auf Seite 16
IBM MQ unterstützt JMS, Java, C++ und .NET. Diese Sprachen und Frameworks verwenden das IBM MQ-Objektmodell. Es stellt Klassen bereit, die über dieselbe Funktionalität verfügen wie Aufrufe und Strukturen von IBM MQ.

Technische Übersicht

„Konzepte für die Anwendungsentwicklung” auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

Zugehörige Verweise

Referenzinformationen zum Entwickeln von Anwendungen

IBM MQ-Datendefinitionsdateien

IBM MQ stellt Datendefinitionsdateien bereit, die Sie zum Schreiben Ihrer Anwendungen verwenden können.

Datendefinitionsdateien sind auch bekannt als:

Sprache	Datendefinitionen
C	Include-Dateien oder Headerdateien
Visual Basic	Moduldateien (nur 32-Bit-Versionen)
COBOL	Kopierdateien
Assembler	Makros
PL/I	Include-Dateien

Die Datendefinitionsdateien, die Ihnen beim Schreiben von Kanalexits helfen sollen, werden im Abschnitt Kopier-, Header-, Include- und Moduldateien von IBM MQ beschrieben.

Die Datendefinitionsdateien, die Ihnen beim Schreiben von Exits für installierbare Services helfen sollen, werden im Abschnitt „Benutzerexits, API-Exits und installierbare IBM MQ-Services” auf Seite 986 beschrieben.

Informationen zu Datendefinitionsdateien, die von C++ unterstützt werden, finden Sie unter C++ verwenden.

IBM i

Informationen zu Datendefinitionsdateien, die unter RPG unterstützt werden, finden Sie im Handbuch IBM i Application Programming Reference (ILE/RPG).

Die Namen der Datendefinitionsdateien haben das Präfix CMQ sowie ein Suffix, das durch die Programmiersprache bestimmt wird:

Suffix	Sprache
a	Assemblersprache
b	Visual Basic
c	C
l	COBOL (ohne Anfangswerte)
p	PL/I
v	COBOL (mit gesetzten Standardwerten)

Installationsbibliothek

z/OS Der Name **thlqual** gibt das übergeordnete Qualifikationsmerkmal der Installationsbibliothek unter z/OS an.

Unter folgenden Überschriften finden Sie in diesem Abschnitt Erläuterungen zu IBM MQ-Datendefinitionsdateien:

- „Include-Dateien in Programmiersprache C” auf Seite 756
- „Visual Basic-Moduldateien” auf Seite 756
- „Kopierdateien in COBOL” auf Seite 756
- z/OS „Makros in Assemblersprache System/390” auf Seite 757
- z/OS „PL/I-Kopfdatendateien” auf Seite 758

Include-Dateien in Programmiersprache C

Die Include-Dateien von IBM MQ in der Programmiersprache C sind unter Headerdateien in Programmiersprache C aufgeführt. Sie werden in folgenden Verzeichnissen oder Bibliotheken installiert:

Plattform	Installationsverzeichnis oder Bibliothek
IBM i	QMQM/H
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
Windows	*MQ_INSTALLATION_PATH*\Tools\c\include
z/OS	**thlqual**.SCSQC370

Dabei steht *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Anmerkung: Unter AIX and Linux sind die Include-Dateien symbolisch in `/usr/include` verknüpft.

Weitere Informationen zur Struktur von Verzeichnissen finden Sie unter Dateisystemunterstützung planen.

Visual Basic-Moduldateien

IBM MQ for Windows stellt vier Moduldateien in Visual Basic bereit.

Sie sind unter Moduldateien in Visual Basic aufgeführt und installiert unter

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

Kopierdateien in COBOL

Für COBOL stellt IBM MQ separate Kopierdateien mit den benannten Konstanten sowie zwei Kopierdateien für die einzelnen Strukturen bereit.

Es gibt für jede Struktur zwei Kopierdateien, weil jede sowohl mit als auch ohne Anfangswerte bereitgestellt wird:

- Verwenden Sie in der WORKING-STORAGE SECTION eines COBOL-Programms die Dateien, die die Strukturfelder mit Standardwerten initialisieren. Diese Strukturen sind in den Kopierdateien definiert, deren Namen den Buchstaben V (Values = Werte) als Suffix haben.
- Verwenden Sie in der LINKAGE SECTION eines COBOL-Programms die Strukturen ohne Anfangswerte. Diese Strukturen sind in Kopierdateien definiert, deren Namen den Buchstaben L (Linkage = Verknüpfung) als Suffix haben.

IBM i Kopierdateien, die Daten- und Schnittstellendefinitionen für IBM i enthalten, werden mittels Prototypaufrufen von MQI für ILE COBOL-Programme bereitgestellt. Die Dateien befinden sich in QMQM/QCBLLESRC und haben Teildateinamen mit dem Suffix L (für Strukturen ohne Anfangswerte) oder dem Suffix V (für Strukturen mit Anfangswerten).

Die COBOL-Kopierdateien von IBM MQ sind im Abschnitt COBOL-Kopierdateien aufgelistet. Sie werden in folgenden Verzeichnissen installiert:

Plattform	Installationsverzeichnis oder Bibliothek
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/

Plattform	Installationsverzeichnis oder Bibliothek
IBM i	QMQM/QCBLLESRC
Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (für Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (für IBM VisualAge COBOL)
z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Schließen Sie nur die Dateien in Ihr Programm ein, die Sie benötigen. Verwenden Sie dazu eine oder mehrere COPY-Anweisungen hinter einer Deklaration der Ebene 01. Das heißt, dass Sie bei Bedarf mehrere Versionen der Strukturen in ein Programm einschließen können. Beachten Sie, dass CMQV eine große Datei ist.

Hier ein Beispiel eines COBOL-Codes zum Einschließen der Kopierdatei CMQMDV:

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

Jede Strukturdeklaration beginnt mit einem Element der Ebene 01; Sie können mehrere Instanzen der Struktur deklarieren, indem Sie hinter der Deklaration der Ebene 01 eine COPY-Anweisung codieren, mit der der Rest der Strukturdeklaration hineinkopiert wird. Fügen Sie mit dem Schlüsselwort IN einen Verweis auf die geeignete Instanz hinzu.

Hier ein Beispiel für einen COBOL-Code zum Einschließen von zwei CMQMDV-Instanzen:

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

Richten Sie die Strukturen an 4-Byte-Grenzen aus. Wenn Sie mit einer COPY-Anweisung eine Struktur hinter einem Element einschließen, das kein Element der Ebene 01 ist, dann stellen Sie sicher, dass die Struktur ein Mehrfaches von 4 Bytes ausgehend vom Anfang des Elements der Ebene 01 ist. Andernfalls kann sich die Leistung Ihrer Anwendung verschlechtern.

Die Strukturen sind im Abschnitt Im MQI verwendete Datentypen beschrieben. In den Beschreibungen der Felder in den Strukturen werden die Namen von Feldern ohne ein Präfix angezeigt. Setzen Sie in COBOL-Programmen den Namen der Struktur, gefolgt von einem Bindestrich, als Präfix vor die Feldnamen, so wie in den COBOL-Deklarationen gezeigt. Die Felder in den Strukturkopierdateien werden auf diese Weise mit einem Präfix versehen.

Die Feldnamen in den Deklarationen in den Strukturkopierdateien sind in Großbuchstaben geschrieben. Sie können stattdessen auch Groß-/Kleinschreibung oder Kleinbuchstaben verwenden. Beispiel: Das Feld *StrucId* der MQGMO-Struktur wird in der COBOL-Deklaration und in der Kopierdatei als MQGMO-STRUCID angezeigt.

Die Strukturen mit dem V-Suffix sind mit Anfangswerten für alle Felder deklariert, sodass Sie nur die Felder einstellen müssen, in denen der Anfangswert nicht dem erforderlichen Wert entspricht.

Makros in Assemblersprache System/390

z/OS

IBM MQ for z/OS stellt zwei Makros in Assemblersprache bereit, die die benannten Konstanten sowie ein Makro zur Generierung der einzelnen Strukturen enthalten.

Sie sind unter z/OSAssembler-Kopierdateien aufgeführt und in **thlqual**.SCSQMACS installiert.

Diese Makros werden mit einem Code wie dem folgenden aufgerufen:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

PL/I-Kopfdatendateien

z/OS

IBM MQ for z/OS stellt Include-Dateien bereit, die alle Definitionen enthalten, die Sie zum Schreiben von IBM MQ-Anwendungen in PL/I benötigen.

Die Dateien sind im Abschnitt PL/I-Kopfdatendateien aufgelistet und werden im Verzeichnis **thlqual**.SCSQPLIC installiert.

Schließen Sie diese Dateien in Ihr Programm ein, wenn Sie den IBM MQ-Stub mit Ihrem Programm verknüpfen (siehe „Preparing your program to run" auf Seite 1076). Schließen Sie nur CMQP mit ein, wenn Sie die IBM MQ-Aufrufe dynamisch verknüpfen möchten (siehe „Dynamically calling the IBM MQ stub" auf Seite 1082). Diese dynamische Verknüpfung kann nur für Stapel- und für IMS-Programme durchgeführt werden.

Prozedurale Anwendungen für die Warteschlangensteuerung erstellen

Dieser Abschnitt enthält Informationen zum Erstellen von Anwendungen für die Warteschlangensteuerung, zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager und zum Öffnen und Schließen von Objekten.

Unter den folgenden Links finden Sie weitere Informationen zum Erstellen von Anwendungen:

- „Message Queue Interface (MQI) - Übersicht" auf Seite 759
- „Verbindung zu einem Warteschlangenmanager herstellen und trennen" auf Seite 773
- „Objekte öffnen und schließen" auf Seite 780
- „Nachrichten in eine Warteschlange einreihen" auf Seite 792
- „Nachrichten aus einer Warteschlange abrufen" auf Seite 808
- „Publish/Subscribe-Anwendungen schreiben" auf Seite 851
- „Objektattribute abfragen und einstellen" auf Seite 897
- „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen" auf Seite 901
- „IBM MQ-Anwendungen durch Auslöser starten" auf Seite 913
- „Mit MQI und Clustern arbeiten" auf Seite 933
- z/OS „Using and writing applications on IBM MQ for z/OS" auf Seite 938
- z/OS „IMS and IMS bridge applications on IBM MQ for z/OS" auf Seite 73

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung" auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

„Anwendungen für IBM MQ entwickeln" auf Seite 5
Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

„Konzepte für den Entwurf von IBM MQ-Anwendungen" auf Seite 52

Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

„Prozedurale Clientanwendungen schreiben” auf Seite 961
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„Prozedurale Anwendung erstellen” auf Seite 1052
Sie können eine IBM MQ-Anwendung in einer von mehreren prozeduralen Sprachen schreiben und die Anwendung auf mehreren unterschiedlichen Plattformen ausführen.

„Prozedurale Programmfehler handhaben” auf Seite 1090
In diesen Informationen werden Fehler erläutert, die den MQI-Aufrufen Ihrer Anwendungen beim Ausgeben eines Aufrufs oder bei der Übergabe einer Nachricht an den Zielort zugeordnet werden.

Zugehörige Tasks

„Prozedurale IBM MQ-Beispielprogramme verwenden” auf Seite 1111
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

Message Queue Interface (MQI) - Übersicht

Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

Die Message Queue Interface setzt sich aus den folgenden Komponenten zusammen:

- *Aufrufen*, über die Programme auf den Warteschlangenmanager und dessen Funktionen zugreifen können
- *Strukturen*, über die Programme Daten an den Warteschlangenmanager übergeben bzw. Daten aus dem Warteschlangenmanager abrufen
- *Elementardatentypen* für die Übergabe von Daten an den Warteschlangenmanager bzw. für den Abruf von Daten aus dem Warteschlangenmanager

z/OS IBM MQ for z/OS stellt darüber hinaus noch Folgendes bereit:

- Zwei zusätzliche Aufrufe, über die z/OS-Stapelprogramme Änderungen festschreiben und zurücksetzen können
- *Datendefinitionsdateien* (manchmal auch als Kopierdateien, Makros, Include-Dateien und Headerdateien bezeichnet), in denen die Werte der von IBM MQ for z/OS bereitgestellten Konstanten definiert werden
- *Stubprogramme* für die Programmverknüpfung mit Ihren Anwendungen
- Eine Reihe von Beispielprogrammen, die die Verwendung der MQI auf der z/OS-Plattform veranschaulichen. Weitere Informationen zu diesen Beispielprogrammen finden Sie im Abschnitt „Using the sample programs for z/OS” auf Seite 1221.

IBM i IBM MQ for IBM i stellt darüber hinaus noch Folgendes bereit:

- *Datendefinitionsdateien* (manchmal auch als Kopierdateien, Makros, Include-Dateien und Headerdateien bezeichnet), in denen die Werte der von IBM MQ for IBM i bereitgestellten Konstanten definiert werden
- Drei Stubprogramme, um eine Programmverknüpfung mit Ihren ILE C-, ILE COBOL- und ILE RPGC-Anwendungen herzustellen
- Eine Reihe von Beispielprogrammen, die die Verwendung der MQI auf der IBM i-Plattform veranschaulichen.

AIX, Linux, and Windows-Systeme stellen darüber hinaus noch Folgendes bereit:

- Aufrufe, über die Systemprogramme von IBM MQ for AIX, Linux, and Windows Änderungen festschreiben und zurücksetzen können.

- *Include-Dateien*, in denen die auf diesen Plattformen bereitgestellten Konstanten definiert sind
- *Bibliotheksdateien*, um Ihre Anwendungen zu verknüpfen
- Eine Reihe von Beispielprogrammen, die die Verwendung der MQI auf diesen Plattformen veranschaulichen Weitere Informationen zu diesen Beispielprogrammen finden Sie im Abschnitt „Beispielprogramme plattformübergreifend verwenden" auf Seite 1112.
- Beispielquellcode und ausführbaren Beispielcode für die Verbindung zu externen Transaktionsmanagern

Unter den folgenden Links finden Sie weitere Informationen zur MQI:

- „MQI-Aufrufe" auf Seite 760
- „Synchronisationspunktaufrufe" auf Seite 761
- „Datenkonvertierung, Datentypen, Datendefinitionen und Strukturen" auf Seite 762
- „IBM MQ-Stubprogramme und -Bibliotheksdateien" auf Seite 763
- „Parameter, die in allen Aufrufen verwendet werden" auf Seite 768
- „Puffer angeben" auf Seite 769
- z/OS „z/OS batch considerations" auf Seite 769
- „Signalverarbeitung unter AIX and Linux" auf Seite 770

Zugehörige Konzepte

„Verbindung zu einem Warteschlangenmanager herstellen und trennen" auf Seite 773
Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Objekte öffnen und schließen" auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen" auf Seite 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen" auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen" auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen" auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„IBM MQ-Anwendungen durch Auslöser starten" auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten" auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS" auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS" auf Seite 73
This information helps you to write IMS applications using IBM MQ.

MQI-Aufrufe

Dieser Abschnitt enthält Informationen zu den MQI-Aufrufen (Message Queue Interface).

Die Aufrufe in der MQI werden in folgende Gruppen unterteilt:

MQCONN, MQCONNX und MQDISC
Mit diesen Aufrufen wird ein Programm (unter Angabe von Optionen oder ohne Angabe von Optionen) mit einem Warteschlangenmanager verbunden bzw. die Verbindung zwischen einem Programm und einem Warteschlangenmanager wieder getrennt.

z/OS Wenn Sie CICS-Programme für z/OS erstellen, sind diese Aufrufe nicht erforderlich. Wenn Ihre Anwendung jedoch auf andere Plattformen portiert werden soll, sollten Sie diese Aufrufe verwenden.

MQOPEN und MQCLOSE
Mit diesen Aufrufen werden Objekte (beispielsweise Warteschlangen) geöffnet und geschlossen.

MQPUT und MQPUT1
Mit diesen Aufrufen werden Nachrichten in eine Warteschlange eingereiht.

MQGET
Mit diesem Aufruf werden Nachrichten in einer Warteschlange angezeigt oder aus einer Warteschlange entfernt.

MQSUB, MQSUBRQ
Mit diesen Aufrufen wird eine Subskription für ein Thema registriert und werden Veröffentlichungen in Zusammenhang mit der Subskription abgerufen.

MQINQ
Mit diesem Aufruf werden die Attribute eines Objekts abgefragt.

MQSET
Mit diesem Aufruf werden einige der Attribute einer Warteschlange gesetzt. Die Attribute anderer Objekttypen können mit diesem Aufruf nicht gesetzt werden.

MQBEGIN, MQCMIT und MQBACK
Diese Aufrufe werden verwendet, wenn IBM MQ eine Arbeitseinheit koordiniert. Mit MQBEGIN wird die Arbeitseinheit gestartet, mit MQCMIT und MQBACK wird sie beendet, wobei die während der Arbeitseinheit vorgenommenen Änderungen festgeschrieben bzw. zurückgesetzt werden. IBM i Mit dem IBM i-Controller für die Festschreibung werden unter IBM MQ for IBM i globale Arbeitseinheiten koordiniert. Zum Starten der Commitsteuerung, zum Festschreiben und zum Zurücksetzen werden native Befehle verwendet.

MQCRTMH, MQBUFMH, MQMHBUF, MQDLTMH
Mit diesen Aufrufen wird eine Nachrichtenkennung erstellt, eine Nachrichtenkennung in einen Puffer konvertiert, ein Puffer in eine Nachrichtenkennung konvertiert und eine Nachrichtenkennung gelöscht.

MQSETMP, MQINQMP, MQDLTMP
Mit diesen Aufrufen wird eine Nachrichteneigenschaft in einer Nachrichtenkennung gesetzt, eine Nachrichteneigenschaft abgerufen und eine Eigenschaft aus einer Nachrichtenkennung gelöscht.

MQCB, MQCB_FUNCTION, MQCTL
Mit diesen Aufrufen wird eine Callback-Funktion registriert und gesteuert.

MQSTAT
Mit diesem Aufruf werden Statusinformationen zu vorangegangenen asynchronen Put-Operationen abgerufen.

Eine Beschreibung der MQI-Aufrufe finden Sie im Abschnitt Beschreibungen der Aufrufe.

Synchronisationspunktaufrufe
Dieser Abschnitt enthält Informationen zu Synchronisationspunktaufrufen auf verschiedenen Plattformen.

Synchronisationspunktaufrufe stehen wie folgt zur Verfügung:

IBM MQ for z/OS Aufrufe
z/OS

IBM MQ for z/OS stellt die Aufrufe MQCMIT und MQBACK bereit.

Mit diesen Aufrufen in z/OS-Stapelprogrammen wird der Warteschlangenmanager angewiesen, alle seit dem letzten Synchronisationspunkt ausgeführten MQGET- und MQPUT-Operationen festzuschreiben (damit werden sie permanent) oder zurückzusetzen. In anderen Umgebungen werden Änderungen wie folgt festgeschrieben bzw. zurückgesetzt:

CICS
Mit Befehlen wie EXEC CICS SYNCPOINT und EXEC CICS SYNCPOINT ROLLBACK.

IMS
Mit den IMS-Synchronisationspunktfunktionen wie dem GU-Aufruf (Get Unique) an die IOPCB-Schnittstelle, dem CHKP-Aufruf (Checkpoint) und dem ROLB-Aufruf (Rollback).

RRS
Je nachdem mit dem MQCMIT- bzw. MQBACK-Aufruf oder mit dem Befehl SRRCMIT bzw. SRRBACK (siehe „Transaction management and recoverable resource manager services" auf Seite 905).

Anmerkung: SRRCMIT und SRRBACK sind native RRS-Befehle, keine MQI-Aufrufe.

IBM i Aufrufe

IBM i

IBM MQ for IBM i stellt die Befehle MQCMIT und MQBACK bereit. Sie können auch die IBM i-Befehle COMMIT und ROLLBACK oder andere Befehle oder Aufrufe verwenden, mit denen die IBM i-Funktionen zur Commitsteuerung wie beispielsweise CICS SYNCPOINT aufgerufen werden.

IBM MQ-Aufrufe auf AIX, Linux, and Windows-Plattformen

ALW

IBM MQ for AIX, Linux, and Windows stellt die MQCMIT- und MQBACK-Aufrufe bereit.

Mit Synchronisationspunktaufrufen in Programmen können Sie den Warteschlangenmanager anweisen, alle seit dem letzten Synchronisationspunkt ausgeführten MQGET- und MQPUT-Operationen festzuschreiben (damit werden sie permanent) oder zurückzusetzen. Änderungen in der CICS-Umgebung werden mit Befehlen wie beispielsweise EXEC CICS SYNCPOINT und EXEC CICS SYNCPOINT ROLLBACK festgeschrieben bzw. zurückgesetzt.

Datenkonvertierung, Datentypen, Datendefinitionen und Strukturen

Dieser Abschnitt enthält Informationen zur Datenkonvertierung, zu Elementardatentypen, zu IBM MQ-Datendefinitionen und zu Strukturen bei Verwendung der MQI.

Datenkonvertierung

Mit dem MQXCNVC-Aufruf (Datenkonvertierung) werden Nachrichtenzeichendaten in einen anderen Zeichensatz konvertiert. Dieser Aufruf wird nur von einem Datenkonvertierungsexitverwendet, außer unter IBM MQ for z/OS.

Informationen zur Syntax bei Verwendung des MQXCNVC-Aufrufs finden Sie unter MQXCNVC - Convert characters, Hinweise zum Erstellen und Aufrufen von Datenkonvertierungsexits finden Sie unter „Datenkonvertierungsexits schreiben" auf Seite 1035.

Elementardatentypen

Für die unterstützten Programmiersprachen stellt die MQI Elementardatentypen oder unstrukturierte Felder bereit.

Eine ausführliche Beschreibung dieser Datentypen finden Sie unter Elementary data types.

IBM MQ-Datendefinitionen

z/OS IBM MQ for z/OS stellt Datendefinitionen in Form von COBOL-Kopierdateien, Assemblermakros, einer PL/I-Include-Datei, einer C-Include-Datei und in Form von C++-Include-Dateien bereit.

IBM i IBM MQ for IBM i stellt Datendefinitionen in Form von COBOL-Kopierdateien, RPG-Kopierdateien, C-Include-Dateien und C++-Include-Dateien bereit.

Die in IBM MQ bereitgestellten Datendefinitionsdateien enthalten Folgendes:

- Definitionen aller IBM MQ-Konstanten und -Rückgabecodes
- Definitionen der IBM MQ-Strukturen und -Datentypen
- Konstantendefinitionen für die Initialisierung der Strukturen
- Funktionsprototypen für jeden Aufruf (nur für PL/I und C)

Eine ausführliche Beschreibung der IBM MQ-Datendefinitionsdateien finden Sie unter „IBM MQ-Datendefinitionsdateien" auf Seite 755.

Strukturen

Strukturen, die zusammen mit den unter „MQI-Aufrufe" auf Seite 760 aufgeführten MQI-Aufrufen verwendet werden, werden in Datendefinitionsdateien für jede unterstützte Programmiersprache bereitgestellt.

Eine Zusammenfassung der Strukturen finden Sie unter Strukturdatentypen .

IBM i z/OS IBM MQ for z/OS und IBM MQ for IBM i stellen Dateien bereit, die Konstanten enthalten, die Sie beim Ausfüllen einiger Felder dieser Strukturen verwenden können. Weitere Informationen hierzu finden Sie im Abschnitt IBM MQ-Datendefinitionen.

IBM MQ-Stubprogramme und -Bibliotheksdateien

In diesem Abschnitt sind die bereitgestellten Stubprogramme und Bibliotheksdateien für jede Plattform aufgeführt.

Weitere Informationen zur Verwendung von Stubprogrammen und Bibliotheksdateien beim Erstellen einer ausführbaren Anwendung finden Sie unter „Prozedurale Anwendung erstellen" auf Seite 1052. Informationen zum Herstellen einer Verknüpfung mit C++-Bibliotheksdateien finden Sie unter C++ verwenden *IBM MQC++ verwenden*.

AIX *IBM MQ for AIX-Bibliotheksdateien*

In IBM MQ for AIX müssen Sie Ihr Programm nicht nur mit den vom Betriebssystem bereitgestellten Bibliotheksdateien verknüpfen, sondern auch mit den MQI-Bibliotheksdateien, die für die Umgebung bereitgestellt sind, in der Ihre Anwendung ausgeführt wird.

Stellen Sie in einer Nicht-Thread-Anwendung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Tabelle 106. Bibliotheken für AIX-Anwendungen, bei denen es sich nicht um Thread-Anwendungen handelt

Bibliotheksdatei	**Umgebung**
libmqm.a	Server für C
libmqic.a und libmqm.a	Client für C
libmqmzf.a	Exits für installierbare Services für C
libmqmxa.a	Server-XA-Schnittstelle
libmqmxa64.a	Alternative Server-XA-Schnittstelle
libmqcxa.a	Client-XA-Schnittstelle
libmqcxa64.a	Alternative Client-XA-Schnittstelle
libmqmcbrt.o	IBM MQ-Laufzeitbibliothek für Micro Focus COBOL-Unterstützung
libmqmcb.a	Server für COBOL

Tabelle 106. Bibliotheken für AIX-Anwendungen, bei denen es sich nicht um Thread-Anwendungen handelt (Forts.)	
Bibliotheksdatei	**Umgebung**
libmqicb.a	Client für COBOL
libimqc23ia.a	Client für C++ (XLC 16)
libimqs23ia.a	Server für C++ (XLC 16)
V 9.4.0 libimqc23ca.a	Client für C++ (XLC 17)
V 9.4.0 libimqs23ca.a	Server für C++ (XLC 17)

V 9.4.0 Bibliotheken, die "ia" enthalten, wurden mit dem XLC 16-Compiler erstellt, während Bibliotheken mit "ca" im Namen mit dem XLC 17-Compiler erstellt wurden.

Stellen Sie in einer Thread-Anwendung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Tabelle 107. Bibliotheksdateien für AIX-Thread-Anwendungen. Eine Tabelle mit zwei Spalten mit einer Auflistung der Bibliotheksdateien und Angabe der jeweiligen Umgebung.	
Bibliotheksdatei	**Umgebung**
libmqm_r.a	Server für C
libmqic_r.a und libmqm_r.a	Client für C
libmqmzf_r.a	Exits für installierbare Services für C
libmqmxa_r.a	Server-XA-Schnittstelle
libmqmxa64_r.a	Alternative Server-XA-Schnittstelle
libmqcxa_r.a	Client-XA-Schnittstelle
libmqcxa64_r.a	Alternative Client-XA-Schnittstelle
libimqc23ia_r.a	Client für C++ (XLC 16)
libimqs23ia_r.a	Server für C++ (XLC 16)
V 9.4.0 libimqc23ca_r.a	Client für C++ (XLC 17)
V 9.4.0 libimqs23ca_r.a	Server für C++ (XLC 17)

V 9.4.0 Bibliotheken mit Namen, die ia enthalten, wurden mit dem XLC 16-Compiler erstellt, während Bibliotheken mit Namen, die ca enthalten, mit dem XLC 17-Compiler erstellt wurden.

Anmerkung: Verknüpfungen mit mehreren Bibliotheken sind nicht möglich. Sie können also beispielsweise nicht gleichzeitig eine Verknüpfung mit einer Bibliothek mit Threads und einer Bibliothek ohne Threads herstellen.

IBM i *IBM MQ for IBM i-Bibliotheksdateien*

Verknüpfen Sie in IBM MQ for IBM i Ihr Programm nicht nur mit den vom Betriebssystem bereitgestellten Bibliotheksdateien, sondern auch mit den MQI-Bibliotheksdateien, die für die Umgebung bereitgestellt sind, in der Ihre Anwendung ausgeführt wird.

Für Nicht-Thread-Anwendungen:

Tabelle 108. Bibliotheken für IBM i-Anwendungen, bei denen es sich nicht um Thread-Anwendungen handelt	
Bibliotheksdatei	**Umgebung**
LIBMQM	Ein Server-/Client-Service-Programm
LIBMQIC	Ein Client-Service-Programm
IMQB23I4	Ein C++-Basisserviceprogramm
IMQS23I4	Ein C++-Server-Service-Programm
LIBMQMZF	Installierbare Exits für die Programmiersprache C

In einer Thread-Anwendung:

Tabelle 109. Bibliotheksdateien für IBM i-Thread-Anwendungen	
Bibliotheksdatei	**Umgebung**
LIBMQM_R	Ein Server-&Client-Service-Programm
IMQB23I4_R	Ein C++-Basisserviceprogramm
IMQS23I4_R	Ein C++-Server-Service-Programm
LIBMQMZF_R	Installierbare Exits für die Programmiersprache C
LIBMQIC_R	Ein Client-Service-Programm

In IBM MQ for IBM i können Sie Ihre Anwendungen in C++ erstellen. Hinweise zur Verknüpfung Ihrer C++-Anwendungen sowie ausführliche Informationen zu allen Aspekten bei der Verwendung von C++ finden Sie unter C++ verwenden.

Linux *IBM MQ for Linux-Bibliotheksdateien*

In IBM MQ für Linux müssen Sie Ihr Programm nicht nur mit den vom Betriebssystem bereitgestellten Bibliotheksdateien verknüpfen, sondern auch mit den MQI-Bibliotheksdateien, die für die Umgebung bereitgestellt sind, in der Ihre Anwendung ausgeführt wird.

Stellen Sie in einer Nicht-Thread-Anwendung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Tabelle 110. Bibliotheken für Linux-Anwendungen, bei denen es sich nicht um Thread-Anwendungen handelt	
Bibliotheksdatei	**Umgebung**
libmqm.so	Server für C
libmqic.so und libmqm.so	Client für C
libmqmzf.so	Exits für installierbare Services für C
libmqmxa.so	Server-XA-Schnittstelle
libmqmxa64.so	Alternative Server-XA-Schnittstelle
libmqcxa.so	Client-XA-Schnittstelle
libmqcxa64.so	Alternative Client-XA-Schnittstelle
libimqc23gl.so	Client für C++
libimqs23gl.so	Server für C++

Stellen Sie in einer Thread-Anwendung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Tabelle 111. Bibliotheksdateien für Linux-Thread-Anwendungen	
Bibliotheksdatei	**Umgebung**
libmqm_r.so	Server für C
libmqic_r.so und libmqm_r.so	Client für C
libmqmzf_r.so	Exits für installierbare Services für C
libmqmxa_r.so	Server-XA-Schnittstelle
libmqmxa64_r.so	Alternative Server-XA-Schnittstelle
libmqcxa_r.so	Client-XA-Schnittstelle
libmqcxa64_r.so	Alternative Client-XA-Schnittstelle
libimqc23gl_r.so	Client für C++
libimqs23gl_r.so	Server für C++

Anmerkung: Verknüpfungen mit mehreren Bibliotheken sind nicht möglich. Sie können also beispielsweise nicht gleichzeitig eine Verknüpfung mit einer Bibliothek mit Threads und einer Bibliothek ohne Threads herstellen.

Windows *IBM MQ for Windows-Bibliotheksdateien*

In IBM MQ for Windows müssen Sie Ihr Programm nicht nur mit den vom Betriebssystem bereitgestellten Bibliotheksdateien verknüpfen, sondern auch mit den MQI-Bibliotheksdateien, die für die Umgebung bereitgestellt sind, in der Ihre Anwendung ausgeführt.

Tabelle 112. Bibliotheksdateien für Windows-Anwendungen	
Bibliotheksdatei	**Umgebung**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Server für C (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Client für C (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	Server-XA-Schnittstelle für C (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	Client-XA-Schnittstelle für C (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	Client-MTS für C (32-Bit)
MQ_INSTALLATI-ON_PATH\Tools\Lib\mqmcics4.lib32	Server-Unterstützung für TXSeries CICS für C (32-Bit)
MQ_INSTALLATI-ON_PATH\Tools\Lib\mqccics4.lib32	Client-Unterstützung für TXSeries CICS für C (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	Exits für installierbare Services für C (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Server für IBM COBOL (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Server für Micro Focus COBOL (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	Client für IBM COBOL (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Client für Micro Focus COBOL (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	Server für C++ (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	Client für C++ (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	Basis für C++ (32-Bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	Client-MTS für C++ (32-Bit)

Tabelle 112. Bibliotheksdateien für Windows-Anwendungen (Forts.)	
Bibliotheksdatei	**Umgebung**
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Server für C (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Client für C (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	Server-XA-Schnittstelle für C (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	Client-XA-Schnittstelle für C (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	Client-MTS für C (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	Server für IBM COBOL (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Server für Micro Focus COBOL (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	Client für IBM COBOL (64-Bit)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Client für Micro Focus COBOL (64-Bit)
MQ_INSTALLATI-ON_PATH\Tools\Lib64\imqs23vn.lib	Server für C++ (64-Bit)
MQ_INSTALLATI-ON_PATH\Tools\Lib64\imqc23vn.lib	Client für C++ (64-Bit)
MQ_INSTALLATI-ON_PATH\Tools\Lib64\imqb23vn.lib	Basis für C++ (64-Bit)
MQ_INSTALLATI-ON_PATH\Tools\Lib64\imqx23vn.lib	Client-MTS für C++ (64-Bit)

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Verwenden Sie `amqmdnet.dll` zum Kompilieren von .NET-Programmen. Weitere Informationen finden Sie unter „IBM MQ .NET-Programme kompilieren” auf Seite 642 im Abschnitt „.NET-Anwendungen entwickeln” auf Seite 582.

Die folgenden Dateien werden aus Gründen der Kompatibilität mit früheren Releases bereitgestellt:

```
mqic32.lib
mqic32xa.lib
```

z/OS *IBM MQ for z/OS stub programs*

Before you can run a program written with IBM MQ for z/OS, you must link-edit it to the stub program supplied with IBM MQ for z/OS for the environment in which you are running the application.

The stub program provides the first stage of the processing of your calls into requests that IBM MQ for z/OS can process.

IBM MQ for z/OS supplies the following stub programs:

CSQBSTUB
Stub program for z/OS batch programs

CSQBRRSI
Stub program for z/OS batch programs using RRS by way of the MQI

CSQBRSTB
Stub program for z/OS batch programs using RRS directly

CSQCSTUB
Stub program for CICS programs

CSQQSTUB
Stub program for IMS programs

CSQXSTUB
Stub program for distributed queuing non-CICS exits

CSQASTUB
Stub program for data-conversion exits

Attention: If you use a stub program other than one listed for a specific environment, it might have unpredictable results.

Note: If you use the CSQBRSTB stub program, link-edit with ATRSCSS from SYS1.CSSLIB. (SYS1.CSSLIB is also known as the *Callable Services Library*). For more information about RRS see "Transaction management and recoverable resource manager services" on page 905.

Alternatively, you can dynamically call the stub from within your program. This technique is described in "Dynamically calling the IBM MQ stub" on page 1082.

In IMS, you might also need to use a special language interface module that is supplied by IBM MQ.

Do not run applications that are link-edited with CSQBSTUB and CSQQSTUB in the same IMS MPP region. This can cause problems such as DFS3607I or CSQQ005E messages. The first MQCONN call in an address space determines which interface is used, therefore CSQQSTUB and CSQBSTUB transactions must run in different IMS message regions.

Parameter, die in allen Aufrufen verwendet werden
Es gibt zwei Parametertypen, die in allen Aufrufen verwendet werden: Kennungen und Rückgabecodes.

Verwendung von Kennungen

In allen MQI-Aufrufen werden eine oder auch mehrere *Kennungen* verwendet. Diese geben je nach Aufruf den Warteschlangenmanager, die Warteschlange oder ein anderes Objekt, die Nachricht oder die Subskription an.

Für Programme, die mit einem Warteschlangenmanager kommunizieren, ist eine eindeutige Kennung erforderlich, über die siediesen Warteschlangenmanager erkennen können. Diese Art Kennung ist eine *Verbindungskennung*, manchmal auch als *Hconn* bezeichnet. Bei CICS-Programmen ist die Verbindungskennung immer null. Bei allen anderen Plattformen oder Programmarten wird die Verbindungskennung vom MQCONN- oder MQCONNX-Aufruf zurückgegeben, wenn das Programm eine Verbindung zum Warteschlangenmanager herstellt. Bei Verwendung anderer Aufrufe übergeben Programme die Verbindungskennung als Eingabeparameter.

Damit ein Programm mit einem IBM MQ-Objekt arbeiten kann, benötigt das Programm eine eindeutige Kennung, über die es dieses Objekt erkennen kann. Diese Art Kennung ist eine *Objektkennung*, manchmal auch als *Hobj* bezeichnet. Diese Kennung wird vom MQOPEN-Aufruf zurückgegeben, wenn das Programm das Objekt öffnet, um damit zu arbeiten. Bei der Verwendung nachfolgender MQPUT-, MQGET-, MQINQ-, MQSET- oder MQCLOSE-Aufrufe übergibt das Programm die Objektkennung als Eingabeparameter.

Ebenso gibt der MQSUB-Aufruf eine *Subskriptionskennung* (oder *Hsub*) zurück, mit der in nachfolgenden MQGET-, MQCB- oder MQSUBRQ-Aufrufen die Subskription gekennzeichnet wird, und bestimmte Aufrufe, die Nachrichteneigenschaften verarbeiten, verwenden eine *Nachrichtenkennung* (oder *Hmsg*).

Rückgabecodes

Von jedem Aufruf werden als Ausgabeparameter ein Beendigungs- und ein Ursachencode zurückgegeben. Bei beiden handelt es sich um sogenannte *Rückgabecodes*.

Aufrufe geben nach ihrer Ausführung einen *Beendigungscode* zurück, der angibt, ob der Aufruf erfolgreich war. Bei diesem Beendigungscode handelt es sich in der Regel um MQCC_OK (der Aufruf war erfolgreich) oder um MQCC_FAILED (der Aufruf ist fehlgeschlagen). Einige Aufrufe geben einen temporären Status (MQCC_WARNING) zurück, der auf eine nur teilweise erfolgreiche Ausführung hindeutet.

Jeder Aufruf gibt darüber hinaus einen *Ursachencode* zurück, der die Ursache für ein Fehlschlagen oder für eine nur teilweise erfolgreiche Ausführung angibt. Es gibt eine Vielzahl von Ursachencodes, die eine Vielzahl von möglichen Ursachen abdecken: dass beispielsweise die Warteschlange voll ist, für

eine Warteschlange keine Get-Operationen zugelassen sind oder eine bestimmte Warteschlange für den Warteschlangenmanager nicht definiert ist. Programme können anhand des Ursachencodes die weitere Vorgehensweise festlegen. Sie können beispielsweise die Benutzer auffordern, ihre Eingaben zu ändern oder einen Aufruf zu wiederholen, oder sie können eine Fehlernachricht an den Benutzer zurückgeben.

Wenn MQCC_OK als Beendigungscode zurückgegeben wird, ist der Ursachencode immer MQRC_NONE.

Die Beendigungs- und Ursachencodes für die einzelnen Aufrufe werden zusammen mit einer Beschreibung des Aufrufs aufgeführt. Wählen Sie dazu in der Liste unter Aufrufbeschreibungen den Aufruf aus, zu dem Sie nähere Informationen wünschen.

Ausführlichere Informationen, einschließlich Vorschlägen für Korrekturmaßnahmen, finden Sie in folgenden Abschnitten:

- z/OS IBM MQ for z/OS -Nachrichten, -Beendigungscodes und -Ursachencodes fürIBM MQ for z/OS
- Nachrichten und Ursachencodes für alle anderen IBM MQ-Plattformen

Puffer angeben

Der Warteschlangenmanager verweist nur bei Bedarf auf Puffer. Ist für einen Aufruf kein Puffer erforderlich oder hat der Puffer eine Länge von null, können Sie einen Nullzeiger auf einen Puffer verwenden.

Zur Größenangabe des Puffers sollte immer die Datenlänge verwendet werden.

Soll in einem Puffer die Ausgabe eines Aufrufs gespeichert werden (beispielsweise die Nachrichtendaten für einen MQGET-Aufruf oder die mit dem MQINQ-Aufruf abgerufenen Attributwerte), wird der Warteschlangenmanager versuchen, einen Ursachencode zurückzugeben, wenn der angegebene Puffer ungültig ist oder es sich bei ihm um einen Nur-Lese-Speicher handelt. Der Warteschlangenmanager wird jedoch unter Umständen nicht immer in der Lage sein, einen Ursachencode zurückzugeben.

z/OS *z/OS batch considerations*

z/OS batch programs that call the MQI can be in either supervisor or problem state.

However, they must meet the following conditions:

- They must be in task mode, not service request block (SRB) mode.
- They must be in Primary address space control (ASC) mode (not Access Register ASC mode).
- They must not be in cross-memory mode. The primary address space number (ASN) must be equal to the secondary ASN and the home ASN.
- They must not be used as MPF exit programs.
- No z/OS locks can be held.
- There can be no function recovery routines (FRRs) on the FRR stack.
- Any program status word (PSW) key can be in force for the MQCONN or MQCONNX call (provided the key is compatible with using storage that is in the TCB key), but subsequent calls that use the connection handle returned by MQCONN or MQCONNX:
  - Must have the same PSW key that was used on the MQCONN or MQCONNX call
  - Must have parameters accessible (for write, where appropriate) under the same PSW key
  - Must be issued under the same task (TCB), but not in any subtask of the task
- They can be in either 24-bit or 31-bit addressing mode. However, if 24-bit addressing mode is in force, parameter addresses must be interpreted as valid 31-bit addresses.

If any of these conditions is not met, a program check might occur. In some cases the call will fail and a reason code will be returned.

Linux AIX *Überlegungen zu AIX and Linux*

Bei der Entwicklung von AIX and Linux-Anwendungen sollten Sie Folgendes beachten.

Linux AIX *Systemaufruf 'fork' unter AIX and Linux*

Bei der Verwendung des Systemaufrufs 'fork' in IBM MQ- Anwendungen sollten Sie Folgendes beachten.

Wenn Ihre Anwendung `fork`verwenden möchte, muss der übergeordnete Prozess dieser Anwendung `fork` aufrufen, bevor IBM MQ aufgerufen wird, z. B. `MQCONN`, oder ein IBM MQ -Objekt mit **ImqQueueManager**erstellen.

Wenn Ihre Anwendung nach Ausgabe eines IBM MQ-Aufrufs einen untergeordneten Prozess erstellen möchte, muss der Anwendungscode eine `fork()`-Funktion mit `exec()` verwenden, damit es sich bei dem untergeordneten Prozess um eine neue Instanz, nicht um eine exakte Kopie des übergeordneten Prozesses handelt.

Verwendet Ihre Anwendung keine `exec()`-Funktion, wird von dem innerhalb des untergeordneten Prozesses ausgegebenen IBM MQ-API-Aufruf der Fehler MQRC_ENVIRONMENT_ERROR zurückgegeben.

Linux AIX *Signalverarbeitung unter AIX and Linux*

Bei AIX and Linux-Systemen wurde insgesamt von einer Nicht-Thread-Prozessumgebung zu einer Multithread-Umgebung übergegangen. Signale und die Verarbeitung von Signalen werden in der Multithread-Umgebung zwar unterstützt, sie sind jedoch nicht unbedingt für die Multithread-Umgebung geeignet und unterliegen einige Einschränkungen.

Bei AIX and Linux-Systemen wurde insgesamt von einer Nicht-Thread-Prozessumgebung zu einer Multithread-Umgebung übergegangen. In der Nicht-Thread-Umgebung konnten einige Funktionen nur durch die Verwendung von Signalen implementiert werden, obwohl für die meisten Anwendungen die Erkennung von Signalen und deren Verarbeitung nicht erforderlich war. In der Multithread-Umgebung unterstützen Basiselemente einige der Funktionen, die in Nicht-Thread-Umgebungen unter Verwendung von Signalen implementiert wurden.

Signale und die Verarbeitung von Signalen werden in der Multithread-Umgebung zwar unterstützt, sie sind jedoch nicht unbedingt für die Multithread-Umgebung geeignet und unterliegen einige Einschränkungen. Bei der Integration von Anwendungscode mit unterschiedlichen Middleware-Bibliotheken (die als Teil der Anwendung ausgeführt werden) in eine Multithread-Umgebung, von der jede versucht, Signale zu verarbeiten, kann dies zu Problemen führen. Traditionell werden Signalhandler (die pro Prozess definiert werden) gespeichert und wiederhergestellt; bei nur einem Ausführungsthread in einem Prozess war dies möglich, in einer Multithread-Umgebung ist dies jedoch nicht möglich. Dies liegt daran, dass eine Vielzahl von Ausführungsthreads versuchen könnten, eine prozessübergreifende Ressource zu speichern und wiederherzustellen, was ein unvorhersehbares Ergebnis zur Folge haben könnte.

Linux AIX *Nicht-Thread-Anwendungen*

Jede MQI-Funktion definiert ihren eigenen Signalhandler für die Signale. Die Handler der Benutzer für diese Signale werden für die Dauer des MQI-Funktionsaufrufs ersetzt. Andere Signale können auf dem üblichen Weg über benutzerdefinierte Handler abgefangen werden.

Jede MQI-Funktion definiert ihren eigenen Signalhandler für diese Signale:

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

Die Handler der Benutzer für diese Signale werden für die Dauer des MQI-Funktionsaufrufs ersetzt. Andere Signale können auf dem üblichen Weg über benutzerdefinierte Handler abgefangen werden. Wenn Sie keinen Handler installieren, werden die Standardaktionen (beispielsweise 'ignore' (Ignorieren), 'core dump' (Kernspeicherauszug) oder 'exit' (Beenden)) übernommen.

Nach der Verarbeitung eines synchronen Signals (SIGSEGV, SIGBUS, SIGFPE, SIGILL) versucht IBM MQ, das Signal vor dem Ausgeben des MQI-Funktionsaufrufs an einen registrierten Signalhandler zu übergeben.

Linux AIX *Thread-Anwendungen*

Ein Thread bleibt von einem MQCONN-Aufruf (oder einem MQCONNX-Aufruf) bis zu einem MQDISC-Aufruf mit IBM MQ verbunden.

Synchrone Signale

Synchrone Signale werden in einem bestimmten Thread ausgegeben.

AIX and Linux-Systeme ermöglichen die sichere Definition eines Signalhandlers für solche Signale für den gesamten Prozess. Für die folgenden Signale definiert IBM MQ jedoch eigene Signalhandler im Anwendungsprozess, solange ein Thread mit IBM MQ verbunden ist:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

Wenn Sie Multithread-Anwendungen erstellen, gibt es für jedes Signal nur einen prozessübergreifenden Signalhandler. Wenn IBM MQ eigene Handler für synchrone Signale definiert, werden für jedes Signal alle zuvor registrierten Handler gespeichert. Nachdem IBM MQ eines der aufgeführten Signale verarbeitet hat, versucht IBM MQ den Signalhandler aufzurufen, der bei der ersten IBM MQ-Verbindung innerhalb des Prozesses gerade aktiv war. Nachdem die Verbindung aller Annwendungsthreads zu IBM MQ wieder getrennt wurde, werden die zuvor registrierten Handler wiederhergestellt.

Da Signalhandler von IBM MQ gespeichert und wiederhergestellt werden, dürfen die Anwendungsthreads keine Signalhandler für diese Signal definieren, solange noch die Möglichkeit besteht, dass ein anderer Thread desselben Prozesses noch mit IBM MQ verbunden ist.

Anmerkung: Wenn eine Anwendung oder eine Middleware-Bibliothek (die als Teil einer Anwendung ausgeführt wird) einen Signalhandler definiert, während ein Thread mit IBM MQ verbunden ist, muss der Signalhandler der Anwendung bei der Verarbeitung des Signals den entsprechenden IBM MQ-Handler aufrufen.

Beim Definieren und Herstellen von Signalhandlern gilt allgemein, dass der Signalhandler, der zuletzt gespeichert wurde, auch zuerst wiederhergestellt werden muss:

- Wenn eine Anwendung nach Herstellen einer Verbindung zu IBM MQ einen Signalhandler definiert, muss der vorherige Signalhandler wiederhergestellt werden, bevor die Anwendung die Verbindung zu IBM MQ trennt.
- Wenn eine Anwendung vor dem Herstellen einer Verbindung zu IBM MQ einen Signalhandler definiert, muss die Verbindung der Anwendung zu IBM MQ getrennt werden, bevor der Signalhandler der Anwendung wiederhergestellt wird.

Anmerkung: Wird diese Regel (dass der zuletzt gespeicherte Signalhandler auch zuerst wiederhergestellt werden muss) nicht eingehalten, kann dies in der Anwendung zu einer unerwarteten Signalverarbeitung und möglicherweise zu einem Verlust von Signalen in der Anwendung führen.

Asynchrone Signale

IBM MQ verwendet asynchrone Signale in Thread-Anwendungen nur, wenn es sich um Clientanwendungen handelt.

Zusätzliche Hinweise zu Thread-Client-Anwendungen

IBM MQ verarbeitet während einer Ein-/Ausgabe-Operation in einem Server die folgenden Signale. Diese Signale werden vom Kommunikationsstack definiert. Die Anwendung darf für diese Signale keine Signalhandler definieren, solange ein Thread mit einem Warteschlangenmanager verbunden ist:

SIGPIPE (für TCP/IP)

Linux AIX *Zusätzliche Informationen bei Verwendung der AIX and Linux-Signalverarbeitung in MQI*

Bei der Verwendung von MQI für die Signalverarbeitung unter AIX and Linux gibt es zusätzliche Aspekte zu Fastpath-Anwendungen, MQI-Funktionsaufrufen in Signalhandlern, Signalen während MQI-Aufrufen, Benutzerexits und und installierbaren Services un VMS-Exit-Handlern zu berücksichtigen.

Fastpath-Anwendungen (gesicherte Anwendungen)

Fastpath-Anwendungen werden in demselben Prozess wie IBM MQ und damit in der Multithread-Umgebung ausgeführt.

In dieser Umgebung werden die synchronen Signale SIGSEGV, SIGBUS, SIGFPE und SIGILL von IBM MQ verarbeitet; alle anderen Signale dürfen nicht an die Fastpath-Anwendung übergeben werden, solange sie mit IBM MQ verbunden ist. Diese Signale müssen blockiert oder aber von der Anwendung verarbeitet werden. Wenn ein Fastpath-Anwendung ein solches Ereignis abfängt, muss der Warteschlangenmanager gestoppt und anschließend erneut gestartet werden, da er andernfalls unter Umständen in einem undefinierten Status verharrt. Ein vollständige Liste der Einschränkungen für Fastpath-Anwendungen bei MQCONNX finden Sie unter „Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen” auf Seite 775.

MQI-Funktionsaufrufe in Signalhandlern

Innerhalb eines Signalhandlers darf keine MQI-Funktion aufgerufen werden.

Wenn versucht wird, eine MQI-Funktion aus einem Signalhandler aufzurufen, solange eine andere MQI-Funktion noch aktiv ist, wird die Fehlernachricht MQRC_CALL_IN_PROGRESS zurückgegeben. Der Versuch, eine MQI-Funktion aus einem Signalhandler aufzurufen, wenn keine andere MQI-Funktion aktiv ist, wird wahrscheinlich aufgrund betriebssystemspezifischer Einschränkungen während des Vorgangs fehlschlagen, da aus oder in einem Handler nur bestimmte Aufrufe ausgegeben werden dürfen.

Bei C++-Destruktormethoden, die unter Umständen automatisch während der Programmbeendigung aufgerufen werden, kann der Aufruf von MQI-Funktionen wahrscheinlich nicht verhindert werden. Ignorieren Sie die eventuell zurückgegebenen Fehlernachrichten des Typs MQRC_CALL_IN_PROGRESS. Wenn ein Signalhandler die Funktion 'exit()' aufruft, setzt IBM MQ nicht festgeschriebene Nachrichten ganz normal unter Synchronisationspunktsteuerung zurück und schließt alle offenen Warteschlangen.

Signale während MQI-Aufrufen

MQI-Funktionen geben keinen EINTR-Fehlercode oder ähnlichen Code an die Anwendungsprogramme zurück.

Wenn während eines MQI-Aufrufs ein Signal auftritt und der Handler *return* aufruft, wird der Aufruf so fortgesetzt, als ob das Signal nicht aufgetreten wäre. Insbesondere MQGET-Aufrufe können nicht durch ein Signal unterbrochen werden, das besagt, dass die Steuerung unverzüglich an die Anwendung zurückgegeben werden soll. Wenn es möglich sein soll, einen MQGET-Aufruf zu unterbrechen, müssen Sie die Warteschlange auf GET_DISABLED setzen; Sie können auch eine Schleife um einen MQGET-Aufruf mit einer endlichen Zeitlimitüberschreitung verwenden (Option MQGMO_WAIT mit gesetztem Feld 'gmo.WaitInterval') und in einer Nicht-Thread-Umgebung mit Ihrem Signalhandler oder in einer Threadumgebung mit einer entsprechenden Funktion ein Flag setzen, das die Schleife unterbricht.

AIX In der AIX-Umgebung müssen für IBM MQ Systemaufrufe, die durch Signale unterbrochen wurden, erneut gestartet werden. Wenn Sie eigene Signalhandler mit 'sigaction(2)' definieren, müssen Sie das Flag SA_RESTART im Feld 'sa_flags' der neuen Aktionsstruktur setzen, da IBM MQ andernfalls Aufrufe, die durch eine Signal unterbrochen wurden, unter Umständen nicht abschließen kann.

Benutzerexits und installierbare Services

Benutzerexits und installierbare Services, die als Teil eines IBM MQ-Prozesses in einer Multithread-Umgebung ausgeführt werden, unterliegen denselben Einschränkungen wie Fastpath-Anwendungen. Sie sind

permanent mit IBM MQ verbunden und verwenden daher keine Signale und keine Betriebssystemaufrufe, die nicht threadsicher sind.

Verbindung zu einem Warteschlangenmanager herstellen und trennen

Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

Die Art und Weise, in der diese Verbindung hergestellt wird, hängt von der Plattform und der Umgebung ab, in der das Programm ausgeführt wird:

Multi **IBM MQ for Multiplatforms**
Programme, die in diesen Umgebungen ausgeführt werden, können mit dem MQI-Aufruf MQCONN eine Verbindung zum Warteschlangenmanager herstellen und mit dem MQDISC-Aufruf die Verbindung zum Warteschlangenmanager trennen. Alternativ können Programme auch den MQCONNX-Aufruf verwenden.

z/OS **IBM MQ for z/OS-Stapel**
Programme, die in dieser Umgebung ausgeführt werden, können mit dem MQI-Aufruf MQCONN eine Verbindung zum Warteschlangenmanager herstellen und mit dem MQDISC-Aufruf die Verbindung zum Warteschlangenmanager trennen. Alternativ können Programme auch den MQCONNX-Aufruf verwenden.

z/OS-Stapelprogramme können nacheinander oder gleichzeitig eine Verbindung zu mehreren Warteschlangenmanagern innerhalb desselben TCB herstellen.

z/OS **IMS**
Die IMS-Steuerregion ist beim Start mit einem oder mehreren Warteschlangenmanagern verbunden. Diese Verbindung wird über IMS-Befehle gesteuert. Informationen zur Steuerung des IMS-Adapters von z/OS finden Sie im Abschnitt IBM MQ for z/OS verwalten. Programmierer, die IMS-Programme für Message-Queuing erstellen, müssen jedoch über den MQI-Aufruf MQCONN den Warteschlangenmanager angeben, zu dem eine Verbindung hergestellt werden soll. Mit dem MQDISC-Aufruf können sie die Verbindung zu diesem Warteschlangenmanager wieder trennen.

Nach einem IMS-Aufruf, der einen Synchronisationspunkt erstellt, und vor der Verarbeitung von Nachrichten anderer Benutzer stellt der IMS-Adapter sicher, dass die Anwendung die Kennungen schließt und die Verbindung zum Warteschlangenmanager trennt. Siehe „Syncpoints in IMS applications” auf Seite 904.

IMS-Programme können nacheinander oder gleichzeitig eine Verbindung zu mehreren Warteschlangenmanagern innerhalb desselben TCB herstellen.

z/OS **CICS Transaction Server für z/OS**
CICS-Programme müssen nichts unternehmen, um eine Verbindung zum Warteschlangenmanager herzustellen, da das CICS-System selbst verbunden ist. Diese Verbindung wird in der Regel bei der Initialisierung automatisch hergestellt, Sie können jedoch auch die mit IBM MQ for z/OS bereitgestellte CKQC-Transaktion verwenden. Weitere Informationen zu CKQC finden Sie im Abschnitt IBM MQ for z/OS verwalten.

CICS-Tasks können nur zu dem Warteschlangenmanager eine Verbindung herstellen, mit dem die CICS-Region verbunden ist.

CICS-Programme können auch die entsprechenden MQI-Aufrufe (MQCONN und MQDISC) verwenden, um eine Verbindung herzustellen bzw. zu trennen. Unter Umständen möchten Sie davon Gebrauch machen, damit diese Anwendungen mit möglichst wenig Änderungen am Programmcode in Nicht-CICS-Umgebungen portiert werden können. Diese Aufrufe werden jedoch *immer* in einer CICS -Umgebung erfolgreich ausgeführt. d. h., der Rückgabecode gibt nicht unbedingt Aufschluss über den tatsächlichen Status der Verbindung zum Warteschlangenmanager.

TXSeries für Windows und Open Systems
Diese Programme müssen nichts unternehmen, um eine Verbindung zum Warteschlangenmanager herzustellen, da das CICS-System selbst verbunden ist. Daher wird immer nur jeweils eine Verbindung

unterstützt. CICS-Anwendungen müssen einen MQCONN-Aufruf absetzen, um eine Verbindungskennung abzurufen, und vor ihrer Beendigung einen MQDISC-Aufruf ausgeben.

Unter den folgenden Links finden Sie weitere Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager:

- „Verbindung zu einem Warteschlangenmanager über den MQCONN-Aufruf herstellen” auf Seite 774
- „Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen” auf Seite 775
- „Programme mit MQDISC von einem Warteschlangenmanager trennen” auf Seite 780

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Objekte öffnen und schließen” auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS” auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS” auf Seite 73
This information helps you to write IMS applications using IBM MQ.

Verbindung zu einem Warteschlangenmanager über den MQCONN-Aufruf herstellen

Dieser Abschnitt enthält Informationen zum Herstellen einer Verbindung zu einem Warteschlangenmanager unter Verwendung des MQCONN-Aufrufs.

Sie können entweder eine Verbindung zu einem bestimmten Warteschlangenmanager oder aber zum Standardwarteschlangenmanager herstellen:

- z/OS Bei IBM MQ for z/OS in der Stapelumgebung wird der Standardwarteschlangenmanager im Modul CSQBDEFV angegeben.
- Multi Für IBM MQ for Multiplatforms ist der Standardwarteschlangenmanager in der Datei mqs.ini angegeben.

z/OS Alternativ können Sie in den MVS-Stapelumgebungen sowie in den TSO- und RRS-Umgebungen von z/OS eine Verbindung zu jedem Warteschlangenmanager innerhalb einer Gruppe mit gemeinsamer Warteschlange herstellen. Die MQCONN- oder MQCONNX-Anforderung wählt ein aktives Mitglied dieser Gruppe aus.

Wenn Sie eine Verbindung zu einem Warteschlangenmanager herstellen, muss dieser lokal für die Task sein, d. h., er muss zu demselben System wie die IBM MQ-Anwendung gehören.

z/OS In der IMS-Umgebung muss der Warteschlangenmanager mit der IMS-Steuerregion verbunden sein und mit der abhängigen Region, die vom Programm verwendet wird. Der Warteschlangenmanager wird bei der Installation von IBM MQ for z/OS im Modul CSQQDEFV angegeben.

Für die TXSeries CICS-Umgebung sowie für TXSeries für Windows und AIX muss der Warteschlangenmanager als XA-Ressource in CICS definiert werden.

Eine Verbindung zum Standardwarteschlangenmanager wird mit dem MQCONN-Aufruf hergestellt, indem ein Name eingegeben wird, der aus Leerzeichen besteht oder mit einem Nullzeichen (X'00') beginnt.

Eine Anwendung muss über eine entsprechende Berechtigung verfügen, damit sie erfolgreich eine Verbindung zu einem Warteschlangenmanager herstellen kann. Weitere Informationen hierzu finden Sie im Abschnitt Sicherheit.

Der MQCONN-Aufruf hat die folgende Ausgabe:

- Eine Verbindungskennung (**Hconn**)
- Einen Beendigungscode
- Einen Ursachencode

Die Verbindungskennung wird in nachfolgenden MQI-Aufrufen verwendet.

Wenn aus dem Ursachencode hervorgeht, dass die Anwendung bereits mit dem Warteschlangenmanager verbunden ist, ist die zurückgegebene Verbindungskennung mit der identisch, die zurückgegeben wurde, als die Anwendung zum ersten Mal verbunden wurde. Die Anwendung darf in diesem Fall keinen MQDISC-Ausruf ausgeben, da die aufrufende Anwendung weiterhin verbunden bleiben soll.

Die Verbindungskennung hat denselben Gültigkeitsbereich wie die Objektkennung (siehe „Objekte mit dem MQOPEN-Aufruf öffnen” auf Seite 782).

Eine Beschreibung der Parameter finden Sie in der Beschreibung des MQCONN-Aufrufs unter MQCONN.

Wenn der Warteschlangenmanager bei der Ausgabe eines MQCONN-Aufrufs im Quiescemodus ist oder gerade heruntergefahren wird, schlägt der Aufruf fehl.

Gültigkeitsbereich des MQCONN- bzw. MQCONNX-Aufrufs

Der Gültigkeitsbereich eines MQCONN- oder MQCONNX-Aufrufs ist in der Regel der Thread, von dem er ausgegeben wurde. d. h., die vom Aufruf zurückgegebene Verbindungskennung gilt nur innerhalb des Threads, von dem der Aufruf ausgegeben wurde. Es kann nur jeweils ein Aufruf unter Angabe der Kennung ausgegeben werden. Wird sie von einem anderen Thread aus verwendet, wird sie als ungültig abgelehnt. Wenn in Ihrer Anwendung mehrere Threads vorhanden sind, die alle IBM MQ-Aufrufe verwenden sollen, muss der MQCONN- oder MQCONNX-Aufruf von jedem dieser Threads ausgegeben werden.

Wenn ein Prozess mehrere MQCONN-Aufrufe ausgibt, muss nicht jeder Aufruf an denselben Warteschlangenmanager abgesetzt. Es kann jedoch nur jeweils eine IBM MQ-Verbindung von einem Thread hergestellt werden. Alternativ können Sie „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX” auf Seite 777 in Betracht ziehen, um die Verwendung mehrerer IBM MQ -Verbindungen von einem einzelnen Thread und einer IBM MQ -Verbindung von einem beliebigen Thread zuzulassen.[7]

Wenn Ihre Anwendung als Client ausgeführt wird, kann sie innerhalb eines Threads eine Verbindung zu mehreren Warteschlangenmanagern herstellen.

Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen

Der MQCONNX-Aufruf ähnelt dem MQCONN-Aufruf, schließt allerdings Optionen ein, um die Aufrufe zu steuern.

[7] Bei Verwendung von Multithread-Anwendungen im Zusammenhang mit IBM MQ for AIX or Linux-Systemen müssen Sie sicherstellen, dass für die Anwendungen eine ausreichende Stackgröße für die Threads definiert wurde. Wenn Multithread-Anwendungen MQI-Aufrufe entweder selbst oder über andere Signalhandler (beispielsweise CICS) ausgeben, sollten Sie eine Stackgröße von 256 KB oder mehr verwenden.

Als Eingabe für MQCONNX können Sie den Namen eines Warteschlangenmanagers z/OS bzw. bei z/OS-Systemen mit gemeinsam genutzter Warteschlange den Namen einer Gruppe mit gemeinsamer Warteschlange angeben. Die Optionen zur Steuerung, wie die Verbindung zum WS-Manager hergestellt wird, werden in einer Struktur bereitgestellt, die als MQCNObezeichnet wird.

Die Ausgabe von MQCONNX enthält folgende Komponenten:

- Eine Verbindungskennung (Hconn)
- Einen Beendigungscode
- Einen Ursachencode

Die Verbindungskennung verwenden Sie für nachfolgende MQI-Aufrufe.

Die Verbindungsoptionen, die im Feld *Options* der MQCNO-Struktur festgelegt sind, ermöglichen die Steuerung mehrerer Attribute der Verbindung. Besonders hervorzuheben sind die folgenden Optionsgruppen:

- Mit den Bindungsoptionen können *vertrauenswürdige Anwendungen* erstellt werden. Vertrauenswürdige Anwendungen implizieren, dass die IBM MQ -Anwendung und der lokale Warteschlangenmanageragent denselben Prozess durchlaufen. Da der Agentenprozess auf den Warteschlangenmanager nicht mehr über eine Schnittstelle zugreifen muss, werden diese Anwendungen zu einer Erweiterung des Warteschlangenmanagers. Dieses Verhalten wird durch Angabe der Option MQCNO_FASTPATH_BINDING angefordert. Weitere Informationen zu den Einschränkungen, die für vertrauenswürdige Anwendungen gelten, finden Sie unter „Einschränkungen für vertrauenswürdige Anwendungen” auf Seite 776.
- Die Optionen für die gemeinsame Nutzung von Handles ermöglichen die Erstellung gemeinsam genutzter Verbindungen. Gemeinsam genutzte Verbindungen können Kennungen zwischen verschiedenen Threads innerhalb desselben Prozesses gemeinsam nutzen. Weitere Informationen zu gemeinsam genutzten Verbindungen finden Sie unter „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX” auf Seite 777.

Mit MQCNO kann die Anwendung auch steuern, wie die Verbindung zum Warteschlangenmanager authentifiziert wird. Authentifizierungsnachweise können in einer MQCSP-Struktur angegeben werden, auf die von der MQCNO-Struktur verwiesen wird.

Eine vollständige Beschreibung der Parameter für den MQCONNX-Aufruf und der Verbindungsattribute, die gesteuert werden können, finden Sie im Abschnitt MQCONNX-Connect-Warteschlangenmanager (erweitert).

Einschränkungen für vertrauenswürdige Anwendungen
Einschränkungen, die für vertrauenswürdige Anwendungen gelten Einige Einschränkungen gelten für alle Plattformen und andere sind plattformspezifisch.

T

- Vertrauenswürdige Anwendungen müssen explizit vom Warteschlangenmanager getrennt werden.
- Sie müssen vertrauenswürdige Anwendungen stoppen, bevor Sie den Warteschlangenmanager mit dem Befehl **endmqm** beenden.
- Mit MQCNO_FASTPATH_BINDING dürfen keine asynchronen Signale und Zeitgeberinterrupts verwendet werden (wie `sigkill`).
- Ein Thread innerhalb einer vertrauenswürdigen Anwendung kann keine Verbindung zu einem Warteschlangenmanager herstellen, so lange ein anderer Thread innerhalb des gleichen Prozesses mit einem anderen Warteschlangenmanager verbunden ist (dies gilt für alle Plattformen).
- Linux AIX Auf AIX and Linux-Systemen müssen Sie für alle MQI-Aufrufe 'mqm' als effektive Benutzer-ID und Gruppen-ID verwenden. Sie können diese IDs vor einem Nicht-MQI-Aufruf, der eine Authentifizierung erfordert, ändern (z. B. vor dem Öffnen einer Datei), müssen sie aber vor dem nächsten MQI-Aufruf wieder auf 'mqm' zurücksetzen.
- IBM i Unter IBM i:

1. Vertrauenswürdige Anwendungen müssen unter dem QMQM-Benutzerprofil ausgeführt werden. Die Zugehörigkeit des Benutzerprofils zur QMQM-Gruppe oder die QMQM-Berechtigung der Anwendung allein ist nicht ausreichend. Eventuell kann das QMQM-Benutzerprofil nicht zur Anmeldung bei interaktiven Jobs verwendet werden oder nicht in den Jobbeschreibungen von Jobs angegeben werden, die vertrauenswürdige Anwendungen ausführen. In diesem Fall besteht eine Möglichkeit darin, die Funktionen der IBM i Profile Swapping API, QSYGETPH, QWTSETP und QSYRLSPH zu verwenden, um den aktuellen Benutzer des Jobs vorübergehend in QMQM zu ändern, während die IBM MQ -Programme ausgeführt werden. Ausführliche Informationen zu diesen Funktionen sowie ein Beispiel für ihre Verwendung finden Sie im Abschnitt Sicherheits-APIs der Dokumentation zu IBM i-*Anwendungsprogrammierschnittstellen* .
2. Vertrauenswürdige Anwendungen dürfen nicht mit Systemanfrageoption 2 oder durch Beenden des zugehörigen Jobs mit ENDJOB abgebrochen werden.

- ALW Auf AIX, Linux, and Windows-Systemen werden vertrauenswürdige 32-Bit-Anwendungen nicht unterstützt. Beim Versuch, eine vertrauenswürdige 32-Bit-Anwendung auszuführen, wird diese auf eine Standard-Binding-Verbindung herabgestuft.

Multi *Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX*

Dieser Abschnitt enthält Informationen zu gemeinsam genutzten Verbindungen mit MQCONNX und einige zu beachtende Hinweise zur Verwendung.

Anmerkung: z/OS Nicht unterstützt unter IBM MQ for z/OS.

Auf Multiplatforms ist eine Verbindung mit MQCONN nur für den Thread verfügbar, der die Verbindung hergestellt hat. Die Optionen des MQCONNX-Aufrufs ermöglichen jedoch eine Verbindung, die von allen Threads eines Prozesses verwendet werden kann. Wenn Ihre Anwendung in einer Transaktionsumgebung ausgeführt wird, in der MQI-Aufrufe im gleichen Thread ausgegeben werden müssen, müssen Sie folgende Standardoptionen verwenden:

MQCNO_HANDLE_SHARE_NONE
Erstellt eine nicht gemeinsam genutzte Verbindung.

In den meisten anderen Umgebungen können Sie eine der folgenden threadunabhängigen, gemeinsam genutzten Verbindungsoptionen verwenden:

MQCNO_HANDLE_SHARE_BLOCK
Erstellt eine gemeinsam genutzte Verbindung. Wenn die Verbindung bei einer MQCNO_HANDLE_SHARE_BLOCK-Verbindung aktuell durch MQI-Aufrufe bei einem anderen Thread verwendet wird, wartet der MQI-Aufruf, bis der aktuelle MQI-Aufruf abgeschlossen ist.

MQCNO_HANDLE_SHARE_NO_BLOCK
Erstellt eine gemeinsam genutzte Verbindung. Wenn eine Verbindung bei einer MQCNO_HANDLE_SHARE_NO_BLOCK-Verbindung aktuell durch einen MQI-Aufruf bei einem anderen Thread verwendet wird, schlägt der MQI-Aufruf mit der Ursache MQRC_CALL_IN_PROGRESS fehl.

Mit Ausnahme der MTS-Umgebung (Microsoft Transaction Server) ist der Standardwert MQCNO_HANDLE_SHARE_NONE. In MTS-Umgebungen ist der Standardwert MQCNO_HANDLE_SHARE_BLOCK.

Vom MQCONNX-Aufruf wird eine Verbindungskennung zurückgegeben. Die Kennung kann durch nachfolgende MQI-Aufrufe von jedem Thread im Prozess verwendet werden, der diese Aufrufe der von MQCONNX zurückgegebenen Kennung zuordnet. MQI-Aufrufe, die eine gemeinsam genutzte Einzelkennung verwenden, werden über Threads serialisiert.

Beispielsweise ist bei einer gemeinsam genutzten Kennung folgende Reihenfolge von Vorgängen möglich:

1. Thread 1 gibt MQCONNX aus und ruft eine gemeinsam genutzte Kennung *h1* ab
2. Thread 1 öffnet eine Warteschlange und gibt eine Abrufanforderung mithilfe von *h1* aus
3. Thread 2 gibt eine Einreihungsanforderung mithilfe von *h1* aus
4. Thread 3 gibt eine Einreihungsanforderung mithilfe von *h1* aus
5. Thread 2 gibt MQDISC mithilfe von *h1* aus

Solange die Kennung von einem Thread genutzt wird, steht die Verbindung den anderen Threads nicht zur Verfügung. Sofern es akzeptabel ist, dass ein Thread wartet, bis ein vorangegangener Aufruf eines anderen Threads abgeschlossen ist, können Sie MQCONNX mit der Option MQCNO_HANDLE_SHARE_BLOCK verwenden.

Allerdings können Blockierungen Schwierigkeiten verursachen. Nehmen wir an, dass in Schritt „2" auf Seite 777 Thread 1 eine Abrufanforderung ausgibt, die auf Nachrichten wartet, die möglicherweise noch nicht eingetroffen sind (ein Abruf mit Wartestatus). In diesem Fall müssen die Threads 2 und 3 ebenfalls solange warten (werden blockiert), bis die Abrufanforderung bei Thread 1 erfolgt. Wenn Sie es vorziehen, dass ein MQI-Aufruf einen Fehler zurückgibt, wenn ein anderer MQI-Aufruf bereits bei der Kennung ausgeführt wird, verwenden Sie MQCONNX mit der Option MQCNO_HANDLE_SHARE_NO_BLOCK.

Hinweise zur Verwendung von gemeinsam genutzten Verbindungen

1. Alle Objektkennungen (Hobj), die durch das Öffnen eines Objekts erstellt werden, werden einem Hconn zugeordnet; bei einem gemeinsam genutzten Hconn werden die Hobjs somit ebenfalls gemeinsam genutzt und können von allen Threads mithilfe von Hconn verwendet werden. Ähnliches gilt für Arbeitseinheiten, die durch ein Hconn gestartet werden, das diesem Hconn zugeordnet ist; daher wird auch dies über Threads mit diesem Hconn gemeinsam genutzt.
2. *Alle* Threads können die Verbindung mit einem gemeinsam genutzten Hconn durch einen MQDISC trennen, nicht nur der Thread, der das entsprechende MQCONNX aufgerufen hat. Das MQDISC beendet Hconn und somit dessen Verfügbarkeit für alle Threads.
3. Ein Einzelthread kann mehrere gemeinsam genutzte Hconns seriell verwenden, z. B. kann MQPUT verwendet werden, um eine Nachricht bei einem gemeinsam genutzten Hconn und anschließend eine weitere Nachricht mithilfe eines weiteren gemeinsam genutzten Hconn einzureihen, wobei jede Operation unter einer anderen lokalen Arbeitseinheit ausgeführt wird.
4. Gemeinsam genutzte Hconns können nicht innerhalb einer globalen Arbeitseinheit verwendet werden.

Multi *Verwendung von MQCONNX-Aufrufoptionen mit MQ_CONNECT_TYPE*

Verwenden Sie diese Informationen, um sich mit den verschiedenen MQCONNX-Aufrufoptionen und deren Verwendung mit der Umgebungsvariablen **MQ_CONNECT_TYPE** vertraut zu machen.

Anmerkung: MQ_CONNECT_TYPE hat nur Auswirkungen auf STANDARD-Bindungen. Bei anderen Bindungen wird **MQ_CONNECT_TYPE** ignoriert.

Unter IBM MQ for Multiplatformskönnen Sie die Umgebungsvariable **MQ_CONNECT_TYPE** in Kombination mit dem Bindungstyp verwenden, der im Feld Options der MQCNO-Struktur angegeben ist, die in einem MQCONNX-Aufruf verwendet wird.

*Tabelle 113. Verwendung von MQCONNX-Aufrufoptionen mit der Umgebungsvariablen **MQ_CONNECT_TYPE***

MQCONNX-Aufrufoption	MQ_CONNECT_TYPE, Umgebungsvariable	Ergebnis
STANDARD	NICHT DEFINIERT	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	FASTPATH	STANDARD
STANDARD	CLIENT	CLIENT
STANDARD	LOKAL	STANDARD

Wenn MQCNO_STANDARD_BINDING nicht angegeben wird, können Sie MQCNO_NONE verwenden, das standardmäßig den Wert MQCNO_STANDARD_BINDING annimmt.

Authentifizierung und Identität für MQCONN und MQCONNX

Verwenden Sie diese Task, um zu erfahren, wie Anwendungen Berechtigungsnachweise für die Authentifizierung bereitstellen können, wenn sie eine Verbindung zu IBM MQherstellen.

Die Standardbenutzeridentität

Wenn eine Anwendung die Schnittstelle für Nachrichtenwarteschlangen (MQI) verwendet, um über MQCONN oder MQCONNX eine Verbindung zu IBM MQ herzustellen, wird immer eine Benutzeridentität eingerichtet und der Verbindung zugeordnet.

Standardmäßig ist die ursprüngliche Benutzeridentität immer die des Betriebssystemprozesses, unter dem die Anwendung läuft. Diese ursprüngliche Identität kann für lokal gebundene oder vertrauenswürdige Anwendungsverbindungen ausreichen.

Wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager mit einem MQCONN-Aufruf herstellt, kann die Anwendung die Standard-Benutzer-ID nicht ändern. Die folgenden Mechanismen können jedoch die Benutzer-ID ändern, die der Verbindung zugeordnet ist:

- Ein clientseitiger oder serverseitiger Sicherheitsexit
- Kanalauthentifizierungsregeln auf dem Warteschlangenmanager.
- Eine Client-Benutzer-ID, die während der gegenseitigen TLS-Authentifizierung eingerichtet wird

MQCONNX zur Bereitstellung von Berechtigungsnachweisen verwenden

MQCONNX gibt einer Anwendung mehr Kontrolle über die Identität, die der Verbindung zugeordnet ist. Eine Anwendung kann eine MQCSP-Struktur als Teil der Verbindungsoptionen bereitstellen, die im Parameter **ConnectOpts** für MQCONNX angegeben sind. Die MQCSP-Struktur kann Berechtigungsnachweise enthalten, die zum Erstellen einer Benutzeridentität verwendet werden. IBM MQ unterstützt die folgenden Berechtigungsnachweise in der MQCSP-Struktur:

- Eine Benutzer-ID und ein Kennwort.
- V 9.4.0 Ab IBM MQ 9.3.4ein Authentifizierungstoken, wenn die Anwendung eine Verbindung zu einem Warteschlangenmanager herstellt, der auf AIX -oder Linux -Systemen ausgeführt wird.

Die Konfiguration der Verbindungsauthentifizierung und Kanalauthentifizierung des Warteschlangenmanagers steuert, wie die von einer Anwendung bereitgestellten Berechtigungsnachweise verarbeitet werden. Diese Konfiguration wirkt sich beispielsweise auf die folgenden Aspekte aus:

- Gibt an, ob die Berechtigungsnachweise in der MQCSP-Struktur validiert werden und wie sie validiert werden.
- Gibt an, ob die Benutzer-ID in den Berechtigungsnachweisen in der MQCSP-Struktur einer anderen Benutzer-ID zugeordnet wird
- Gibt an, ob der authentifizierte Benutzer dann als Kontext für die Anwendung übernommen wird

Weitere Informationen zur Verbindungsauthentifizierung finden Sie unter Verbindungsauthentifizierung. Weitere Informationen zur Kanalauthentifizierung finden Sie im Abschnitt Kanalauthentifizierungsdatensätze.

Einige der in der Programmiersprache C geschriebenen Beispielprogramme, die die MQI verwenden, veranschaulichen, wie die MQCSP-Struktur zur Bereitstellung von Authentifizierungsnachweisen verwendet wird. Weitere Informationen finden Sie in den folgenden Beispielprogrammen:

- „Die Get-Beispielprogramme” auf Seite 1148
- „Die Put-Beispielprogramme” auf Seite 1161
- „Das Beispielprogramm 'Browser'” auf Seite 1135
- „TLS-Beispielprogramm” auf Seite 1178

Zugehörige Informationen

Benutzer mit der MQCSP-Struktur identifizieren und authentifizieren

MQCSP - Sicherheitsparameter
Benutzer identifizieren und authentifizieren

Programme mit MQDISC von einem Warteschlangenmanager trennen

In diesem Abschnitt erfahren Sie, wie Programme mit MQDISC von einem Warteschlangenmanager getrennt werden.

Wenn ein Programm mit einem MQCONN- oder MQCONNX-Aufruf eine Verbindung zu einem Warteschlangenmanager hergestellt und seine Interaktion mit dem Warteschlangenmanager beendet hat, trennt es die Verbindung zum Warteschlangenmanager automatisch, außer in den folgenden Fällen:

- z/OS In CICS Transaction Server for z/OS-Anwendungen, in denen dieser Aufruf optional ist, es sei denn, MQCONNX wurde verwendet und Sie möchten das Verbindungstag vor der Beendigung der Anwendung freigeben.
- IBM i Unter IBM MQ for IBM i. Dort wird bei der Abmeldung vom Betriebssystem ein impliziter MQDISC-Aufruf vorgenommen.

Als Eingabe für den MQDISC-Aufruf müssen Sie die Verbindungskennung (Hconn) bereitstellen, die bei der Herstellung der Verbindung zum Warteschlangenmanager von MQCONN bzw. MQCONNX zurückgegeben wurde.

Unter CICS auf Multiplatforms ist die Verbindungskennung (Hconn) nach dem Aufruf von MQDISC nicht mehr gültig und Sie können keine weiteren MQI-Aufrufe ausgeben, bis Sie MQCONN oder MQCONNX erneut aufrufen. MQDISC führt ein implizites MQCLOSE für alle Objekte aus, die noch mit dieser Kennung offen sind.

z/OS Bei Ausgabe eines MQDISC-Aufrufs für einen mit z/OS verbundenen Client wird zwar eine implizite Commitoperation ausgeführt, doch alle zu diesem Zeitpunkt noch offenen Warteschlangenkennungen werden erst geschlossen, wenn der Kanal tatsächlich beendet wird.

z/OS Wenn Sie MQCONNX zur Verbindung unter IBM MQ for z/OS verwenden, beendet MQDISC auch die Gültigkeit des von MQCONNX eingerichteten Verbindungstags. Wenn mit einem Verbindungstag in einer CICS-, IMS- oder RRS-Anwendung allerdings eine aktive Arbeitseinheit mit Wiederherstellung verknüpft ist, wird MQDISC mit dem Ursachencode MQRC_CONN_TAG_NOT_RELEASED abgelehnt.

Beschreibungen der Parameter finden Sie in der Beschreibung des MQDISC-Aufrufs im Abschnitt MQDISC.

Wenn kein MQDISC ausgegeben wird

Eine standardmäßige, nicht gemeinsam genutzte Verbindung (Hconn) wird getrennt, sobald der für den Verbindungsaufbau verantwortliche Thread endet. Eine gemeinsam genutzte Verbindung wird nur dann implizit zurückgesetzt und getrennt, wenn der gesamte Prozess endet. Wenn der Thread, durch den die gemeinsam genutzte Verbindung (Hconn) hergestellt wurde, endet, während die Verbindung noch besteht, kann die Verbindung weiterhin genutzt werden.

Berechtigungsprüfung

MQCLOSE- und MQDISC-Aufrufe führen üblicherweise keine Berechtigungsprüfung durch.

Im normalen Verlauf der Ereignisse schließt bzw. trennt ein Job, der die Berechtigung hat, ein IBM MQ-Objekt zu öffnen oder zu verbinden, das Objekt auch wieder. Selbst wenn einem Job, der eine Verbindung zu einem IBM MQ-Objekt hergestellt oder ein solches Objekt geöffnet hat, die Berechtigung entzogen wird, werden MQCLOSE- und MQDISC-Aufrufe akzeptiert.

Objekte öffnen und schließen

In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

Um eine der folgenden Operationen auszuführen, müssen Sie zuerst das relevante IBM MQ -Objekt *öffnen* :

- Nachrichten in eine Warteschlange einreihen
- Nachrichten aus einer Warteschlange abrufen
- Attribute eines Objekts festlegen
- Attribute eines Objekts abfragen

Zum Öffnen eines Objekts verwenden Sie den MQOPEN-Aufruf, wobei Sie mit den Optionen des Aufrufs festlegen, was Sie mit dem Objekt machen möchten. Die einzige Ausnahme hiervon ist, wenn Sie eine Einzelnachricht in eine Warteschlange einreihen und die Warteschlange danach sofort wieder schließen möchten. In diesem Fall können Sie das *Öffnen* umgehen, indem Sie den MQPUT1-Aufruf verwenden (siehe „Einzelnachricht mit dem MQPUT1-Aufruf in eine Warteschlange einreihen” auf Seite 802).

Bevor Sie ein Objekt mit dem MQOPEN-Aufruf öffnen, müssen Sie Ihr Programm mit einem Warteschlangenmanager verbinden. Eine ausführliche Beschreibung hierzu für alle Umgebungen finden Sie im Abschnitt „Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773.

Es gibt vier Typen von IBM MQ-Objekten, die geöffnet werden können:

- Warteschlange
- Namensliste
- Prozessdefinition
- Warteschlangenmanager

Mit dem MQOPEN-Aufruf öffnen Sie alle diese Objekte auf ähnliche Weise. Weitere Informationen zu IBM MQ-Objekten finden Sie unter Objekttypen.

Sie können das gleiche Objekt auch mehrmals öffnen, wobei Sie jedes Mal eine neue Objektkennung erhalten. Vielleicht möchten Sie Nachrichten einer Warteschlange mithilfe einer Kennung durchsuchen und andere Nachrichten aus der gleichen Warteschlange mithilfe einer anderen Kennung entfernen. Dadurch sparen Sie Ressourcen zum Schließen und erneuten Öffnen des gleichen Objekts. Sie können eine Warteschlange auch zum gleichzeitigen Durchsuchen *und* Entfernen von Nachrichten öffnen.

Ebenso können Sie mit einem MQOPEN-Aufruf auch mehrere Objekte öffnen und diese mit einem MQCLOSE-Aufruf wieder schließen. Informationen hierzu finden Sie im Abschnitt „Verteilerlisten” auf Seite 803.

Beim Versuch, ein Objekt zu öffnen, prüft der Warteschlangenmanager, ob Sie zum Öffnen dieses Objekts mit den im MQOPEN-Aufruf angegebenen Optionen berechtigt sind.

Objekte werden automatisch geschlossen, wenn ein Programm die Verbindung zum Warteschlangenmanager trennt. In der IMS-Umgebung wird eine Trennung erzwungen, wenn ein Programm aufgrund des IMS-Aufrufs GU (get unique) eine Verarbeitung für einen neuen Benutzer beginnt. Unter IBM i werden Objekte bei Beendigung eines Jobs automatisch geschlossen.

Es zählt jedoch zu den guten Programmierpraktiken, Objekte, die Sie geöffnet haben, auch wieder zu schließen. Hierzu verwenden Sie den MQCLOSE-Aufruf.

Unter den folgenden Links erhalten Sie weitere Informationen zum Öffnen und Schließen von Objekten:

- „Objekte mit dem MQOPEN-Aufruf öffnen” auf Seite 782
- „Dynamische Warteschlangen erstellen” auf Seite 790
- „Ferne Warteschlangen öffnen” auf Seite 791
- „Objekte mit dem MQCLOSE-Aufruf schließen” auf Seite 792

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773

Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Nachrichten in eine Warteschlange einreihen" auf Seite 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen" auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen" auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen" auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„IBM MQ-Anwendungen durch Auslöser starten" auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten" auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS" auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS" auf Seite 73
This information helps you to write IMS applications using IBM MQ.

Objekte mit dem MQOPEN-Aufruf öffnen

In diesem Abschnitt erfahren Sie, wie Objekte mit dem MQOPEN-Aufruf geöffnet werden.

Als Eingabe für den MQOPEN-Aufruf ist Folgendes erforderlich:

- Eine Verbindungskennung. Für CICS -Anwendungen unter z/OSkönnen Sie die Konstante MQHC_DEF_HCONN (mit dem Wert null) angeben oder die vom MQCONN-oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden. Verwenden Sie für Multiplatforms immer die Verbindungskennung, die vom MQCONN-oder MQCONNX-Aufruf zurückgegeben wurde.
- Eine Beschreibung des zu öffnenden Objekts in der Objektdeskriptorstruktur (MQOD)
- Eine oder mehrere Optionen zur Steuerung der Aktion des Aufrufs

Die Ausgabe von MQOPEN enthält folgende Komponenten:

- Eine Objektkennung, die Ihren Zugriff auf das Objekt darstellt; verwenden Sie diese Kennung als Eingabe für nachfolgende MQI-Aufrufe
- Eine geänderte Objektdeskriptorstruktur, wenn Sie eine dynamische Warteschlange erstellen (sofern von Ihrer Plattform unterstützt)
- Einen Beendigungscode.
- Einen Ursachencode.

Lebensdauer der Objektkennung

Die Lebensdauer einer Objektkennung (Hobj) ist die Gleiche wie die Lebensdauer einer Verbindungskennung (Hconn).

Nähere Informationen hierzu finden Sie in den Abschnitten „Gültigkeitsbereich des MQCONN- bzw. MQCONNX-Aufrufs" auf Seite 775 und „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX" auf Seite 777. In einigen Umgebungen sind jedoch zusätzliche Einschränkungen zu berücksichtigen:

CICS
In einem CICS-Programm können Sie die Kennung nur in der CICS-Task verwenden, aus der der MQOPEN-Aufruf erfolgt ist.

IMS und z/OS Batch
In der IMS -und z/OS Batch -Umgebung können Sie die Kennung innerhalb derselben Task, aber nicht innerhalb von Subtasks verwenden.

Beschreibungen der MQOPEN-Parameter finden Sie im Abschnitt MQOPEN.

In den nachfolgenden Abschnitten werden die Informationen beschrieben, die Sie als Eingaben für MQOPEN bereitstellen müssen.

Objekte (in der MQOD-Struktur) identifizieren

Das zu öffnende Objekt identifizieren Sie in der MQOD-Struktur. Diese Struktur ist ein Eingabeparameter für den MQOPEN-Aufruf. (Wenn Sie mit dem MQOPEN-Aufruf eine dynamische Warteschlange erstellen, wird die Struktur vom Warteschlangenmanager geändert.)

Ausführliche Informationen zur MQOD-Struktur finden Sie im Abschnitt MQOD.

Informationen zur Verwendung der MQOD-Struktur für Verteilerliste finden Sie im Abschnitt „MQOD-Struktur verwenden" auf Seite 804 unter „Verteilerlisten" auf Seite 803.

Namensauflösung
So löst der MQOPEN-Aufruf die Namen von Warteschlangen und Warteschlangenmanagern auf.

Anmerkung: Der Aliasname eines Warteschlangenmanagers ist eine Definition einer fernen Warteschlange ohne ein RNAME-Feld.

Beim Öffnen einer IBM MQ-Warteschlange führt der MQOPEN-Aufruf eine Funktion zur Auflösung des von Ihnen bereitgestellten Warteschlangennamens durch. Dadurch wird bestimmt, in welcher Warteschlange der Warteschlangenmanager die nachfolgenden Operationen ausführt. Das bedeutet, dass der Aufruf den Namen entweder in einer lokalen Warteschlange oder einer Übertragungswarteschlange auslöst, wenn Sie den Namen einer Aliaswarteschlange oder einer fernen Warteschlange in Ihrem Objektdeskriptor (MQOD) angeben. Wenn eine Warteschlange für irgendeine Art von Eingabe, Anzeige oder Einstellung geöffnet ist, erfolgt die Auflösung in eine lokale Warteschlange, wenn eine vorhanden ist, und schlägt fehl, wenn keine vorhanden ist. Die Auflösung erfolgt nur in eine nicht lokale Warteschlange, wenn Sie nur für Ausgabedaten, nur für Abfragen oder nur für Ausgabedaten und Abfragen geöffnet ist. Im Abschnitt Tabelle 114 auf Seite 783 finden Sie eine Übersicht des Namensauflösungsprozesses. Der Name, den Sie im Feld *ObjectQMgrName* angeben, wird *vor* dem im Feld *ObjectName* bereitgestellten Namen aufgelöst.

Tabelle 114 auf Seite 783 zeigt ebenfalls, wie Sie eine lokale Definition einer fernen Warteschlange verwenden können, um einen Aliasnamen für den Namen eines Warteschlangenmanagers zu definieren. Dies erlaubt Ihnen auszuwählen, welche Übertragungswarteschlange verwendet wird, wenn Sie Nachrichten in eine ferne Warteschlange einreihen, damit Sie zum Beispiel eine einzelne Übertragungswarteschlange für Nachrichten verwenden können, die für viele ferne Warteschlangen bestimmt sind.

Lesen Sie sich zur Verwendung der folgenden Tabelle die zwei linken Spalten unter der Überschrift **Eingabe in MQOD** durch und wählen Sie den zutreffenden Fall aus. Anschließend finden Sie in der jeweiligen Zeile die entsprechenden Anweisungen. Wenn Sie die Anweisungen in den Spalten **Aufgelöste Namen** befolgen, können Sie entweder zur Spalte **Eingabe in MQOD** zurückgehen und die Werte gemäß Anweisung eingeben oder Sie können die Tabelle mit den gelieferten Ergebnissen schließen. Eventuell müssen Sie zum Beispiel *ObjectName* eingeben.

Tabelle 114. Warteschlangennamen bei Verwendung von MQOPEN auflösen

Eingabe in MQOD	Eingabe in MQOD	Aufgelöste Namen	Aufgelöste Namen	Aufgelöste Namen
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue

Tabelle 114. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)

Eingabe in MQOD	**Eingabe in MQOD**	**Aufgelöste Namen**	**Aufgelöste Namen**	**Aufgelöste Namen**
Leer oder lokaler Warteschlangenmanager	Lokale Warteschlange ohne Clusterattribut	Lokaler Warteschlangenmanager	Eingabe *ObjectName*	Nicht zutreffend (lokale Warteschlange verwendet)
Leerer Warteschlangenmanager	Lokale Warteschlange mit Clusterattribut	Durch Auslastungsmanagement gewählter Cluster-Warteschlangenmanager oder bei PUT-Vorgang gewählter spezifischer Cluster-Warteschlangenmanager	Eingabe *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE und die verwendete lokale Warteschlange SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Lokaler Warteschlangenmanager	Lokale Warteschlange mit Clusterattribut	Lokaler Warteschlangenmanager	Eingabe *ObjectName*	Nicht zutreffend (lokale Warteschlange verwendet)
Leer oder lokaler Warteschlangenmanager	Modellwarteschlange	Lokaler Warteschlangenmanager	Erstellter Name	Nicht zutreffend (lokale Warteschlange verwendet)
Leer oder lokaler Warteschlangenmanager	Aliaswarteschlange mit oder ohne Clusterattribut	Namensauflösung erneut mit unverändertem *ObjectQMgrName* ausführen und *ObjectName* mit Einstellung auf *BaseQName* im Definitionsobjekt der Aliaswarteschlange eingeben. Darf sich nicht in einen lokal definierten Aliasnamen auflösen, in dem *ObjectQMgrName* angegeben ist, kann sich aber in einen Cluster-Aliasnamen auflösen (der auf einem anderen Warteschlangenmanager bereitgestellt wird), in dem *ObjectQMgrName* leer ist.		
Lokaler Warteschlangenmanager	Aliaswarteschlange mit Clusterattribut	Der Aliasname darf nicht in eine Clusterwarteschlange aufgelöst werden, die nicht lokal definiert ist, oder die denselben *ObjectName* wie der Alias hat.		

Tabelle 114. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)

Eingabe in MQOD	**Eingabe in MQOD**	**Aufgelöste Namen**	**Aufgelöste Namen**	**Aufgelöste Namen**
Leerer Warteschlangenmanager	Aliaswarteschlange mit Clusterattribut	Der Aliasname kann in eine Clusterwarteschlange mit gleichem *ObjectName* wie der Alias aufgelöst werden.		
Leer oder lokaler Warteschlangenmanager	Lokale Definition einer fernen Warteschlange	Namensauflösung erneut mit *ObjectQMgrName* mit Einstellung auf *RemoteQMgrName* und *ObjectName* mit Einstellung auf *RemoteQName* ausführen. Darf keine fernen Warteschlangen auflösen		Name von *XmitQName*-Attribut, wenn nicht leer; anderenfalls *RemoteQMgrName* in das Definitionsobjekt einer fernen Warteschlange. SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Leerer Warteschlangenmanager	Kein übereinstimmendes lokales Objekt; Clusterwarteschlange gefunden	Durch Auslastungsmanagement gewählter Cluster-Warteschlangenmanager oder bei PUT-Vorgang gewählter spezifischer Cluster-Warteschlangenmanager	Eingabe *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Leer oder lokaler Warteschlangenmanager	Kein übereinstimmendes lokales Objekt. Clusterwarteschlange nicht gefunden.		Fehler, Warteschlange nicht gefunden	Nicht zutreffend
Name von Warteschlangenmanager in gleicher Warteschlangengruppe für gemeinsame Nutzung wie lokaler Warteschlangenmanager	Lokale gemeinsam genutzte Warteschlange	Lokaler Warteschlangenmanager	Eingabe *ObjectName*	Nicht zutreffend
Name einer lokalen Übertragungswarteschlange	(Nicht aufgelöst)	Eingabe *ObjectQMgrName*	Eingabe *ObjectName*	Eingabe *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)

Tabelle 114. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)

Eingabe in MQOD	**Eingabe in MQOD**	**Aufgelöste Namen**	**Aufgelöste Namen**	**Aufgelöste Namen**
Aliasdefinition für Warteschlangenmanager (*RemoteQMgrName* kann der lokale Warteschlangenmanager sein)	(Nicht aufgelöste, ferne Warteschlange)	Namensauflösung erneut mit *ObjectQMgrName* mit Einstellung *RemoteQMgrName* ausführen. Darf nicht in ferne Warteschlange aufgelöst werden	Eingabe *ObjectName*	Name von *XmitQName*-Attribut, wenn nicht leer; anderenfalls *RemoteQMgrName* in das Definitionsobjekt einer fernen Warteschlange. SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Warteschlangenmanager ist nicht der Name eines lokalen Objekts; Cluster-Warteschlangenmanager oder Warteschlangenmanager-Aliasname gefunden	(Nicht aufgelöst)	*ObjectQMgrName* oder bestimmter Cluster-Warteschlangenmanager bei PUT-Vorgang ausgewählt	Eingabe *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Warteschlangenmanager ist nicht der Name eines lokalen Objekts; keine Clusterobjekte gefunden	(Nicht aufgelöst)	Eingabe *ObjectQMgrName*	Eingabe *ObjectName*	*DefXmitQName*-Attribut des Warteschlangenmanagers, wo *DefXmitQName* unterstützt wird. SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)

Anmerkungen:

1. *BaseQName* - Dies ist der Name der Basiswarteschlange aus der Definition der Aliaswarteschlange.
2. *RemoteQName* - Dies ist der Name der fernen Warteschlange aus der lokalen Definition der fernen Warteschlange.
3. *RemoteQMgrName* - Dies ist der Name des fernen Warteschlangenmanagers aus der lokalen Definition der fernen Warteschlange.
4. *XmitQName* - Dies ist der Name der Übertragungswarteschlange aus der lokalen Definition der fernen Warteschlange.
5. z/OS Für IBM MQ for z/OS -Warteschlangenmanager, die Teil einer Gruppe mit gemeinsamer Warteschlange sind, kann der Name der Gruppe mit gemeinsamer Warteschlange anstelle des Namens des lokalen Warteschlangenmanagers in Tabelle 114 auf Seite 783verwendet werden. Wenn der lokale Warteschlangenmanager die Zielwarteschlange nicht öffnen kann bzw. eine Nachricht nicht in die Warteschlange eingereiht werden kann, wird die Nachricht entweder über die gruppeninterne Warteschlangensteuerung oder einen IBM MQ-Kanal an die Warteschlange mit dem angegebenen ObjectQMgrName weitergeleitet.
6. In der Spalte *ObjectName* der Tabelle bezieht sich CLUSTER sowohl auf das Attribut CLUSTER als auch auf das Attribut CLUSNL der Warteschlange.
7. SYSTEM.QSG.TRANSMIT.QUEUE wird verwendet, wenn lokale und ferne Warteschlangenmanager in derselben Gruppe mit gemeinsamer Warteschlange sind; gruppeninterne Warteschlangensteuerung wird aktiviert.
8. Wenn Sie jedem Clustersenderkanal eine andere Clusterübertragungswarteschlange zugeordnet haben, lautet der Name der Clusterübertragungswarteschlange möglicherweise nicht SYSTEM.CLUS-

TER.TRANSMIT.QUEUE. Weitere Informationen zu mehreren Clusterübertragungswarteschlangen finden Sie im Abschnitt Clustering: Konfiguration von Clusterübertragungswarteschlangen planen.

9. Wenn der Warteschlangenmanager nicht der Name eines lokalen Objekts ist und Clusterwarteschlangenmanager oder ein Warteschlangenmanager-Aliasname gefunden werden, tritt folgende Situation ein.

   Wenn Sie mit **ObjectQMgrName** einen Warteschlangenmanagernamen angegeben haben und dem lokalen Warteschlangenmanager mehrere Clusterkanäle mit unterschiedlichen Clusternamen bekannt sind, über die diese Zieladresse erreicht werden kann, kann unabhängig vom Clusternamen der Zielwarteschlange irgendeiner dieser Kanäle zur Übertragung der Nachricht verwendet werden.

   Dies kann ein unerwartetes Verhalten sein, wenn erwartet wurde, dass Nachrichten für diese Warteschlange ausschließlich über einen Kanal mit demselben Clusternamen wie die Warteschlange gesendet werden.

   Der Wert **ObjectQMgrName** hat in diesem Fall jedoch Vorrang, und für den Clusterlastausgleich werden alle Kanäle in Betracht gezogen, über die der Warteschlangenmanager erreicht werden kann, ohne dass der Clustername, zu dem die Kanäle gehören, dabei eine Rolle spielt.

Beim Öffnen einer Aliaswarteschlange wird zugleich die Basiswarteschlange geöffnet, in die der Aliasname aufgelöst wird, und beim Öffnen einer fernen Warteschlange wird zugleich die Übertragungswarteschlange geöffnet. Daher können Sie weder die von Ihnen angegebene Warteschlange löschen, noch die Warteschlange, in die sie auflöst, während die andere offen ist.

Obwohl eine Aliaswarteschlange in eine andere lokal definierte Aliaswarteschlange (egal ob in einem Cluster gemeinsam genutzt oder nicht) nicht aufgelöst werden kann, ist das Auflösen in eine über Fernzugriff definierte Cluster-Aliaswarteschlange zulässig und kann daher als Basiswarteschlange angegeben werden.

Der aufgelöste Name der Warteschlange bzw. des Warteschlangenmanagers werden in der MQOD-Struktur in den Feldern *ResolvedQName* und *ResolvedQMgrName* gespeichert.

Weitere Informationen zur Namensauflösung in einer Umgebung mit verteilter Steuerung von Warteschlangen finden Sie im Abschnitt Was ist die Auflösung von Warteschlangennamen?.

Optionen des MQOPEN-Aufrufs verwenden

Geben Sie im Parameter **Options** des MQOPEN-Aufrufs eine oder mehrere Optionen an, um die Art des Zugriffs zu steuern, der Ihnen auf das Objekt, das Sie öffnen, eingeräumt wird. Mit diesen Optionen können Sie sich folgende Berechtigungen einräumen:

- Warteschlange öffnen und festlegen, dass alle Nachrichten, die in diese Warteschlange eingereiht werden, an die gleiche Instanz dieser Warteschlange übergeben werden
- Warteschlange öffnen, um Nachrichten darin einzureihen
- Warteschlange öffnen, um Nachrichten darin zu durchsuchen
- Warteschlange öffnen, um Nachrichten daraus zu entfernen
- Objekt öffnen, um dieses abzufragen und dessen Attribute festzulegen (Sie können allerdings nur die Attribute von Warteschlangen festlegen)
- Thema oder Themenzeichenfolge öffnen, um Nachrichten darin zu veröffentlichen
- Kontextinformationen mit einer Nachricht verknüpfen
- Alternative Benutzer-ID für Sicherheitsprüfungen angeben
- Aufruf steuern, wenn der Warteschlangenmanager stillgelegt ist

MQOPEN-Option für Clusterwarteschlangen
Die Bindung für die Warteschlangenkennung wird aus dem Warteschlangenattribut **DefBind** übernommen, das den Wert MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED oder MQBND_BIND_ON_GROUP haben kann.

Sollen alle Nachrichten, die mit MQPUT in eine Warteschlange eingereiht werden, über die gleiche Route an den gleichen Warteschlangenmanager geleitet werden, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BIND_ON_OPEN.

Soll das Ziel zum Zeitpunkt der Einreihung mit MQPUT angegeben werden, also auf Nachrichtenbasis, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BIND_NOT_FIXED.

Sollen alle Nachrichten einer Nachrichtengruppe, die mit MQPUT in eine Warteschlange eingereiht werden, der gleichen Zielinstanz zugeordnet werden, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BIND_ON_GROUP.

Bei Verwendung von Nachrichtengruppen mit Clustern muss entweder MQOO_BIND_ON_OPEN oder MQOO_BIND_ON_GROUP angegeben werden, um sicherzustellen, dass alle Nachrichten in der Gruppe an demselben Ziel verarbeitet werden.

Wenn Sie keine dieser Optionen angeben, wird der Standardwert MQOO_BIND_AS_Q_DEF verwendet.

Wenn Sie in der MQOD den Namen eines Warteschlangenmanagers angeben, wird die Warteschlangeninstanz dieses Warteschlangenmanagers ausgewählt. Ist der Warteschlangenmanagername leer, kann jede Instanz ausgewählt werden. Weitere Informationen finden Sie im Abschnitt „MQOPEN und Cluster" auf Seite 934.

Wenn Sie mit einer QALIAS-Definition eine Clusterwarteschlange öffnen, werden einige Warteschlangenattribute durch die Aliaswarteschlange und nicht durch die Basiswarteschlange festgelegt. Clusterattribute gehören zu den Attributen der Basiswarteschlangendefinition, die durch die Aliaswarteschlange überschrieben werden. Im folgenden Snippet wird die Clusterwarteschlange beispielsweise mit MQOO_BIND_NOT_FIXED und nicht mit MQOO_BIND_ON_OPEN geöffnet. Die Definition der Clusterwarteschlange steht im gesamten Cluster zur Verfügung, während die Definition der Aliaswarteschlange nur lokal für den Warteschlangenmanager gilt.

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

MQOPEN-Option zum Einreihen von Nachrichten
Mit der Option MQOO_OUTPUT können Sie eine Warteschlange oder ein Thema für das Einreihen von Nachrichten öffnen.

MQOPEN-Option zum Durchsuchen von Nachrichten
Wenn Sie eine Warteschlange öffnen möchten, um die Nachrichten darin zu *durchsuchen*, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BROWSE.

Dadurch wird ein *Anzeigecursor* erstellt, mit dem der Warteschlangenmanager die nächste Nachricht in der Warteschlange ermittelt. Weitere Informationen finden Sie unter „Nachrichten in einer Warteschlange durchsuchen (Browsing)" auf Seite 844.

Anmerkung:

1. Die Nachrichten einer fernen Warteschlange können Sie mit der Option MQOO_BROWSE nicht durchsuchen; öffnen Sie mit dieser Option keine ferne Warteschlange.
2. Beim Öffnen einer Verteilerliste können Sie diese Option nicht angeben. Weitere Informationen zu Verteilerlisten finden Sie im Abschnitt „Verteilerlisten" auf Seite 803.
3. Wenn Sie kooperatives Browsing verwenden, sollten Sie MQOO_BROWSE mit der Option MQOO_CO_OP verwenden (siehe Optionen).

MQOPEN-Optionen zum Entfernen von Nachrichten
Das Öffnen einer Warteschlange zum Entfernen von Nachrichten können Sie durch drei Optionen steuern.

In einem MQOPEN-Aufruf können Sie jeweils nur eine dieser Optionen verwenden. Mit diesen Optionen bestimmen Sie, ob Ihr Programm exklusiven oder gemeinsamen Zugriff auf die Warteschlange hat. *Exklusiver Zugriff* bedeutet Folgendes, solange Sie die Warteschlange nicht wieder geschlossen haben: Nur Sie können Nachrichten daraus entfernen. Falls ein anderes Programm versucht, die Warteschlange zu

öffnen, um Nachrichten daraus zu entfernen, schlägt dessen MQOPEN-Aufruf fehl. *Gemeinsamer Zugriff* bedeutet, dass mehrere Programme Nachrichten daraus entfernen können.

Es empfiehlt sich, die Art des Zugriffs zu übernehmen, die für die Warteschlange bei deren Definition vorgesehen wurde. Die Warteschlangendefinition beinhaltet u. a. die Einstellung der Attribute **Shareability** und **DefInputOpenOption**. Zur Übernahme dieser Zugriffseinstellung verwenden Sie die Option MQOO_INPUT_AS_Q_DEF. Im Abschnitt Tabelle 115 auf Seite 789 wird erläutert, wie sich die Einstellung dieser Attribute auf die Art des Zugriffs auswirkt, die Sie bei Verwendung dieser Option erhalten.

Tabelle 115. Auswirkung der Attribute und Optionen des MQOPEN-Aufrufs auf den Warteschlangenzugriff

Warteschlangenattribute		**Art des Zugriffs durch MQOPEN-Optionen**		
Shareability	**DefInputOpenOption**	**AS_Q_DEF**	**SHARED**	**EXCLUSIVE**
SHAREABLE	SHARED	Gemeinsam genutzt	Gemeinsam genutzt	exklusiv
SHAREABLE	EXCLUSIVE	exklusiv	Gemeinsam genutzt	exklusiv
NOT_SHAREABLE*	SHARED*	exklusiv	exklusiv	exklusiv
NOT_SHAREABLE	EXCLUSIVE	exklusiv	exklusiv	exklusiv

Anmerkung: * Auch wenn Sie eine Warteschlange mit dieser Attributkombination definieren können, wird die Standardeingabeoption für das Öffnen durch das Shareability-Attribut überschrieben.

Alternativen:

- Wenn Sie wissen, dass Ihre Anwendung auch dann funktioniert, wenn andere Programme zur selben Zeit Nachrichten aus der Warteschlange entfernen, verwenden Sie die Option MQOO_INPUT_SHARED. Im Abschnitt Tabelle 115 auf Seite 789 erfahren Sie, wie Sie in bestimmten Fällen selbst bei dieser Option exklusiven Zugriff auf die Warteschlange erhalten.
- Wenn Sie wissen, dass Ihre Anwendung nur dann funktioniert, wenn andere Programme zur selben Zeit keine Nachrichten aus der Warteschlange entfernen können, verwenden Sie die Option MQOO_INPUT_EXCLUSIVE.

Anmerkung:

1. Aus einer fernen Warteschlange können Sie keine Nachrichten entfernen. Daher können Sie eine ferne Warteschlange mit keiner der MQOO_INPUT_*-Optionen öffnen.
2. Beim Öffnen einer Verteilerliste können Sie diese Option nicht angeben. Weitere Informationen finden Sie in „Verteilerlisten” auf Seite 803.

MQOPEN-Optionen zum Einstellen und Abfragen von Attributen
Mit der Option MQOO_SET können Sie eine Warteschlange öffnen, um ihre Attribute festzulegen.

Eine Definition der Attribute anderer Objekttypen ist nicht möglich (siehe „Objektattribute abfragen und einstellen” auf Seite 897).

Zum Öffnen eines Objekts zum Abfragen seiner Attribute verwenden Sie die Option MQOO_INQUIRE.

Anmerkung: Beim Öffnen einer Verteilerliste können Sie diese Option nicht angeben.

MQOPEN-Optionen für den Nachrichtenkontext
Wenn Sie beim Einreihen einer Nachricht in eine Warteschlange die Möglichkeit haben möchten, der Nachricht Kontextinformationen zuzuordnen, müssen Sie beim Öffnen der Warteschlange eine der Nachrichtenkontextoptionen verwenden.

Die Optionen ermöglichen eine Differenzierung zwischen Kontextinformationen zum *Benutzer*, der die Nachricht erstellt hat, und Kontextinformationen zu der *Anwendung*, die die Nachricht erstellt hat. Außerdem können Sie entscheiden, ob Sie beim Einreihen der Nachricht in die Warteschlange die Kontextinfor-

mationen selbst festlegen möchten oder ob der Kontext automatisch aus einer anderen Warteschlangenkennung übernommen werden soll.

Zugehörige Konzepte

„Nachrichtenkontext” auf Seite 50
Nachrichtenkontext-Informationen ermöglichen der Anwendung, die die Nachricht abruft, Informationen über den Absender der Nachricht zu erhalten.

„Nachrichtenkontextinformationen steuern” auf Seite 799
Wenn Sie den MQPUT- oder MQPUT1-Aufruf verwenden, um eine Nachricht in eine Warteschlange einzureihen, können Sie angeben, dass der Warteschlangenmanager dem Nachrichtendeskriptor einige Standard-Kontextinformationen hinzufügen soll. Anwendungen, die über die entsprechende Berechtigungsstufe verfügen, können zusätzliche Kontextinformationen enthalten. Über das Optionenfeld in der MQPMO-Struktur können Sie die Kontextinformationen steuern.

MQOPEN-Option für eine alternative Benutzerberechtigung
Beim Versuch, ein Objekt mit einem MQOPEN-Aufruf zu öffnen, prüft der Warteschlangenmanager, ob Sie zum Öffnen dieses Objekts berechtigt sind. Falls Sie keine Berechtigung besitzen, schlägt der Aufruf fehl.

Für Serverprogramme muss der Warteschlangenmanager aber unter Umständen die Berechtigung des Benutzers prüfen, für den die Programme arbeiten, nicht die Berechtigung des Servers selbst. Dazu müssen diese Programme die Option MQOO_ALTERNATE_USER_AUTHORITY des MQOPEN-Aufrufs verwenden und die alternative Benutzer-ID im Feld *AlternateUserId* der MQOD-Struktur angeben. Normalerweise erhält der Server die Benutzer-ID aus den Kontextinformationen der Nachricht, die er verarbeitet.

z/OS *MQOPEN option for queue manager quiescing*
If you use the MQOPEN call when the queue manager is in a quiescing state, the call might fail, depending on which environment you are using.

In the CICS environment on z/OS, if you use the MQOPEN call when the queue manager is in a quiescing state, the call always fails.

In other z/OS and Multiplatforms environments, the call fails when the queue manager is quiescing only if you use the MQOO_FAIL_IF_QUIESCING option of the MQOPEN call.

MQOPEN-Option zum Auflösen der Namen lokaler Warteschlangen
Wenn Sie eine lokale Alias- oder Modellwarteschlange öffnen, wird die lokale Warteschlange zurückgegeben.

Wenn Sie jedoch eine ferne Warteschlange oder eine Clusterwarteschlange öffnen, werden die Felder *ResolvedQName* und *ResolvedQMgrName* der MQOD-Struktur mit den Namen der fernen Warteschlange und des fernen Warteschlangenmanagers gefüllt, die in der Definition der fernen Warteschlange gefunden wurden, oder mit der ausgewählten fernen Clusterwarteschlange.

Verwenden Sie die Option MQOO_RESOLVE_LOCAL_Q des MQOPEN-Aufrufs, um die *ResolvedQName* in der MQOD-Struktur mit dem Namen der lokalen Warteschlange zu füllen, die geöffnet wurde. In ähnlicher Weise wird *ResolvedQMgrName* mit dem Namen des lokalen Warteschlangenmanagers gefüllt, auf dem sich die lokale Warteschlange befindet. Dieses Feld steht allerdings nur in Version 3 der MQOD-Struktur zur Verfügung; vor Version 3 wird MQOO_RESOLVE_LOCAL_Q ohne Rückgabe eines Fehlers ignoriert.

Wenn Sie beim Öffnen beispielsweise eine ferne Warteschlange MQOO_RESOLVE_LOCAL_Q angeben, ist *ResolvedQName* der Name der Übertragungswarteschlange, in die Nachrichten eingereiht werden. *ResolvedQMgrName* ist der Name des lokalen Warteschlangenmanagers, der die Übertragungswarteschlange hostet.

Dynamische Warteschlangen erstellen

Verwenden Sie eine dynamische Warteschlange, wenn die Warteschlange nach Beendigung der Anwendung nicht mehr benötigt wird.

Sie können beispielsweise als Empfangswarteschlange für Antworten eine dynamische Warteschlange verwenden. Den Namen der Empfangswarteschlange für Antworten geben Sie beim Einreihen einer

Nachricht in eine Warteschlange im Feld *ReplyToQ* der MQMD-Struktur an (siehe „Nachrichten mit der MQMD-Struktur definieren" auf Seite 794).

Zum Erstellen einer dynamischen Warteschlange verwenden Sie eine Schablone, eine sogenannte Modellwarteschlange, in Verbindung mit dem Aufruf MQOPEN. Eine Modellwarteschlange erstellen Sie mit den Befehlen bzw. mit den Operations- und Bedienfeldern von IBM MQ. Die von Ihnen erstellte dynamische Warteschlange übernimmt die Attribute der Modellwarteschlange.

Den Namen der Modellwarteschlange geben Sie beim Aufruf von MQOPEN im Feld *ObjectName* der MQOD-Struktur an. Bei Beendigung des Aufrufs wird für das Feld *ObjectName* der Name der erstellten dynamischen Warteschlange übernommen. Im Feld *ObjectQMgrName* wird dann der Name des lokalen Warteschlangenmanagers eingetragen.

Der Name der zu erstellenden dynamischen Warteschlange kann auf drei Arten angegeben werden:

- Im Feld *DynamicQName* der MQOD-Struktur können Sie den gewünschten vollständigen Namen angeben.
- Sie können ein Präfix (mit weniger als 33 Zeichen) für den Namen angeben und die weitere Benennung dem Warteschlangenmanager überlassen. Der Warteschlangenmanager generiert dann einen eindeutigen Namen und dennoch behalten Sie eine gewisse Kontrolle. (So könnten Sie beispielsweise für jeden Benutzer ein bestimmtes Präfix verwenden oder Warteschlangen mit einem bestimmten Namenspräfix eine spezielle Sicherheitsklassifikation zuteilen.) Wenn Sie dieses Verfahren anwenden möchten, geben Sie im Feld *DynamicQName* als letztes Zeichen abgesehen von Leerzeichen einen Stern (*) an. Geben Sie nicht nur einen einzelnen Stern (*) als Name der dynamischen Warteschlange an.
- Sie können auch die Generierung des vollständigen Namens dem Warteschlangenmanager überlassen. Geben Sie für dieses Verfahren im Feld *DynamicQName* an erster Zeichenposition einen Stern (*) an.

Weitere Informationen zu diesen Verfahren finden Sie unter der Beschreibung des Felds DynamicQName.

Zusätzliche Angaben über dynamische Warteschlangen enthält der Abschnitt Dynamische und Modellwarteschlangen.

Ferne Warteschlangen öffnen

Eine ferne Warteschlange ist eine Warteschlange, die von einem anderen Warteschlangenmanager verwaltet wird als demjenigen, mit dem die Anwendung verbunden ist.

Zum Öffnen einer fernen Warteschlange verwenden Sie wie für eine lokale Warteschlange einen MQOPEN-Aufruf. Sie können den Namen der Warteschlange wie folgt angeben:

1. Geben Sie im Feld *ObjectName* der MQOD-Struktur den Namen der fernen Warteschlange an, der dem *Lokal*-Warteschlangenmanager bekannt ist.

   Anmerkung: Lassen Sie das Feld *ObjectQMgrName* in diesem Fall leer.
2. Geben Sie im Feld *ObjectName* der MQOD-Struktur den Namen der fernen Warteschlange an, der dem *Fern*-Warteschlangenmanager bekannt ist. Geben Sie im Feld *ObjectQMgrName* Folgendes an:

   - Den Namen der Übertragungswarteschlange, die den gleichen Namen wie der ferne Warteschlangenmanager hat. Name und Schreibweise (Groß-/Kleinschreibung bzw. eine Kombination daraus) müssen *exakt* übereinstimmen.
   - Den Namen eines Warteschlangenmanager-Aliasobjekts, der sich auf den Zielwarteschlangenmanager oder die Übertragungswarteschlange auflösen lässt.

   Dadurch kennt der Warteschlangenmanager sowohl das Ziel der Nachricht als auch die Übertragungswarteschlange, in die die Nachricht gestellt werden muss, damit sie ihr Ziel erreicht.
3. Wenn *DefXmitQname* unterstützt wird, geben Sie im Feld *ObjectName* der MQOD-Struktur den Namen der fernen Warteschlange an, der dem *Fern*-Warteschlangenmanager bekannt ist.

   Anmerkung: Setzen Sie das Feld *ObjectQMgrName* auf den Namen des fernen Warteschlangenmanagers (in diesem Fall darf es nicht leer bleiben).

Beim Aufruf von MQOPEN werden nur lokale Namen geprüft; der letzte Punkt der Überprüfung ist die Validierung, dass die angegebene Übertragungswarteschlange vorhanden ist.

Eine Zusammenfassung dieser Methoden finden Sie im Abschnitt Tabelle 114 auf Seite 783.

Objekte mit dem MQCLOSE-Aufruf schließen

Zum Schließen eines Objekts verwenden Sie den MQCLOSE-Aufruf.

Wenn das Objekt eine Warteschlange ist, beachten Sie Folgendes:

- Eine temporäre dynamische Warteschlange muss vor dem Schließen nicht geleert werden.

  Beim Schließen einer temporären dynamischen Warteschlange wird die Warteschlange mit allen darin enthaltenen Nachrichten gelöscht, selbst wenn für diese Warteschlange noch nicht festgeschriebene MQGET-, MQPUT- oder MQPUT1-Aufrufe ausstehen.

- z/OS Wenn unter IBM MQ for z/OS für die zu schließende Warteschlange MQGET-Anforderungen mit einer MQGMO_SET_SIGNAL-Option ausstehen, so werden diese abgebrochen.
- Wenn Sie die Warteschlange mit der Option MQOO_BROWSE geöffnet haben, wird der Anzeigecursor gelöscht.

Da das Schließen von Warteschlangen unabhängig von Synchronisationspunkten ist, können Sie Warteschlangen vor oder nach Synchronisationspunkten schließen.

Als Eingabe für den MQCLOSE-Aufruf ist Folgendes erforderlich:

- Eine Verbindungskennung. Verwenden Sie dieselbe Verbindungskennung, die zum Öffnen verwendet wurde. Für CICS -Anwendungen unter z/OSkönnen Sie alternativ die Konstante MQHC_DEF_HCONN angeben (die den Wert null hat).
- Die Kennung des Objekts, das Sie schließen möchten. Diese entnehmen Sie der Ausgabe des MQOPEN-Aufrufs.
- MQCO_NONE im Feld *Options* (es sei denn, Sie schließen eine permanente dynamische Warteschlange).
- Die Steueroption, mit der festgelegt wird, ob der Warteschlangenmanager die Warteschlange auch dann löschen soll, wenn sie noch Nachrichten enthält (beim Schließen einer permanenten dynamischen Warteschlange).

Die Ausgabe von MQCLOSE enthält folgende Komponenten:

- Einen Beendigungscode
- Einen Ursachencode
- Die Objektkennung zurückgesetzt auf den Wert MQHO_UNUSABLE_HOBJ

Beschreibungen der MQCLOSE-Parameter finden Sie im Abschnitt MQCLOSE.

Nachrichten in eine Warteschlange einreihen

In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

Zum Einreihen einer Nachricht in eine Warteschlange verwenden Sie den MQPUT-Aufruf. Nach dem einleitenden MQOPEN-Aufruf können Sie MQPUT wiederholt verwenden, um mehrere Nachrichten in die gleiche Warteschlange einzureihen. Um die Warteschlange nach dem Einreihen der Nachrichten wieder zu schließen, rufen Sie MQCLOSE auf.

Wenn Sie eine Einzelnachricht in eine Warteschlange einreihen und die Warteschlange danach sofort wieder schließen möchten, können Sie auch MQPUT1 verwenden. MQPUT1 führt die gleichen Funktionen aus wie die folgende Aufrufsequenz:

- MQOPEN
- MQPUT
- MQCLOSE

Wenn Sie mehrere Nachrichten in eine Warteschlange einreihen möchten, ist der MQPUT-Aufruf im Allgemeinen effizienter. Dies hängt allerdings auch von der Größe der Nachrichten und der verwendeten Plattform ab.

Unter den folgenden Links erhalten Sie weitere Informationen zum Einreihen von Nachrichten in eine Warteschlange:

- „Nachrichten mit dem MQPUT-Aufruf in eine lokale Warteschlange einreihen” auf Seite 793
- „Nachrichten in eine ferne Warteschlange einreihen” auf Seite 799
- „Nachrichteneigenschaften festlegen” auf Seite 799
- „Nachrichtenkontextinformationen steuern” auf Seite 799
- „Einzelnachricht mit dem MQPUT1-Aufruf in eine Warteschlange einreihen” auf Seite 802
- „Verteilerlisten” auf Seite 803
- „Fälle, in denen der PUT-Aufruf fehlschlägt” auf Seite 808

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773
Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Objekte öffnen und schließen” auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS” auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS” auf Seite 73
This information helps you to write IMS applications using IBM MQ.

Nachrichten mit dem MQPUT-Aufruf in eine lokale Warteschlange einreihen

In diesem Abschnitt erfahren Sie, wie Nachrichten mit dem MQPUT-Aufruf in eine lokale Warteschlange eingereiht werden.

Als Eingabe für den MQPUT-Aufruf ist Folgendes erforderlich:

- Eine Verbindungskennung (Hconn)
- Eine Warteschlangenkennung (Hobj)
- Eine Beschreibung der Nachricht, die Sie in die Warteschlange einreihen möchten, in Form einer Nachrichtendeskriptorstruktur (MQMD)
- Steuerinformationen in Form einer Nachrichteneinreihungsoptionsstruktur (MQPMO)

- Die Länge der in der Nachricht enthaltenen Daten (MQLONG)
- Die eigentlichen Nachrichtendaten

Die Ausgabe von MQPUT enthält folgende Komponenten:

- Einen Ursachencode (MQLONG)
- Einen Beendigungscode (MQLONG)

Nach erfolgreicher Beendigung des Aufrufs werden auch Ihre Optionsstruktur und Ihre Nachrichtendeskriptorstruktur zurückgegeben. Während des Aufrufs werden in der Optionsstruktur die Warteschlange und der Warteschlangenmanager eingetragen, an die die Nachricht gesendet wurde. Falls Sie anfordern, dass der Warteschlangenmanager einen eindeutigen Wert für die Kennung der eingereihten Nachricht generiert (durch Angabe einer binären Null im Feld *MsgId* der MQMD-Struktur), fügt der Aufruf diesen Wert vor der Rückgabe dieser Struktur im Feld *MsgId* ein. Dieser Wert muss vor dem nächsten MQPUT-Aufruf zurückgesetzt werden.

Eine genaue Beschreibung des MQPUT-Aufrufs finden Sie im Abschnitt MQPUT.

Genaue Informationen zu den für den MQPUT-Aufruf erforderlichen Eingaben finden Sie unter folgenden Links:

- „Kennungen angeben” auf Seite 794
- „Nachrichten mit der MQMD-Struktur definieren” auf Seite 794
- „Optionen in der MQPMO-Struktur angeben” auf Seite 794
- „Die Daten der Nachricht” auf Seite 797
- „Nachrichten einreihen; Nachrichtenkennungen verwenden” auf Seite 799

Kennungen angeben

Als Verbindungskennung (*Hconn*) können Sie in CICS on z/OS-Anwendungen die Konstante MQHC_DEF_HCONN (mit dem Wert null) angeben oder die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden. Für andere Anwendungen müssen Sie immer die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden.

Unabhängig von der Umgebung geben Sie als Warteschlangenkennung (*Hobj*) immer die vom MQOPEN-Aufruf zurückgegebene Kennung ein.

Nachrichten mit der MQMD-Struktur definieren

Die Nachrichtendeskriptorstruktur (MQMD) ist ein Ein-/Ausgabeparameter für MQPUT- und MQPUT1-Aufrufe. Damit definieren Sie die Nachricht, die Sie in eine Warteschlange einreihen.

Wenn für die Nachricht MQPRI_PRIORITY_AS_Q_DEF oder MQPER_PERSISTENCE_AS_Q_DEF angegeben ist und die Warteschlange eine Clusterwarteschlange ist, werden die Werte der Warteschlange verwendet, auf die sich MQPUT auflöst. Ist diese Warteschlange für MQPUT inaktiviert, schlägt der Aufruf fehl. Weitere Informationen hierzu finden Sie im Abschnitt Warteschlangenmanagercluster konfigurieren.

Anmerkung: Verwenden Sie vor dem Einreihen einer neuen Nachricht MQPMO_NEW_MSG_ID und MQPMO_NEW_CORREL_ID, um sicherzustellen, dass *MsgId* und *CorrelId* eindeutig sind. Die Werte dieser Felder werden bei einem erfolgreichen MQPUT zurückgegeben.

Eine Einführung in die in der MQMD-Struktur beschriebenen Nachrichteneigenschaften finden Sie im Abschnitt „IBM MQ-Nachrichten” auf Seite 19, eine Beschreibung der Struktur selbst im Abschnitt MQMD.

Optionen in der MQPMO-Struktur angeben

Verwenden Sie die MQPMO-Struktur (Nachrichteneinreihungsoptionsstruktur) zur Übergabe von Optionen an MQPUT- und MQPUT1-Aufrufe.

In den folgenden Abschnitten erhalten Sie Informationen zum Ausfüllen der Felder dieser Struktur. Eine Beschreibung der Struktur finden Sie im Abschnitt MQPMO.

Die Struktur enthält die folgenden Felder:

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

Diese Felder enthalten Folgendes:

StrucId
Dieses Feld kennzeichnet die Struktur als Nachrichteneinreihungsoptionsstruktur. Das Feld ist vierstellig. Geben Sie hier immer die MQPMO_STRUC_ID an.

Version
Dieses Feld enthält die Versionsnummer der Struktur. Der Standardwert ist MQPMO_VERSION_1. Bei Angabe von MQPMO_VERSION_2 können Sie Verteilerlisten verwenden (siehe „Verteilerlisten” auf Seite 803). Bei Angabe von MQPMO_VERSION_3 können Sie Nachrichtenkennungen und Nachrichteneigenschaften verwenden. Bei Angabe von MQPMO_CURRENT_VERSION verwendet Ihre Anwendung immer die aktuellste Version.

Optionen
Dieses Feld steuert Folgendes:

- Ob die Put-Operation in eine Arbeitseinheit eingeschlossen ist
- Wie viele Kontextinformationen die Nachricht enthält
- Woher die Kontextinformationen stammen
- Ob der Aufruf fehlschlägt, wenn der Warteschlangenmanager stillgelegt ist
- Ob Gruppierung oder Segmentierung möglich sind
- Die Generierung einer neuen Nachrichten-ID und Korrelations-ID
- Die Reihenfolge, in der Nachrichten und Segmente in eine Warteschlange eingereiht werden
- Ob die Namen lokaler Warteschlangen aufgelöst werden

Wenn Sie für das Feld *Options* den Standardwert (MQPMO_NONE) übernehmen, werden der eingereihten Nachricht die standardmäßigen Kontextinformationen zugeordnet.

Die Art und Weise, wie der Aufruf mit Synchronisationspunkten arbeitet, wird von der Plattform bestimmt. Der Standardwert für die Synchronisationspunktsteuerung ist yes für z/OS und no für Multiplatforms.

Context
Dieses Feld gibt die Warteschlangenkennung an, der die Kontextinformationen entnommen werden sollen (sofern im Feld *Options* angefordert).

Eine Einführung in den Nachrichtenkontext finden Sie im Abschnitt „Nachrichtenkontext” auf Seite 50. Informationen zur Verwendung der MQPMO-Struktur zur Steuerung der Kontextinformationen einer Nachricht finden Sie im Abschnitt „Nachrichtenkontextinformationen steuern” auf Seite 799.

ResolvedQName
Dieses Feld enthält (nach der Auflösung eines Aliasnamens) den Namen der Warteschlange, die zum Einreihen der Nachricht geöffnet wurde. Dies ist ein Ausgabefeld.

ResolvedQMgrName
Dieses Feld enthält den Namen des Warteschlangenmanagers (ggf. nach Auflösung eines Aliasnamens), der Eigner der Warteschlange in *ResolvedQName* ist. Dies ist ein Ausgabefeld.

Die MQPMO-Struktur kann auch für Verteilerlisten erforderliche Felder enthalten (siehe „Verteilerlisten" auf Seite 803). Wenn Sie diese Funktion nutzen möchten, müssen Sie Version 2 der MQPMO-Struktur verwenden. Dazu gehören die folgenden Felder:

RecsPresent
Dieses Feld enthält die Anzahl der Warteschlangen in der Verteilerliste (d. h. die Anzahl der Nachrichteneinreihungsdatensätze (Put Message Records, MQPMR) und der zugehörigen Antwortdatensätze (Response Records, MQRR)).

Sie können den gleichen Wert eingeben wie die Anzahl der mit MQOPEN angegebenen Objektdatensätze (Object Records). Ist der Wert allerdings niedriger als die Anzahl der mit dem MQOPEN-Aufruf bereitgestellten Objektdatensätze oder geben Sie gar keine Nachrichteneinreihungsdatensätze an, so werden die Werte der nicht definierten Warteschlangen den Standardwerten des Nachrichtendeskriptors entnommen. Ist der Wert hingegen höher als die Anzahl der bereitgestellten Objektdatensätze, so werden die überschüssigen Nachrichteneinreihungsdatensätze ignoriert.

Folgendes wird empfohlen:

- Wenn Sie von jedem Ziel einen Bericht oder eine Antwort erhalten möchten, so geben Sie den gleichen Wert ein, der auch in der MQOR-Struktur angegeben ist, und verwenden Sie MQPMRs mit *MsgId*-Feldern. Initialisieren Sie diese *MsgId*-Felder entweder mit Null oder geben Sie MQPMO_NEW_MSG_ID an.

  Nach dem Einreihen der Nachricht in die Warteschlange stehen in den MQPMRs die vom Warteschlangenmanager erstellten *MsgId*-Werte zur Verfügung. Anhand dieser Werte können Sie die Berichte bzw. Antworten den einzelnen Zielen zuordnen.
- Wenn Sie keine Berichte oder Antworten erhalten möchten, wählen Sie eine der folgenden Optionen aus:

  1. Wenn Sie Ziele ermitteln möchten, die sofort fehlschlagen, können Sie im Feld *RecsPresent* den Wert aus der MQOR-Struktur eingeben und die MQRRs zur Identifikation dieser Ziele bereitstellen. Geben Sie in diesem Fall jedoch keine MQPMRs ein.
  2. Wenn Sie die fehlgeschlagenen Ziele nicht ermitteln möchten, geben Sie im Feld *RecsPresent* Null ein und stellen Sie weder MQPMRs noch MQRRs bereit.

Anmerkung: Wenn Sie MQPUT1 verwenden, muss die Anzahl der Antwortdatensatzverweise (Response Record Pointers) und die Anzahl der Antwortdatensatzoffsets (Response Record Offsets) null sein.

Eine ausführliche Beschreibung der Nachrichteneinreihungsdatensätze (MQPMR) und der Antwortdatensätze (MQRR) finden Sie in den Abschnitten MQPMR und MQRR.

PutMsgRecFields
Dieses Feld gibt an, welche Felder in jedem Nachrichteneinreihungsdatensatz (MQPMR) vorhanden sind. Eine Liste dieser Felder finden Sie im Abschnitt „MQPMR-Struktur verwenden" auf Seite 807.

PutMsgRecOffset* und *PutMsgRecPtr
Zur Adressierung der Nachrichteneinreihungsdatensätze werden Verweise (i. d. R. in C) und Offsets (i. d. R. in COBOL) verwendet (eine Übersicht über die MQPMR-Struktur finden Sie im Abschnitt „MQPMR-Struktur verwenden" auf Seite 807).

Verwenden Sie das Feld *PutMsgRecPtr* zur Angabe eines Verweises auf den ersten Nachrichteneinreihungsdatensatz bzw. das Feld *PutMsgRecOffset* zur Angabe des Offsets des ersten Nachrichteneinreihungsdatensatzes. Es handelt sich hierbei um das Offset vom Anfang der MQPMO-Struktur. Geben Sie je nach Einstellung des Felds *PutMsgRecFields* entweder im Feld *PutMsgRecOffset* oder im Feld *PutMsgRecPtr* einen Wert ungleich null ein.

ResponseRecOffset und ResponseRecPtr
Auch für die Adressierung von Antwortdatensätzen verwenden Sie Verweise und Offsets (weitere Informationen zu Antwortdatensätzen finden Sie im Abschnitt „MQRR-Struktur verwenden" auf Seite 806).

Verwenden Sie das Feld *ResponseRecPtr* zur Angabe eines Verweises auf den ersten Antwortdatensatz bzw. das Feld *ResponseRecOffset* zur Angabe des Offsets des ersten Antwortdatensatzes. Es handelt sich hierbei um das Offset vom Anfang der MQPMO-Struktur. Geben Sie entweder im Feld *ResponseRecOffset* oder im Feld *ResponseRecPtr* einen Wert ungleich null ein.

Anmerkung: Wenn Sie MQPUT1 zum Einreihen von Nachrichten in eine Verteilerliste verwenden, muss *ResponseRecPtr* leer oder null sein und *ResponseRecOffset* muss null sein.

Version 3 der MQPMO-Struktur enthält zusätzlich die folgenden Felder:

OriginalMsgHandle
Die Verwendung dieses Felds hängt vom Wert im Feld *Action* ab. Wenn Sie eine neue Nachricht mit zugehörigen Nachrichteneigenschaften einreihen, geben Sie in diesem Feld die Nachrichtenkennung ein, die Sie zuvor erstellt haben und für die Sie Eigenschaften festgelegt haben. Wenn Sie eine Nachricht weiterleiten, auf eine Nachricht antworten oder einen Bericht zu einer zuvor abgerufenen Nachricht generieren, enthält dieses Feld die Nachrichtenkennung jener Nachricht.

NewMsgHandle
Wenn Sie eine neue Nachrichtenkennung (*NewMsgHandle*) angeben, überschreiben deren Eigenschaften sämtliche Eigenschaften der ursprünglichen Nachrichtenkennung (*OriginalMsgHandle*). Weitere Informationen finden Sie im Abschnitt Action (MQLONG).

Action
In diesem Feld geben Sie die Art der durchzuführenden Einreihung an. Die gültigen Werte lauten wie folgt:

MQACTP_NEW
Dies ist eine neue, mit noch keiner anderen Nachricht verknüpfte Nachricht.

MQACTP_FORWARD
Diese Nachricht wurde zuvor abgerufen und wird jetzt weitergeleitet.

MQACTP_REPLY
Diese Nachricht ist eine Antwort auf eine zuvor abgerufene Nachricht.

MQACTP_REPORT
Diese Nachricht ist ein Bericht zu einer zuvor abgerufenen Nachricht.

Weitere Informationen finden Sie im Abschnitt Action (MQLONG).

PubLevel
Wenn diese Nachricht eine Veröffentlichung ist, können Sie in diesem Feld festlegen, welche Subskriptionen diese Nachricht erhalten. Nur Subskriptionen mit einem *SubLevel* kleiner oder gleich diesem Wert erhalten diese Veröffentlichung. Der Standardwert ist 9 (die höchste Ebene). Bei diesem Wert erhalten alle Subskriptionen unabhängig von deren *SubLevel* die Veröffentlichung.

Die Daten der Nachricht

Geben Sie im Parameter **Buffer** des MQPUT-Aufrufs die Adresse des Puffers ein, der Ihre Daten enthält. Die Daten Ihrer Nachricht können jeden beliebigen Inhalt haben. Die Datenmenge der Nachrichten wirkt sich jedoch auf die Leistung der Anwendung aus, die die Nachrichten verarbeitet.

Die maximale Datengröße wird durch Folgendes bestimmt:

- Das Attribut **MaxMsgLength** des Warteschlangenmanagers
- Das Attribut **MaxMsgLength** der Warteschlange, in die Sie die Nachricht einreihen
- Die Größe aller durch IBM MQ hinzugefügten Nachrichtenheader (einschließlich des Headers für nicht zustellbare Nachrichten (MQDLH) und des Headers für Verteilerlisten (MQDH))

Das Attribut **MaxMsgLength** des Warteschlangenmanagers gibt die Nachrichtengröße an, die der Warteschlangenmanager verarbeiten kann. Sein Standardwert ist für alle IBM MQ-Produkte ab Version 6 100 MB.

Den Wert dieses Attributs können Sie durch Ausführung des MQINQ-Aufrufs am Warteschlangenmanagerobjekt ermitteln. Bei großen Nachrichten sollten Sie diesen Wert ändern.

Das Attribut **MaxMsgLength** einer Warteschlange gibt die maximale Größe einer Nachricht an, die in die Warteschlange eingereiht werden kann. Wenn Sie versuchen, eine größere Nachricht einzureihen, schlägt der MQPUT-Aufruf fehl. Beim Einreihen von Nachrichten in eine ferne Warteschlange wird die maximale Nachrichtengröße, die in die Warteschlange eingereiht werden kann, durch das Attribut **MaxMsgLength** der fernen Warteschlange, der Übertragungswarteschlangen auf dem Weg zum Ziel und der verwendeten Kanäle festgelegt.

Bei einer MQPUT-Operation muss die Größe der Nachricht kleiner oder gleich dem Wert des Attributs **MaxMsgLength** sowohl der Warteschlange als auch des Warteschlangenmanagers sein. Auch wenn die Werte dieser Attribute unabhängig voneinander sind, empfiehlt es sich, das Attribut *MaxMsgLength* der Warteschlange auf einen Wert kleiner oder gleich dem Wert des Warteschlangenmanagers zu setzen.

IBM MQ fügt Nachrichten unter folgenden Bedingungen Headerinformationen hinzu:

- Wenn Sie eine Nachricht in eine ferne Warteschlange einreihen, fügt IBM MQ der Nachricht eine Übertragungsheaderstruktur (MQXQH) hinzu. Diese Struktur enthält den Namen der Zielwarteschlange und den Namen deren Warteschlangenmanagers.
- Wenn IBM MQ eine Nachricht nicht an eine ferne Warteschlange zustellen kann, versucht es, die Nachricht in die Warteschlange für nicht zustellbare Nachrichten einzureihen. Dabei wird der Nachricht eine MQDLH-Struktur (Header für nicht zustellbare Nachrichten) hinzugefügt. Diese Struktur enthält den Namen der Zielwarteschlange sowie den Grund, weshalb die Nachricht in die Warteschlange für nicht zustellbare Nachrichten eingereiht wurde.
- Wenn Sie versuchen, eine Nachricht an mehrere Zielwarteschlangen zu senden, fügt IBM MQ der Nachricht einen MQDH-Header hinzu. Dieser beschreibt die Daten einer Nachricht, die sich in einer Übertragungswarteschlange befindet und zu einer Verteilerliste gehört. Berücksichtigen Sie dies bei der Auswahl des optimalen Werts für die maximale Nachrichtenlänge.
- Wenn es sich bei der Nachricht um ein Segment oder um eine Nachricht einer Gruppe handelt, fügt IBM MQ eventuell einen MQMDE-Header hinzu.

Nähere Beschreibungen dieser Strukturen finden Sie in den Abschnitten MQDH und MQMDE.

Wenn Ihre Nachrichten bereits die maximal für diese Warteschlangen zulässige Größe haben und diese Header zusätzlich hinzukommen, schlägt die Einreihung fehl, da die Nachrichten danach zu groß sind. So tragen Sie dazu bei, ein Fehlschlagen der Einreihung zu verhindern:

- Achten Sie darauf, dass Ihre Nachrichten kleiner sind, als die Einstellungen der Attribute **MaxMsgLength** der Übertragungswarteschlange und der Warteschlange für nicht zustellbare Nachrichten erlauben. Lassen Sie mindestens den Wert der Konstanten MQ_MSG_HEADER_LENGTH frei (bei großen Verteilerlisten auch mehr).
- Stellen Sie sicher, dass das Attribut **MaxMsgLength** der Warteschlange für nicht zustellbare Nachrichten auf denselben Wert gesetzt ist wie das Attribut *MaxMsgLength* des Warteschlangenmanagers, der der Eigner der Warteschlange für nicht zustellbare Nachrichten ist.

Nähere Beschreibungen der Attribute für den Warteschlangenmanager und der Konstanten für die Steuerung von Nachrichtenwarteschlangen finden Sie im Abschnitt Attribute für den Warteschlangenmanager.

z/OS Informationen zur Handhabung nicht zugestellter Nachrichten in einer Umgebung mit verteilter Steuerung von Warteschlangen finden Sie im Abschnitt Nicht zugestellte/unverarbeitete Nachrichten.

Nachrichten einreihen; Nachrichtenkennungen verwenden

In der MQPMO-Struktur stehen zwei Nachrichtenkennungen zur Verfügung: *OriginalMsgHandle* und *NewMsgHandle*. Die Beziehung zwischen diesen Nachrichtenkennungen wird durch den Wert des MQPMO-Felds *Action* definiert.

Weitere Informationen finden Sie im Abschnitt Action (MQLONG). Zum Einreihen einer Nachricht ist eine Nachrichtenkennung nicht unbedingt erforderlich. Ihr Zweck ist die Zuordnung von Eigenschaften zu einer Nachricht. Sie ist also nur erforderlich, wenn Sie Nachrichteneigenschaften verwenden.

Nachrichten in eine ferne Warteschlange einreihen

Wenn Sie eine Nachricht in eine ferne Warteschlange einreihen möchten (d. h. in eine Warteschlange, die von einem anderen Warteschlangenmanager verwaltet wird als demjenigen, mit dem die Anwendung verbunden ist), müssen Sie gegenüber der lokalen Warteschlange lediglich zusätzlich beachten, wie Sie den Namen der Warteschlange beim Öffnen angeben.Weitere Informationen hierzu finden Sie im Abschnitt „Ferne Warteschlangen öffnen” auf Seite 791. Ansonsten unterscheidet sich die Verwendung des Aufrufs MQPUT bzw. MQPUT1 nicht gegenüber einer lokalen Warteschlange.

Weitere Informationen zur Verwendung von fernen Warteschlangen und Übertragungswarteschlangen finden Sie im Abschnitt IBM MQVerfahren der verteilten Steuerung von Warteschlangen.

Nachrichteneigenschaften festlegen

Rufen Sie für jede Eigenschaft, die Sie einstellen möchten, MQSETMP auf. Beim Einreihen einer Nachricht brauchen Sie dann in der MQPMO-Struktur nur noch die Nachrichtenkennung und die Aktionsfelder festzulegen.

Zur Zuordnung von Eigenschaften zu einer Nachricht muss die Nachricht eine Nachrichtenkennung haben. Diese erstellen Sie mit einem MQCRTMH-Funktionsaufruf. Rufen Sie danach für jede Eigenschaft, die Sie für die betreffende Nachricht einstellen möchten, MQSETMP mit dieser Nachrichtenkennung auf. Die Verwendung von MQSETMP wird im Beispielprogramm amqsstma.c veranschaulicht.

Wenn es sich bei der Nachricht, die Sie mit MQPUT oder MQPUT1 in eine Warteschlange einreihen, um eine neue Nachricht handelt, setzen Sie das Feld 'OriginalMsgHandle' in der MQPMO-Struktur auf den Wert dieser Nachrichtenkennung und das MQPMO-Feld 'Action' auf MQACTP_NEW (Standardwert).

Wenn es sich um eine zuvor abgerufene Nachricht handelt, die Sie nun weiterleiten, beantworten oder einen Bericht dazu senden, fügen Sie im Feld 'OriginalMsgHandle' der MQPMO-Struktur die ursprüngliche Kennung der Nachricht und im Feld 'NewMsgHandle' die neue Nachrichtenkennung ein. Legen Sie dann im Feld 'Action' die gewünschte Aktion (MQACTP_FORWARD, MQACTP_REPLY oder MQACTP_REPORT) fest.

Wenn eine zuvor abgerufene Nachricht bereits Eigenschaften in einem MQRFH2-Header aufweist, können Sie diese mit dem Aufruf MQBUFMH in Eigenschaften der Nachrichtenkennung umwandeln.

Wenn Sie eine Nachricht in eine Warteschlange eines Warteschlangenmanagers einer Version vor IBM WebSphere MQ 7.0 einreihen, die noch keine Nachrichteneigenschaften verarbeiten kann, können Sie im Parameter PropertyControl der Kanaldefinition angeben, wie die Eigenschaften behandelt werden sollen.

Nachrichtenkontextinformationen steuern

Wenn Sie den MQPUT- oder MQPUT1-Aufruf verwenden, um eine Nachricht in eine Warteschlange einzureihen, können Sie angeben, dass der Warteschlangenmanager dem Nachrichtendeskriptor einige Standard-Kontextinformationen hinzufügen soll. Anwendungen, die über die entsprechende Berechtigungsstufe verfügen, können zusätzliche Kontextinformationen enthalten. Über das Optionenfeld in der MQPMO-Struktur können Sie die Kontextinformationen steuern.

Mithilfe der Nachrichtenkontextinformationen kann die Anwendung, die die Nachricht abruft, den Absender der Nachricht ermitteln. Alle Kontextinformationen werden in den Kontextfeldern des Nachrichtendeskriptors gespeichert. Der Informationstyp gehört zu Identität-, Ursprungs- und Benutzerkontextinformaionen

Verwenden Sie zum Steuern der Kontextinformationen das Feld *Options* in der MQPMO-Struktur.

Wenn Sie Sie keine Optionen für Kontextinformationen angeben, überschreibt der Warteschlangenmanager die Kontextinformationen, die eventuell bereits im Nachrichtendeskriptor vorhanden sind, mit den

Identität- und Kontextinformationen, die der Warteschlangenmanager für Ihre Nachricht erstellt hat. Das ist dasselbe wie die Angabe der Option MQPMO_DEFAULT_CONTEXT.Möglicherweise benötigen Sie diese Standardkontextinformationen, wenn Sie eine neue Nachricht erstellen (z. B. bei der Verarbeitung von Benutzereingaben von einer Abfrageanzeige).

Wenn Sie keine Kontextinformationen benötigen, die ihrer Nachricht zugeordnet werden, verwenden Sie die Option MQPMO_NO_CONTEXT. Beim Einreihen einer Nachricht ohne Kontext werden alle Berechtigungsprüfungen, die durch IBM MQ erfolgen, mithilfe einer leeren Benutzer-ID durchgeführt. Einer leeren Benutzer-ID kann keine explizite Berechtigung zu IBM MQ-Ressourcen zugewiesen werden; sie wird aber als Element der besonderen Gruppe "nobody" behandelt. Weitere Informationen zu dieser besonderen Gruppe nobody finden Sie im Abschnitt Referenzinformationen zu installierbaren Services.

Die Kontexteinstellung können Sie mit dem Aufruf MQOPEN, gefolgt von MQPUT mit der Option MQOO_ und der Option MQPMO_ vornehmen, wie in den folgenden Abschnitten gezeigt. Ebenso können Sie die Kontexteinstellung nur mit einem MQPUT1-Aufruf vornehmen, in welchem Fall Sie, wie nachfolgend gezeigt, lediglich die Option MQPMO_ auswählen müssen.

In den folgenden Abschnitten dieses Themas wird die Verwendung von Identitätskontext, Benutzerkontext und allem Kontext erklärt.

- „Identitätskontext übergeben” auf Seite 800
- „Benutzerkontext übergeben” auf Seite 801
- „Gesamten Kontext übergeben” auf Seite 801
- „Identitätskontext einstellen” auf Seite 801
- „Benutzerkontext einstellen” auf Seite 801
- „Gesamten Kontext einstellen” auf Seite 801

Identitätskontext übergeben

Generell sollten Programme Identitätskontextinformationen von Nachricht zu Nachricht um eine Anwendung herum übergeben, bis die Daten ihr Ziel erreichen.

Programme sollten den Ursprungskontext immer ändern, wenn die Daten geändert werden. Anwendungen, die Kontextinformationen ändern oder einstellen möchten, müssen über die entsprechende Berechtigungsstufe verfügen. Der Warteschlangenmanager prüft diese Berechtigung, wenn die Anwendungen die Warteschlange öffnen; sie müssen über die Berechtigung verfügen, um die zutreffenden Kontextoptionen für den MQOPEN-Aufruf zu verwenden.

Wenn Ihre Anwendung eine Nachricht abruft, die Daten der Nachricht verarbeitet und sie anschließend in eine andere Nachricht einreiht (möglicherweise für die Verarbeitung durch eine andere Anwendung), muss die Anwendung die Identitätskontextinformationen von der ursprünglichen Nachricht an die neue Nachricht übergeben. Sie können dem Warteschlangenmanager ermöglichen, die ursprüngliche Kontextinformation zu erstellen.

Verwenden Sie beim Öffnen der Warteschlange für den Abruf der Nachricht die Option MQOO_SAVE_ALL_CONTEXT, um die Kontextinformationen von der ursprünglichen Nachricht zu speichern. Zusätzlich zu allen anderen Optionen können Sie dies mit dem MQOPEN-Aufruf verwenden. Beachten Sie allerdings, dass Sie Kontextinformationen nicht speichern können, wenn Sie nur die Nachricht durchsuchen.

Wenn Sie die zweite Nachricht erstellen:

- Öffnen Sie die Warteschlange mithilfe der Option MQOO_PASS_IDENTITY_CONTEXT (zusätzlich zur Option MQOO_OUTPUT).
- Geben Sie im Feld *Context* der Nachrichteneinreihungsoptionsstruktur (MQPMO) die Kennung der Warteschlange an, aus der die Kontextinformationen stammen.
- Geben Sie im Feld *Options* der MQPMO-Struktur die Option MQPMO_PASS_IDENTITY_CONTEXT an.

Benutzerkontext übergeben

Es ist nicht möglich, nur Benutzerkontext zu übergeben. Wenn Sie beim Einreihen einer Nachricht Benutzerkontext übergeben möchten, geben Sie MQPMO_PASS_ALL_CONTEXT an. Alle Eigenschaften im Benutzerkontext werden genau so wie der Ursprungskontext übergeben.

Wenn eine MQPUT oder MQPUT1 auftritt und der Kontext übergeben wird, wird der gesamte Benutzerkontext von der abgerufenen Nachricht an die eingereihte Nachricht übergeben. Alle Benutzerkontexteigenschaften, die von der einreihenden Anwendung geändert wurden, werden mit ihren ursprünglichen Werten eingereiht. Alle Benutzerkontexteigenschaften, die von der einreihenden Anwendung gelöscht wurden, werden in der eingereihten Nachricht wiederhergestellt. Alle Benutzerkontexteigenschaften, die der Nachricht von der einreihenden Anwendung hinzugefügt wurden, werden beibehalten.

Gesamten Kontext übergeben

Wenn Ihre Anwendung eine Nachricht abruft und die Nachrichtendaten (unverändert) in eine andere Nachricht eingliedert, muss die Anwendung die gesamten Kontextinformationen (Identität, Ursprung und Benutzer) der ursprünglichen Nachricht an die neue Nachricht übergeben. Ein Beispiel einer Anwendung, die das tun könnte, ist ein Nachrichtenübermittler, der Nachrichten von einer Warteschlange zur anderen verschiebt.

Wenden Sie dasselbe Verfahren an, dass Sie bei der Übergabe von Identitätskontext verwenden, es sei denn, Sie verwenden die MQOPEN-Option MQOO_PASS_ALL_CONTEXT und die Put-Nachrichtenoption MQPMO_PASS_ALL_CONTEXT.

Identitätskontext einstellen

Wenn Sie die Identitätskontextinformationen für eine Nachricht einstellen möchten:

- Öffnen Sie die Warteschlange mit der Option MQOO_SET_IDENTITY_CONTEXT.
- Reihen Sie die Nachricht in der Warteschlange ein, indem Sie die Option MQPMO_SET_IDENTITY_CONTEXT angeben. Geben Sie im Nachrichtendeskriptor die Identitätskontextinformationen ein, die Sie benötigen.

Anmerkung: Wenn Sie bei einigen (allerdings nicht allen) Identitätskontextfeldern die Optionen MQOO_SET_IDENTITY_CONTEXT und MQPMO_SET_IDENTITY_CONTEXT einstellen, ist es wichtig festzustellen, dass der Warteschlangenmanager keines der anderen Felder einstellt.

Wenn Sie die Optionen für den Nachrichtenkontext ändern möchten, müssen Sie über die entsprechenden Berechtigungen zum Aufrufen des Aufrufs verfügen. Zur Ausführung von MQOO_SET_IDENTITY_CONTEXT oder MQPMO_SET_IDENTITY_CONTEXT benötigen Sie zum Beispiel die Berechtigung `+setid`.

Benutzerkontext einstellen

Wenn Sie eine Eigenschaft im Benutzerkontext einstellen möchten, stellen Sie das Kontextfeld des Nachrichteneigenschaftsdeskriptors (MQPD) MQPD_USER_CONTEXT ein, wenn Sie den MQSETMP-Aufruf ausführen.

Sie müssen über keine Sonderberechtigung verfügen, um eine Eigenschaft im Benutzerkontext einzustellen. Der Benutzerkontext hat keine MQOO_SET_*- oder MQPMO_SET_*-Kontextoptionen.

Gesamten Kontext einstellen

Wenn Sie sowohl die Identitäts- als auch die Ursprungskontextinformationen für eine Nachricht einstellen möchten:

1. Öffnen Sie die Warteschlange mit der Option MQOO_SET_ALL_CONTEXT.
2. Reihen Sie die Nachricht in der Warteschlange ein, indem Sie die Option MQPMO_SET_ALL_CONTEXT angeben. Geben Sie im Nachrichtendeskriptor die Identitäts- und Ursprungskontextinformationen ein, die Sie benötigen.

Für jede Art von Kontexteinstellung wird eine entsprechende Berechtigung benötigt.

Zugehörige Konzepte

„Nachrichtenkontext" auf Seite 50
Nachrichtenkontext-Informationen ermöglichen der Anwendung, die die Nachricht abruft, Informationen über den Absender der Nachricht zu erhalten.

Zugehörige Verweise

„MQOPEN-Optionen für den Nachrichtenkontext" auf Seite 789
Wenn Sie beim Einreihen einer Nachricht in eine Warteschlange die Möglichkeit haben möchten, der Nachricht Kontextinformationen zuzuordnen, müssen Sie beim Öffnen der Warteschlange eine der Nachrichtenkontextoptionen verwenden.

Einzelnachricht mit dem MQPUT1-Aufruf in eine Warteschlange einreihen

Verwenden Sie den MQPUT1-Aufruf, wenn die Warteschlange nach dem Einreihen einer einzelnen Nachricht sofort geschlossen werden soll. So verwendet eine Serveranwendung, die eine Antwort an jede der Warteschlangen sendet, vermutlich einen MQPUT1-Aufruf.

MQPUT1 hat die gleiche Funktion wie der Aufruf von MQOPEN gefolgt von MQPUT und einem abschließenden MQCLOSE. Der einzige Unterschied in der Syntax eines MQPUT- und eines MQPUT1-Aufrufs besteht darin, dass Sie für MQPUT eine Objektkennung angeben, während Sie für MQPUT1 wie in MQOPEN eine Objektdeskriptorstruktur (MQOD) angeben (siehe Sie).„Objekte (in der MQOD-Struktur) identifizieren" auf Seite 783). Nur so weiß der MQPUT1-Aufruf, welche Warteschlange er öffnen soll (bei MQPUT muss die Warteschlange bereits geöffnet sein).

Als Eingabe für den MQPUT1-Aufruf ist Folgendes erforderlich:

- Eine Verbindungskennung.
- Eine Beschreibung des zu öffnenden Objekts in Form einer Objektdeskriptorstruktur (MQOD)
- Eine Beschreibung der Nachricht, die Sie in die Warteschlange einreihen möchten, in Form einer Nachrichtendeskriptorstruktur (MQMD)
- Steuerinformationen in Form einer Nachrichteneinreihungsoptionsstruktur (MQPMO)
- Die Länge der in der Nachricht enthaltenen Daten (MQLONG)
- Die Adresse der Nachrichtendaten

Die Ausgabe von MQPUT1 enthält folgende Komponenten:

- Einen Beendigungscode
- Einen Ursachencode

Nach erfolgreicher Beendigung des Aufrufs werden auch Ihre Optionsstruktur und Ihre Nachrichtendeskriptorstruktur zurückgegeben. Während des Aufrufs werden in der Optionsstruktur die Warteschlange und der Warteschlangenmanager eingetragen, an die die Nachricht gesendet wurde. Falls Sie anfordern, dass der Warteschlangenmanager einen eindeutigen Wert für die Kennung der eingereihten Nachricht generiert (durch Angabe einer binären Null im Feld *MsgId* der MQMD-Struktur), fügt der Aufruf diesen Wert vor der Rückgabe dieser Struktur im Feld *MsgId* ein.

Anmerkung: MQPUT1 kann nicht mit dem Namen einer Modellwarteschlange verwendet werden; ist aber bereits eine Modellwarteschlange geöffnet, so können Sie ein MQPUT1 an die dynamische Warteschlange ausgeben.

MQPUT1 hat die folgenden sechs Eingabeparameter:

Hconn
Dies ist eine Verbindungskennung. Für CICS-Anwendungen können Sie die Konstante MQHC_DEF_HCONN (mit dem Wert null) angeben oder die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden. Für andere Programme müssen Sie immer die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden.

ObjDesc
Dies ist eine Objektdeskriptorstruktur (MQOD).

Geben Sie in den Feldern *ObjectName* und *ObjectQMgrName* den Namen der Warteschlange an, in die die Nachricht eingereiht werden soll, sowie den Namen des Warteschlangenmanagers, der Eigner dieser Warteschlange ist.

Das Feld *DynamicQName* wird im MQPUT1-Aufruf ignoriert, da dieser keine Modellwarteschlangen verwenden kann.

Im Feld *AlternateUserId* können Sie für die Überprüfung der Berechtigung zum Öffnen der Warteschlange eine alternative Benutzer-ID eingeben.

MsgDesc
Dies ist eine Nachrichtendeskriptorstruktur (MQMD). Verwenden Sie diese Struktur wie beim MQPUT-Aufruf zur Definition der Nachricht, die Sie in die Warteschlange einreihen möchten.

PutMsgOpts
Dies ist eine Nachrichteneinreihungsoptionsstruktur (MQPMO). Verwenden Sie diese Struktur wie beim MQPUT-Aufruf (siehe „Optionen in der MQPMO-Struktur angeben” auf Seite 794).

Wenn das Feld *Options* auf null gesetzt ist, verwendet der Warteschlangenmanager Ihre eigene Benutzer-ID zur Überprüfung der Berechtigung für den Zugriff auf die Warteschlange. In diesem Fall ignoriert der Warteschlangenmanager eine im Feld *AlternateUserId* der MQOD-Struktur angegebene alternative Benutzer-ID.

BufferLength
Dies ist die Nachrichtenlänge.

Buffer
Dies ist der Pufferbereich mit dem Text der Nachricht.

Bei einem Cluster funktioniert MQPUT1 genauso, als wäre die Option MQOO_BIND_NOT_FIXED gesetzt. Um herauszufinden, wohin eine Nachricht gesendet wurde, müssen Anwendungen die aufgelösten Felder der MQPMO-Struktur (nicht der MQOD-Struktur) verwenden. Weitere Informationen hierzu finden Sie im Abschnitt Warteschlangenmanagercluster konfigurieren.

Eine genaue Beschreibung des MQPUT1-Aufrufs finden Sie im Abschnitt MQPUT1.

Multi *Verteilerlisten*

Unter IBM MQ for Multiplatformskönnen Sie mit Verteilerlisten eine Nachricht in einem einzigen MQPUT- oder MQPUT1 -Aufruf an mehrere Ziele einreihen. Ein einzelner MQOPEN-Aufruf kann mehrere Warteschlangen öffnen und ein einzelner MQPUT-Aufruf kann dann jeder dieser Warteschlangen eine Nachricht zustellen. Die etwas allgemein gehaltenen Informationen aus den für diesen Prozess verwendeten MQI-Strukturen können durch spezifische Informationen zu den einzelnen Zielen der Verteilerliste ersetzt werden.

Achtung: Verteilerlisten unterstützen nicht die Verwendung von Aliaswarteschlangen, die auf Themenobjekte verweisen. Wenn eine Aliaswarteschlange auf ein Topic-Objekt in einer Verteilerliste verweist, gibt IBM MQ MQRC_ALIAS_BASE_Q_TYPE_ERROR zurück.

Bei Ausgabe eines MQOPEN-Aufrufs werden die generischen Informationen dem Objektdeskriptor (MQOD) entnommen. Wenn Sie im Feld *Version* MQOD_VERSION_2 und im Feld *RecsPresent* einen Wert größer als null angeben, kann *Hobj* als Kennung einer Liste (einer oder mehrerer Warteschlangen) statt einer einzelnen Warteschlange definiert werden. In diesem Fall werden spezifische Informationen durch die Objektdatensätze (MQORs) festgelegt, die Details zum Ziel enthalten (*ObjectName* und *ObjectQMgrName*).

Die Objektkennung (*Hobj*) wird an den MQPUT-Aufruf übergeben. Dies ermöglicht Ihnen die Einreihung von Nachrichten in die Warteschlangen einer Liste statt nur in eine einzelne Warteschlange.

Beim Einreihen einer Nachricht in die Warteschlangen (MQPUT) werden die generischen Informationen der Nachrichteneinreihungsoptionsstruktur (MQPMO) und dem Nachrichtendeskriptor (MQMD) entnommen. Die spezifischen Informationen werden in Form von Nachrichteneinreihungsdatensätzen (MQPMR) bereitgestellt.

Antwortdatensätze (MQRR) können einen Beendigungscode und einen zu jeder Zielwarteschlange spezifischen Ursachencode erhalten.

Abbildung 56 auf Seite 804 zeigt die Funktionsweise von Verteilerlisten.

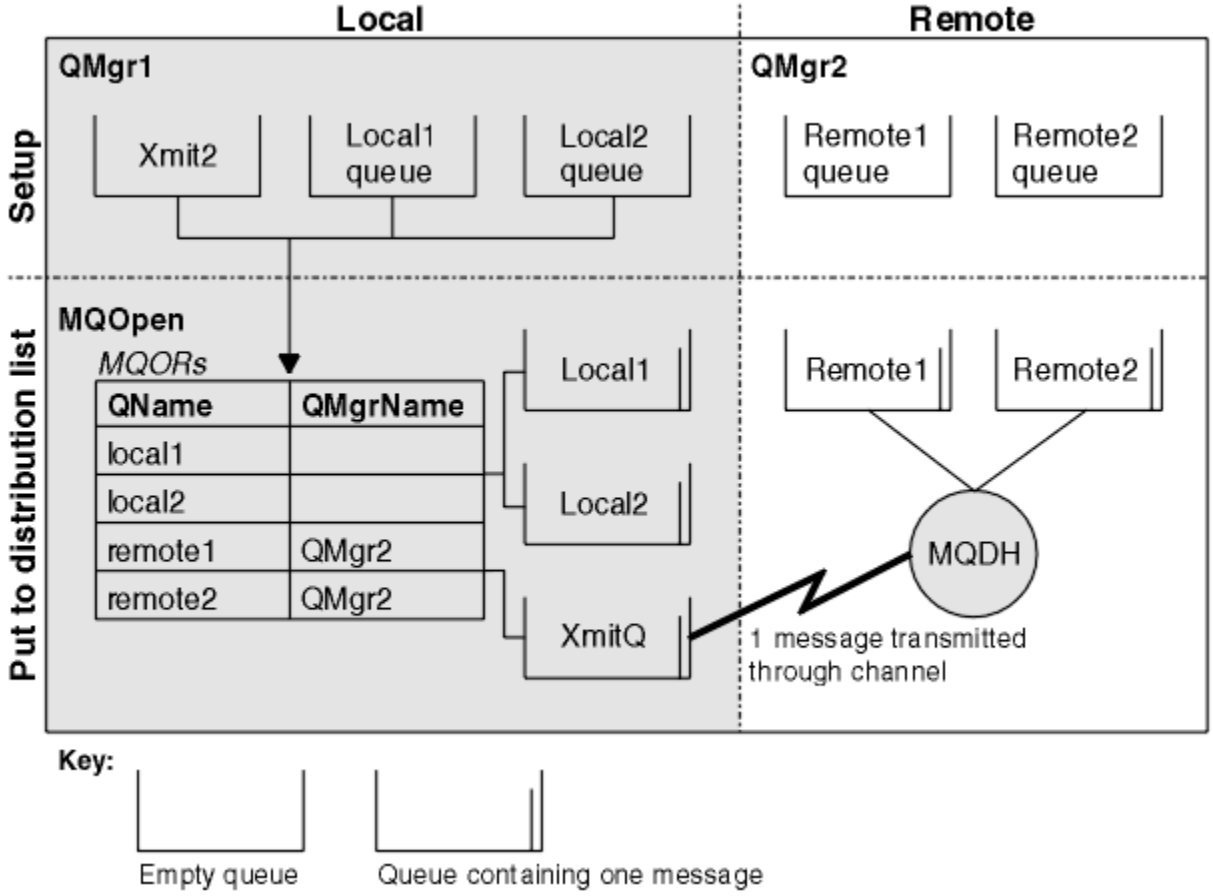


Abbildung 56. So funktionieren Verteilerlisten

Verteilerlisten öffnen
Zum Öffnen einer Verteilerliste verwenden Sie den MQOPEN-Aufruf, wobei Sie mit den Optionen des Aufrufs festlegen, was Sie mit der Liste machen möchten.

Als Eingabe für den MQOPEN-Aufruf ist Folgendes erforderlich:

- Eine Verbindungskennung (siehe „Nachrichten in eine Warteschlange einreihen” auf Seite 792)
- Generische Informationen der Objektdeskriptorstruktur (MQOD)
- Den Namen jeder zu öffnenden Warteschlange in der Objektdeskriptorstruktur (MQOR)

Die Ausgabe von MQOPEN enthält folgende Komponenten:

- Eine Objektkennung, die Ihren Zugriff auf die Verteilerliste darstellt
- Einen generischen Beendigungscode
- Einen generischen Ursachencode
- Antwortdatensätze (optional) mit einem Beendigungscode und einer Ursache für jedes Ziel

MQOD-Struktur verwenden

Die zu öffnenden Warteschlangen identifizieren Sie in der MQOD-Struktur.

Zur Definition einer Verteilerliste müssen Sie im Feld *Version* MQOD_VERSION_2, im Feld *RecsPresent* einen Wert größer als null und im Feld *ObjectType* den Wert MQOT_Q angeben. Eine Beschreibung aller Felder der MQOD-Struktur finden Sie im Abschnitt MQOD.

MQOR-Struktur verwenden

Geben Sie für jedes Ziel eine MQOR-Struktur an.

Die Struktur enthält den Namen der Zielwarteschlange und den Namen des Warteschlangenmanagers. Die Felder *ObjectName* und *ObjectQMgrName* der MQOD-Struktur werden für Verteilerlisten nicht verwendet. Es muss mindestens ein Objektdatensatz vorhanden sein. Wenn *ObjectQMgrName* leer bleibt, wird der lokale Warteschlangenmanager verwendet. Weitere Informationen zu diesen Feldern finden Sie im Abschnitt ObjectName und ObjectQMgrName.

Die Zielwarteschlangen können Sie auf zwei Weisen angeben:

- Verwenden Sie das Offset-Feld *ObjectRecOffset*.

  In diesem Fall muss die Anwendung ihre eigene Struktur mit einer MQOD-Struktur gefolgt von der Feldgruppe der MQOR-Datensätze (mit so vielen Feldgruppenelementen wie nötig) deklarieren und in *ObjectRecOffset* den Offset des ersten Elements der Feldgruppe vom Anfang der MQOD angeben. Vergewissern Sie sich, dass diese relative Position korrekt ist.

  Die Verwendung der integrierten, durch die Programmiersprache bereitgestellten Funktionen wird empfohlen, sofern diese in allen Umgebungen, in denen die Anwendung ausgeführt wird, vorhanden sind. Der folgende Code veranschaulicht dieses Verfahren für die Programmiersprache COBOL:

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  Verwenden Sie alternativ die Konstante MQOD_CURRENT_LENGTH, wenn die Programmiersprache die erforderlichen integrierten Funktionen nicht in allen relevanten Umgebungen unterstützt. Dieses Verfahren wird durch folgenden Code veranschaulicht:

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

  Dies funktioniert jedoch nur richtig, wenn die MQOD-Struktur und die Feldgruppe mit den MQOR-Datensätzen lückenlos sind; falls der Compiler zwischen der MQOD-Struktur und der MQOR-Feldgruppe Sprungbytes einfügt, so müssen diese dem Wert im Feld *ObjectRecOffset* hinzugefügt werden.

  Die Verwendung von *ObjectRecOffset* wird bei Programmiersprachen empfohlen, die den Datentyp 'Pointer' nicht unterstützen bzw. diesen Datentyp auf eine Art und Weise implementieren, die sich nicht auf andere Umgebungen übertragen lässt (wie dies z. B. in der Programmiersprache COBOL der Fall ist).

- Verwenden Sie das Zeigerfeld *ObjectRecPtr*.

  In diesem Fall kann die Anwendung die Feldgruppe der MQOR-Strukturen separat von der MQOD-Struktur deklarieren und *ObjectRecPtr* auf die Adresse der Feldgruppe setzen. Der folgende Code veranschaulicht dieses Verfahren für die Programmiersprache C:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  Die Verwendung von *ObjectRecPtr* wird bei Programmiersprachen empfohlen, die den Datentyp 'Pointer' auf eine Art und Weise unterstützen, die sich auf andere Umgebungen übertragen lässt (wie dies z. B. in der Programmiersprache C der Fall ist).

Unabhängig vom gewählten Verfahren müssen und dürfen Sie nur eines von beidem, *ObjectRecOffset* oder *ObjectRecPtr*, verwenden. Der Aufruf schlägt mit Ursachencode MQRC_OBJECT_RECORDS_ERROR fehl, wenn die Werte in beiden Feldern null bzw. in beiden Feldern ungleich null sind.

MQRR-Struktur verwenden

Diese Strukturen sind zielspezifisch; jeder Antwortdatensatz enthält ein *CompCode*- und ein *Reason*-Feld für jede Warteschlange einer Verteilerliste. Zur Ermittlung von Fehlerursachen müssen Sie diese Struktur verwenden.

Wenn Sie zum Beispiel den Ursachencode MQRC_MULTIPLE_REASONS erhalten und Ihre Verteilerliste enthält fünf Zielwarteschlangen, können Sie mit dieser Struktur ermitteln, auf welche Warteschlangen sich das gemeldete Problem bezieht. Haben Denn haben Sie einen Beendigungscode und einen Ursachencode für jedes Ziel, können Sie Fehler leichter lokalisieren.

Weitere Informationen zur MQRR-Struktur finden Sie im Abschnitt MQRR.

Abbildung 57 auf Seite 806 zeigt, wie Sie eine Verteilerliste in C öffnen.

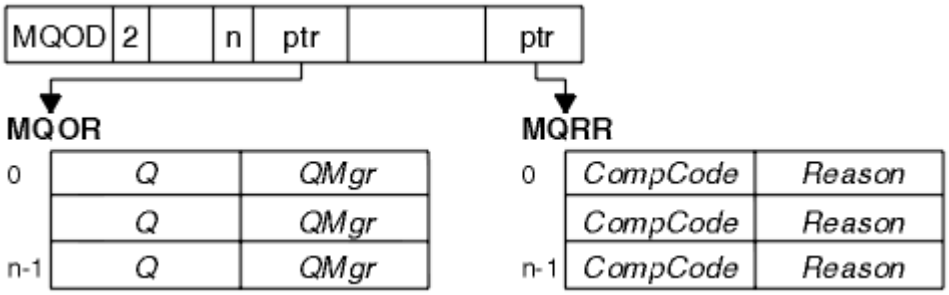


Abbildung 57. Verteilerliste in C öffnen

Abbildung 58 auf Seite 806 zeigt, wie Sie eine Verteilerliste in COBOL öffnen.

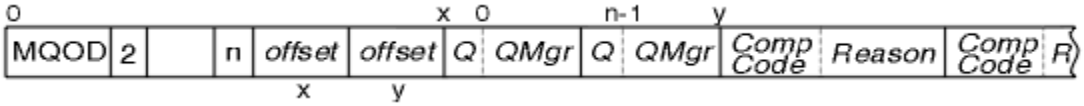


Abbildung 58. Verteilerliste in COBOL öffnen

MQOPEN-Optionen verwenden

Beim Öffnen einer Verteilerliste können Sie die folgenden Optionen angeben:

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (optional)
- MQOO_ALTERNATE_USER_AUTHORITY (optional)
- MQOO_*_CONTEXT (optional)

Eine Beschreibung dieser Optionen finden Sie im Abschnitt „Objekte öffnen und schließen” auf Seite 780.

Nachrichten in eine Verteilerliste einreihen
Zum Einreihen von Nachrichten in eine Verteilerliste können Sie MQPUT oder MQPUT1 verwenden.

Als Eingabe ist Folgendes erforderlich:

- Eine Verbindungskennung (siehe „Nachrichten in eine Warteschlange einreihen” auf Seite 792)
- Eine Objektkennung; wenn eine Verteilerliste mit MQOPEN geöffnet wurde, erlaubt die Kennung *Hobj* nur das Einreihen in die Liste
- Eine Nachrichtendeskriptorstruktur (MQMD); eine Beschreibung dieser Struktur finden Sie im Abschnitt MQMD

- Steuerinformationen in Form einer Nachrichteneinreihungsoptionsstruktur (MQPMO); Informationen zum Ausfüllen der Felder der MQPMO-Struktur finden Sie im Abschnitt „Optionen in der MQPMO-Struktur angeben” auf Seite 794
- Steuerinformationen in Form von Nachrichteneinreihungsdatensätzen (MQPMR)
- Die Länge der in der Nachricht enthaltenen Daten (MQLONG)
- Die eigentlichen Nachrichtendaten

Die Ausgabe enthält folgende Komponenten:

- Einen Beendigungscode
- Einen Ursachencode
- Antwortdatensätze (optional)

MQPMR-Struktur verwenden

Diese Struktur ist optional und stellt für einige Felder, die Sie auf andere Weise identifizieren wollen als die im MQMD identifizierten Felder, zielspezifische Informationen bereit.

Eine Beschreibung dieser Felder finden Sie im Abschnitt MQPMR.

Der Inhalt jedes Datensatzes hängt von den Informationen im Feld *PutMsgRecFields* der MQPMO-Struktur ab. Sehen Sie sich hierzu das Beispielprogramm AMQSPTL0.C an (eine Beschreibung finden Sie im Abschnitt „Das Distribution List-Beispielprogramm” auf Seite 1146), das die Verwendung von Verteilerlisten veranschaulicht. In diesem Beispiel werden die Werte für *MsgId* und *CorrelId* der MQPMR-Struktur entnommen. Der betreffende Abschnitt des Beispielprogramms sieht so aus:

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

Dies setzt jedoch voraus, dass *MsgId* und *CorrelId* für jedes Ziel einer Verteilerliste angegeben werden. Die Nachrichteneinreihungsdatensätze werden als Feldgruppe bereitgestellt.

Abbildung 59 auf Seite 807 zeigt, wie Sie in C eine Nachricht in eine Verteilerliste einreihen.

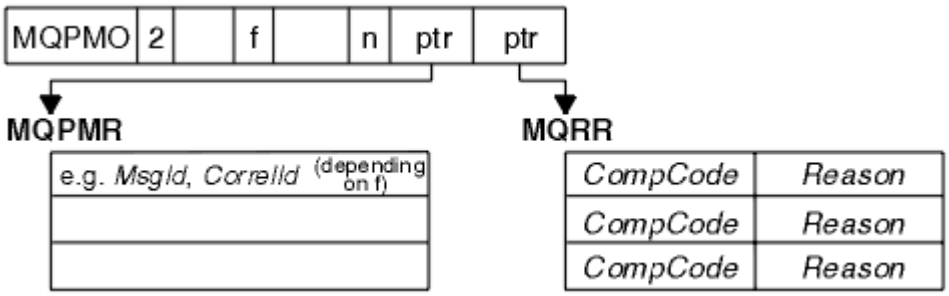


Abbildung 59. Nachricht in C in eine Verteilerliste einreihen

Abbildung 60 auf Seite 807 zeigt, wie Sie in COBOL eine Nachricht in eine Verteilerliste einreihen.

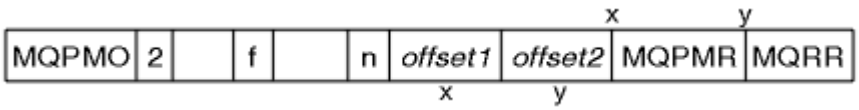


Abbildung 60. Nachricht in COBOL in eine Verteilerliste einreihen

MQPUT1 verwenden

Bei Verwendung von MQPUT1 müssen Sie Folgendes beachten:

1. Die Werte der Felder *ResponseRecOffset* und *ResponseRecPtr* müssen leer oder null sein.
2. Die Adressen der Antwortdatensätze müssen, sofern sie erforderlich sind, im MQOD angegeben sein.

Fälle, in denen der PUT-Aufruf fehlschlägt

Wenn zwischen der Ausgabe eines MQOPEN- und eines MQPUT-Aufrufs bestimmte Attribute einer Warteschlange mit der Option FORCE eines Befehls geändert werden, schlägt der MQPUT-Aufruf fehl und gibt den Ursachencode MQRC_OBJECT_CHANGED zurück.

Der Warteschlangenmanager markiert die Objektkennung als ungültig. Dies passiert auch, wenn die Änderungen während der Verarbeitung eines MQPUT1-Aufrufs vorgenommen werden oder wenn die Änderungen auf jede Warteschlange zutreffen, auf die sich der Warteschlangenname auflösen lässt. Die Attribute, die sich solchermaßen auf die Objektkennung auswirken, werden in der Beschreibung des MQOPEN-Aufrufs im Abschnitt MQOPEN aufgelistet. Falls Ihr Aufruf den Ursachencode MQRC_OBJECT_CHANGED zurückgibt, schließen Sie die Warteschlange, öffnen Sie sie erneut und wiederholen Sie dann den Einreihungsversuch.

Wenn für eine Warteschlange, in die Sie versuchen, Nachrichten einzureihen, Put-Operationen unzulässig sind (das Gleiche gilt für Warteschlangen, auf die sich der Warteschlangenname auflöst), schlägt der MQPUT- bzw. der MQPUT1-Aufruf mit dem Ursachencode MQRC_PUT_INHIBITED fehl. Möglicherweise lässt sich die Nachricht bei einer späteren Wiederholung des Aufrufs erfolgreich einreihen, wenn die Anwendung es zulässt, dass andere Programme die Warteschlangenattribute regelmäßig ändern.

Wenn die Warteschlange, in die Sie versuchen, eine Nachricht einzureihen, voll ist, schlägt der MQPUT- bzw. der MQPUT1-Aufruf mit dem Ursachencode MQRC_Q_FULL fehl.

Wenn eine dynamische (temporäre oder permanente) Warteschlange gelöscht wurde, schlagen MQPUT-Aufrufe, die eine zuvor erworbene Objektkennung verwenden, mit dem Ursachencode MQRC_Q_DELETED fehl. In diesem Fall sollten Sie die Objektkennung schließen, da sie von keinem Nutzen mehr ist.

Bei Verteilerlisten kann eine einzelne Anforderung auch mehrere Beendigungs- und Ursachencodes mit sich bringen. Diese können nicht allein durch die Ausgabefelder *CompCode* und *Reason* des MQOPEN- und MQPUT-Aufrufs verarbeitet werden.

Wenn Sie Nachrichten über Verteilerlisten mehreren Zielen zustellen, enthalten die Antwortdatensätze die spezifischen *CompCode* und *Reason*-Einträge für jedes Ziel. Wenn Sie den Beendigungscode MQCC_FAILED erhalten, wurde die Nachricht in keiner Zielwarteschlange erfolgreich eingereiht. Wenn der Beendigungscode MQCC_WARNING lautet, wurde die Nachricht in mindestens einer Zielwarteschlange erfolgreich eingereiht. Wenn Sie den Rückgabecode MQRC_MULTIPLE_REASONS erhalten, sind die Ursachencodes nicht für alle Ziele identisch. Aus diesem Grund wird die Verwendung der MQRR-Struktur empfohlen. Dieser können Sie entnehmen, welche Warteschlangen einen Fehler verursacht haben, und sofern dies der Fall ist, die Ursache für den Fehler.

Nachrichten aus einer Warteschlange abrufen

In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

Die Nachrichten einer Warteschlange können Sie auf zwei Weisen abrufen:

1. Sie können eine Nachricht aus der Warteschlange entfernen, so dass sie für andere Programme nicht mehr sichtbar ist.
2. Sie können eine Nachricht kopieren, so dass die ursprüngliche Nachricht in der Warteschlange verbleibt. Dies wird auch als *Browsing* bezeichnet. Nach dem Browsen können Sie die Nachricht entfernen.

In beiden Fällen verwenden Sie den MQGET-Aufruf. Zunächst aber muss Ihre Anwendung mit dem Warteschlangenmanager verbunden werden, und Sie müssen die Warteschlange (zur Eingabe, zum Browsen oder für beides) mit dem MQOPEN-Aufruf öffnen. Eine Beschreibung dieser Operationen finden Sie in den

Abschnitten „Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773 und „Objekte öffnen und schließen” auf Seite 780.

Nach dem Öffnen der Warteschlange können Sie den MQGET-Aufruf wiederholt verwenden, um weitere Nachrichten aus dieser Warteschlange zu entfernen bzw. um diese zu browsen. Um die Warteschlange nach dem Abrufen der Nachrichten wieder zu schließen, rufen Sie MQCLOSE auf.

Unter den folgenden Links erhalten Sie weitere Informationen zum Abrufen von Nachrichten aus einer Warteschlange:

- „Nachrichten mit dem MQGET-Aufruf aus einer Warteschlange abrufen” auf Seite 810
- „Abrufreihenfolge der Nachrichten aus einer Warteschlange” auf Seite 814
- „Bestimmte Nachricht abrufen” auf Seite 825
- „Leistung nicht persistenter Nachrichten verbessern” auf Seite 827
- z/OS „Type of index” auf Seite 831
- „Nachrichten über 4 MB verarbeiten” auf Seite 832
- „Warten auf Nachrichten” auf Seite 838
- z/OS „Signaling” auf Seite 839
- z/OS „Backout überspringen” auf Seite 840
- „Anwendungsdatenkonvertierung” auf Seite 843
- „Nachrichten in einer Warteschlange durchsuchen (Browsing)” auf Seite 844
- „Fälle, in denen der MQGET-Aufruf fehlschlägt” auf Seite 850

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773
Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Objekte öffnen und schließen” auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Objektattribute abfragen und einstellen” auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS” auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS” auf Seite 73
This information helps you to write IMS applications using IBM MQ.

Nachrichten mit dem MQGET-Aufruf aus einer Warteschlange abrufen

Der MQGET-Aufruf ruft eine Nachricht aus einer offenen lokalen Warteschlange ab. Aus den Warteschlangen anderer Systeme kann er keine Nachrichten abrufen.

Als Eingabe für den MQGET-Aufruf ist Folgendes erforderlich:

- Eine Verbindungskennung.
- Eine Warteschlangenkennung
- Eine Beschreibung der Nachricht, die Sie aus der Warteschlange abrufen möchten, in Form einer Nachrichtendeskriptorstruktur (MQMD)
- Steuerinformationen in Form einer Nachrichtenabrufoptionsstruktur (MQGMO)
- Die Größe des Puffers, den Sie zum Speichern der Nachricht zugewiesen haben (MQLONG)
- Die Adresse des für die Nachricht bestimmten Speichers

Die Ausgabe von MQGET enthält folgende Komponenten:

- Einen Ursachencode
- Einen Beendigungscode
- Die Nachricht in dem angegebenen Pufferbereich, sofern der Aufruf erfolgreich abgeschlossen wurde
- Ihre Optionsstruktur mit dem Namen der Warteschlange, aus der die Nachricht abgerufen wurde, sowie dem Namen des zugehörigen Warteschlangenmanagers
- Ihre Nachrichtendeskriptorstruktur mit Informationen zur abgerufenen Nachricht
- Die Länge der Nachricht (MQLONG)

Eine genaue Beschreibung des MQGET-Aufrufs finden Sie im Abschnitt MQGET.

In den nachfolgenden Abschnitten werden die Informationen beschrieben, die Sie als Eingaben für MQGET bereitstellen müssen.

- „Verbindungskennungen angeben” auf Seite 810
- „Nachrichten mit der MQMD-Struktur und dem MQGET-Aufruf beschreiben” auf Seite 810
- „MQGET-Optionen in der MQGMO-Struktur angeben” auf Seite 811
- „Größe des Pufferbereichs angeben” auf Seite 813

Verbindungskennungen angeben

z/OS Für CICS-Anwendungen unter z/OS können Sie die Konstante MQHC_DEF_HCONN (mit dem Wert null) angeben oder die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden. Für andere Anwendungen müssen Sie immer die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden.

Verwenden Sie die vom MQOPEN-Aufruf zurückgegebene Warteschlangenkennung (*Hobj*).

Nachrichten mit der MQMD-Struktur und dem MQGET-Aufruf beschreiben

Zur Identifizierung der Nachricht, die Sie aus einer Warteschlange abrufen möchten, verwenden Sie die Nachrichtendeskriptorstruktur (MQMD).

Dies ist ein Ein-/Ausgabeparameter für den MQGET-Aufruf. Eine Einführung in die in der MQMD-Struktur beschriebenen Nachrichteneigenschaften finden Sie im Abschnitt „IBM MQ-Nachrichten” auf Seite 19, eine Beschreibung der Struktur selbst im Abschnitt MQMD.

Wenn Sie wissen, welche Nachricht Sie aus der Warteschlange abrufen möchten, lesen Sie „Bestimmte Nachricht abrufen” auf Seite 825.

Wenn Sie keine bestimmte Nachricht angeben, ruft MQGET die *erste* Nachricht aus der Warteschlange ab. Lesen Sie auch im Abschnitt „Abrufreihenfolge der Nachrichten aus einer Warteschlange” auf Seite 814,

wie sich die Priorität einer Nachricht, das Attribut **MsgDeliverySequence** der Warteschlange und die Option MQGMO_LOGICAL_ORDER auf die Reihenfolge der Nachrichten in der Warteschlange auswirken.

Anmerkung: Wenn Sie MQGET mehrmals verwenden möchten, um zum Beispiel die Nachrichten einer Warteschlange nacheinander abzurufen, müssen Sie die Felder *MsgId* und *CorrelId* dieser Struktur nach jedem Aufruf auf null setzen. Dadurch wird die Kennung der zuvor abgerufenen Nachricht aus diesen Feldern gelöscht.

Möchten Sie Ihre Nachrichten hingegen gruppieren, muss die *GroupId* für die Nachrichten derselben Gruppe identisch sein. Der Aufruf kann dann nach Nachrichten mit derselben Kennung wie derjenigen der vorangegangenen Nachricht suchen und so die gesamte Gruppe zusammenstellen.

MQGET-Optionen in der MQGMO-Struktur angeben

Die MQGMO-Struktur ist eine Ein-/Ausgabevariable für die Übergabe von Optionen an den MQGET-Aufruf. In den folgenden Abschnitten erhalten Sie Informationen zum Ausfüllen einiger Felder dieser Struktur.

Eine Beschreibung der MQGMO-Struktur finden Sie im Abschnitt MQGMO.

StrucId
StrucId ist ein vierstelliges Feld zur Kennzeichnung der Struktur als eine Nachrichtenabfrageoptionsstruktur. Geben Sie hier immer MQGMO_STRUC_ID an.

Version
*Version*enthält die Versionsnummer der Struktur. MQGMO_VERSION_1 ist der Standardwert. Wenn Sie die Felder von Version 2 verwenden oder Nachrichten in logischer Reihenfolge abrufen möchten, geben Sie MQGMO_VERSION_2 an. Wenn Sie die Felder von Version 3 verwenden oder Nachrichten in logischer Reihenfolge abrufen möchten, geben Sie MQGMO_VERSION_3 an. Bei MQGMO_CURRENT_VERSION verwendet Ihre Anwendung immer die aktuellste Version.

Options
Innerhalb Ihres Codes können Sie die Optionen in beliebiger Reihenfolge auswählen. Im Feld *Options* wird jede Option durch ein Bit dargestellt.

Im Feld *Options* kann Folgendes festgelegt werden:

- Ob der MQGET-Aufruf den Eingang einer Nachricht in der Warteschlange abwartet, bevor er beendet wird (siehe „Warten auf Nachrichten” auf Seite 838)
- Ob die Get-Operation in eine Arbeitseinheit eingeschlossen ist
- Ob eine nicht persistente Nachricht auch außerhalb eines Synchronisationspunkts abgerufen werden kann, wodurch sich die Nachrichtenübertragung beschleunigt
- z/OS Unter IBM MQ for z/OS, ob die abgerufene Nachricht durch 'Backout überspringen' markiert ist (siehe „Backout überspringen” auf Seite 840)
- Ob die Nachricht aus der Warteschlange entfernt oder nur durchsucht wird (Browsing)
- Ob die Nachricht durch einen Anzeigecursor oder durch andere Auswahlkriterien ausgewählt wird
- Ob der Aufruf als erfolgreich gilt, auch wenn die Nachricht länger als der Puffer ist
- z/OS Unter IBM MQ for z/OS, ob der Aufruf beendet werden darf; diese Option legt auch ein Signal fest, das anzeigt, dass Sie bei Eingang einer Nachricht eine Benachrichtigung wünschen
- Ob der Aufruf fehlschlägt, wenn der Warteschlangenmanager stillgelegt ist
- z/OS Unter IBM MQ for z/OS, ob der Aufruf fehlschlägt, wenn die Verbindung stillgelegt ist
- Ob die Konvertierung der Anwendungsnachrichtendaten erforderlich ist (siehe „Anwendungsdatenkonvertierung” auf Seite 843)
- Die Reihenfolge, in der Nachrichten und Nachrichtensegmente aus einer Warteschlange abgerufen werden z/OS (außer unter IBM MQ for z/OS)
- Ob nur vollständige logische Nachrichten abgerufen werden können z/OS (außer unter IBM MQ for z/OS)

- Ob die Nachrichten einer Gruppe nur dann abgerufen werden können, wenn *alle* Nachrichten der Gruppe verfügbar sind
- Ob die Segmente einer logischen Nachricht nur dann abgerufen werden können, wenn *alle* Segmente der logischen Nachricht verfügbar sind z/OS (außer unter IBM MQ for z/OS)

Wenn Sie für das Feld *Options* den Standardwert (MQGMO_NO_WAIT) belassen, funktioniert der MQGET-Aufruf wie folgt:

- Wenn die Warteschlange keine Nachricht mit Ihren Auswahlkriterien enthält, wartet der Aufruf nicht auf den Eingang einer Nachricht, sondern wird sofort beendet. z/OS Unter IBM MQ for z/OS wird auch kein Signal gesetzt, das anzeigt, dass Sie bei Eingang einer Nachricht eine Benachrichtigung wünschen.
- Der Umgang des Aufrufs mit Synchronisationspunkten wird durch die Plattform gesteuert:

Plattform	**Unter Synchronisationspunktsteuerung**
IBM i	Nein
Systeme mit AIX and Linux	Nein
z/OS z/OS z/OS	Ja
Systeme mit Windows	Nein

- z/OS Unter IBM MQ for z/OS wird die abgerufene Nachricht nicht durch 'Backout überspringen' markiert.
- Die ausgewählte Nachricht wird aus der Warteschlange entfernt (nicht durchsucht).
- Es ist keine Konvertierung der Anwendungsnachrichtendaten erforderlich.
- Der Aufruf schlägt fehl, wenn die Nachricht länger als der Puffer ist.

WaitInterval
Das Feld *WaitInterval* gibt die maximale Zeit in Millisekunden an, die ein MQGET-Aufruf bei Verwenden der Option MQGMO_WAIT auf den Eingang einer Nachricht in der Warteschlange wartet. Wenn innerhalb der in *WaitInterval* angegebenen Zeit keine Nachricht eingeht, wird der Aufruf mit einem Ursachencode beendet, der angibt, dass in der Warteschlange keine Nachricht mit den Auswahlkriterien vorhanden war.

z/OS Wenn Sie unter IBM MQ for z/OSdie Option MQGMO_SET_SIGNAL verwenden, gibt das Feld *WaitInterval* die Zeit an, für die das Signal gesetzt wird.

Weitere Informationen zu diesen Optionen finden Sie in „Warten auf Nachrichten” auf Seite 838 z/OS und „Signaling” auf Seite 839 .

z/OS ***Signal1***
Signal1 wird nur unter IBM MQ for z/OSunterstützt.

Wenn Sie mit der Option MQGMO_SET_SIGNAL bei Eingang einer geeigneten Nachricht eine Benachrichtigung für Ihre Anwendung anfordern, geben Sie den Signaltyp im Feld *Signal1* an. In IBM MQ auf allen anderen Plattformen ist das Feld *Signal1* reserviert und sein Wert ist nicht signifikant.

z/OS Weitere Informationen finden Sie unter „Signaling” auf Seite 839.

Signal2
Das Feld *Signal2* ist auf allen Plattformen reserviert und sein Wert ohne Bedeutung.

z/OS Weitere Informationen finden Sie unter „Signaling” auf Seite 839.

ResolvedQName
ResolvedQName ist ein Ausgabefeld, in dem der Warteschlangenmanager (ggf. nach Auflösung eines Aliasnamens) den Namen der Warteschlange zurückgibt, aus der die Nachricht abgerufen wurde.

MatchOptions
MatchOptions legt die Auswahlkriterien für MQGET fest.

GroupStatus
GroupStatus gibt an, ob die abgerufene Nachricht zu einer Gruppe gehört.

SegmentStatus
SegmentStatus gibt an, ob das abgerufene Element zu einer logischen Nachricht gehört.

Segmentation
Segmentation gibt an, ob für die abgerufene Nachricht eine Segmentierung erlaubt ist.

MsgToken

MsgToken kennzeichnet die Nachricht eindeutig.

ReturnedLength
ReturnedLength ist ein Ausgabefeld, in dem der Warteschlangenmanager die Länge der Nachrichtendaten (in Byte) zurückgibt.

MsgHandle
Die Kennung für eine Nachricht, die mit den Eigenschaften der Nachricht belegt wird, die aus der Warteschlange abgerufen wird. Die Kennung wurde zuvor durch einen MQCRTMH-Aufruf erstellt. Alle Eigenschaften, die bereits der Kennung zugewiesen sind, werden vor dem Abrufen der Nachricht gelöscht.

Größe des Pufferbereichs angeben

Geben Sie im Parameter **BufferLength** des MQGET-Aufrufs die Größe des Pufferbereichs an, in dem die abgerufenen Nachrichtendaten gespeichert werden sollen. Die Größe kann auf drei Arten bestimmt werden:

1. Vielleicht wissen Sie bereits, wie groß die Nachrichten dieses Programms in der Regel sind. Geben Sie in diesem Fall einen Puffer mit dieser Größe an.

   Wenn der MQGET-Aufruf abgeschlossen werden soll, auch wenn die Nachricht für den Puffer zu groß ist, können Sie jedoch die Option MQGMO_ACCEPT_TRUNCATED_MSG der MQGMO-Struktur verwenden. In diesem Fall geschieht Folgendes:

   - Es werden so viele Nachrichtendaten wie möglich in den Puffer gefüllt
   - Der Aufruf gibt als Beendigungscode eine Warnung zurück
   - Die Nachricht wird aus der Warteschlange entfernt (der Rest der Nachricht wird verworfen) oder der Anzeigecursor wird vorgerückt (beim Browsen der Warteschlange)
   - Die tatsächliche Länge der Nachricht wird in *DataLength* zurückgegeben.

   Ohne diese Option wird der Aufruf zwar auch mit einer Warnung beendet, jedoch wird die Nachricht nicht aus der Warteschlange entfernt (bzw. der Anzeigecursor wird nicht vorgerückt).
2. Geben Sie eine geschätzte Puffergröße oder evtl. sogar null Byte an und verwenden Sie dafür die Option MQGMO_ACCEPT_TRUNCATED_MSG *nicht*. Bei einem Fehlschlagen von MQGET (z. B. weil der Puffer zu klein war) wird die Länge der Nachricht im Parameter **DataLength** des Aufrufs zurückgegeben. (Auch jetzt werden so viele Nachrichtendaten wie möglich in den Puffer gefüllt, allerdings wird die Verarbeitung des Aufrufs nicht abgeschlossen.) Speichern Sie die *MsgId* dieser Nachricht und wiederholen Sie dann den MQGET-Aufruf mit einem ausreichend großen Pufferbereich und der *MsgId* aus dem ersten Aufruf.

   Wenn Ihr Programm einer Warteschlange Nachrichten zustellt, die auch von anderen Programmen verwendet wird, kann es passieren, dass eines dieser anderen Programme eine von Ihnen gewünschte Nachricht daraus entfernt, bevor Ihr Programm einen weiteren MQGET-Aufruf ausgeben kann. Ihr Programm vergeudet dann auf der Suche nach einer Nachricht, die gar nicht mehr vorhanden ist, sinnlos

Zeit. Um dies zu vermeiden, empfiehlt es sich, die Warteschlange zunächst mit *BufferLength* gleich null und der Option MQGMO_ACCEPT_TRUNCATED_MSG nach der gewünschten Nachricht zu durchsuchen. Dadurch wird der Anzeigecursor unter die gewünschte Nachricht gesetzt. Danach können Sie die Nachricht mit einem erneuten MQGET, dieses Mal aber mit der Option MQGMO_MSG_UNDER_CURSOR, abrufen. Falls nun ein anderes Programm die Nachricht zwischen diesen beiden MQGET-Aufrufen entfernt, schlägt der zweite MQGET-Aufruf sofort fehl (ohne nochmalige Durchsuchung der Warteschlange), da sich unter dem Anzeigecursor keine Nachricht befindet.

3. Das Attribut *MaxMsgLength* einer *Warteschlange* legt die maximale Länge der von dieser Warteschlange akzeptierten Nachrichten fest. Das Attribut *MaxMsgLength* des *Warteschlangenmanagers* legt hingegen die maximale Länge der von diesem Warteschlangenmanager akzeptierten Nachrichten fest. Wenn Sie die erwartete Nachrichtenlänge nicht kennen, können Sie das Attribut **MaxMsgLength** mit dem MQINQ-Aufruf abfragen und dann einen Puffer mit dieser Größe festlegen.

   Die Puffergröße sollte aus Leistungsgründen so nahe wie möglich an der tatsächliche Nachrichtengröße liegen.

   Weitere Informationen zum Attribut **MaxMsgLength** finden Sie im Abschnitt „Maximale Nachrichtenlänge heraufsetzen" auf Seite 832.

Abrufreihenfolge der Nachrichten aus einer Warteschlange

Sie können die Reihenfolge steuern, in der Nachrichten aus einer Warteschlange abgerufen werden. In diesem Abschnitt erhalten Sie nähere Informationen zu den einzelnen Optionen.

Priority

Ein Programm kann einer Nachricht beim Einreihen in eine Warteschlange eine Priorität zuweisen (siehe „Nachrichtenprioritäten" auf Seite 27). Nachrichten mit gleicher Priorität werden in der Warteschlange in ihrer Eingangsreihenfolge gespeichert, nicht in ihrer Commitreihenfolge.

Der Warteschlangenmanager verwaltet Warteschlangen in Abhängigkeit des Warteschlangenattributs **MsgDeliverySequence** in strikter FIFO-Sequenz (First In - First Out) oder in FIFO-Sequenz mit Prioritäten. Sobald eine Nachricht in einer Warteschlange eintrifft, wird sie unmittelbar nach der zuletzt eingefügten Nachricht mit derselben Priorität eingefügt.

Programme können aus einer Warteschlange entweder die erste Nachricht oder eine bestimmte Nachricht unabhängig von deren Priorität abrufen. Es kann zum Beispiel sein, dass ein Programm die Antwort auf eine bestimmte Nachricht bearbeiten möchte, die es zuvor gesendet hat. Weitere Informationen finden Sie unter „Bestimmte Nachricht abrufen" auf Seite 825.

Wenn eine Anwendung eine komplette Nachrichtenfolge in eine Warteschlange einreiht, kann eine andere Anwendung diese Nachrichten unter den folgenden Voraussetzungen in der gleichen Reihenfolge abrufen, in der sie eingereiht wurden:

- Die Nachrichten haben alle die gleiche Priorität
- Die Nachrichten wurden alle innerhalb der gleichen Arbeitseinheit oder unabhängig von Arbeitseinheiten eingereiht
- Es handelt sich um eine lokale Warteschlange der einreihenden Anwendung

Wenn diese Bedingungen nicht zutreffen, die Anwendungen die Nachrichten aber in einer bestimmten Reihenfolge abrufen müssen, müssen die Anwendungen den Nachrichtendaten entweder Sequenzinformationen hinzufügen oder eine Methode einrichten, die den Empfang einer Nachricht bestätigt, bevor die nächste Nachricht gesendet wird.

z/OS In IBM MQ for z/OS können Sie die Geschwindigkeit von MQGET-Operationen aus einer Warteschlange mit dem Warteschlangenattribut *IndexType* beschleunigen. Weitere Informationen finden Sie in „Type of index" auf Seite 831.

Logische und physische Reihenfolge

Innerhalb jeder Prioritätsstufe können Nachrichten in Warteschlangen in *physischer* oder *logischer* Reihenfolge auftreten.

Die physische Reihenfolge ist die Reihenfolge, in der die Nachrichten in einer Warteschlange eintreffen. Eine logische Reihenfolge besteht, wenn sich alle Nachrichten und Segmente innerhalb einer Gruppe nacheinander in ihrer logischen Sortierung in der Position befinden, die von der physischen Position des ersten Elements, das zur Gruppe gehört, festgelegt wird.

Eine Beschreibung von Gruppen, Nachrichten und Segmenten finden Sie im Abschnitt „Nachrichtengruppen” auf Seite 46. Diese physischen und logischen Anordnungen können aus folgenden Gründen Unterschiede aufweisen:

- Gruppen können bei einem Ziel zu ähnlichen Zeiten von verschiedenen Anwendungen kommend eintreffen und verlieren somit jegliche eindeutige physische Reihenfolge.
- Sogar innerhalb einer einzelnen Gruppe kann sich die Reihenfolge der Nachrichten durch Umleitung oder Verzögerung einiger Nachrichten in der Gruppe ändern.

Die logische Reihenfolge kann beispielsweise wie Abbildung Abbildung 61 auf Seite 815 aussehen:

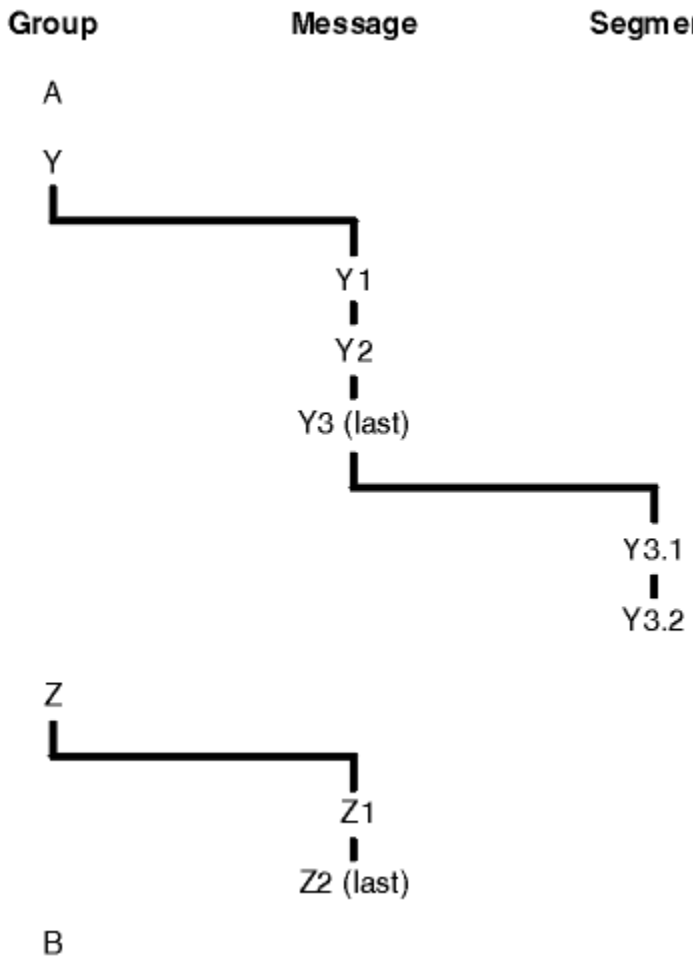


Abbildung 61. Logische Reihenfolge in einer Warteschlange

Diese Nachrichten können in der folgenden logischen Reihenfolge in einer Warteschlange auftreten:

1. Nachricht A (nicht in einer Gruppe)
2. Logische Nachricht 1 aus Gruppe Y
3. Logische Nachricht 2 aus Gruppe Y
4. Segment 1 von der (letzten) logischen Nachricht 3 aus Gruppe Y
5. (Letztes) Segment 2 von (letzter) logischer Nachricht 3 aus Gruppe Y
6. Logische Nachricht 1 aus Gruppe Z
7. (Letzte) logische Nachricht 2 aus Gruppe Z

8. Nachricht B (nicht in einer Gruppe)

Die physische Reihenfolge könnte allerdings ganz anders aussehen. Die physische Position des *ersten* Elements innerhalb jeder Gruppe bestimmt die logische Position der gesamten Gruppe. Wenn z. B. die Gruppen Y und Z zu ähnlichen Zeiten eintreffen und Nachricht 2 aus Gruppe Z Nachricht 1 überholt, sieht die Reihenfolge aus wie Abbildung Abbildung 62 auf Seite 816:

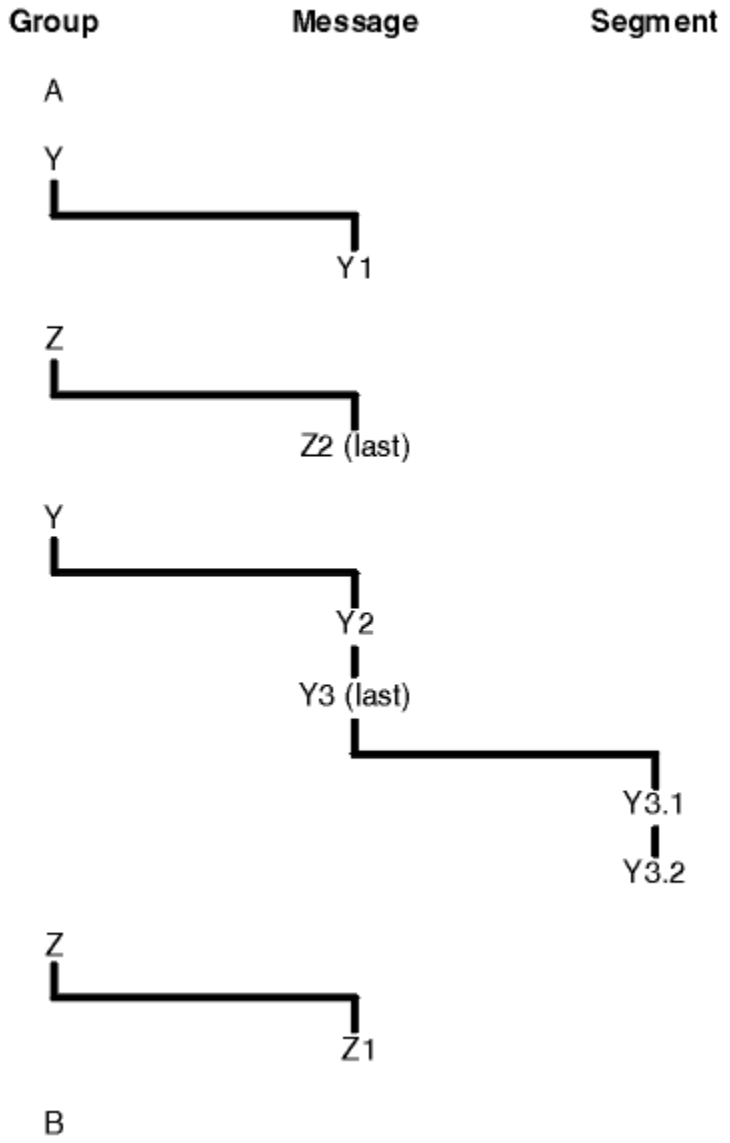


Abbildung 62. Physische Reihenfolge in einer Warteschlange

Diese Nachrichten treten in der Warteschlange in der folgenden logischen Reihenfolge auf:

1. Nachricht A (nicht in einer Gruppe)
2. Logische Nachricht 1 aus Gruppe Y
3. Logische Nachricht 2 aus Gruppe Z
4. Logische Nachricht 2 aus Gruppe Y
5. Segment 1 von der (letzten) logischen Nachricht 3 aus Gruppe Y
6. (Letztes) Segment 2 von (letzter) logischer Nachricht 3 aus Gruppe Y
7. Logische Nachricht 1 aus Gruppe Z
8. Nachricht B (nicht in einer Gruppe)

Anmerkung: z/OS In IBM MQ for z/OS wird die physische Reihenfolge der Nachrichten in der Warteschlange nicht garantiert, wenn die Warteschlange durch die Gruppen-ID indiziert wird.

Beim Abrufen von Nachrichten können Sie MQGMO_LOGICAL_ORDER angeben, um Nachrichten in logischer Reihenfolge anstatt in physischer Reihenfolge abzurufen.

Wenn Sie einen MQGET-Aufruf mit MQGMO_BROWSE_FIRST und MQGMO_LOGICAL_ORDER ausgeben, müssen nachfolgende MQGET-Aufrufe mit MQGMO_BROWSE_NEXT auch MQGMO_LOGICAL_ORDER angeben. Wenn das MQGET mit MQGMO_BROWSE_FIRST umgekehrt nicht MQGMO_LOGICAL_ORDER angibt, müssen die folgenden MQGETs mit MQGMO_BROWSE_NEXT es ebenso wenig angeben.

Die Gruppen- und Segmentinformationen, die der Warteschlangenmanager für MQGET-Aufrufe beibehält, die Nachrichten in der Warteschlange durchsuchen, decken sich nicht mit den Gruppen- und Segmentinformationen, die der Warteschlangenmanager für MQGET-Aufrufe beibehält, die Nachrichten aus der Warteschlangen entfernen. Wenn Sie MQGMO_BROWSE_FIRST angeben, ignoriert der Warteschlangenmanager die Gruppen- und Segmentinformationen für das Durchsuchen und scannt die Warteschlange, als ob es keine aktuelle Gruppe und keine aktuelle logische Nachricht gäbe.

Anmerkung: Verwenden Sie keinen MQGET-Aufruf, um einen Suchvorgang *über das Ende* der Nachrichtengruppe (oder einer logischen Nachricht, die nicht in einer Gruppe ist) hinaus durchzuführen, ohne MQGMO_LOGICAL_ORDER anzugeben. Wenn beispielsweise die letzte Nachricht in der Gruppe *vor* der ersten Nachrichten in der Warteschlangegruppe steht, führt ein Suchvorgang mit MQGMO_BROWSE_NEXT über das Ende der Gruppe hinaus, bei dem MQMO_MATCH_MSG_SEQ_NUMBER angegeben wird und der Parameter *MsgSeqNumber* auf 1 gesetzt ist (um die erste Nachricht in der nächsten Gruppe zu suchen) dazu, das erneut die erste Nachricht in der bereits durchsuchten Gruppe zurückgegeben wird. Dieser Effekt kann sofort eintreten oder mehrere MQGET-Aufrufe später (wenn Zwischengruppen vorhanden sind).

Vermeiden Sie, dass durch *zweifaches* Öffnen der Warteschlange für den Suchvorgang die Möglichkeit einer unendlichen Schleife entsteht:

- Verwenden Sie die erste Kennung, um nur die erste Nachricht in jeder Gruppe anzuzeigen.
- Verwenden Sie die zweite Kennung, um nur die Nachrichten in einer bestimmten Gruppe anzuzeigen.
- Verwenden Sie die MQMO_*-Optionen, um den zweiten Anzeigecursor zur Position des ersten Anzeigecursors zu bewegen, bevor die Nachrichten in der Gruppe durchsucht werden.
- Verwenden Sie MQGMO_BROWSE_NEXT nicht, um einen Suchvorgang über das Ende einer Gruppe hinaus durchzuführen.

Weitere Informationen hierzu finden Sie in den Abschnitten MQGET, MQMD und Regeln zur Überprüfung von MQI-Optionen.

Bei den meisten Anwendungen werden Sie wahrscheinlich entweder eine logische oder physische Reihenfolge für den Suchvorgang wählen. Wenn Sie allerdings zwischen diesen beiden Modi wechseln wollen, beachten Sie, dass Ihre Position innerhalb der logischen Reihenfolge bei der ersten Ausgabe eines Suchvorgangs mit QGMO_LOGICAL_ORDER hergestellt wird.

Wenn das erste Element innerhalb der Gruppe zu diesem Zeitpunkt nicht vorhanden ist, gilt die Gruppe, in der Sie sind, nicht als Teil der logischen Reihenfolge.

Sobald sich der Anzeigecursor innerhalb einer Gruppe befindet, kann er innerhalb derselben Gruppe fortfahren, sogar wenn die erste Nachricht entfernt wird. Zu Beginn können Sie jedoch nie in eine Gruppe mithilfe von MQGMO_LOGICAL_ORDER hineingehen, wenn das erste Element nicht vorhanden ist.

MQPMO_LOGICAL_ORDER

Über die Option MQPMO teilt die Anwendung dem Warteschlangenmanager mit, wie sie Nachrichten in Gruppen und Segmente von logischen Nachrichten einreiht. Sie kann nur für den Aufruf MQPUT angegeben werden; für den Aufruf MQPUT1 ist sie nicht gültig.

Mit MQPMO_LOGICAL_ORDER wird angegeben, dass die Anwendung aufeinanderfolgende MQPUT-Aufrufe für Folgendes verwendet:

1. Segmente werden in jede logische Nachricht nach Systemoffset in aufsteigender Reihenfolge (beginnend bei 0 und ohne Lücken) eingefügt.

2. Alle Segmente werden zunächst vollständig in eine logische Nachricht eingefügt, bevor sie in die nächste logische Nachricht eingefügt werden.
3. Die logischen Nachrichten werden nach Nachrichtenfolgenummer in aufsteigender Reihenfolge (beginnend bei 1 und ohne Lücken) in die einzelnen Nachrichtengruppen eingefügt. IBM MQ erhöht die Nachrichtenfolgenummer automatisch.
4. Alle logischen Nachrichten werden zunächst vollständig in eine Nachrichtengruppe eingefügt, bevor sie in die nächste Nachrichtengruppe eingefügt werden.

Da die Anwendung dem Warteschlangenmanager mitgeteilt hat, wie sie Nachrichten in Gruppen und Segmenten logischer Nachrichten einreiht, muss die Anwendung die Gruppen- und Segmentinformationen über jeden MQPUT-Aufruf nicht pflegen und aktualisieren, weil der Warteschlangenmanager diese Informationen pflegt und aktualisiert. Insbesondere bedeutet dies, dass die Anwendung die Felder *GroupId*, *MsgSeqNumber* und *Offset* nicht in MQMD einstellen muss, da der Warteschlangenmanagergruppe diese Felder automatisch auf die richtigen Werte setzt. Die Anwendung muss in MQMD lediglich das Feld *MsgFlags* einstellen, um anzugeben, ob Nachrichten zu einer Gruppe gehören oder Segmente logischer Nachrichten sind, und um die letzte Nachricht einer Gruppe bzw. das letzte Segment einer logischen Nachricht anzugeben.

Nach dem Start einer Nachrichtengruppe oder logischen Nachricht müssen alle nachfolgenden MQPUT-Aufrufe in MQMD im Feld *MsgFlags* die entsprechenden MQMF_*-Flags angeben. Wenn die Anwendung eine Nachricht einzureihen versucht, die nicht einer Gruppe ist, wenn es eine nicht abgeschlossene Nachrichtengruppe gibt, oder eine Nachricht einreiht, die kein Segment ist, wenn es eine nicht abgeschlossene Nachrichtengruppe gibt, schlägt der Aufruf mit dem Ursachencode MQRC_INCOMPLETE_GROUP bzw. ggf. MQRC_INCOMPLETE_MSG fehl. Der Warteschlangenmanager behält jedoch die Informationen über die aktuelle Nachrichtengruppe oder die aktuelle logische Nachricht bei und die Anwendung kann sie beenden, indem sie eine Nachricht sendet (möglicherweise ohne Anwendungsnachrichtendaten). Darin wird MQMF_LAST_MSG_IN_GROUP bzw. MQMF_LAST_SEGMENT angegeben, bevor der MQPUT-Aufruf erneut ausgegeben wird, um die Nachricht einzureihen, die nicht in der Gruppe und kein Segment ist.

Abbildung 62 auf Seite 816 zeigt die gültigen Options- und Flag-Kombinationen sowie die vom Warteschlangenmanager im jeweiligen Fall verwendeten Werte der Felder *GroupId*, *MsgSeqNumber* und *Offset*. Kombinationen aus Optionen und Flags, die nicht in der Tabelle aufgeführt sind, sind nicht gültig. Die Spalten in der Tabelle haben folgende Bedeutungen; Beides bedeutet Ja oder Nein:

LOG ORD
Zeigt an, ob die Option MQPMO_LOGICAL_ORDER beim Aufruf angegeben wird.

MIG
Zeigt an, ob die Option MQMF_MSG_IN_GROUP oder MQMF_LAST_MSG_IN_GROUP beim Aufruf angegeben wird.

SEG
Zeigt an, ob die Option MQMF_SEGMENT oder MQMF_LAST_SEGMENT beim Aufruf angegeben wird.

SEG OK
Zeigt an, ob die Option MQMF_SEGMENTATION_ALLOWED beim Aufruf angegeben wird.

Cur grp
Zeigt an, ob eine aktuelle Nachrichtengruppe vor dem Aufruf vorhanden ist.

Cur log msg
Zeigt an, ob eine aktuelle logische Nachricht vor dem Aufruf vorhanden ist.

Sonstige Spalten
Zeigen die vom Warteschlangenmanager verwendeten Werte. Vorherige zeigt den Wert an, der für das Feld in der vorherigen Nachricht für die Warteschlangenkennung verwendet wird.

Tabelle 116. MQPUT-Optionen zu Nachrichten in Gruppen und Segmenten von logischen Nachrichten.

Angegebene Option	Angegebene Option	Angegebene Option	Angegebene Option	Gruppen- und log-msg-Status vor dem Aufruf	Gruppen- und log-msg-Status vor dem Aufruf	Vom Warteschlangenmanager verwendete Werte	Vom Warteschlangenmanager verwendete Werte	Vom Warteschlangenmanager verwendete Werte
LOG ORD	MIG	SEG	SEG OK	Cur grp	Cur log msg	**GroupId**	**MsgSeqNumber**	**Offset**
Ja	Nein	Nein	Nein	Nein	Nein	MQGI_NONE	1	0
Ja	Nein	Nein	Ja	Nein	Nein	Neue Gruppen-ID	1	0
Ja	Nein	Ja	Eines	Nein	Nein	Neue Gruppen-ID	1	0
Ja	Nein	Ja	Eines	Nein	Ja	Vorige Gruppen-ID	1	Voriger Offset + vorige Segmentlänge
Ja	Ja	Eines	Eines	Nein	Nein	Neue Gruppen-ID	1	0
Ja	Ja	Eines	Eines	Ja	Nein	Vorige Gruppen-ID	Vorige Folgenummer + 1	0
Ja	Ja	Ja	Eines	Ja	Ja	Vorige Gruppen-ID	Vorige Folgenummer	Voriger Offset + vorige Segmentlänge
Nein	Nein	Nein	Nein	Eines	Eines	MQGI_NONE	1	0
Nein	Nein	Nein	Ja	Eines	Eines	Neue Gruppen-ID, falls MQGI_NONE; andernfalls der Wert im Feld	1	0
Nein	Nein	Ja	Eines	Eines	Eines	Neue Gruppen-ID, falls MQGI_NONE; andernfalls der Wert im Feld	1	Wert in Feld
Nein	Ja	Nein	Eines	Eines	Eines	Neue Gruppen-ID, falls MQGI_NONE; andernfalls der Wert im Feld	Wert in Feld	0
Nein	Ja	Ja	Eines	Eines	Eines	Neue Gruppen-ID, falls MQGI_NONE; andernfalls der Wert im Feld	Wert in Feld	Wert in Feld

Anmerkung:

- MQPMO_LOGICAL_ORDER ist im MQPUT1-Aufruf nicht gültig.

- Falls MQPMO_NEW_MSG_ID oder MQMI_NONE angegeben ist, generiert der Warteschlangenmanager für das Feld *MsgId* eine neue Nachrichten-ID; andernfalls verwendet er den im Feld angegebenen Wert.
- Falls MQPMO_NEW_CORREL_ID angegeben ist, generiert der Warteschlangenmanager für das Feld *CorrelId* eine neue Korrelations-ID; andernfalls verwendet er den im Feld angegebenen Wert.

Bei Angabe von MQPMO_LOGICAL_ORDER setzt der Warteschlangenmanager voraus, dass alle Nachrichten einer Gruppe bzw. alle Segmente einer logischen Nachricht mit dem gleichen Wert im MQMD-Feld *Persistence* eingereiht werden, d. h., alle Nachrichten bzw. Segmente müssen persistent oder alle müssen nicht persistent sein. Ist dies nicht der Fall, schlägt der MQPUT-Aufruf mit dem Ursachencode MQRC_INCONSISTENT_PERSISTENCE fehl.

Die Option MQPMO_LOGICAL_ORDER beeinträchtigt die Arbeitseinheiten wie folgt:

- Wenn die erste physische Nachricht in einer Gruppe oder logischen Nachricht innerhalb einer Arbeitseinheit eingereiht wird, müssen alle anderen physischen Nachrichten in der Gruppe oder logischen Nachricht innerhalb der Arbeitseinheit eingereiht werden, wenn dieselbe Warteschlangenkennung verwendet wird. Sie müssen jedoch nicht innerhalb derselben Arbeitseinheit eingereiht werden, um einer Nachrichtengruppe oder logischen Nachricht, die viele physische Nachrichten beinhaltet, eine Aufteilung über mindestens zwei aufeinanderfolgende Einheiten für die Warteschlangenkennung zu ermöglichen.
- Wenn die erste physische Nachricht in einer Arbeitseinheit oder logischen Nachricht nicht innerhalb einer Arbeitseinheit eingereiht wird, so kann keine der weiteren physischen Nachrichten der Gruppe oder der logischen Nachricht innerhalb einer Arbeitseinheit eingereiht werden, sofern die gleiche Warteschlangenkennung verwendet wird.

Ist dies nicht der Fall, schlägt der MQPUT-Aufruf mit dem Ursachencode MQRC_INCONSISTENT_UOW fehl.

Bei Angabe von MQPMO_LOGICAL_ORDER darf die mit dem MQPUT-Aufruf bereitgestellte MQMD-Version nicht älter als MQMD_VERSION_2 sein. Ist dies nicht der Fall, schlägt der Aufruf mit dem Ursachencode MQRC_WRONG_MD_VERSION fehl.

Wenn MQPMO_LOGICAL_ORDER nicht angegeben wird, können Nachrichten in Gruppen und Segmenten von logischen Nachrichten in jeder Reihenfolge eingereiht werden und es ist nicht erforderlich, vollständige Nachrichtengruppen oder vollständige logische Nachrichten einzureihen. Die Anwendung muss sicherstellen, dass die Felder *GroupId*, *MsgSeqNumber*, *Offset* und *MsgFlags* über die entsprechenden Werte verfügen.

Verwenden Sie dieses Verfahren, um eine Nachrichtengruppe oder logische Nachricht zwischendurch erneut zu starten, nachdem ein Systemfehler aufgetreten ist. Beim Neustart des Systems kann die Anwendung die Felder *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags* und *Persistence* auf die korrekten Werte setzen und dann den MQPUT-Aufruf nach Bedarf mit MQPMO_SYNCPOINT oder MQPMO_NO_SYNCPOINT, jedoch ohne Angabe von MQPMO_LOGICAL_ORDER ausgeben. Wenn dieser Aufruf erfolgreich ist, behält der Warteschlangenmanager die Gruppen- und Segmentinformationen bei und nachfolgende MQPUT-Aufrufe können mithilfe dieser Warteschlangenkennung MQPMO_LOGICAL_ORDER wie üblich angeben.

Die vom Warteschlangenmanager für den MQPUT-Aufruf beibehaltenen Gruppen- und Segmentinformationen haben nichts mit den für den MQGET-Aufruf beibehaltenen Informationen zu tun.

Die Anwendung kann für eine bestimmte Warteschlangenkennung MQPUT-Aufrufe, die MQPMO_LOGICAL_ORDER angeben, mit MQPUT-Aufrufen kombinieren, die es nicht angeben. Allerdings sind folgende Punkte zu beachten:

- Wenn MQPMO_LOGICAL_ORDER nicht angegeben wird, hat jeder erfolgreiche MQPUT-Aufruf zur Folge, dass der Warteschlangenmanager die Gruppen- und Segmentinformationen für die Warteschlangenkennung auf die von der Anwendung festgelegten Werte einstellt, indem die vorhandenen Gruppen- und Segmentinformationen, die vom Warteschlangenmanager für die Warteschlangenkennung beibehalten werden, ersetzt werden.
- Wenn MQPMO_LOGICAL_ORDER nicht angegeben wird, schlägt der Aufruf nicht fehl, wenn es eine aktuelle Nachrichtengruppe oder logische Nachricht gibt; der Aufruf wird möglicherweise mit einem

MQCC_WARNING-Beendigungscode erfolgreich abgeschlossen. Tabelle 117 auf Seite 821 gibt die verschiedenen Fälle an, die auftreten können. Ist der Beendigungscode in diesen Fällen nicht MQCC_OK, so lautet der Ursachencode abhängig von der Fehlerursache wie folgt:

- MQRC_INCOMPLETE_GROUP
- MQRC_INCOMPLETE_MSG
- MQRC_INCONSISTENT_PERSISTENCE
- MQRC_INCONSISTENT_UOW

Anmerkung: Die Gruppen- und Segmentinformationen für den MQPUT1-Aufruf werden nicht durch den Warteschlangenmanager geprüft.

Tabelle 117. Ergebnis, wenn der MQPUT- oder MQCLOSE-Aufruf nicht mit den Gruppen- und Segmentinformationen übereinstimmt

Aktueller Aufruf ist	Vorheriger Aufruf war MQPUT mit MQPMO_LOGICAL_ORDER	Vorheriger Aufruf war MQPUT ohne MQPMO_LOGICAL_ORDER
MQPUT mit MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQPUT ohne MQPMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE mit nicht beendeter Gruppe oder logischen Nachricht	MQCC_WARNING	MQCC_OK

Für Anwendungen, die Nachrichten und Segmente in eine logische Reihenfolge bringen, wird MQPMO_LOGICAL_ORDER angegeben, da es die einfachste Option ist. Bei dieser Option muss die Anwendung die Gruppen- und Segmentinformationen nicht verwalten, da der Warteschlangenmanager diese Informationen verwaltet. Wenn jedoch spezielle Anwendungen mehr Steuerungsmöglichkeiten erfordern als die Option MQPMO_LOGICAL_ORDER ermöglicht, dann können Sie diese Option weglassen. In diesem Fall müssen Sie jedoch vor jedem MQPUT- bzw. MQPUT1-Aufruf sicherstellen, dass die Felder *GroupId*, *MsgSeqNumber*, *Offset* und *MsgFlags* in MQMD die korrekten Werte enthalten.

Eine Anwendung, die z. B. physische Nachrichten, die sie empfängt, ohne Rücksicht darauf, ob diese Nachrichten in Gruppen oder Segmenten von logischen Nachrichten sind, weiterleiten möchte, darf aus zwei Gründen MQPMO_LOGICAL_ORDER nicht angeben:

- Wenn die Nachrichten abgerufen und eingereiht werden, wird den Nachrichten durch Angabe von MQPMO_LOGICAL_ORDER eine neue Gruppen-ID zugewiesen, durch die es für den Absender der Nachrichten schwer oder unmöglich werden kann, Antworten oder Berichtsnachrichten, die aus der Nachrichtengruppe hervorgehen, zuzuordnen.
- In einem komplexen Netz aus verschiedenen Pfaden zwischen sendenden und empfangenden Warteschlangenmanagern kann es dazu kommen, dass physische Nachrichten nicht in der richtigen Reihenfolge eintreffen. Wenn MQPMO_LOGICAL_ORDER und MQGMO_LOGICAL_ORDER im MQGET-Aufruf nicht angegeben wird, kann die weiterleitende Anwendung jede physische Nachricht sobald sie eingeht abrufen und weiterleiten, ohne darauf zu warten, dass die nächste Nachricht der logischen Reihenfolge eintrifft.

Anwendungen, die Berichtsnachrichten für Nachrichten in Gruppen oder Segmenten von logischen Nachrichten erstellen, dürfen beim Einreihen der Berichtsnachricht ebenfalls nicht MQPMO_LOGICAL_ORDER angeben.

MQPMO_LOGICAL_ORDER kann mit jeder anderen MQPMO_*-Option angegeben werden.

Einreihen logisch sortierter Gruppen in einer Clusterwarteschlange (MQOO_BIND_ON_GROUP)

Die Option MQOO_BIND_ON_OPEN stellt sicher, dass alle Nachrichten von dieser Anwendung und somit alle Gruppen zu einer einzigen Instanz weitergeleitet werden. Der Nachteil liegt darin, dass der Datenver-

kehr nicht über mehrere Instanzen einer Clusterwarteschlange verteilt wird. Wenn Sie den Lastausgleich ermöglichen und gleichzeitig Nachrichtengruppen intakt halten möchten, müssen Sie folgende Optionen einstellen:

- Im MQPUT-Aufruf muss MQPMO_LOGICAL_ORDER angegeben werden
- Im MQOPEN-Aufruf muss eine der beiden folgenden Optionen angegeben werden:
  - MQOO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF und in der Warteschlangendefinition muss DEFBIND(GROUP) angegeben sein

Lastausgleich wird dann *zwischen Gruppen* von Nachrichten gesteuert, ohne MQCLOSE und MQOPEN von der Warteschlange zu erfordern. *Zwischen Gruppen* bedeutet, dass MQMF_MSG_IN_GROUP in MQMD(v2) oder MQMDE eingestellt wird und es keine teilweise vollständige Gruppe in Bearbeitung gibt. Wenn eine Gruppe in Bearbeitung ist, werden der aufgelöste Warteschlangenmanager und Warteschlangenname in der Objektkennung wiederverwendet.

Wenn die vorherige Nachricht MQPMO_LOGICAL_ORDER war und/oder MQMF_MSG_IN_GROUP eingestellt wurde, aber die aktuelle Nachricht nicht Teil der Gruppe war, schlägt der PUT-Aufruf mit MQRC_INCOMPLETE_GROUP fehl.

Wenn ein einzelner MQPUT nicht MQPMO_LOGICAL_ORDER angibt und keine aktuelle Gruppe aktiv ist, wird der Lastausgleich für diese Nachricht gesteuert (als ob der MQOPEN-Aufruf MQOO_BIND_NOT_FIXED angegeben hätte).

Bei Nachrichten, die mit MQOO_BIND_ON_GROUP an ein bestimmtes Ziel gebunden sind, erfolgt keine Neuzuordnung. Weitere Informationen zu Neuzuordnung finden Sie im Abschnitt „Nachrichtengruppen” auf Seite 46.

Logische Nachrichten gruppieren

Es gibt zwei Hauptgründe für die Verwendung logischer Nachrichten in einer Gruppe:

- Die Nachrichten müssen in einer bestimmten Reihenfolge verarbeitet werden.
- Die Nachrichten einer Gruppe müssen alle auf gleiche Art und Weise verarbeitet werden.

In beiden Fällen müssen Sie die gesamte Gruppe in einer abrufenden Anwendungsinstanz (Get-Operation) abrufen.

Angenommen, eine Gruppe besteht aus vier logischen Nachrichten. Die einreihende Anwendung (Put-Operation) sieht dann wie folgt aus:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

Die abrufende Anwendung gibt für die erste Nachricht der Gruppe die Option MQGMO_ALL_MSGS_AVAILABLE an. Dadurch wird sichergestellt, dass die Verarbeitung nicht beginnt, bevor alle Nachrichten der Gruppe eingetroffen sind. Die Option MQGMO_ALL_MSGS_AVAILABLE wird für die nachfolgenden Nachrichten der Gruppe ignoriert.

Nach dem Abrufen der ersten logischen Nachricht der Gruppe können Sie mit MQGMO_LOGICAL_ORDER sicherstellen, dass die verbleibenden logischen Nachrichten der Gruppe der Reihenfolge nach abgerufen werden.

Die abrufende Anwendung (Get-Operation) sieht dann wie folgt aus:

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
```

```
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

Weitere Beispiele zur Gruppierung von Nachrichten finden Sie in den Abschnitten „Segmentierung logischer Nachrichten durch die Anwendung" auf Seite 835 und „Gruppe in einer Arbeitseinheit einreihen und abrufen" auf Seite 823.

Achtung: Wenn Publish/Subscribe zum Senden von Nachrichten an ein Thema (oder zum Einreihen von Nachrichten in ein Topic-Alias) verwendet wird, ist die Gruppierung und Segmentierung von Nachrichten nicht zulässig.

Da Subskriptionen unabhängig von Veröffentlichungsaktivitäten erstellt und entfernt werden können, kann nicht sichergestellt werden, dass ein Subskribent eine vollständige Nachrichtengruppe oder alle Segmente einer Nachricht empfangen würde (siehe RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP).

Informationen zur Ermöglichung einer Anwendung, anzufordern, dass eine Gruppe von Nachrichten bei Verwendung von Clusterwarteschlangen derselben Zielinstanz zugeordnet wird, finden Sie im Abschnitt DefBind.

Gruppe in einer Arbeitseinheit einreihen und abrufen

Im vorherigen Fall können Nachrichten oder Segmente den Knoten erst verlassen (bei einem fernen Ziel) bzw. abgerufen werden, wenn die Gruppe vollständig eingereiht und die Arbeitseinheit festgeschrieben ist. Dies ist aber eventuell nicht das, was Sie wünschen, wenn das Einreihen der Gruppe sehr lange dauert oder wenn der Warteschlangenspeicher auf dem Knoten knapp wird. In diesem Fall haben Sie die Möglichkeit, die Gruppe beim Einreihen auf mehrere Arbeitseinheiten aufzuteilen.

Bei einer Gruppe, die auf mehrere Arbeitseinheiten aufgeteilt ist, kann bereits mit dem Commit einzelner Arbeitseinheiten begonnen werden, selbst wenn die einreihende Anwendung für den restlichen Teil der Gruppe fehlschlägt. Die Anwendung muss allerdings die mit jeder Arbeitseinheit festgeschriebenen Statusinformationen speichern, um eine unvollständig eingereihte Gruppe nach einem Neustart wieder aufnehmen zu können. Der einfachste Ort, diese Informationen zu speichern, ist eine STATUS-Warteschlange. Diese bleibt leer, wenn die vollständige Gruppe erfolgreich eingereiht werden kann.

Bei Segmenten ist die Logik ähnlich. In diesem Fall muss der Parameter **StatusInfo** jedoch den Wert für *Offset* einschließen.

Nachfolgend sehen Sie ein Beispiel für die Einreihung einer Gruppe in mehreren Arbeitseinheiten:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
```

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

Wenn alle Arbeitseinheiten festgeschrieben sind, wurde die Gruppe vollständig und erfolgreich eingereiht und die STATUS-Warteschlange ist leer. Andernfalls muss die Gruppe an der in den Statusinformationen angegebenen Stelle wieder aufgenommen werden. MQPMO_LOGICAL_ORDER kann nicht für den ersten PUT-Vorgang verwendet werden, jedoch jederzeit danach.

Die Wiederaufnahme der Verarbeitung sieht wie folgt aus:

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

Eventuell möchten Sie in der abrufenden Anwendung mit der Verarbeitung der Nachrichten einer Gruppe beginnen, bevor die Gruppe vollständig angekommen ist. Dadurch verbessern Sie die Antwortzeiten für die Nachrichten der Gruppe und reduzieren den für die Gruppe erforderlichen Speicherplatz. Die Vorteile zeigen sich aber erst richtig, wenn Sie für jede Nachrichtengruppe mehrere Arbeitseinheiten verwenden. Aus Wiederherstellungsgründen müssen Sie jede Nachricht einer Arbeitseinheit abrufen.

Dies setzt aber wie bei der einreihenden Anwendung voraus, dass beim Festschreiben der einzelnen Arbeitseinheiten automatisch an einem bestimmten Ort Statusinformationen aufgezeichnet werden. Auch hier ist der einfachste Ort, diese Informationen zu speichern, eine STATUS-Warteschlange. Wenn die vollständige Gruppe erfolgreich verarbeitet wurde, bleibt diese Warteschlange leer.

Anmerkung: Für die Arbeitseinheiten zwischen der ersten und letzten Einheit können Sie MQGET-Aufrufe aus der STATUS-Warteschlange umgehen, indem Sie mit dem Flag MQMF_SEGMENT angeben, dass jedes MQPUT in die STATUS-Warteschlange ein Segment einer Nachricht ist, anstatt für jede Arbeitseinheit eine völlig neue Nachricht einzureihen. Bei der letzten Arbeitseinheit wird ein Abschlusssegment (MQMF_LAST_SEGMENT) in die Statuswarteschlange eingereiht. Danach werden die Statusinformationen durch ein MQGET mit der Option MQGMO_COMPLETE_MSG gelöscht.

Während der Neustartverarbeitung verwenden Sie zum Abrufen einer möglichen Statusnachricht besser nicht einen einzelnen MQGET-Aufruf, sondern durchsuchen die Statuswarteschlange mit MQGMO_LOGICAL_ORDER bis zum letzten Segment (d. h., bis keine weiteren Segmente mehr zurückgegeben werden). Geben Sie außerdem für die erste Arbeitseinheit nach dem Neustart beim Einreihen des Statussegments explizit den Offset an.

Im folgenden Beispiel werden nur die Nachrichten einer Gruppe berücksichtigt. Dabei wird vorausgesetzt, dass der Anwendungspuffer, unabhängig davon, ob die Nachricht segmentiert wurde, groß genug für die gesamte Nachricht ist. Daher wird in jedem MQGET MQGMO_COMPLETE_MSG angegeben. Das gleiche Prinzip gilt auch bei Segmenten (in diesem Fall muss der Parameter 'StatusInfo' jedoch den Wert für *Offset* einschließen).

Der Einfachheit halber gehen wir davon aus, dass pro Arbeitseinheit (UOW) maximal vier Nachrichten abgerufen werden:

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
```

```
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
      MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
   /* Come here if there was only 1 message in the group */
   MQCMIT
```

Wenn alle Arbeitseinheiten festgeschrieben sind, wurde die Gruppe vollständig und erfolgreich abgerufen und die STATUS-Warteschlange ist leer. Andernfalls muss die Gruppe an der in den Statusinformationen angegebenen Stelle wieder aufgenommen werden. MQGMO_LOGICAL_ORDER kann nicht für den ersten GET-Vorgang verwendet werden, jedoch jederzeit danach.

Die Wiederaufnahme der Verarbeitung sieht wie folgt aus:

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   /* The next message on the group must be retrieved by matching
      the sequence number and group ID with those retrieved from the
      status information. */
   GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
   MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
         MQMD.GroupId      = value from Status message,
         MQMD.MsgSeqNumber = value from Status message plus 1
   msgs = 1
   /* Process this message */
   ...

   /* Now normal processing is resumed */
   /* Retrieve remaining messages in the group */
   do while ( GroupStatus == MQGS_MSG_IN_GROUP )

      /* Process up to 4 messages in the group */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                  | MQGMO_LOGICAL_ORDER
      do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
         MQGET
         msgs = msgs + 1
         /* Process this message */
         ...

      /* Have retrieved last message or 4 messages  */
      /* Update status message if not last in group */
      MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
      if ( GroupStatus == MQGS_MSG_IN_GROUP )
         StatusInfo = GroupId,MsgSeqNumber from MQMD
         MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
      MQCMIT
      msgs = 0
```

Bestimmte Nachricht abrufen

Es gibt verschiedene Möglichkeiten, eine bestimmte Nachricht aus einer Warteschlange abzurufen. Diese wären: Auswahl nach MsgId und CorrelId, Auswahl nach GroupId, MsgSeqNumber und Offset oder Aus-

wahl nach MsgToken. Ebenso können Sie beim Öffnen einer Warteschlange eine Auswahlzeichenfolge verwenden.

Verwenden Sie die Felder `MsgId` und `CorrelId` der MQMD-Struktur, wenn Sie eine bestimmte Nachricht abrufen wollen. Da diese Felder aber explizit durch die Anwendung festgelegt werden können, kann es sein, dass die von Ihnen angegebenen Werte keine eindeutige Nachricht identifizieren. Tabelle 118 auf Seite 826 zeigt, welche Nachrichten bei den möglichen Einstellungen dieser Felder abgerufen werden. Wenn Sie im Parameter **`GetMsgOpts`** des MQGET-Aufrufs MQGMO_MSG_UNDER_CURSOR angeben, werden diese Felder ignoriert.

Tabelle 118. Nachrichten- und Korrelations-IDs verwenden

Zum Abrufen von	**`MsgId`**	**`CorrelId`**
Erste Nachricht der Warteschlange	`MQMI_NONE`	`MQCI_NONE`
Erste Nachricht, die mit *`MsgId`* übereinstimmt	Ungleich null	`MQCI_NONE`
Erste Nachricht, die mit *`CorrelId`* übereinstimmt	`MQMI_NONE`	Ungleich null
Erste Nachricht, die mit *`MsgId`* und *`CorrelId`* übereinstimmt	Ungleich null	Ungleich null

In all diesen Fällen bedeutet *erste* die erste Nachricht, die mit den Auswahlkriterien übereinstimmt (außer bei Angabe von MQGMO_BROWSE_NEXT, in welchem Fall die *nächste* Nachricht der Sequenz gemeint ist, die mit den Auswahlkriterien übereinstimmt).

In der Rückgabe setzt MQGET die Felder *`MsgId`* und *`CorrelId`* auf die Nachrichten- und Korrelations-ID der zurückgegebenen Nachricht (sofern eine zurückgegeben wird).

Wenn das Feld *`Version`* der MQMD-Struktur auf 2 gesetzt ist, können Sie die Felder *`GroupId`*, *`MsgSeqNumber`* und *`Offset`* verwenden. Tabelle 119 auf Seite 826 zeigt, welche Nachrichten bei den möglichen Einstellungen dieser Felder abgerufen werden.

Tabelle 119. Gruppen-ID verwenden

Zum Abrufen von	**Abgleichoptionen**
Erste Nachricht der Warteschlange	MQMO_NONE
Erste Nachricht, die mit `MsgId` übereinstimmt	MQMO_MATCH_MSG_ID
Erste Nachricht, die mit `CorrelId` übereinstimmt	MQMO_MATCH_CORREL_ID
Erste Nachricht, die mit `GroupId` übereinstimmt	MQMO_MATCH_GROUP_ID
Erste Nachricht, die mit `MsgSeqNumber` übereinstimmt	MQMO_MATCH_MSG_SEQ_NUMBER
Erste Nachricht, die mit `MsgToken` übereinstimmt	MQMO_MATCH_MSG_TOKEN
Erste Nachricht, die mit `Offset` übereinstimmt	MQMO_MATCH_OFFSET

Anmerkungen:

1. MQMO_MATCH_XXX bedeutet, dass das Feld *XXX* der MQMD-Struktur auf den abzugleichenden Wert gesetzt ist.
2. Die MQMO-Flags können auch in Kombination verwendet werden. MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER und MQMO_MATCH_OFFSET können zum Beispiel gemeinsam zur Rückgabe des Segments verwendet werden, das durch die Felder `GroupId`, `MsgSeqNumber` und `Offset` identifiziert wird.
3. Die Angabe von MQGMO_LOGICAL_ORDER wirkt sich auf die Nachricht aus, die Sie abzurufen versuchen, da diese Option abhängig von Statusinformationen ist, die für die Warteschlangenkennung festgelegt werden. Informationen hierzu finden Sie in den Abschnitten „Logische und physische Reihenfolge” auf Seite 814 und Optionen.

Normalerweise ruft der MQGET-Aufruf die erste Nachricht aus einer Warteschlange ab. Wenn Sie beim Aufruf von MQGET eine bestimmte Nachricht angeben, muss der Warteschlangenmanager die Warteschlange so lange durchsuchen, bis es diese Nachricht findet. Dies kann die Leistung Ihrer Anwendung beeinträchtigen.

Wenn Sie ab Version 2 der MQGMO-Struktur die Flags MQMO_MATCH_MSG_ID und MQMO_MATCH_CORREL_ID nicht angeben, brauchen Sie das Feld `MsgId` bzw. `CorrelId` zwischen den einzelnen MQGET-Aufrufen nicht mehr zurückzusetzen.

z/OS Unter IBM MQ for z/OS kann die Geschwindigkeit von MQGET-Operationen aus einer Warteschlange mit dem Warteschlangenattribut `IndexType` beschleunigt werden. Weitere Informationen finden Sie unter „Type of index” auf Seite 831.

Durch Angabe des 'MsgToken' und der 'MatchOption' MQMO_MATCH_MSG_TOKEN in der MQGMO-Struktur können Sie eine bestimmte Nachricht aus einer Warteschlange abrufen. Das 'MsgToken' wurde durch den MQPUT-Aufruf zurückgegeben, durch den die Nachricht ursprünglich in die Warteschlange eingereiht wurde, bzw. durch vorangegangene MQGET-Operationen, und ändert sich außer bei einem Neustart des Warteschlangenmanagers nicht.

Wenn Sie nur an einem Teil der Nachrichten aus der Warteschlange interessiert sind, können Sie die zu verarbeitenden Nachrichten durch eine Auswahlzeichenfolge im MQOPEN- oder MQSUB-Aufruf angeben. MQGET ruft dann die nächste Nachricht ab, die dieser Auswahlzeichenfolge entspricht. Weitere Informationen zu Auswahlzeichenfolgen finden Sie im Abschnitt „Selectors” auf Seite 32.

Leistung nicht persistenter Nachrichten verbessern

Wenn ein Client eine Nachricht von einem Server benötigt, sendet er eine Anforderung an den Server. Für jede von ihm verarbeitete Nachricht sendet der Client eine eigene Anforderung. Zur Verbesserung der Leistung eines Clients, der nicht persistente Nachrichten verarbeitet, kann er für die Verwendung von *Vorauslesen* konfiguriert und es so vermieden werden, dass er diese Anforderungsnachrichten senden muss. Mit Vorauslesen können Nachrichten an einen Client gesendet werden, ohne dass eine Anwendung sie anfordern muss.

Wenn Vorauslesen aktiviert ist, werden die Nachrichten in einen Hauptspeicherpuffer auf dem Client gesendet, der als *Vorauslesepuffer* bezeichnet wird. Der Client besitzt für jede geöffnete Warteschlange mit aktiviertem Vorauslesen einen Vorauslesepuffer. Die Nachrichten im Vorauslesepuffer sind nicht als persistent definiert. Der Client sendet dem Server regelmäßig aktuelle Informationen über das von ihm verarbeitete Datenvolumen.

Bei einem MQOPEN-Aufruf mit MQOO_READ_AHEAD aktiviert der IBM MQ-Client die Vorausleseoption nur, wenn bestimmte Bedingungen erfüllt sind. Zu diesen Bedingungen gehören:

- Die Clientanwendung muss kompiliert und mit den IBM MQ MQ-Client-Thread-Bibliotheken verknüpft sein.
- Der Clientkanal muss das TCP/IP-Protokoll verwenden.
- In den Client- und Serverkanaldefinitionen muss für den Kanal ein Wert ungleich null für den Parameter SHARECNV (gemeinsame Dialognutzung) angegeben sein.

Durch die Verwendung von Vorauslesen kann die Leistung verbessert werden, wenn nicht persistente Nachrichten von einer Clientanwendung verarbeitet werden. Diese Leistungsverbesserung betrifft sowohl MQI- als auch JMS-Anwendungen. Clientanwendungen, die MQGET oder asynchrone Verarbeitung verwenden, profitieren von den Leistungsverbesserungen, wenn sie nicht persistente Nachrichten verarbeiten.

Nicht alle Clientanwendungen sind von ihrem Design her für die Verwendung von Vorauslesen geeignet, da beim Vorauslesen nicht alle Optionen unterstützt werden. Einige Optionen müssen zwischen den einzelnen MQGET-Aufrufen auch konsistent sein. Wenn ein Client seine Auswahlkriterien zwischen MQGET-Aufrufen ändert, werden im Vorauslesepuffer gespeicherte Nachrichten im Vorauslesepuffer des Clients zurückgelassen.

Wenn kein Rückstand zurückgelassener Nachrichten mit den vorherigen Auswahlkriterien mehr erforderlich ist, kann auf dem Client zum Löschen dieser Nachrichten ein konfigurierbares Löschintervall einge-

richtet werden. Das Löschintervall ist eine der Optimierungsoptionen für das Vorauslesen, die auf dem Client eingestellt werden. Sie können diese Optionen Ihren Anforderungen entsprechend einstellen.

Beim Neustart einer Clientanwendung können Nachrichten im Vorauslesepuffer verloren gehen. Umgekehrt kann eine Nachricht, die in einen Vorauslesepuffer verschoben wurde, aus ihrer Warteschlange entfernt werden, ohne zugleich aus dem Puffer entfernt zu werden. Ein MQGET-Aufruf, der Vorauslesen verwendet, kann also eine Nachricht zurückgeben, die gar nicht mehr existiert.

Vorauslesen wird nur für Clientbindungen ausgeführt. Für alle anderen Bindungen wird dieses Attribut ignoriert.

Vorauslesen hat keine Auswirkung auf die Auslösung. Es wird keine Auslösenachricht generiert, wenn eine Nachricht vom Client voraus gelesen wird. Beim Vorauslesen werden keine Abrechnungs- und Statistikdaten generiert.

Vorauslesen mit Publish/Subscribe-Messaging verwenden

Wenn eine subskribierende Anwendung eine Zielwarteschlange angibt, an die Veröffentlichungen gesendet werden sollen, wird der Wert DEFREADA dieser Warteschlange als Standardwert für das Vorauslesen verwendet.

Fordert eine subskribierende Anwendung hingegen die Verwaltung des Veröffentlichungsziels durch IBM MQ, wird eine verwaltete Warteschlange als dynamische Warteschlange auf Basis eines vordefinierten Modells erstellt. In diesem Fall wird der Wert DEFREADA der Modellwarteschlange als Standardwert für das Vorauslesen verwendet. Dabei wird die Standardmodellwarteschlange SYSTEM.DURABLE.PUBLICATIONS.MODEL bzw. SYSTEM.NONDURABLE.PUBLICATIONS.MODEL verwendet, wenn keine andere Modellwarteschlange für dieses oder ein übergeordnetes Thema definiert ist.

Zugehörige Konzepte

„Leistung für nicht persistente Nachrichten unter AIX optimieren” auf Seite 830
Wenn Sie AIX Version 5.3 oder höher verwenden, empfiehlt sich für nicht persistente Nachrichten vermutlich ein Heraufsetzen des Optimierungsparameters auf die volle Leistung.

Zugehörige Tasks

„Vorauslesen aktivieren und inaktivieren” auf Seite 830
Vorauslesen ist standardmäßig inaktiviert. Es kann aber auf Warteschlangen- oder Anwendungsebene aktiviert werden.

Zugehörige Verweise

„MQGET-Optionen und Vorauslesen” auf Seite 828
Wenn Vorauslesen aktiviert ist, werden nicht alle MQGET-Optionen unterstützt. Einige Optionen müssen zwischen den einzelnen MQGET-Aufrufen auch konsistent sein.

MQGET-Optionen und Vorauslesen
Wenn Vorauslesen aktiviert ist, werden nicht alle MQGET-Optionen unterstützt. Einige Optionen müssen zwischen den einzelnen MQGET-Aufrufen auch konsistent sein.

Bei einem MQOPEN-Aufruf mit MQOO_READ_AHEAD aktiviert der IBM MQ-Client die Vorausleseoption nur, wenn bestimmte Bedingungen erfüllt sind. Zu diesen Bedingungen gehören:

- Die Clientanwendung muss kompiliert und mit den IBM MQ MQ-Client-Thread-Bibliotheken verknüpft sein.
- Der Clientkanal muss das TCP/IP-Protokoll verwenden.
- In den Client- und Serverkanaldefinitionen muss für den Kanal ein Wert ungleich null für den Parameter SHARECNV (gemeinsame Dialognutzung) angegeben sein.

In folgender Tabelle ist angegeben, welche Optionen beim Vorauslesen unterstützt werden, und ob diese Optionen zwischen einzelnen MQGET-Aufrufen geändert werden können.

Tabelle 120. MQGET-Optionen und Vorauslesen			
MQGET-Werte und -Optionen	**Bei aktiviertem Vorauslesen zulässig und zwischen MQGET-Aufrufen änderbar [5]**	**Bei aktiviertem Vorauslesen zulässig, kann aber nicht zwischen MQGET-Aufrufen ausgetauscht werden[1]**	**Bei aktiviertem Vorauslesen nicht erlaubt [2]**
MQMD-Werte für MQGET	MsgId[3] CorrelId [3]	Encoding CodedCharSetId	
MQGMO-Optionen für MQGET	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _UNDER_CURSOR • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_NEXT • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENT • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_TRUNCATED_MSG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • MQGMO_SYNCPOINT • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR [4] • MQGMO_LOCK • MQGMO_UNLOCK • MQGMO_LOGICAL_ORDER • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ AVAILABLE

Anmerkungen:

1. Wenn diese Optionen zwischen MQGET-Aufrufen geändert werden, wird 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. Falls eine Clientanwendung die MsgId- und CorrelId-Werte zwischen MQGET-Aufrufen ändert, wurden eventuell bereits Nachrichten mit den früheren Werten an den Client gesendet und verbleiben dort so lange im Vorauslesepuffer, bis sie verarbeitet (oder automatisch gelöscht) werden.
4. MQGMO_MSG_UNDER_CURSOR ist bei aktiviertem Vorauslesen nicht möglich. Vorauslesen wird inaktiviert, wenn beim Öffnen einer Warteschlange MQOO_BROWSE sowie MQOO_INPUT_SHARED oder MQOO_INPUT_EXCLUSIVE angegeben werden.
5. Wenn Vorauslesen aktiviert ist, bestimmt der erste MQGET-Aufruf, ob die Nachrichten einer Warteschlange durchsucht oder abgerufen werden. Verwendet die Clientanwendung danach ein MQGET mit geänderten Optionen, also Durchsuchen nach einem ursprünglichen Abrufen bzw. umgekehrt Abrufen nach einem ursprünglichen Durchsuchen, wird ein Fehler mit dem Ursachencode MQRC_OPTIONS_CHANGED zurückgegeben.

Wenn ein Client seine Auswahlkriterien zwischen MQGET-Aufrufen ändert, werden im Vorauslesepuffer gespeicherte Nachrichten, die den ursprünglichen Auswahlkriterien entsprechen, nicht von der Clientanwendung verarbeitet und verbleiben im Vorauslesepuffer des Clients. Ab einer gewissen Menge solchermaßen im Vorauslesepuffer des Clients zurückgelassener Nachrichten gehen die Vorteile des Vorauslesens verloren, da für jede verarbeitete Nachricht eine separate Anforderung an den Server erforderlich wird. Ob der Einsatz der Vorauslesefunktion effizient ist, können Sie dem Statusparameter READA der Verbindung entnehmen.

Das Vorauslesen kann bei Anforderung durch eine Anwendung unterdrückt werden, wenn die im ersten MQGET-Aufruf angegebenen Optionen inkompatibel sind. In diesem Fall gibt der Verbindungsstatus an, dass Vorauslesen unterdrückt ist.

Falls Sie aufgrund dieser für MQGET geltenden Einschränkungen der Meinung sind, dass sich eine Anwendung aufgrund ihres Designs nicht für Vorauslesen eignet, verwenden Sie MQOPEN mit der Option MQOO_READ_AHEAD_NO. Alternativ können Sie den Standardvorauslesewert der zu öffnenden Warteschlange auch auf NO oder DISABLED setzen.

Vorauslesen aktivieren und inaktivieren
Vorauslesen ist standardmäßig inaktiviert. Es kann aber auf Warteschlangen- oder Anwendungsebene aktiviert werden.

Informationen zu diesem Vorgang

Bei einem MQOPEN-Aufruf mit MQOO_READ_AHEAD aktiviert der IBM MQ-Client die Vorausleseoption nur, wenn bestimmte Bedingungen erfüllt sind. Zu diesen Bedingungen gehören:

- Die Clientanwendung muss kompiliert und mit den IBM MQ MQ-Client-Thread-Bibliotheken verknüpft sein.
- Der Clientkanal muss das TCP/IP-Protokoll verwenden.
- In den Client- und Serverkanaldefinitionen muss für den Kanal ein Wert ungleich null für den Parameter SHARECNV (gemeinsame Dialognutzung) angegeben sein.

So aktivieren Sie Vorauslesen:

- Zum Konfigurieren von Vorauslesen auf Warteschlangenebene setzen Sie das Warteschlangenattribut DEFREADA auf YES.
- Zum Konfigurieren von Vorauslesen auf Anwendungsebene führen sie die folgenden Schritte aus:
  - Wenn Vorauslesen wann immer möglich verwendet werden soll, verwenden Sie den Funktionsaufruf MQOPEN mit der Option MQOO_READ_AHEAD. Wenn allerdings das Warteschlangenattribut DEFREADA auf DISABLED gesetzt ist, kann die Clientanwendung kein Vorauslesen verwenden.
  - Wenn Vorauslesen nur dann verwendet werden soll, wenn diese Funktion für eine Warteschlange aktiviert ist, verwenden Sie den Funktionsaufruf MQOPEN mit der Option MQOO_READ_AHEAD_AS_Q_DEF.

Wenn sich eine Clientanwendung aufgrund ihres Designs nicht für Vorauslesen eignet, können Sie die Funktion wie folgt inaktivieren:

- Auf Warteschlangenebene durch Setzen des Warteschlangenattributs DEFREADA auf NO, wenn Vorauslesen nur auf explizite Anforderung einer Clientanwendung verwendet werden soll, oder auf DISABLED, wenn Vorauslesen unabhängig von einer entsprechenden Anforderung der Clientanwendung auf gar keinen Fall verwendet werden soll.
- Auf Anwendungsebene durch Verwendung des Funktionsaufrufs MQOPEN mit der Option MQOO_NO_READ_AHEAD.

Zwei MQCLOSE-Optionen legen fest, was mit Nachrichten geschieht, die sich beim Schließen der Warteschlange noch im Vorauslesepuffer befinden.

- Verwenden Sie MQCO_IMMEDIATE, um die Nachrichten im Vorauslesepuffer zu verwerfen.
- Verwenden Sie MQCO_QUIESCE, um sicherzustellen, dass die Nachrichten im Vorauslesepuffer vor dem Schließen der Warteschlange von der Anwendung verarbeitet werden. Wenn MQCLOSE mit der Option MQCO_QUIESCE ausgegeben wird und sich noch Nachrichten im Vorauslesepuffer befinden, wird MQRC_READ_AHEAD_MSGS mit MQCC_WARNING zurückgegeben.

Leistung für nicht persistente Nachrichten unter AIX optimieren
Wenn Sie AIX Version 5.3 oder höher verwenden, empfiehlt sich für nicht persistente Nachrichten vermutlich ein Heraufsetzen des Optimierungsparameters auf die volle Leistung.

Wenn der Optimierungsparameter sofort in Kraft treten soll, führen Sie folgenden Befehls als Rootbenutzer aus:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Wenn der Optimierungsparameter sofort in Kraft treten und auch nach einem Neustart noch wirksam sein soll, führen Sie folgenden Befehls als Rootbenutzer aus:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

Nicht persistente Nachrichten verbleiben normalerweise nur im Hauptspeicher. Allerdings gibt es Umstände, unter denen AIX nicht persistente Nachrichten zu bestimmten Zeiten auf die Festplatte schreibt. Nachrichten, die auf die Festplatte geschrieben werden, stehen für MQGET so lange nicht zur Verfügung, bis der Speichervorgang auf die Festplatte abgeschlossen ist. Dieser Grenzwert wird durch den vorgeschlagenen Optimierungsbefehl geändert. Anstatt Nachrichten auf die Festplatte zu schreiben, wenn sich in der Warteschlange 16 KB an Daten angesammelt haben, werden die Nachrichten nach Ausführung dieses Befehls erst auf die Festplatte geschrieben, wenn der Realspeicher des Computers nahezu voll ist. Da es sich hier um eine globale Änderung handelt, können davon auch andere Softwarekomponenten betroffen sein.

Wenn Sie unter AIX Multithread-Anwendungen insbesondere auf Multiprozessormaschinen verwenden, empfehlen wir Ihnen zur Leistungsoptimierung und für eine solide Zeitplanung dringend, vor dem Start der Anwendung die Einstellung 'AIXTHREAD_SCOPE=S' in der Datei `.profile` für die ID 'mqm' bzw. in der Umgebung festzulegen. For example:

```
export AIXTHREAD_SCOPE=S
```

Bei der Einstellung AIXTHREAD_SCOPE=S werden mit Standardattributen erstellte Benutzerthreads in den systemweiten Zuteilungsbereich gestellt. Wenn ein Benutzerthread mit systemweitem Zuteilungsbereich erstellt wird, wird er an einen Kernel-Thread gebunden und vom Kernel geplant. Der zugrunde liegende Kernel-Thread wird mit keinem anderen Benutzerthread gemeinsam genutzt.

Dateideskriptoren

Bei der Ausführung eines Prozesses mit mehreren Threads, z. B. der Agentenprozess, wird unter Umständen der veränderliche Grenzwert für Dateideskriptoren erreicht. Bei diesem Grenzwert werden der IBM MQ-Ursachencode `MQRC_UNEXPECTED_ERROR (2195)` und (sofern genügend Dateideskriptoren vorhanden sind) eine IBM MQ FFST™-Datei ausgegeben.

Dieses Problem kann umgangen werden, indem das Verarbeitungslimit für die Anzahl der Dateideskriptoren erhöht wird. Setzen Sie dazu das Attribut `nofiles` unter `/etc/security/limits` bzw. in der Standard-Zeilengruppe für die Benutzer-ID 'mqm' auf 10.000.

Systemressourcengrenzen

Setzen Sie den Grenzwert für Systemressourcen für Daten- und Stacksegmente auf 'unlimited'. Geben Sie hierzu folgenden Befehl in einer Eingabeaufforderung ein:

```
ulimit -d unlimited
ulimit -s unlimited
```

Type of index

The queue attribute, *IndexType*, specifies the type of index that the queue manager maintains to increase the speed of MQGET operations on the queue.

Note: Supported only on IBM MQ for z/OS.

You have five options:

Value	Description
NONE	No index is maintained. Use this when retrieving messages sequentially (see “Priority” on page 814).
GROUPID	An index of group identifiers is maintained. You must use this index type if you want logical ordering of message groups (see “Logische und physische Reihenfolge” on page 814).

Value	Description
MSGID	An index of message identifiers is maintained. Use this when retrieving messages using the *MsgId* field as a selection criterion on the MQGET call (see “Bestimmte Nachricht abrufen” on page 825).
MSGTOKEN	An index of message tokens is maintained.
CORRELID	An index of correlation identifiers is maintained. Use this when retrieving messages using the *CorrelId* field as a selection criterion on the MQGET call (see “Bestimmte Nachricht abrufen” on page 825).

Note:

1. If you are indexing using the MSGID option or CORRELID option, set the relative **MsgId** or **CorrelId** parameters in the MQMD. It is not beneficial to set both.
2. Browse uses the index mechanism to find a message if a queue matches all the following conditions:
   - It has index type MSGID, CORRELID, or GROUPID
   - It is browsed with the same type of id
   - It has messages of only one priority
3. Avoid queues (indexed by *MsgId* or *CorrelId*) containing thousands of messages because this affects restart time. (This does not apply to nonpersistent messages as they are deleted at restart.)
4. MSGTOKEN is used to define queues managed by the z/OS workload manager.

For a full description of the **IndexType** attribute, see IndexType. For further information on the **IndexType** attribute, see “Design and performance considerations for z/OS applications” on page 69.

Nachrichten über 4 MB verarbeiten

Nachrichten können für die Anwendung, die Warteschlange, oder den Warteschlangenmanager zu groß sein. Je nach Umgebung bietet IBM MQ verschiedene Möglichkeiten, Nachrichten zu verarbeiten, die größer als 4 MB sind.

Das Attribut **MaxMsgLength** kann auf allen IBM MQ-Systemen ab V6 bis auf 100 MB gesetzt werden. Legen Sie hier einen Wert entsprechend der Größe der Nachrichten der jeweiligen Warteschlange fest. Unter IBM MQ for Multiplatformshaben Sie außerdem folgende Möglichkeiten:

1. Verwenden Sie segmentierte Nachrichten. (Nachrichten können von der Anwendung oder vom Warteschlangenmanager segmentiert werden.)
2. Verwenden Sie Referenznachrichten.

Diese Methoden werden im verbleibenden Teil dieses Abschnitts näher beschrieben.

Maximale Nachrichtenlänge heraufsetzen

Das Warteschlangenmanagerattribut **MaxMsgLength** definiert die maximale Länge einer Nachricht, die von einem Warteschlangenmanager verarbeitet werden kann. Entsprechend gibt das Warteschlangenattribut **MaxMsgLength** die maximale Länge einer Nachricht an, die von einer Warteschlange verarbeitet werden kann. Welche standardmäßige maximale Nachrichtenlänge unterstützt wird, hängt von der jeweiligen Umgebung ab.

Multi Unter IBM MQ for Multiplatformskönnen Sie beide Attribute manuell festlegen. Der gültige Bereich des Warteschlangenmanagerattributs liegt zwischen 32768 Byte und 100 MB.

Achtung: z/OS Unter IBM MQ for z/OS ist das Warteschlangenmanagerattribut **MaxMsgLength** mit 100 MB fest codiert.

Starten Sie Ihre Anwendungen und Kanäle nach einer Änderung eines oder beider **MaxMsgLength**-Attribute neu, um sicherzustellen, dass die Änderungen wirksam werden.

Nach einer Änderung darf die Größe einer Nachricht den kleineren Wert der beiden **MaxMsgLength**-Attribute nicht überschreiten. Bereits vorhandene Nachrichten verbleiben aber in der Warteschlange, auch wenn sie größer als einer der beiden Werte sind.

Falls eine Nachricht zu groß für die Warteschlange ist, wird MQRC_MSG_TOO_BIG_FOR_Q zurückgegeben. Ist sie zu groß für den Warteschlangenmanager, wird MQRC_MSG_TOO_BIG_FOR_Q_MGR zurückgegeben.

Diese Methode der Verarbeitung großer Nachrichten ist einfach und komfortabel. Dennoch sollten Sie vor einer Änderung der Attribute folgende Punkte berücksichtigen:

- Die einheitliche Konfiguration der Warteschlangenmanager geht verloren. Die maximale Größe der Nachrichtendaten wird durch das Attribut *MaxMsgLength* aller Warteschlangen (einschließlich Übertragungswarteschlangen) bestimmt, in die eine Nachricht eingereiht wird. Dieser Wert entspricht standardmäßig meist dem *MaxMsgLength*-Wert des Warteschlangenmanagers, besonders bei Übertragungswarteschlangen. Bei Nachrichten, die an einen fernen Warteschlangenmanager übertragen werden, lässt sich daher nur schwer vorhersagen, ob die Nachricht zu groß ist.
- Der Bedarf an Systemressourcen nimmt zu. Die Anwendungen benötigen beispielsweise größere Puffer und auf einigen Plattformen auch mehr gemeinsam genutzten Speicher. Auf den Warteschlangenspeicher wirkt sich der erhöhte Bedarf nur aus, wenn er tatsächlich für größere Nachrichten benötigt wird.
- Die Stapelverarbeitung der Kanäle wird beeinträchtigt. Hinsichtlich des Stapelzählers gilt auch eine große Nachricht nur als eine Nachricht, deren Übertragung dauert aber länger, wodurch sich die Antwortzeiten auch für die anderen Nachrichten erhöhen.

Multi *Nachrichtensegmentierung*

In diesem Abschnitt erfahren Sie, wie Nachrichten segmentiert werden. Dieses Feature wird unter IBM MQ for z/OS oder nicht von Anwendungen unterstützt, die IBM MQ classes for JMSverwenden.

Die Erhöhung der maximalen Nachrichtenlänge, wie im Abschnitt „Maximale Nachrichtenlänge heraufsetzen” auf Seite 832 beschrieben, hat auch Nachteile. Außerdem kann eine Nachricht, trotz Erhöhung dieses Werts, noch zu groß für eine Warteschlange oder einen Warteschlangenmanager sein. In diesem Fall bietet sich Ihnen die Möglichkeit der Nachrichtensegmentierung. Informationen zu Segmenten finden Sie im Abschnitt „Nachrichtengruppen” auf Seite 46.

Die nächsten Abschnitte beschäftigen sich dagegen mit typischen Verwendungsbeispielen für segmentierte Nachrichten. Beim Einreihen und Abrufen (mit Entfernen) von Nachrichten wird davon ausgegangen, dass die MQPUT- bzw. MQGET-Aufrufe *immer* für eine Arbeitseinheit gelten. Um unvollständige Gruppen im Netz zu vermeiden, sollten Sie diese Methode möglichst bevorzugt verwenden. In den Beispielen wird von einem einphasigen Commit des Warteschlangenmanagers ausgegangen, jedoch sind auch andere Koordinationstechniken möglich.

Für die abrufenden Anwendungen wird zudem vorausgesetzt, dass, wenn mehrere Server die gleiche Warteschlange verarbeiten, jeder Server den gleichen Code ausführt, so dass einem Server niemals eine Nachricht oder ein Segment abgeht, das er (aufgrund der vorangegangenen Angabe von MQGMO_ALL_MSGS_AVAILABLE oder MQGMO_ALL_SEGMENTS_AVAILABLE) in der Warteschlange erwartet.

Achtung: Wenn Publish/Subscribe zum Senden von Nachrichten an ein Thema (oder zum Einreihen von Nachrichten in ein Topic-Alias) verwendet wird, ist die Gruppierung und Segmentierung von Nachrichten nicht zulässig.

Da Subskriptionen unabhängig von Veröffentlichungsaktivitäten erstellt und entfernt werden können, kann nicht sichergestellt werden, dass ein Subskribent eine vollständige Nachrichtengruppe oder alle Segmente einer Nachricht empfangen würde (siehe RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP).

Segmentierte Nachricht in einer Arbeitseinheit einreihen und abrufen

Eine segmentierte Nachricht, die sich über eine Arbeitseinheit erstreckt, können Sie, wie in „Gruppe in einer Arbeitseinheit einreihen und abrufen” auf Seite 823 beschrieben, einreihen und abrufen.

Allerdings können Sie keine segmentierten Nachrichten in globalen Arbeitseinheiten einreihen und abrufen.

Multi *Segmentierung und Wiederzusammensetzung durch den Warteschlangenmanager*

Dies ist das einfachste Szenario, in dem eine Anwendung eine Nachricht einreiht, die von einer anderen Anwendung abgerufen wird. Eventuell handelt es sich um eine sehr große Nachricht - nicht zu groß für die einreihende oder abrufende Anwendung, deren Puffer zur Verarbeitung der Nachricht ausreicht, jedoch zu groß für den Warteschlangenmanager oder eine Warteschlange, in die die Nachricht eingereiht werden muss.

Die einzige Änderung, die in diesem Fall erforderlich ist, besteht darin, dass die einreihende Anwendung dem Warteschlangenmanager erlauben muss, eine Nachricht bei Bedarf zu segmentieren:

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

Entsprechend muss die abrufende Anwendung anfordern, dass der Warteschlangenmanager die Nachricht, sofern sie segmentiert wurde, wieder zusammensetzt:

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

In diesem einfachsten Szenario muss die Anwendung das Feld GroupId vor dem MQPUT-Aufruf auf MQGI_NONE zurücksetzen, so dass der Warteschlangenmanager für jede Nachricht eine eindeutige Gruppen-ID generieren kann. Andernfalls kann es passieren, dass nicht zusammengehörige Nachrichten die gleiche Gruppen-ID aufweisen, was in Folge zu einer fehlerhaften Verarbeitung führt.

Der Anwendungspuffer muss groß genug für die wieder zusammengesetzte Nachricht sein (es sei denn, Sie haben MQGMO_ACCEPT_TRUNCATED_MSG angegeben).

Wenn das MAXMSGLEN-Attribut einer Warteschlange in Hinsicht auf die Nachrichtensegmentierung geändert werden muss, sollten Sie Folgendes berücksichtigen:

- Die Mindestsegmentlänge, die eine lokale Warteschlange unterstützt, beträgt 16 Byte.
- Bei Übertragungswarteschlangen muss MAXMSGLEN auch die Größe der Header berücksichtigen. Der Wert sollte mindestens 4000 Byte größer sein, als die maximal erwartete Größe der Benutzerdaten eines Nachrichtensegments, das in die Übertragungswarteschlange eingereiht werden kann.

Falls Datenkonvertierung erforderlich ist, muss diese vermutlich durch Angabe von MQGMO_CONVERT durch die abrufende Anwendung ausgeführt werden. Dies sollte möglichst direkt über die gesamte Datei geschehen, da der Datenkonvertierungsexit mit der abgeschlossenen Nachricht ausgegeben wird. Versuchen Sie nicht, Daten in einem Senderkanal zu konvertieren, wenn die Nachricht segmentiert ist und das Datenformat ausschließt, dass der Datenkonvertierungsexit die Konvertierung an unvollständigen Daten durchführt.

Multi *Anwendungssegmentierung*

Anwendungssegmentierung wird verwendet, wenn die Segmentierung durch den Warteschlangenmanager nicht sinnvoll erscheint oder die Anwendungen Datenkonvertierung innerhalb bestimmter Segmentgrenzen erfordern.

Die Segmentierung durch die Anwendung wird in erster Linie aus den folgenden beiden Gründen verwendet:

1. Die Segmentierung durch den Warteschlangenmanager allein reicht nicht aus, weil die Nachricht auch für die Verarbeitung in einem einzigen Anwendungspuffer zu groß ist.
2. Die Datenkonvertierung muss durch die Senderkanäle vorgenommen werden und das Format setzt voraus, dass die einreihende Anwendung die Segmentgrenzen festlegt, damit die Konvertierung eines einzelnen Segments möglich ist.

Ist keine Datenkonvertierung erforderlich oder verwendet die abrufende Anwendung immer MQGMO_COMPLETE_MSG, kann jedoch auch die Segmentierung durch den Warteschlangenmanager erfolgen (geben Sie in diesem Fall MQMF_SEGMENTATION_ALLOWED an). Im Beispiel segmentiert die Anwendung die Nachricht in vier Segmente:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

Ohne Angabe von MQPMO_LOGICAL_ORDER muss die Anwendung den *Offset* und die Länge jedes Segments festlegen. In diesem Fall wird der logische Status nicht automatisch verwaltet.

Die abrufende Anwendung kann nicht garantieren, dass ihr Puffer groß genug für jede wieder zusammengesetzte Nachricht ist. Sie muss daher in der Lage sein, Segmente auch einzeln zu verarbeiten.

Diese Anwendung beginnt bei segmentierten Nachrichten keine Verarbeitung, so lange nicht alle Segmente einer logischen Nachricht vorhanden sind. Daher ist für das erste Segment MQGMO_ALL_SEGMENTS_AVAILABLE angegeben. Geben Sie allerdings MQGMO_LOGICAL_ORDER an und eine aktuelle logische Nachricht ist vorhanden, wird MQGMO_ALL_SEGMENTS_AVAILABLE ignoriert.

Verwenden Sie nach dem Abrufen des ersten Segments einer logischen Nachricht die Option MQGMO_LOGICAL_ORDER, um sicherzustellen, dass die verbleibenden Segmente der logischen Nachricht in der richtigen Reihenfolge abgerufen werden.

Dies gilt nicht für Nachrichten anderer Gruppen. Sollten solche Nachrichten eintreffen, werden sie in der Reihenfolge verarbeitet, in der das erste Segment jeder Nachricht in der Warteschlange eingeht.

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi *Segmentierung logischer Nachrichten durch die Anwendung*

Die Nachrichten müssen in logischer Reihenfolge in einer Gruppe verwaltet werden und einige oder auch alle Nachrichten sind so groß, dass sie durch die Anwendung segmentiert werden müssen.

In unserem Beispiel wird eine Gruppe mit vier logischen Nachrichten eingereiht. Alle mit Ausnahme der dritten Nachricht sind groß und müssen segmentiert werden. Die Segmentierung erfolgt durch die einreihende Anwendung:

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

In der abrufenden Anwendung wird das erste MQGET mit MQGMO_ALL_MSGS_AVAILABLE ausgeführt. Das bedeutet, dass erst dann Nachrichten oder Segmente einer Gruppe abgerufen werden, wenn die Gruppe vollständig eingereiht ist. Nach dem Abrufen der ersten physischen Nachricht einer Gruppe wird mit MQGMO_LOGICAL_ORDER sichergestellt, dass die verbleibenden Segmente und Nachrichten der Gruppe der Reihenfolge nach abgerufen werden:

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
             (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the GroupStatus
      and SegmentStatus information to see what has been returned */
   ...

MQCMIT
```

Anmerkung: Geben Sie allerdings MQGMO_LOGICAL_ORDER an und eine aktuelle Gruppe ist vorhanden, wird MQGMO_ALL_MSGS_AVAILABLE ignoriert.

Multi *Referenznachrichten und Übertragungen großer Objekte*

Referenznachrichten ermöglichen die Übertragung eines großen Objekts von einem Knoten zu einem anderen, ohne das Objekt in IBM MQ -Warteschlangen auf dem Quellen-oder Zielknoten zu speichern. Besonders vorteilhaft ist dies, wenn die Daten in einem anderen Format vorliegen, wie dies z. B. bei E-Mail-Anwendungen der Fall ist.

Um diese Übertragungsmethode zu aktivieren, geben Sie einen Nachrichtenexit an beiden Enden eines Kanals an. Weitere Informationen hierzu finden Sie im Abschnitt „Kanalnachrichtenexitprogramme” auf Seite 1031.

IBM MQ gibt das Format eines Referenznachrichtenheaders (MQRMH) vor. Eine Beschreibung hierzu finden Sie im Abschnitt MQRMH. Dieser erhält einen definierten Formatnamen, dem die tatsächlichen Daten unmittelbar folgen können.

Zur Initiierung der Übertragung eines großen Objekts kann eine Anwendung eine Nachricht mit einem Referenznachrichtenheader, jedoch ohne Daten, einreihen. Wenn diese Nachricht den Knoten verlässt, ruft der Nachrichtenexit das zugehörige Objekt auf eine geeignete Weise ab und hängt es an die Referenznachricht an. Danach gibt er die Nachricht (nun größer als zuvor) an den sendenden Nachrichtenkanalagenten (MCA) zur Übertragung an den empfangenden MCA zurück.

Am empfangenden MCA ist ein weiterer Nachrichtenexit konfiguriert. Wenn dieser Exit eine solche Nachricht erhält, erstellt er das Objekt anhand der angehängten Objektdaten, leitet die Referenznachricht jedoch *ohne* diese Daten weiter. Die Referenznachricht kann nun bei einer Anwendung eingehen, und diese Anwendung weiß, dass das Objekt (oder zumindest der Teil davon, der durch diese Referenznachricht dargestellt wird) auf diesem Knoten erstellt wurde.

Die maximale Menge an Objektdaten, die ein sendender Nachrichtenexit an eine Referenznachricht anhängen kann, wird durch die verhandelte maximale Nachrichtenlänge für den Kanal beschränkt. Der Exit darf für jede weitergeleitete Nachricht nur eine Nachricht an den MCA zurückgeben, weshalb die einreihende Anwendung für die Übertragung eines Objekts auch mehrere Nachrichten einreihen kann. Jede Nachricht muss die *logische* Länge und den Offset des Objekts angeben, das an die Nachricht angehängt wird. In Fällen, in denen jedoch die Gesamtgröße des Objekts bzw. die für den Kanal maximal zulässige Größe nicht bekannt ist, sollten Sie den sendenden Nachrichtenexit so konfigurieren, dass die einreihende Anwendung nur die erste Nachricht einreiht und der Exit an diese so viele Daten wie möglich anhängt und danach selbst die nächste Nachricht in die Übertragungswarteschlange einreiht.

Allerdings sollten Sie bei Anwendung dieser Methode für große Nachrichten die folgenden Punkte berücksichtigen:

- Der MCA und der Nachrichtenexit laufen unter einer IBM MQ-Benutzer-ID. Der Nachrichtenexit (und daher auch die Benutzer-ID) muss auf das Objekt zugreifen können, um es an der Sendeseite abzurufen

oder an der Empfangsseite zu erstellen. Unter Umständen ist dies nur möglich, wenn das Objekt allgemein zugänglich ist. Dadurch kann ein Sicherheitsproblem entstehen.

- Wenn die Referenznachricht mit den angehängten Massendaten mehrere Warteschlangenmanager passieren muss, bevor sie ihr Ziel erreicht, stehen die Massendaten in den IBM MQ-Warteschlangen auf den zwischengeschalteten Knoten zur Verfügung. Jedoch müssen in diesem Fall weder eine spezielle Unterstützung noch weitere Exits bereitgestellt werden.
- Bei Weiterleitungen oder Warteschlangen für nicht zustellbare Nachrichten ist das Design der Nachrichtenexits recht anspruchsvoll. Eventuell geraten die einzelnen Teile des Objekts in einem solchen Fall bei der Zustellung aus der Reihe.
- Wenn eine Referenznachricht ihr Ziel erreicht, stellt der Nachrichtenexit der Empfangsseite das Objekt wieder her. Diese Zusammensetzung ist jedoch nicht mit der Arbeitseinheit des MCA synchronisiert. Bei einem Backout des Stapels trifft daher eine weitere Referenznachricht mit dem gleichen Objektteil mit einem späteren Stapel ein, und der Nachrichtenexit versucht unter Umständen diesen Objektteil noch einmal wiederherzustellen. Handelt es sich bei dem Objekt zum Beispiel um eine aufeinanderfolgende Reihe von Datenbankaktualisierungen, wäre ein solches Vorkommnis nicht akzeptabel. Der Nachrichtenexit müsste in einem solchen Fall protokollieren, welche Aktualisierungen bereits angewendet wurden. Dazu wäre unter Umständen eine IBM MQ-Warteschlange erforderlich.
- Je nach Eigenschaften des Objekttyps müssen die Nachrichtenexits und die Anwendungen bei der Führung der Nutzungszähler zusammenarbeiten, damit das Objekt gelöscht werden kann, wenn es nicht mehr benötigt wird. Eventuell ist auch eine Instanzkennung erforderlich. Hierfür ist ein Feld im Header der Referenznachricht vorgesehen (siehe MQRMH).
- Wenn eine Referenznachricht als Verteilerliste eingereiht wird, muss das Objekt für jede daraus folgende Verteilerliste bzw. für jedes Einzelziel auf diesem Knoten abrufbar sein. Eventuell müssen Sie Nutzungszähler führen. Berücksichtigen Sie auch die Möglichkeit, dass ein Knoten für einige Ziele der Verteilerliste der letzte Knoten, für andere Ziele hingegen nur ein Zwischenknoten sein kann.
- Massendaten werden normalerweise nicht konvertiert, da die Konvertierung *vor* dem Aufruf des Nachrichtenexits stattfindet. Aus diesem Grund darf vom initiierenden Senderkanal keine Konvertierung angefordert werden. Wird die Referenznachricht über einen Zwischenknoten weitergeleitet, so werden die Massendaten, sofern angefordert, beim Verlassen des Zwischenknotens konvertiert.
- Referenznachrichten können nicht segmentiert werden.

MQRMH- und MQMD-Struktur verwenden

Eine Beschreibung der Felder im Header einer Referenznachricht und im Nachrichtendeskriptor finden Sie in den Abschnitten MQRMH und MQMD.

Setzen Sie das Feld *Format* in der MQMD-Struktur auf MQFMT_REF_MSG_HEADER. Das MQHREF-Format wird, wenn in MQGET angefordert, automatisch mit allen noch folgenden Massendaten von IBM MQ konvertiert.

Hier ein Beispiel zur Verwendung der Felder *DataLogicalOffset* und *DataLogicalLength* der MQRMH-Struktur:

Eine einreihende Anwendung reiht eine Referenznachricht mit folgendem Inhalt ein:

- Keine physischen Daten
- *DataLogicalLength* = 0 (diese Nachricht stellt das gesamte Objekt dar)
- *DataLogicalOffset* = 0.

Das Objekt sei 70.000 Byte groß, und der sendende Nachrichtenexit sendet die ersten 40.000 Byte in einer Referenznachricht mit folgendem Inhalt über den Kanal:

- 40.000 Byte physischer Daten nach dem MQRMH
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (vom Anfang des Objekts).

Danach fügt er eine weitere Nachricht mit folgendem Inhalt in die Übertragungswarteschlange ein:

- Keine physischen Daten
- *DataLogicalLength* = 0 (zum Ende des Objekts). Sie könnten hier auch den Wert 30.000 angeben.
- *DataLogicalOffset* = 40000 (ab diesem Punkt).

Sobald dieser Nachrichtenexit vom sendenden Nachrichtenexit erkannt wird, werden die verbleibenden 30.000 Byte an Daten angehängt und die Felder werden wie folgt eingestellt:

- 30.000 Byte physischer Daten nach dem MQRMH
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (ab diesem Punkt).

Außerdem wird das Flag MQRMHF_LAST gesetzt.

Eine Beschreibung der für Referenznachrichten bereitgestellten Beispielprogramme finden Sie im Abschnitt „Beispielprogramme plattformübergreifend verwenden” auf Seite 1112.

Warten auf Nachrichten

Wenn ein Programm warten soll, bis eine Nachricht in einer Warteschlange eintrifft, geben Sie im Feld *Options* der MQGMO-Struktur die Option MQGMO_WAIT an.

Im Feld *WaitInterval* der MQGMO-Struktur können Sie die maximale Zeit (in Millisekunden) angeben, die ein MQGET-Aufruf auf den Eingang einer Nachricht in der Warteschlange wartet.

Wenn die Nachricht nicht innerhalb dieser Zeit eintrifft, wird der MQGET-Aufruf mit dem Ursachencode MQRC_NO_MSG_AVAILABLE beendet.

Mit der Konstante MQWI_UNLIMITED können Sie im Feld *WaitInterval* auch ein unbegrenztes Warteintervall festlegen. Allerdings können Ereignisse außerhalb Ihrer Kontrolle dazu führen, dass das Programm sehr lange wartet. Sie sollten diese Konstante daher mit Vorsicht verwenden. Bei IMS-Anwendungen darf kein unbegrenztes Warteintervall festgelegt werden, da dies eine Beendigung des IMS-Systems verhindert. (Eine Beendigung von IMS setzt voraus, dass alle abhängigen Regionen beendet sind.) Stattdessen empfiehlt sich in IMS-Anwendungen ein begrenztes Warteintervall. Wenn der Aufruf dann nach diesem Intervall ohne Rückgabe einer Nachricht beendet wird, können Sie einen weiteren MQGET-Aufruf mit der Warteoption ausgeben.

Anmerkung: Wenn mehrere Programme an derselben gemeinsam genutzten Warteschlange darauf warten, eine Nachricht zu *entfernen*, wird nur ein Programm durch eine eingehende Nachricht aktiviert. Wenn jedoch mehrere Programme darauf warten, eine Nachricht mittels Browsing anzuzeigen, können alle Programme aktiviert werden. Weitere Informationen finden Sie unter der Beschreibung des Felds *Options* der MQGMO-Struktur in MQGMO.

Wenn sich der Status der Warteschlange oder des Warteschlangenmanagers vor Ablauf des Warteintervalls ändert, finden folgende Aktionen statt:

- Wenn der Warteschlangenmanager in den Quiescestatus wechselt und Sie die Option MQGMO_FAIL_IF_QUIESCING verwendet haben, wird das Warteintervall aufgehoben und der MQGET-Aufruf mit dem Ursachencode MQRC_Q_MGR_QUIESCING beendet. Ohne diese Option wartet der Aufruf weiterhin.
- z/OS Unter z/OS: Wenn die Verbindung (für eine CICS- oder IMS-Anwendung) in den Quiescestatus wechselt und Sie die Option MQGMO_FAIL_IF_QUIESCING verwendet haben, wird das Warteintervall aufgehoben und der MQGET-Aufruf mit dem Ursachencode MQRC_CONN_QUIESCING beendet. Ohne diese Option wartet der Aufruf weiterhin.
- Wenn der Warteschlangenmanager zum Herunterfahren gezwungen oder abgebrochen wird, wird der MQGET-Aufruf mit dem Ursachencode MQRC_Q_MGR_STOPPING oder MQRC_CONNECTION_BROKEN beendet.
- Wenn die Attribute der Warteschlange (oder einer Warteschlange, auf die sich der Warteschlangenname auflöst) geändert werden, so dass die GET-Anforderungen nun unterdrückt werden, wird das Warteintervall aufgehoben und der MQGET-Aufruf mit dem Ursachencode MQRC_GET_INHIBITED beendet.

- Wenn die Attribute der Warteschlange (oder einer Warteschlange, auf die sich der Warteschlangenname auflöst) auf eine Weise geändert werden, dass die Option FORCE erforderlich wird, wird das Warteintervall aufgehoben und der MQGET-Aufruf mit dem Ursachencode MQRC_OBJECT_CHANGED beendet.

z/OS Soll Ihre Anwendung an mehreren Warteschlangen warten, verwenden Sie die Signalfunktion von IBM MQ for z/OS (siehe „Signaling" auf Seite 839). Weitere Informationen zu den Bedingungen, unter denen diese Aktionen auftreten, finden Sie im Abschnitt MQGMO.

Signaling

Signaling is supported only on IBM MQ for z/OS.

Signaling is an option on the MQGET call to allow the operating system to notify (or *signal*) a program when an expected message arrives on a queue. This is like the *get with wait* function described in topic "Warten auf Nachrichten" on page 838 because it allows your program to continue with other work while waiting for the signal. However, if you use signaling, you can free the application thread and rely on the operating system to notify the program when a message arrives.

To set a signal

To set a signal, do the following in the MQGMO structure that you use on your MQGET call:

1. Set the MQGMO_SET_SIGNAL option in the *Options* field.
2. Set the maximum life of the signal in the *WaitInterval* field. This sets the length of time (in milliseconds) for which you want IBM MQ to monitor the queue. Use the MQWI_UNLIMITED value to specify an unlimited life.

   Note: IMS applications must not specify an unlimited wait interval because this would prevent the IMS system from terminating. (When IMS terminates, it requires all dependent regions to end.) Instead, IMS applications can examine the state of the ECB at regular intervals (see step 3). A program can have signals set on several queue handles at the same time:
3. Specify the address of the *Event Control Block* (ECB) in the *Signal1* field. This notifies you of the result of your signal. The ECB storage must remain available until the queue is closed.

Note: You cannot use the MQGMO_SET_SIGNAL option with the MQGMO_WAIT option.

When the message arrives

When a suitable message arrives, a completion code is returned to the ECB.

The completion code describes one of the following:

- The message that you set the signal for has arrived on the queue. The message is not reserved for the program that requested a signal, so the program must issue an MQGET call again to get the message.

  Note: Another application could get the message in the time between your receiving the signal and issuing another MQGET call.
- The wait interval you set has expired and the message you set the signal for did not arrive on the queue. IBM MQ has canceled the signal.
- The signal has been canceled. This happens, for example, if the queue manager stops, or the attribute of the queue is changed, so that MQGET calls are no longer allowed.

When a suitable message is already on the queue, the MQGET call completes in the same way as an MQGET call without signaling. Also, if an error is detected immediately, the call completes and the return codes are set.

When the call is accepted and no message is immediately available, control is returned to the program so that it can continue with other work. None of the output fields in the message descriptor are set, but the **CompCode** parameter is set to MQCC_WARNING and the **Reason** parameter is set to MQRC_SIGNAL_REQUEST_ACCEPTED.

For information about what IBM MQ can return to your application when it makes an MQGET call using signaling, see MQGET.

If the program has no other work to do while it is waiting for the ECB to be posted, it can wait for the ECB using:

- For a CICS Transaction Server for z/OS program, the EXEC CICS WAIT EXTERNAL command
- For batch and IMS programs, the z/OS WAIT macro

If the state of the queue or the queue manager changes while the signal is set (that is, the ECB has not yet been posted), the following actions occur:

- If the queue manager enters the quiescing state, and you used the MQGMO_FAIL_IF_QUIESCING option, the signal is canceled. The ECB is posted with the MQEC_Q_MGR_QUIESCING completion code. Without this option, the signal remains set.
- If the queue manager is forced to stop, or is canceled, the signal is canceled. The signal is delivered with the MQEC_WAIT_CANCELED completion code.
- If the attributes of the queue (or a queue to which the queue name resolves) are changed so that get requests are now inhibited, the signal is canceled. The signal is delivered with the MQEC_WAIT_CANCELED completion code.

Note:

1. If more than one program has set a signal on the same shared queue to remove a message, only one program is activated by a message arriving. However, if more than one program is waiting to browse a message, all the programs can be activated. The rules that the queue manager follows when deciding which applications to activate are the same as those for waiting applications: for more information, see the description of the *Options* field of the MQGMO structure in MQGMO - Get-message options.
2. If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally. For more information, see the description of the *Options* field of the MQGMO structure in MQGMO - Get-message options.
3. Under some conditions, it is possible both for an MQGET call to retrieve a message and for a signal (resulting from the arrival of the same message) to be delivered. This means that when your program issues another MQGET call (because the signal was delivered), there could be no message available. Design your program to test for this situation.

For information about how to set a signal, see the description of the MQGMO_SET_SIGNAL option and the *Signal1* field in Signal1.

z/OS *Backout überspringen*

Mit der Option **MQGMO_MARK_SKIP_BACKOUT** des MQGET-Aufrufs verhindern Sie, dass ein Anwendungsprogramm in eine *MQGET-error-backout*-Schleife gerät.

Innerhalb einer Arbeitseinheit kann ein Anwendungsprogramm einen oder mehrere MQGET-Aufrufe ausgeben, um Nachrichten aus einer Warteschlange abzurufen. Falls das Anwendungsprogramm dabei einen Fehler feststellt, kann es die Arbeitseinheit mittels Backout zurücksetzen. Dadurch werden alle Ressourcen, die im Zuge dieser Arbeitseinheit aktualisiert wurden, auf den Status vor Beginn der Arbeitseinheit zurückgesetzt, und die von den MQGET-Aufrufen abgerufenen Nachrichten werden wiederhergestellt.

Nach der Wiederherstellung stehen diese Nachrichten wieder nachfolgenden, vom Anwendungsprogramm ausgegebenen MQGET-Aufrufen zur Verfügung. In vielen Fällen stellt dies kein Problem für das Anwendungsprogramm dar. In Fällen allerdings, in denen der Fehler, der zu dem Backout geführt hat, nicht behoben oder umgangen werden kann, kann eine Wiederherstellung der Nachricht in der Warteschlange bewirken, dass für das Anwendungsprogramm eine *MQGET-error-backout*-Schleife beginnt.

Zur Vermeidung dieses Problems sollten Sie im MQGET-Aufruf die Option MQGMO_MARK_SKIP_BACKOUT angeben. Dadurch wird angegeben, dass die MQGET-Anforderung nicht in die von der Anwendung veranlassten Backouts involviert sein soll. Wenn es bei Verwendung dieser Option zu einem Backout kommt, werden die Aktualisierungen an anderen Ressourcen wie gefordert zurückgesetzt, die gekenn-

zeichnete Nachricht wird hingegen so behandelt, als ob sie unter einer neuen Arbeitseinheit abgerufen worden wäre.

Das Anwendungsprogramm muss in diesem Fall einen eigenen IBM MQ-Aufruf zum Festschreiben oder zum Zurücksetzen der neuen Arbeitseinheit ausgeben. Das Programm kann zum Beispiel eine Ausnahmebehandlung durchführen, zum Beispiel den Absender der Nachricht informieren, dass die Nachricht verworfen wurde, und die Arbeitseinheit festschreiben, damit die Nachricht aus der Warteschlange entfernt wird. Wenn die neue Arbeitseinheit dann, aus welchem Grund auch immer, mittels Backout zurückgesetzt wird, wird die Nachricht in der Warteschlange wiederhergestellt.

Innerhalb einer Arbeitseinheit kann nur für eine MQGET-Anforderung angegeben sein, dass das Backout übersprungen wird; allerdings kann die Arbeitseinheit noch zahlreiche weitere Nachrichten enthalten, für die die Option MQGMO_MARK_SKIP_BACKOUT nicht angegeben ist. Ist bereits für eine Nachricht innerhalb einer Arbeitseinheit MQGMO_MARK_SKIP_BACKOUT angegeben, so schlagen alle weiteren MQGET-Aufrufe innerhalb der gleichen Arbeitseinheit, für die MQGMO_MARK_SKIP_BACKOUT ebenfalls angegeben wird, mit dem Ursachencode MQRC_SECOND_MARK_NOT_ALLOWED fehl.

Anmerkung:

1. Für die solchermaßen gekennzeichnete Nachricht wird das Backout nur übersprungen, wenn die Arbeitseinheit, in der die Nachricht enthalten ist, durch eine Anwendungsanforderung für deren Backout beendet wird. Wenn die Arbeitseinheit aus einem anderen Grund zurückgesetzt wird, wird die Nachricht genauso aus der Warteschlange zurückgesetzt, als ob für sie kein Überspringen des Backouts angegeben wäre.
2. Das Überspringen des Backouts wird von gespeicherten Db2-Prozeduren in durch RRS gesteuerten Arbeitseinheiten nicht unterstützt. Hier schlägt ein MQGET-Aufruf mit der Option MQGMO_MARK_SKIP_BACKOUT mit dem Ursachencode MQRC_OPTION_ENVIRONMENT_ERROR fehl.

Abbildung 63 auf Seite 842 zeigt einen typischen Ablauf in einem Anwendungsprogramm, wenn das Backout mit einer MQGET-Anforderung übersprungen werden soll.

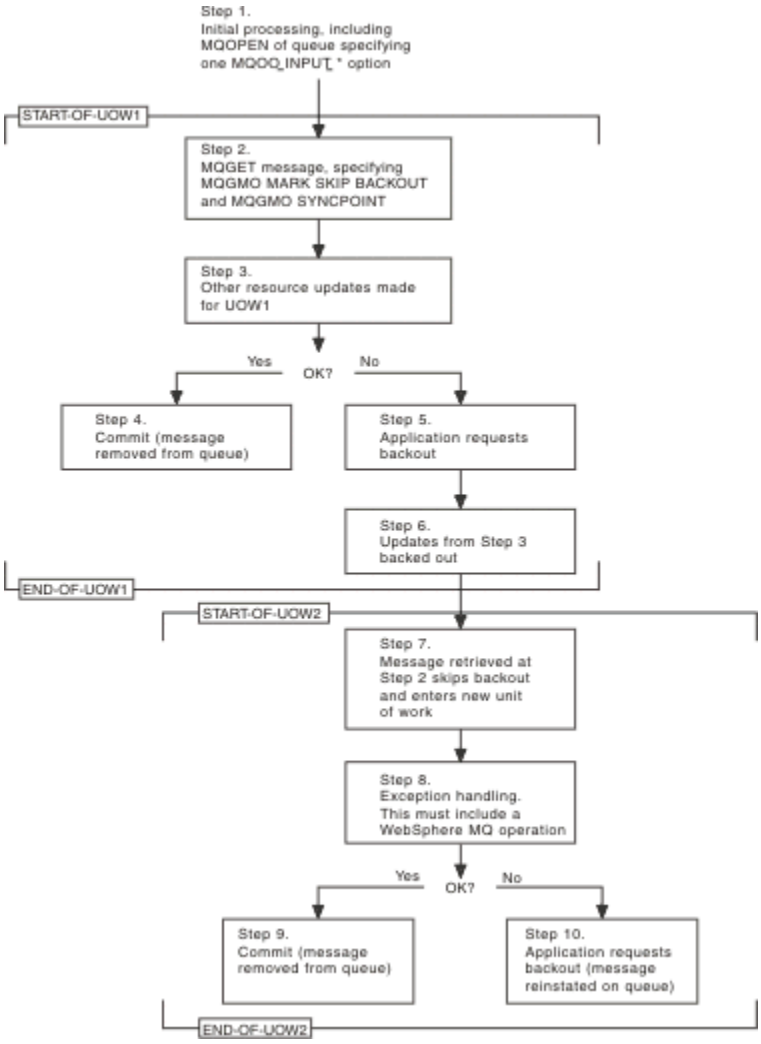


Abbildung 63. Backout mit MQGMO_MARK_SKIP_BACKOUT überspringen

Schritte in Abbildung 63 auf Seite 842:

Schritt 1
Der erste Verarbeitungsschritt beginnt innerhalb der Transaktion mit einem MQOPEN-Aufruf zum Öffnen der Warteschlange (mit einer MQOO_INPUT_*-Option, damit in Schritt 2 Nachrichten aus der Warteschlange abgerufen werden können).

Schritt 2
MQGET wird mit MQGMO_SYNCPOINT und MQGMO_MARK_SKIP_BACKOUT aufgerufen. MQGMO_SYNCPOINT ist erforderlich, da sich MQGET innerhalb einer Arbeitseinheit befinden muss, damit MQGMO_MARK_SKIP_BACKOUT wirksam wird. In Abbildung 63 auf Seite 842 wird diese Arbeitseinheit als UOW1 bezeichnet.

Schritt 3
Im Rahmen von UOW1 werden weitere Ressourcenaktualisierungen vorgenommen. Diese können weitere MQGET-Aufrufe (ohne MQGMO_MARK_SKIP_BACKOUT) einschließen.

Schritt 4
Alle Aktualisierungen aus den Schritten 2 und 3 werden wie angefordert durchgeführt. Das Anwendungsprogramm schreibt die Aktualisierungen fest und UOW1 endet. Die in Schritt 2 abgerufene Nachricht wird aus der Warteschlange entfernt.

Schritt 5
Einige Aktualisierungen aus den Schritten 2 und 3 konnten nicht wie angefordert durchgeführt werden. Das Anwendungsprogramm fordert die Zurücksetzung der während dieser Schritte vorgenommenen Aktualisierungen an.

Schritt 6
Die in Schritt 3 vorgenommenen Aktualisierungen werden zurückgesetzt.

Schritt 7
Das Backout für die in Schritt 2 ausgegebene MQGET-Anforderung wird übersprungen, und die Anforderung wird in eine neue Arbeitseinheit, UOW2, verschoben.

Schritt 8
Als Folge der Zurücksetzung von UOW1 führt UOW2 eine Ausnahmebehandlung durch (z. B. einen MQPUT-Aufruf an eine andere Warteschlange mit dem Hinweis, dass ein Problem aufgetreten ist, das zur Zurücksetzung von UOW1 geführt hat.)

Schritt 9
Schritt 8 wird wie angefordert ausgeführt, das Anwendungsprogramm schreibt den Vorgang fest und UOW2 endet. Da die MQGET-Anforderung Teil von UOW2 ist (siehe Schritt 7), bewirkt dieses Commit, dass die Nachricht aus der Warteschlange entfernt wird.

Schritt 10
Schritt 8 wird nicht wie angefordert ausgeführt, weshalb das Anwendungsprogramm UOW2 zurücksetzt. Da die MQGET-Anforderung Teil von UOW2 ist (siehe Schritt 7), wird sie ebenfalls zurückgesetzt und in der Warteschlange wiederherstellt. Sie steht nun (genauso wie alle anderen Nachrichten der Warteschlange) anderen MQGET-Aufrufen aus diesem oder einem anderen Anwendungsprogramm zur Verfügung.

Anwendungsdatenkonvertierung

Bei Bedarf konvertieren die Nachrichtenkanalagenten (MCAs) Nachrichtendeskriptor und Headerdaten in den erforderlichen Zeichensatz und die benötigte Codierung. Dabei kann die Konvertierung an beiden Seiten der Verbindung (also durch den lokalen oder den fernen MCA) erfolgen.

Wenn eine Anwendung Nachrichten in eine Warteschlange einreiht, fügt der lokale Warteschlangenmanager den Nachrichtendeskriptoren Steuerinformationen hinzu, damit die Nachrichten während der Verarbeitung durch die Warteschlangenmanager und MCAs gesteuert werden können. Je nach Umgebung werden die Datenfelder der Nachrichtenheader mit dem Zeichensatz und der Codierung des lokalen Systems erstellt.

Beim Verschieben von Nachrichten zwischen Systemen müssen die Anwendungsdaten eventuell in den Zeichensatz und die Codierung des empfangenden Systems konvertiert werden. Diese Konvertierung kann durch die MCAs des sendenden Systems vorgenommen werden oder in Anwendungsprogrammen auf dem empfangenden System erfolgen. Wenn das empfangende System eine Datenkonvertierung unterstützt, sollten Sie zur Konvertierung der Anwendungsdaten die Anwendungsprogramme verwenden und sich nicht auf die bereits auf dem sendenden System vorgenommene Konvertierung verlassen.

Anwendungsdaten werden innerhalb eines Anwendungsprogramms konvertiert, wenn Sie im Feld *Options* der an einen MQGET-Aufruf übergebenen MQGMO-Struktur die Option MQGMO_CONVERT angeben und *sämtliche* der folgenden Bedingungen zutreffen:

- Der Wert im Feld *CodedCharSetId* bzw. *Encoding* der MQMD-Struktur der Nachricht in der Warteschlange unterscheidet sich vom Wert im Feld *CodedCharSetId* bzw. *Encoding* der MQMD-Struktur des MQGET-Aufrufs.
- Das Feld *Format* der MQMD-Struktur der Nachricht ist nicht auf MQFMT_NONE gesetzt.
- Die im MQGET-Aufruf angegebene *BufferLength* ist nicht null.
- Die Nachrichtendatenlänge ist nicht null.
- Der Warteschlangenmanager unterstützt die Konvertierung zwischen den Feldern *CodedCharSetId* und *Encoding* der MQMD-Struktur der Nachricht und des MQGET-Aufrufs. Details zu den IDs der codierten Zeichensätze und den unterstützten Maschinencodierungen finden Sie in den Abschnitten CodedCharSetId und Codierung.
- Der Warteschlangenmanager unterstützt die Konvertierung des Nachrichtenformats. Wenn im Feld *Format* der MQMD-Struktur der Nachricht eines der integrierten Formate angegeben ist, kann der Warteschlangenmanager die Nachricht konvertieren. Handelt es sich bei *Format* um keines der integrierten Formate, müssen Sie zur Konvertierung der Nachricht ein Datenkonvertierungsexit schreiben.

Wenn der sendende MCA die Daten konvertieren soll, geben Sie in der Definition der Sender- oder Serverkanäle, für die eine Konvertierung erforderlich ist, das Schlüsselwort CONVERT(YES) an. Bei einem Fehlschlagen der Datenkonvertierung wird die Nachricht an die Warteschlange für nicht zustellbare Nachrichten des sendenden Warteschlangenmanagers gesendet. Der Grund wird im Feld *Feedback* der MQDLH-Struktur angegeben. Kann die Nachricht nicht in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden, wird der Kanal geschlossen und die nicht konvertierte Nachricht verbleibt in der Übertragungswarteschlange. Durch eine Datenkonvertierung innerhalb der Anwendungsprogramme (anstatt durch die sendenden MCAs) wird eine solche Situation vermieden.

In der Regel werden Nachrichtendaten, die vom integrierten Format oder dem Datenkonvertierungsexit als *Zeichendaten* behandelt werden, vom codierten Zeichensatz der Nachricht in das angeforderte Format konvertiert. *Numerische* Daten werden dagegen in die angeforderte Codierung konvertiert.

Nähere Informationen zu den Konvertierungskonventionen für integrierte Formate sowie Informationen zur Erstellung eigener Datenkonvertierungsexits finden Sie im Abschnitt „Datenkonvertierungsexits schreiben” auf Seite 1035. Informationen zu den Sprachunterstützungstabellen und den unterstützten Maschinencodierungen finden Sie auch in den Abschnitten Landessprachen und Maschinencodierungen.

Konvertierung von EBCDIC-Zeilenvorschubzeichen

Wenn Sie sicherstellen wollen, dass die Daten, die Sie von einer EBCDIC-Plattform an eine ASCII-Plattform senden, identisch mit den Daten sind, die Sie wieder zurück erhalten, müssen Sie die Konvertierung der EBCDIC-Zeilenvorschubzeichen steuern.

Dies erreichen Sie mit einem plattformabhängigen Switch, der IBM MQ zur Verwendung unveränderter Konvertierungstabellen zwingt. Allerdings sollte Ihnen bewusst sein, dass dies zu einem inkonsistenten Verhalten führen kann.

Grund hierfür ist, dass das EBCDIC-Zeilenvorschubzeichen nicht auf allen Plattformen (bzw. durch alle Konvertierungstabellen) gleich konvertiert wird. Auf einer ASCII-Plattform werden die Daten daher unter Umständen nicht korrekt formatiert angezeigt. Dies wiederum erschwert zum Beispiel die Fernverwaltung eines IBM i-Systems von einer ASCII-Plattform mit RUNMQSC.

Weitere Informationen zur Konvertierung von EBCDIC-Daten in das ASCII-Format finden Sie im Abschnitt Datenkonvertierung.

Nachrichten in einer Warteschlange durchsuchen (Browsing)

In diesem Abschnitt erfahren Sie, wie die Nachrichten einer Warteschlange mit dem MQGET-Aufruf durchsucht werden.

So durchsuchen Sie die Nachrichten einer Warteschlange mit dem MQGET-Aufruf:

1. Rufen Sie MQOPEN mit der Option MQOO_BROWSE auf, um die Warteschlange zum Durchsuchen zu öffnen.
2. Zum Anzeigen der ersten Nachricht der Warteschlange rufen Sie MQGET mit der Option MQGMO_BROWSE_FIRST auf. Um die gewünschte Nachricht zu finden, rufen Sie MQGET wiederholt mit der Option MQGMO_BROWSE_NEXT auf, um die einzelnen Nachrichten nacheinander anzuzeigen.

   Sie müssen die Felder *MsgId* und *CorrelId* der MQMD-Struktur nach jedem MQGET-Aufruf auf Nullwerte setzen, um alle Nachrichten zu sehen.
3. Rufen Sie zum Schließen der Warteschlange MQCLOSE auf.

Anzeigecursor

Wenn Sie eine Warteschlange mit MQOPEN zum Durchsuchen (Browsing) öffnen, richtet der Aufruf für alle MQGET-Aufrufe mit einer der Suchoptionen einen Anzeigecursor ein. Den Anzeigecursor können Sie sich als logischen Zeiger vorstellen, der sich vor der ersten Nachricht der Warteschlange befindet.

Sie können innerhalb des gleichen Programms auch mehrere Anzeigecursor einrichten, indem Sie für die gleiche Warteschlange mehrere MQOPEN-Anforderungen ausgeben.

Wenn Sie MQGET zum Browsen aufrufen wollen, müssen Sie in Ihrer MQGMO-Struktur eine der folgenden Optionen angeben:

MQGMO_BROWSE_FIRST
Ruft eine Kopie der ersten Nachricht ab, die die in Ihrer MQMD-Struktur angegebenen Bedingungen erfüllt.

MQGMO_BROWSE_NEXT
Ruft eine Kopie der nächsten Nachricht ab, die die in Ihrer MQMD-Struktur angegebenen Bedingungen erfüllt.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Ruft eine Kopie der Nachricht ab, auf die der Cursor gerade zeigt, d. h. eine Kopie der Nachricht, die zuletzt mit MQGMO_BROWSE_FIRST oder MQGMO_BROWSE_NEXT abgerufen wurde.

In allen Fällen verbleibt die Nachricht in der Warteschlange.

Unmittelbar nach dem Öffnen einer Warteschlange befindet sich der Anzeigecursor vor der ersten Nachricht der Warteschlange. Wenn Sie also direkt nach dem MQOPEN-Aufruf einen MQGET-Aufruf ausgeben, können Sie auch mit der Option MQGMO_BROWSE_NEXT zur ersten Nachricht browsen. Sie müssen in diesem Fall nicht MQGMO_BROWSE_FIRST angeben.

Die Reihenfolge, in der Nachrichten aus der Warteschlange kopiert werden, wird durch das Attribut **MsgDeliverySequence** der Warteschlange festgelegt. (Weitere Informationen finden Sie im Abschnitt „Abrufreihenfolge der Nachrichten aus einer Warteschlange” auf Seite 814.)

- „Warteschlangen in FIFO-Reihenfolge (First In First Out)” auf Seite 845
- „Warteschlangen in Prioritätsreihenfolge” auf Seite 845
- „Nicht festgeschriebene Nachrichten” auf Seite 846
- „Änderung an der Warteschlangenreihenfolge” auf Seite 846
- z/OS „Warteschlangenindex verwenden” auf Seite 846

Warteschlangen in FIFO-Reihenfolge (First In First Out)

Bei dieser Reihenfolge ist die erste Nachricht in einer Warteschlange die älteste Nachricht der Warteschlange.

Zum Anzeigen der Nachrichten in dieser Sequenz geben Sie MQGMO_BROWSE_NEXT an. In dieser Sequenz sehen Sie auch alle Nachrichten, die während des Browsens in die Warteschlange eingereiht werden, da neue Nachrichten bei Warteschlangen in FIFO-Reihenfolge am Ende angefügt werden. Erkennt der Cursor, dass er am Ende der Warteschlange angelangt ist, bleibt er dort stehen und gibt MQRC_NO_MSG_AVAILABLE zurück. Sie können den Cursor dort belassen oder ihn mit der Option MQGMO_BROWSE_FIRST an den Anfang der Warteschlange zurücksetzen.

Warteschlangen in Prioritätsreihenfolge

Bei dieser Reihenfolge ist die erste Nachricht in einer Warteschlange die älteste Nachricht der Warteschlange, die zum Zeitpunkt des MQOPEN-Aufrufs die höchste Priorität hatte.

Zum Anzeigen der Nachrichten in dieser Sequenz geben Sie MQGMO_BROWSE_NEXT an.

Der Anzeigecursor zeigt in der Reihenfolge ihrer Priorität (von der höchsten zur niedrigsten) jeweils auf die nächste Nachricht. Dabei zeigt er auch während des Browsens eingereihte Nachrichten an, so lange diese eine Priorität kleiner oder gleich der durch den aktuellen Anzeigecursor gekennzeichneten Nachricht haben.

Nachrichten mit einer höheren Priorität, die erst während des Browsens eingereiht werden, können nur wie folgt angezeigt werden:

- Erneutes Öffnen der Warteschlange zum Browsen, so dass ein neuer Anzeigecursor eingerichtet wird
- Verwenden der Option MQGMO_BROWSE_FIRST

Nicht festgeschriebene Nachrichten

Eine nicht festgeschriebene Nachricht ist bei einem Browse-Vorgang nicht sichtbar, so dass sie vom Anzeigecursor übersprungen wird.

Nachrichten innerhalb einer Arbeitseinheit können erst nach dem Festschreiben der Arbeitseinheit mittels Browsing angezeigt werden. Da Nachrichten ihre Position innerhalb der Warteschlange auch bei einem Commit nicht ändern, können zunächst nicht festgeschriebene und daher übersprungene Nachrichten nicht angezeigt werden, selbst wenn deren Commit mittlerweile *stattgefunden* hat, es sei denn, Sie starten die Suche mit MQGMO_BROWSE_FIRST von neuem.

Änderung an der Warteschlangenreihenfolge

Bei einer Änderung der Zustellungsreihenfolge der Nachrichten von Priorität auf FIFO, während sich bereits Nachrichten in der Warteschlange befinden, ändert sich die Reihenfolge der bereits in der Warteschlange enthaltenen Nachrichten nicht. Später hinzugefügte Nachrichten übernehmen die Standardpriorität der Warteschlange.

Warteschlangenindex verwenden

z/OS

Wenn Sie unter IBM MQ for z/OSeine indexierte Warteschlange durchsuchen, die nur Nachrichten mit einer einzigen Priorität enthält (persistent und/oder nicht persistent), verwendet der Warteschlangenmanager den Index zum Durchsuchen, wenn bestimmte Formen des Durchsuchens verwendet werden.

Die folgenden Browsing-Formen werden verwendet, wenn eine indizierte Warteschlange nur Nachrichten der gleichen Priorität enthält:

1. Wenn die Warteschlange durch die MSGID indiziert ist, werden Browse-Anforderungen, die in der MQMD-Struktur eine MSGID übergeben, zur Bestimmung der Zielnachricht unter Beachtung des Index verarbeitet.
2. Wenn die Warteschlange durch die CORRELID indiziert ist, werden Browse-Anforderungen, die in der MQMD-Struktur eine CORRELID übergeben, zur Bestimmung der Zielnachricht unter Beachtung des Index verarbeitet.
3. Wenn die Warteschlange durch die GROUPID indiziert ist, werden Browse-Anforderungen, die in der MQMD-Struktur eine GROUPID übergeben, zur Bestimmung der Zielnachricht unter Beachtung des Index verarbeitet.

Wenn die Browse-Anforderung in der MQMD-Struktur weder eine MSGID, noch eine CORRELID oder GROUPID übergibt, wird die Warteschlange indiziert und eine Nachricht wird zurückgegeben. Danach wird der Indexeintrag für die Nachricht gesucht und die darin enthaltenen Informationen werden zur Aktualisierung des Anzeigecursors verwendet. Selbst bei Verwendung einer großen Auswahl an Indexwerten bedeutet dies für die Browse-Anforderung keinen bedeutenden Mehraufwand.

Nachrichten durchsuchen, wenn die Nachrichtenlänge unbekannt ist

Wenn Sie eine Nachricht suchen, deren Größe Sie nicht kennen, und zum Auffinden keines der Felder *MsgId*, *CorrelId* oder *GroupId* verwenden möchten, können Sie die Option MQGMO_BROWSE_MSG_UNDER_CURSOR verwenden:

1. Geben Sie einen MQGET-Aufruf mit folgenden Optionen aus:
   - Entweder die Option MQGMO_BROWSE_FIRST oder MQGMO_BROWSE_NEXT
   - Die Option MQGMO_ACCEPT_TRUNCATED_MSG
   - Puffergröße gleich null

   Anmerkung: Falls es wahrscheinlich ist, dass die Nachricht auch durch ein anderes Programm abgerufen wird, sollten Sie eventuell auch die Option MQGMO_LOCK verwenden. Sie sollten dann MQRC_TRUNCATED_MSG_ACCEPTED zurückerhalten.
2. Weisen Sie den benötigten Speicher entsprechend dem als *DataLength* zurückgegebenen Wert zu.

3. Geben Sie einen MQGET-Aufruf mit der Option MQGMO_BROWSE_MSG_UNDER_CURSOR aus.

Nun wird auf die zuletzt abgerufene Nachricht gezeigt; der Anzeigecursor hat sich jedoch nicht bewegt. Sie können die Nachricht nun mit MQGMO_LOCK sperren oder eine bestehende Sperre mit MQGMO_UNLOCK aufheben.

Dieser Aufruf schlägt allerdings fehl, wenn seit dem Öffnen der Warteschlange noch kein MQGET mit MQGMO_BROWSE_FIRST oder MQGMO_BROWSE_NEXT erfolgreich ausgeführt wurde.

Mittels Browsing gefundene Nachricht entfernen
Bereits angezeigte Nachrichten können Sie aus einer Warteschlange entfernen, sofern Sie die Warteschlange sowohl zum Durchsuchen als auch zum Entfernen von Nachrichten geöffnet haben. (Sie müssen im MQOPEN-Aufruf eine der Optionen MQOO_INPUT_* sowie MQOO_BROWSE angegeben haben.)

Um die Nachricht zu entfernen, rufen Sie MQGET erneut auf, geben Sie jedoch im Feld *Options* der MQGMO-Struktur MQGMO_MSG_UNDER_CURSOR an.In diesem Fall ignoriert der MQGET-Aufruf die Felder *MsgId*, *CorrelId* und *GroupId* der MQMD-Struktur.

Zwischen Ihrer Anforderung zum Anzeigen und zum Entfernen kann es passieren, dass Nachrichten aus der Warteschlange, darunter auch die Nachricht unter ihrem Anzeigecursor, durch ein anderes Programm entfernt werden. In diesem Fall gibt der MQGET-Aufruf einen Ursachencode mit dem Hinweis zurück, dass die Nachricht nicht verfügbar ist.

Nachrichten in logischer Reihenfolge durchsuchen

Im Abschnitt „Logische und physische Reihenfolge” auf Seite 814 wird der Unterschied zwischen der logischen und physischen Reihenfolge von Nachrichten in einer Warteschlange erklärt. Diese Unterscheidung ist besonders beim Durchsuchen einer Warteschlange wichtig, da Nachrichten in der Regel nicht gelöscht werden und der Browse-Vorgang nicht immer am Anfang der Warteschlange beginnt.

Wenn eine Anwendung die verschiedenen Nachrichten einer Gruppe in logischer Reihenfolge durchsucht, sollte die logische Reihenfolge auch beim Übergang zur nächsten Gruppe berücksichtigt werden, da die letzte Nachricht der einen Gruppe physisch möglicherweise erst *nach* der ersten Nachricht der nächsten Gruppe eingetroffen ist. Mit der Option MQGMO_LOGICAL_ORDER stellen Sie beim Durchsuchen einer Warteschlange sicher, dass die logische Reihenfolge beachtet wird.

MQGMO_ALL_MSGS_AVAILABLE (bzw. MQGMO_ALL_SEGMENTS_AVAILABLE) sollten Sie bei Browse-Vorgängen mit Vorsicht einsetzen. Betrachten Sie den Fall einer Anzeige der logischen Nachrichten mit der Option MQGMO_ALL_MSGS_AVAILABLE. Bei dieser Option ist eine logische Nachricht nur verfügbar, wenn auch die verbleibenden Nachrichten der Gruppe bereits eingetroffen sind. Sind diese noch nicht vorhanden, wird die Nachricht übersprungen. Dies bedeutet jedoch, dass die fehlenden Nachrichten, wenn sie schließlich eintreffen, von einem MQGMO_BROWSE_NEXT-Vorgang nicht erkannt werden.

Beispiel: Die folgenden logischen Nachrichten seien vorhanden:

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

und der Browse-Aufruf wird mit der Option MQGMO_ALL_MSGS_AVAILABLE ausgeführt. In diesem Fall wird die erste logische Nachricht der Gruppe 456 zurückgegeben, wobei der Anzeigecursor auf dieser logischen Nachricht verbleibt. Wenn nun die zweite (letzte) Nachricht von Gruppe 123 eintrifft:

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
Logical message 1 (not last) of group 456  <=== browse cursor
Logical message 2 (last)     of group 456
```

und die gleiche MQGMO_BROWSE_NEXT-Funktion ausgeführt wird, wird nicht erkannt, dass Gruppe 123 mittlerweile vollständig ist, da sich die erste Nachricht dieser Gruppe *vor* dem Anzeigecursor befindet.

In einigen Fällen (z. B. wenn Nachrichten beim Abrufen entfernt werden sollen, sobald die Gruppe vollständig vorhanden ist) können Sie MQGMO_ALL_MSGS_AVAILABLE in Kombination mit der Option

MQGMO_BROWSE_FIRST verwenden. Andernfalls müssen Sie den Browse-Vorgang wiederholen, damit zuvor nicht erkannte, erst später eingetroffene Nachrichten erkannt werden. Durch MQGMO_WAIT mit MQGMO_BROWSE_NEXT und MQGMO_ALL_MSGS_AVAILABLE werden diese Nachrichten nicht erkannt. (Dies kann auch mit Nachrichten einer höheren Priorität passieren, wenn diese erst nach einem abgeschlossenen Browse-Vorgang eintreffen.)

In den nächsten Abschnitten finden Sie Beispiele für das Browsen (Durchsuchen bzw. Anzeigen von Nachrichten) bei nicht segmentierten Nachrichten. Bei segmentierten Nachrichten erfolgt das Browsen nach den gleichen Prinzipien.

Nachrichten in Gruppen durchsuchen
In diesem Beispiel durchsucht die Anwendung alle Nachrichten der Warteschlange in logischer Reihenfolge.

Die Nachrichten in einer Warteschlange können gruppiert sein. Bei gruppierten Nachrichten sollte die Anwendung erst mit der Verarbeitung einer Gruppe beginnen, wenn alle Nachrichten der Gruppe eingetroffen sind. Aus diesem Grund ist für die erste Nachricht der Gruppe MQGMO_ALL_MSGS_AVAILABLE angegeben; für alle weiteren Nachrichten der Gruppe ist diese Option unnötig.

In diesem Beispiel wird MQGMO_WAIT verwendet. Die WAIT-Bedingung kann zwar allein dadurch erfüllt sein, dass eine neue Gruppe eintrifft, aus den im Abschnitt „Nachrichten in logischer Reihenfolge durchsuchen” auf Seite 847 beschriebenen Gründen ist sie jedoch nicht erfüllt, wenn der Anzeigecursor bereits die erste logische Nachricht der Gruppe übergeben hat, und die verbleibenden Nachrichten erst danach eintreffen. Nichtsdestotrotz stellt eine ausreichend lange Wartezeit sicher, dass die Anwendung nicht jedes Mal in eine Schleife gerät, während sie auf neue Nachrichten oder Segmente wartet.

Mit MQGMO_LOGICAL_ORDER wird sichergestellt, dass das Browsing in logischer Reihenfolge erfolgt. Dies wird hier also anders gehandhabt als im MQGET-Beispiel mit Entfernen der Nachrichten, in dem ja jede Gruppe als Ganzes entfernt werden soll, weshalb MQGMO_LOGICAL_ORDER nicht bei der Suche nach der ersten (oder einzigen) Nachricht einer Gruppe verwendet wird.

Es wird davon ausgegangen, dass der Anwendungspuffer groß genug für die gesamte Nachricht ist, unabhängig davon, ob sie segmentiert wurde. Daher wird in jedem MQGET MQGMO_COMPLETE_MSG angegeben.

Das folgende Beispiel sucht nach den logischen Nachrichten einer Gruppe:

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group */
   ...
```

Diese Gruppe wird wiederholt, bis MQRC_NO_MSG_AVAILABLE zurückgegeben wird.

Nachrichten durchsuchen, abrufen und entfernen
In diesem Beispiel durchsucht die Anwendung zunächst alle logischen Nachrichten einer Gruppe, bevor sie entscheidet, ob die Gruppe insgesamt abgerufen und entfernt werden soll.

Der erste Teil dieses Beispiels ist mit dem vorangegangenen identisch. Nachdem jedoch die gesamte Gruppe gefunden wurde, gehen wir wieder zurück an den Anfang, um die Gruppe abzurufen und zu entfernen.

Da in diesem Beispiel jede Gruppe als Ganzes entfernt wird, wird MQGMO_LOGICAL_ORDER nicht bei der Suche nach der ersten (oder einzigen) Nachricht einer Gruppe verwendet.

Das folgende Beispiel sucht nach den logischen Nachrichten einer Gruppe, um sie anschließend abzurufen und zu entfernen:

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...

if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...

      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                  | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
...
```

Mehrmalige Zustellung bereits angezeigter Nachrichten verhindern
Durch bestimmte Öffnungs- und Nachrichtenabrufoptionen können Sie bereits angezeigte Nachrichten als abgerufen markieren, so dass sie durch die gleiche oder eine kooperierende Anwendung nicht erneut abgerufen werden. Um diese Nachrichten dem Browsing wieder zur Verfügung zu stellen, kann diese Markierung auch explizit oder automatisch wieder von den Nachrichten entfernt werden.

Mittels Browsing können Sie die Nachrichten einer Warteschlange in einer anderen Reihenfolge anzeigen als in der Reihenfolge, in der Sie sie abrufen und entfernen würden. Insbesondere können Sie die gleichen Nachrichten auch mehrmals abrufen und anzeigen, was nicht möglich ist, wenn sie abgerufen und entfernt werden. Um gerade dies aber zu verhindern, empfiehlt es sich, die Nachrichten beim Browsen zu *markieren*, so dass sie nicht erneut abgerufen werden. Dies wird auch als *Browsen mit Markierung* bezeichnet. Zum Markieren bereits angezeigter Nachrichten verwenden Sie MQGET mit der Option MQGMO_MARK_BROWSE_HANDLE, und zum Abrufen nur der noch nicht markierten Nachrichten verwenden Sie MQGET mit der Option MQGMO_UNMARKED_BROWSE_MSG. Wenn Sie gar die Kombination der Optionen MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG und MQGMO_MARK_BROWSE_HANDLE verwenden und MQGET mehrmals ausführen, erhalten Sie nacheinander jede einzelne Nachricht der Warteschlange. Dadurch verhindern Sie die mehrmalige Zustellung der gleichen Nachrichten, selbst wenn MQGMO_BROWSE_FIRST verwendet wird, um sicherzustellen, dass keine Nachricht übersprungen wird. Die Funktion dieser Optionskombination wird vollständig auch durch die Konstante MQGMO_BROWSE_HANDLE übernommen. Wenn die Warteschlange keine Nachrichten mehr enthält, die noch nicht angezeigt wurden, wird MQRC_NO_MSG_AVAILABLE zurückgegeben.

Wenn mehrere Anwendungen die gleiche Warteschlange durchsuchen, können diese die Warteschlange mit den Optionen MQOO_CO_OP und MQOO_BROWSE öffnen. Die durch jedes MQOPEN zurückgegebene Objektkennung wird als Teil einer zusammenwirkenden Gruppe betrachtet. Jede Nachricht, die durch einen MQGET-Aufruf mit der Option MQGMO_MARK_BROWSE_CO_OP zurückgegeben wird, gilt als für diesen kooperierenden Satz an Kennungen markiert.

Die Markierungen von Nachrichten, die schon längere Zeit markiert sind, können vom Warteschlangenmanager automatisch entfernt werden, so dass die Nachrichten wieder für das Browsing zur Verfügung

stehen. Dabei gibt das Warteschlangenmanagerattribut MsgMarkBrowseInterval die Zeit in Millisekunden an, die eine Nachricht für einen solchen kooperierenden Satz an Kennungen markiert bleibt. Ein MsgMarkBrowseInterval von -1 bedeutet, dass die Markierungen der Nachrichten nicht automatisch aufgehoben werden.

Sobald der einzelne Prozess bzw. der Satz kooperierender Prozesse, die Nachrichten markieren, endet, wird die Markierung der markierten Nachrichten wieder entfernt.

Beispiele für kooperatives Browsing

Zum Durchsuchen der Nachrichten in einer Warteschlange und zum Initiieren eines Konsumenten auf Basis des Nachrichteninhalts können Sie mehrere Kopien einer Zuteileranwendung ausführen. In diesem Fall öffnen Sie die Warteschlange in jeder Zuteileranwendung mit MQOO_CO_OP. Damit geben Sie an, dass die Zuteileranwendungen zusammenarbeiten und sich der von den anderen Zuteileranwendungen markierten Nachrichten bewusst sind. Jede Zuteileranwendung gibt nun mehrere MQGET-Aufrufe mit den Optionen MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG und MQGMO_MARK_BROWSE_CO_OP (bzw. funktional gleichwertig mit der einzelnen Konstante MQGMO_BROWSE_CO_OP) aus. Danach ruft jede Zuteileranwendung nur diejenigen Nachrichten ab, die von noch keiner kooperierenden Anwendung markiert wurden. Die Zuteileranwendung initialisiert einen Konsumenten und übergibt das vom MQGET zurückgegebene MsgToken an diesen Konsumenten, der die Nachricht dann aus der Warteschlange abruft und dort entfernt. Falls der Konsument den MQGET-Aufruf der Nachricht mit einem Backout zurücksetzt, steht die Nachricht für die Browser wieder zur Zuteilung zur Verfügung, weil sie nun nicht mehr markiert ist. Auch wenn der Konsument die Nachricht nicht mit einem MQGET abruft und aus der Warteschlange entfernt, wird die Markierung der Nachricht nach Verstreichen des MsgMarkBrowseInterval vom Warteschlangenmanager für den kooperierenden Satz an Kennungen aufgehoben, so dass die Nachricht neu zugeteilt werden kann.

Statt von mehreren Kopien der gleichen Zuteileranwendung kann die Warteschlange auch von verschiedenen Zuteileranwendungen durchsucht werden, wobei zum Beispiel jede für die Verarbeitung einer bestimmten Untergruppe der Nachrichten geeignet sein kann. In diesem Fall öffnen Sie die Warteschlange in jeder Zuteileranwendung mit MQOO_CO_OP. Damit geben Sie an, dass die Zuteileranwendungen zusammenarbeiten und sich der von den anderen Zuteileranwendungen markierten Nachrichten bewusst sind.

- Wenn die Reihenfolge der Nachrichtenverarbeitung für eine einzelne Zuteileranwendung wichtig ist, gibt jede Zuteileranwendung mehrere MQGET-Aufrufe mit den Optionen MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG und MQGMO_MARK_BROWSE_HANDLE (oder MQGMO_BROWSE_HANDLE) aus. Wenn die nächste gefundene Nachricht für die jeweilige Zuteileranwendung geeignet ist, gibt die Anwendung ein MQGET mit den Optionen MQMO_MATCH_MSG_TOKEN und MQGMO_MARK_BROWSE_CO_OP sowie dem vom vorherigen MQGET-Aufruf zurückgegebenen MsgToken aus. Ist der Aufruf erfolgreich, so initialisiert die Zuteileranwendung den Konsumenten und übergibt ihm das MsgToken.
- Wenn die Reihenfolge der Nachrichtenverarbeitung unwichtig ist und davon ausgegangen werden kann, dass die Zuteileranwendung die meisten Nachrichten, die sie vorfindet, verarbeiten kann, können Sie auch die Optionen MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG und MQGMO_MARK_BROWSE_CO_OP (oder MQGMO_BROWSE_CO_OP) verwenden. Findet die Zuteileranwendung eine Nachricht vor, die sie nicht verarbeiten kann, so hebt sie die Markierung der Nachricht durch ein MQGET mit den Optionen MQMO_MATCH_MSG_TOKEN und MQGMO_UNMARK_BROWSE_CO_OP sowie dem zuvor zurückgegebenen MsgToken auf.

Fälle, in denen der MQGET-Aufruf fehlschlägt

Wenn zwischen der Ausgabe eines MQOPEN- und eines MQGET-Aufrufs bestimmte Attribute einer Warteschlange mit der Option FORCE eines Befehls geändert werden, schlägt der MQGET-Aufruf fehl und gibt den Ursachencode MQRC_OBJECT_CHANGED zurück.

Der Warteschlangenmanager markiert die Objektkennung als ungültig. Dies passiert auch, wenn die Änderungen auf jede Warteschlange zutreffen, auf die sich der Warteschlangenname auflösen lässt. Die Attribute, die sich solchermaßen auf die Objektkennung auswirken, werden in der Beschreibung des MQOPEN-Aufrufs im Abschnitt MQOPEN aufgelistet. Falls Ihr Aufruf den Ursachencode MQRC_OB-

JECT_CHANGED zurückgibt, schließen Sie die Warteschlange, öffnen Sie sie erneut und wiederholen Sie dann den Abrufversuch.

Wenn für eine Warteschlange, aus der Sie versuchen, Nachrichten abzurufen, Get-Operationen unzulässig sind (das Gleiche gilt für Warteschlangen, auf die sich der Warteschlangenname auflöst), schlägt der MQGET-Aufruf mit dem Ursachencode MQRC_GET_INHIBITED fehl. Dies geschieht, auch wenn Sie den MQGET-Aufruf zum Durchsuchen (Browsing) verwenden. Möglicherweise lässt sich die Nachricht bei einer späteren Wiederholung des MQGET-Aufrufs erfolgreich abrufen, wenn die Anwendung es zulässt, dass andere Programme die Warteschlangenattribute regelmäßig ändern.

Wenn eine dynamische (temporäre oder permanente) Warteschlange gelöscht wurde, schlagen MQGET-Aufrufe, die eine zuvor erworbene Objektkennung verwenden, mit dem Ursachencode MQRC_Q_DELETED fehl.

Publish/Subscribe-Anwendungen schreiben

Beginnen Sie nun, Ihre eigenen IBM MQ-Publish/Subscribe-Anwendungen zu schreiben.

Eine Übersicht über Publish/Subscribe-Konzepte finden Sie unter Publish/Subscribe-Messaging.

In den folgenden Abschnitten finden Sie Informationen zum Schreiben der verschiedenen Typen von Publish/Subscribe-Anwendungen:

- „Veröffentlichungsanwendungen erstellen” auf Seite 852
- „Subskribentenanwendungen erstellen” auf Seite 859
- „Publish/Subscribe-Lebenszyklen” auf Seite 879
- „Publish/Subscribe-Nachrichteneigenschaften” auf Seite 884
- „Nachrichtenreihenfolge bestimmen” auf Seite 886
- „Veröffentlichungen abfangen” auf Seite 887
- „Veröffentlichungsoptionen” auf Seite 895
- „Subskriptionsoptionen” auf Seite 895

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

„Anwendungen für IBM MQ entwickeln” auf Seite 5
Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

„Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 52
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

„Prozedurale Anwendungen für die Warteschlangensteuerung erstellen” auf Seite 758
Dieser Abschnitt enthält Informationen zum Erstellen von Anwendungen für die Warteschlangensteuerung, zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager und zum Öffnen und Schließen von Objekten.

„Prozedurale Clientanwendungen schreiben” auf Seite 961
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„Prozedurale Anwendung erstellen” auf Seite 1052
Sie können eine IBM MQ-Anwendung in einer von mehreren prozeduralen Sprachen schreiben und die Anwendung auf mehreren unterschiedlichen Plattformen ausführen.

„Prozedurale Programmfehler handhaben” auf Seite 1090

In diesen Informationen werden Fehler erläutert, die den MQI-Aufrufen Ihrer Anwendungen beim Ausgeben eines Aufrufs oder bei der Übergabe einer Nachricht an den Zielort zugeordnet werden.

Zugehörige Tasks

„Prozedurale IBM MQ-Beispielprogramme verwenden" auf Seite 1111
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

Veröffentlichungsanwendungen erstellen

Sehen Sie sich zwei Beispiele an, bevor Sie Veröffentlichungsanwendungen erstellen. Das erste Beispiel wurde möglichst getreu nach einer Punkt-zu-Punkt-Anwendung modelliert, die Nachrichten in eine Warteschlange einreiht, das zweite Beispiel und geläufigere Muster für Veröffentlichungsanwendungen veranschaulicht, wie Themen dynamisch erstellt werden.

Beim Schreiben einer einfachen IBM MQ-Veröffentlichungsanwendung gehen Sie im Prinzip genauso vor wie bei der Entwicklung einer IBM MQ-Punkt-zu-Punkt-Anwendung, die Nachrichten in eine Warteschlange einreiht (Tabelle 121 auf Seite 852). Der einzige Unterschied besteht darin, dass die Nachrichten mittels MQPUT nicht in eine Warteschlange, sondern in ein Thema eingereiht werden.

Tabelle 121. Vergleich zwischen Punkt-zu-Punkt- und Publish/Subscribe-IBM MQ-Programmmuster

Schritt	**Punkt-zu-Punkt-MQ-Aufruf**	**Publish-MQ-Aufruf**
Verbindung zu einem Warteschlangenmanager herstellen	MQCONN	MQCONN
Warteschlange öffnen	MQOPEN	
Thema öffnen		MQOPEN
Nachricht(en) einreihen	MQPUT	MQPUT
Thema schließen		MQCLOSE
Warteschlange schließen	MQCLOSE	
Verbindung zu Warteschlangenmanager trennen	MQDISC	MQDISC

Dies soll nun an zwei Anwendungsbeispielen veranschaulicht werden, die Börsenkurse veröffentlichen. Im ersten Beispiel („Beispiel 1: Veröffentlichungsanwendung für ein festes Thema" auf Seite 853), das noch sehr dem Muster des Einreihens von Nachrichten in eine Warteschlange nachempfunden ist, erstellt der Administrator eine Themendefinition auf ähnliche Weise, wie er eine Warteschlange erstellt. Der Programmierer codiert MQPUT so, dass die Nachrichten statt in eine Warteschlange in das Thema geschrieben werden. Im zweiten Beispiel („Beispiel 2: Veröffentlichungsanwendung für ein variables Thema" auf Seite 856) sieht das Interaktionsmuster des Programms mit IBM MQ ähnlich aus. Der einzige Unterschied besteht darin, dass nicht der Administrator, sondern der Programmierer das Thema bereitstellt, in das die Nachrichten geschrieben werden. In der Praxis bedeutet das in der Regel, dass die Themenzeichenfolge inhaltsdefiniert ist bzw. von einer anderen Quelle bereitgestellt wird, beispielsweise durch eine Benutzereingabe in einem Browser.

Zugehörige Konzepte

„Subskribentenanwendungen erstellen" auf Seite 859
Sehen Sie sich, bevor Sie selbst Subskribentenanwendungen erstellen, die in dieser Dokumentation bereitgestellten Beispiele an: eine IBM MQ-Anwendung, die Nachrichten aus einer Warteschlange konsumiert, eine Anwendung, die eine Subskription erstellt und keine Kenntnisse über die Warteschlangen voraussetzt, und schließlich eine Anwendung, die sowohl Warteschlangen als auch Subskriptionen verwendet.

Zugehörige Verweise

TOPIC DEFINI

ANZEIGETHEMA
ANZEIGETPSTATUS

Beispiel 1: Veröffentlichungsanwendung für ein festes Thema
Ein IBM MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Administrator definiertes Thema.

Anmerkung: Der kompakte Codierungsstil wurde aus Gründen der besseren Lesbarkeit verwendet, ist aber nicht für die Übernahme in die Produktion geeignet.

Die Ausgabe ist in Abbildung 65 auf Seite 854 abgebildet.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &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.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        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);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
               publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Abbildung 64. Einfaches IBM MQ-Veröffentlichungsprogramm für ein festes Thema

---

```
X:\Publish1\Debug>PublishStock
Optional parameters:  TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters:  TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Abbildung 65. Ausgabe des ersten Beispiels für eine Veröffentlichungsanwendung

Die nachfolgend erläuterten Codezeilen veranschaulichen einzelne Aspekte der Entwicklung einer Veröffentlichungsanwendung für IBM MQ.

char topicNameDefault[] = "IBMSTOCKPRICE";
: Im Programm wird ein Standardthemenname definiert. Diesen Namen können Sie überschreiben, indem Sie als erstes Argument im Programm den Namen eines anderen Themenobjekts angeben.

MQCHAR resTopicStr[151];
: `resTopicStr` wird von `td.ResObjectString.VSPtr` referenziert und von `MQOPEN` zur Rückgabe der aufgelösten Themenzeichenfolge verwendet. Für den Nullabschluss sollte die Länge von `resTopicStr` um ein Zeichen länger sein als die in `td.ResObjectString.VSBufSize` übergebene Länge.

memset (resTopicStr, 0, sizeof(resTopicStr));
: Initialisieren Sie `resTopicStr` mit Null, um sicherzustellen, dass die in `MQCHARV` zurückgegebene, aufgelöste Themenzeichenfolge mit Null abgeschlossen wird.

td.ObjectType = MQOT_TOPIC
: Für Publish/Subscribe gibt es einen neuen Objekttyp: das *Themenobjekt*.

td.Version = MQOD_VERSION_4;
: Zur Verwendung des neuen Objekttyps müssen Sie mindestens *Version 4* des Objektdeskriptors verwenden.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
: `topicName` ist der Name eines Themenobjekts, manchmal auch als administratives Themenobjekt oder Verwaltungsthemenobjekt bezeichnet. In diesem Beispiel muss das Themenobjekt bereits in IBM MQ Explorer oder mit dem folgenden MQSC-Befehl erstellt worden sein:

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
: Die aufgelöste Themenzeichenfolge wird im letzten `printf` des Programms zurückgemeldet. Die `MQCHARV ResObjectString`-Struktur muss so eingerichtet werden, dass IBM MQ die aufgelöste Zeichenfolge zurück an das Programm meldet.

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
: Öffnet das Thema für die Ausgabe (genauso wie eine Warteschlange für die Ausgabe geöffnet wird).

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
: Wenn Sie möchten, dass neue Subskribenten die Veröffentlichung empfangen können, geben Sie in der Veröffentlichungsanwendung `MQPMO_RETAIN` ein. Wenn danach ein Subskribent gestartet wird, erhält er die neueste Veröffentlichung, die bereits vor dem Start des Subskribenten veröffentlicht wurde, als erste übereinstimmende Veröffentlichung. Alternativ können Sie Subskribenten nur die Veröffentlichungen bereitstellen, die erst nach dem Start des jeweiligen Subskribenten veröffentlicht wurden. Darüber hinaus kann der Subskribent durch Angabe von `MQSO_NEW_PUBLICATIONS_ONLY` in seiner Subskription den Empfang früherer (ständiger) Veröffentlichungen ablehnen.

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
: Erhöhen Sie die Länge der an `MQPUT` übergebenen Zeichenfolge um 1, damit das Null-Abschlusszeichen als Teil des Nachrichtenpuffers an IBM MQ übergeben wird.

Was zeigt dieses erste Beispiel? Das Beispiel ist so nahe wie möglich dem herkömmlichen und bewährten Entwicklungsmuster für IBM MQ-Punkt-zu-Punkt-Programme nachempfunden. Ein wichtiger Aspekt bei der Programmierung nach dem IBM MQ-Programmierungsmuster ist, dass sich der Programmierer nicht darum kümmern muss, wohin die Nachrichten gesendet werden. Die Aufgabe des Programmierers besteht darin, eine Verbindung mit einem Warteschlangenmanager herzustellen und dafür zu sorgen, dass ihm die Nachrichten übergeben werden, die an die Empfänger verteilt werden sollen. Beim Punkt-zu-Punkt-Konzept öffnet der Programmierer eine Warteschlange (möglicherweise auch eine Aliaswarteschlange), die vom Administrator konfiguriert wurde. Die Aliaswarteschlange leitet die Nachrichten an eine Zielwarteschlange des lokalen oder eines fernen Warteschlangenmanagers weiter. Solange die Nachrichten auf ihre Zustellung warten, werden sie irgendwo zwischen der Quelle und dem Ziel in Warteschlangen gespeichert.

Beim Publish/Subscribe-Muster öffnet der Programmierer statt einer Warteschlange ein Thema. In unserem Beispiel ist das Thema einer vom Administrator festgelegten Themenzeichenfolge zugeordnet. Der Warteschlangenmanager leitet eine Veröffentlichung über Warteschlangen an lokale oder ferne Subskribenten weiter, deren Subskriptionen mit der Themenzeichenfolge der Veröffentlichung übereinstimmen. Im Falle einer ständigen Veröffentlichung bewahrt der Warteschlangenmanager die aktuellste Kopie der Veröffentlichung auf, selbst wenn es für sie zur Zeit keine Subskribenten gibt. Die ständige Veröffentlichung steht dadurch für die sofortige Weiterleitung an künftige Subskribenten zur Verfügung. Die Veröffentlichungsanwendung spielt bei der Auswahl oder der Weiterleitung der Veröffentlichung an ihr Ziel keine Rolle. Ihre Aufgabe ist es, die Veröffentlichungen zu erstellen und den Themen zuzuordnen, die vom Administrator definiert wurden.

Dieses Beispiel für ein festes Thema ist für Publish/Subscribe-Anwendungen eher ungewöhnlich: es ist statisch. Es setzt einen Administrator voraus, der die Themenzeichenfolgen definiert und die Themen für die Veröffentlichung ändert. In der Regel müssen Publish/Subscribe-Anwendungen die Themenstruktur ganz oder zumindest teilweise kennen. Vielleicht ändern sich die Themen zu häufig oder es gibt so viele Themenkombinationen, dass es für einen Administrator sehr mühsam wäre, für jede Themenzeichenfolge, die für Veröffentlichungen benötigt wird, einen Themenknoten zu definieren. Vielleicht stehen die Themenzeichenfolgen auch vor der Veröffentlichung noch nicht fest. Eine Veröffentlichungsanwendung kann zur Definition einer Themenzeichenfolge zum Beispiel Informationen aus dem Veröffentlichungsinhalt übernehmen oder es stehen ihr Informationen zu den für die Veröffentlichung benötigten Themenzeichenfolgen aus anderen Quellen zur Verfügung, beispielsweise aus den Eingaben eines Benutzers in einem Browser. Im nächsten Beispiel wird die dynamische Erstellung von Themen in der Veröffentlichungsanwendung veranschaulicht, damit Sie auch dynamischere Veröffentlichungsmethoden entwickeln können.

Themen verbinden Veröffentlichungsanwendungen und Subskribenten. Die Festlegung der Regeln bzw. der Architektur für die Benennung der Themen und die Organisation der Themen in Themenstrukturen ist ein wichtiger Schritt bei der Entwicklung einer Publish/Subscribe-Lösung. Überprüfen Sie sorgfältig, in welchem Ausmaß die Veröffentlichungs- und Subskribentenprogramme durch die Organisation des Themenbaums miteinander und mit dem Inhalt des Themenbaums verknüpft sind. Untersuchen Sie, ob sich Änderungen am Themenbaum auf die Veröffentlichungs- und Subskribentenanwendungen auswirken und wie Sie diese Auswirkung minimieren können. In die Architektur des IBM MQ-Publish/Subscribe-Modells ist das Konzept eines administrativen Themenobjekts integriert, das die Stammkomponente bzw. die Stammunterstruktur eines Themas bereitstellt. Über dieses Themenobjekt können Sie die Stammkomponente der Themenstruktur administrativ definieren. Dadurch erleichtern Sie sich die Anwendungsprogrammierung sowie den Betrieb der Anwendung und folglich auch die Anwendungswartung. Wenn Sie beispielsweise mehrere Publish/Subscribe-Anwendungen implementieren, deren Themenstrukturen voneinander isoliert sind, können Sie durch die administrative Definition der Stammkomponente der Themenstruktur die Isolation der Themenstrukturen weiterhin garantieren, selbst wenn die von den einzelnen Anwendungen verwendeten Konventionen zur Themenbenennung nicht konsistent sind.

In der Praxis decken Veröffentlichungsanwendungen ein Spektrum von der alleinigen Verwendung fester Themen (wie in diesem Beispiel) bis hin zur Verwendung variabler Themen (wie im nächsten Beispiel) ab. „Beispiel 2: Veröffentlichungsanwendung für ein variables Thema” auf Seite 856 veranschaulicht zudem die gemeinsame Verwendung von Themen und Themenzeichenfolgen.

Zugehörige Konzepte

„Beispiel 2: Veröffentlichungsanwendung für ein variables Thema” auf Seite 856

Ein WebSphere MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Programm definiertes Thema.

„Subskribentenanwendungen erstellen” auf Seite 859
Sehen Sie sich, bevor Sie selbst Subskribentenanwendungen erstellen, die in dieser Dokumentation bereitgestellten Beispiele an: eine IBM MQ-Anwendung, die Nachrichten aus einer Warteschlange konsumiert, eine Anwendung, die eine Subskription erstellt und keine Kenntnisse über die Warteschlangen voraussetzt, und schließlich eine Anwendung, die sowohl Warteschlangen als auch Subskriptionen verwendet.

Beispiel 2: Veröffentlichungsanwendung für ein variables Thema
Ein WebSphere MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Programm definiertes Thema.

Anmerkung: Der kompakte Codierungsstil wurde aus Gründen der besseren Lesbarkeit verwendet, ist aber nicht für die Übernahme in die Produktion geeignet.

Die Ausgabe ist in Abbildung 67 auf Seite 857 abgebildet.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName, topicSt
ring);
    do {
        MQCONN(qmName, &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);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        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);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Abbildung 66. Einfaches IBM MQ-Veröffentlichungsprogramm für ein variables Thema

```
X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Abbildung 67. Ausgabe des zweiten Beispiels für eine Veröffentlichungsanwendung

Auch zu diesem Beispiel gibt es einige Anmerkungen:

char topicNameDefault[] = "STOCKS";
Der Standardthemenname STOCKS legt einen Teil der Themenzeichenfolge fest. Sie können diesen Themennamen überschreiben, indem Sie ihn dem Programm als erstes Argument bereitstellen; Sie können aber auch die Verwendung des Themennamens ganz inaktivieren, indem Sie als ersten Parameter / eingeben.

char topicString[101] = "IBM/PRICE";
IBM/PRICE ist die Standardthemenzeichenfolge. Sie können diese Themenzeichenfolge überschreiben, indem Sie sie dem Programm als zweites Argument bereitstellen.

Der Warteschlangenmanager kombiniert die vom Themenobjekt STOCKS bereitgestellte Themenzeichenfolge "NYSE" mit der vom Programm bereitgestellten Themenzeichenfolge "IBM/PRICE" und fügt dazwischen ein "/" ein. Die aufgelöste Themenzeichenfolge lautet folglich "NYSE/IBM/PRICE". Die sich daraus ergebende Themenzeichenfolge ist identisch mit der im Themenobjekt IBMSTOCKPRICE definierten Themenzeichenfolge und hat auch die gleiche Auswirkung.

Das mit der aufgelösten Themenzeichenfolge verbundene administrative Themenobjekt ist nicht unbedingt identisch mit dem Themenobjekt, das von der Veröffentlichungsanwendung an MQOPEN übergeben wird. IBM MQ entnimmt der Themenstruktur in der aufgelösten Themenzeichenfolge implizit, welches administrative Themenobjekt die mit der Veröffentlichung verbundenen Attribute definiert.

Angenommen, es gibt zwei Themenobjekte A und Bund A definiert Thema "a"und B definiert Thema "a/b" (Abbildung 68 auf Seite 858). Wenn das Veröffentlichungsprogramm auf das Themenobjekt A verweist und die Themenzeichenfolge "b"bereitstellt und das Thema in die Themenzeichenfolge "a/b"auflöst, übernimmt die Veröffentlichung die Eigenschaften des Themenobjekts B , da das Thema der für Bdefinierten Themenzeichenfolge "a/b" entspricht.

if (strcmp(argv[1],"/"))
argv[1] ist der optional bereitgestellte Themenname. "/" ist als Themenname ungültig. Hier gibt diese Zeichenfolge an, dass kein Themenname vorliegt und die Themenzeichenfolge vollständig vom Programm bereitgestellt wird. Der Ausgabe in Abbildung 67 auf Seite 857 können Sie entnehmen, dass die gesamte Themenzeichenfolge dynamisch vom Programm bereitgestellt wird.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
Im Standardfall muss der optionale Themenname (topicName) bereits in IBM MQ Explorer oder mit dem folgenden MQSC-Befehl erstellt worden sein:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
Die Themenzeichenfolge ist ein MQCHARV-Feld des Themendeskriptors.

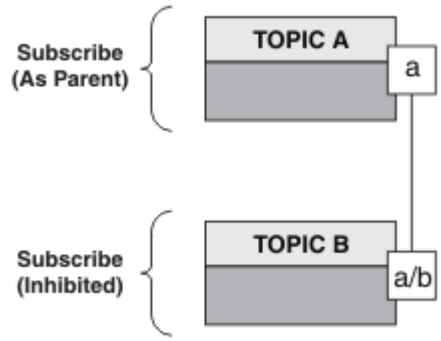


Abbildung 68. Zuordnung der Themenobjekte

Was zeigt das zweite Beispiel? Auch wenn der Code nahezu identisch mit dem Code des ersten Beispiels ist - nur zwei Zeilen unterscheiden sich -, ergibt sich daraus ein völlig neues Programm. Der Programmierer bestimmt die Ziele, an die die Veröffentlichungen gesendet werden. Der Administrator hat einen mini-

malen Eingabeaufwand, um Subskribentenanwendungen zu entwerfen. Außerdem müssen keine Themen oder Warteschlangen vordefiniert werden, um Veröffentlichungen von Veröffentlichungskomponenten an Subskribenten weiterzuleiten.

Beim Punkt-zu-Punkt-Messaging-Konzept müssen die Warteschlangen bereits definiert sein, um Nachrichten weiterleiten zu können. Beim Publish/Subscribe-Konzept ist dies nicht erforderlich, auch wenn IBM MQ Publish/Subscribe mit seinem vorhandenen Warteschlangensystem implementiert. Dadurch gelten die Vorteile der zuverlässigen Zustellung, der Transaktionalität und der losen Kopplung, die mit dem Messaging- und Warteschlangenkonzept verbunden sind, auch für Publish/Subscribe-Anwendungen.

Der Entwickler muss entscheiden, ob den Veröffentlichungs- und Subskribentenprogrammen die zugrunde liegende Themenstruktur bekannt sein soll. Ebenso muss er entscheiden, ob den Subskribentenprogrammen das Warteschlangensystem bekannt sein soll. Sehen Sie sich als nächstes die Beispiele für Subskribentenanwendungen an. Die Beispiele wurden passend zu den Beispielen für die Veröffentlichungsanwendungen entwickelt - in der Regel wird für die Veröffentlichung und Subskription `NYSE/IBM/PRICE` verwendet.

Zugehörige Konzepte

„Beispiel 1: Veröffentlichungsanwendung für ein festes Thema” auf Seite 853
Ein IBM MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Administrator definiertes Thema.

„Subskribentenanwendungen erstellen” auf Seite 859
Sehen Sie sich, bevor Sie selbst Subskribentenanwendungen erstellen, die in dieser Dokumentation bereitgestellten Beispiele an: eine IBM MQ-Anwendung, die Nachrichten aus einer Warteschlange konsumiert, eine Anwendung, die eine Subskription erstellt und keine Kenntnisse über die Warteschlangen voraussetzt, und schließlich eine Anwendung, die sowohl Warteschlangen als auch Subskriptionen verwendet.

Subskribentenanwendungen erstellen

Sehen Sie sich, bevor Sie selbst Subskribentenanwendungen erstellen, die in dieser Dokumentation bereitgestellten Beispiele an: eine IBM MQ-Anwendung, die Nachrichten aus einer Warteschlange konsumiert, eine Anwendung, die eine Subskription erstellt und keine Kenntnisse über die Warteschlangen voraussetzt, und schließlich eine Anwendung, die sowohl Warteschlangen als auch Subskriptionen verwendet.

Diese drei Konsumenten- bzw. Subskribentenarten sind in Tabelle 122 auf Seite 860 mit den für sie typischen IBM MQ-Funktionsaufrufsequenzen aufgeführt.

1. Die erste Subskribentenart, MQ-Veröffentlichungskonsument, ist identisch mit einem Punkt-zu-Punkt-MQ-Programm, das nur MQGET durchführt. Die Anwendung hat keinerlei Kenntnisse davon, dass sie Veröffentlichungen konsumiert - sie liest lediglich Nachrichten aus einer Warteschlange. Die Subskription, durch die Veröffentlichungen an die Warteschlange weitergeleitet werden, wird administrativ in IBM MQ Explorer oder mit einem Befehl erstellt.
2. Die zweite Subskribentenart ist das für Subskribentenanwendungen am häufigsten verwendete Muster. Die Subskribentenanwendung erstellt die Subskription und erhält daraufhin Veröffentlichungen. Die Warteschlangenverwaltung erfolgt über den Warteschlangenmanager. Dies wird als *verwalteter Subskribent* bezeichnet.
3. Bei der dritten Subskribentenart ist die Subskribentenanwendung für das Angeben der Warteschlange zuständig, in der Veröffentlichungen gespeichert werden. Sie entscheidet, wann die Warteschlange geöffnet und geschlossen wird. Außerdem gibt die Subskribentenanwendung die Subskriptionen aus, durch die die Warteschlange mit Veröffentlichungen gefüllt wird. Dies wird als *nicht verwalteter Subskribent* bezeichnet.

Zum besseren Verständnis dieser Subskribentenarten empfiehlt es sich, die in Tabelle 122 auf Seite 860 aufgeführten C-Beispielprogramme zu untersuchen. Die Beispiele wurden passend zu den im Abschnitt „Veröffentlichungsanwendungen erstellen” auf Seite 852 als Beispiele vorgestellten Veröffentlichungsanwendungen entwickelt.

Tabelle 122. Vergleich zwischen Punkt-zu-Punkt- und Subscribe-IBM MQ-Programmmustern				
Schritt	**MQ-Nachrichtenkonsument**	**„Beispiel 1: Konsument einer MQ-Veröffentlichung" auf Seite 860**	**„Beispiel 2: Verwalteter MQ-Subskribent" auf Seite 863**	**„Beispiel 3: Nicht verwalteter MQ-Subskribent" auf Seite 869**
Verbindung zu einem Warteschlangenmanager herstellen	MQCONN	MQCONN	MQCONN	MQCONN
Warteschlange öffnen	MQOPEN	MQOPEN		MQOPEN
Abonnieren			MQSUB	MQSUB
Nachricht(en) abrufen	MQGET	MQGET	MQGET	MQGET
Warteschlange schließen	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
Subskription schließen			MQCLOSE	MQCLOSE
Verbindung zu Warteschlangenmanager trennen	MQDISC	MQDISC	MQDISC	MQDISC

Die Verwendung von MQCLOSE ist optional und kann zur Freigabe von Ressourcen, zur Übergabe von MQCLOSE-Optionen oder lediglich aus Gründen der Symmetrie zu MQOPEN erfolgen. Da es eher unwahrscheinlich ist, dass Sie im Falle des verwalteten MQ-Subskribenten beim Schließen der Subskriptionswarteschlange die MQCLOSE-Optionen angeben müssen und das Argument der Symmetrie irrelevant ist, wird die Subskriptionswarteschlange in Beispiel 2: Verwalteter MQ-Subskribent nicht explizit geschlossen.

Zum Verständnis der Publish/Subscribe-Anwendungsmuster können Sie sich auch die Interaktionen zwischen den beteiligten Entitäten ansehen. Hierfür eignen sich besonders Lebenslinien- bzw. UML-Ablaufdiagramme. Drei Beispiele für Lebenslinien werden im Abschnitt „Publish/Subscribe-Lebenszyklen" auf Seite 879 beschrieben.

Beispiel 1: Konsument einer MQ-Veröffentlichung
Der MQ-Veröffentlichungskonsument ist ein IBM MQ-Nachrichtenkonsument, der nicht selbst Themen subskribiert.

Führen Sie zur Erstellung der in diesem Beispiel verwendeten Subskription und Veröffentlichungswarteschlange die folgenden Befehle aus oder definieren Sie die Objekte in IBM MQ Explorer.

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

Die Subskription `IBMSTOCKPRICESUB` referenziert das für das Veröffentlichungsbeispiel erstellte Themenobjekt `IBMSTOCK` sowie die lokale Warteschlange `STOCKTICKER`. Das Themenobjekt `IBMSTOCK` definiert die in der Subskription verwendete Themenzeichenfolge `NYSE/IBM/PRICE`. Beachten Sie, dass das Themenobjekt und die Warteschlange für den Empfang der Veröffentlichungen vor der Erstellung der Subskription definiert werden müssen.

Das Muster des MQ-Veröffentlichungskonsumenten weist eine Reihe nützlicher Facetten auf:

1. Multiprocessing: Verteilung des Arbeitsaufwands für das Lesen der Veröffentlichungen. Die Veröffentlichungen werden sämtlichst in die Warteschlange eingereiht, die dem Subskriptionsthema zugeordnet

ist. Wenn MQOO_INPUT_SHARED verwendet wird, kann die Warteschlange von mehreren Konsumenten geöffnet werden.

2. Zentral verwaltete Subskriptionen: Die Anwendungen erstellen keine eigenen Subskriptionsthemen oder Subskriptionen. Vielmehr legt der Administrator fest, wohin die Veröffentlichungen gesendet werden.
3. Subskriptionskonzentration: Mehrere verschiedene Subskriptionen können an eine einzige Warteschlange gesendet werden.
4. Permanenz der Subskriptionen: Die Warteschlange empfängt alle Veröffentlichungen, unabhängig davon, ob die Konsumenten aktiv sind.
5. Migration und Koexistenz: Der Konsumentencode funktioniert in einem Punkt-zu-Punkt- und in einem Publish/Subscribe-Szenario gleichermaßen gut.

Die Subskription erstellt eine Beziehung zwischen der Themenzeichenfolge NYSE/IBM/PRICE und der Warteschlange STOCKTICKER. Sobald die Subskription erstellt ist, werden Veröffentlichungen, einschließlich der aktuellen ständigen Veröffentlichung, an STOCKTICKER weitergeleitet.

Eine administrativ erstellte Subskription kann verwaltet oder nicht verwaltet sein. Eine verwaltete Subskription wird ebenso wie eine nicht verwaltete Subskription sofort nach ihrer Erstellung wirksam. Verwalteten Subskriptionen stehen jedoch nicht alle Facetten des Musters zur Verfügung. Informationen hierzu finden Sie unter „Beispiel 3: Nicht verwalteter MQ-Subskribent” auf Seite 869.

Anmerkung: Der kompakte Codierungsstil wurde aus Gründen der besseren Lesbarkeit verwendet, ist aber nicht für die Übernahme in die Produktion geeignet.

Die Ergebnisse werden in Abbildung 70 auf Seite 862 gezeigt.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){           /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000, subscription
Queue);
do {
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
md.Encoding    = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
memset(publication, 0, sizeof(publicationBuffer));
MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
&CompCode, &Reason);
if (Reason == MQRC_NONE)
printf("Received publication \"%s\"\n", publication);
}
while (CompCode == MQCC_OK);
if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Abbildung 69. MQ-Veröffentlichungskonsument

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

Abbildung 70. Ausgabe des MQ-Veröffentlichungskonsumenten

Hinsichtlich IBM MQ sollten Sie bei der Programmierung in der Sprache *C* die folgenden Standardtipps beachten:

memset(publication, 0, sizeof(publicationBuffer));
Achten Sie darauf, dass die Nachricht eine abschließende Null aufweist; dadurch erleichtern Sie sich die Formatierung mittels `printf`. Das Beispiel für die Veröffentlichungsanwendung enthält die abschließende Null in dem an MQPUT übergebenen Nachrichtenpuffer, da `strlen(publication)` um 1 erhöht wurde. Die Einstellung von MQCHAR-Puffern auf Null ist bei IBM MQ-C-Programmen, die die Puffer zum Speichern von Zeichenfolgen verwenden, eine empfehlenswerte Programmierpraxis. Dadurch wird sichergestellt, dass auf ein Zeichenarray, dass den Puffer nicht vollständig füllt, eine Null folgt.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
Reservieren Sie eine Null am Ende des Nachrichtenpuffers, um sicherzustellen, dass die Antwortnachricht in dem Fall, dass `if (messlen == strlen(publication));` gleich 'true' ist, eine abschließende Null aufweist. Dieser Tipp ist eine sinnvolle Ergänzung zum vorangegangenen Tipp. Er stellt sicher, dass `publicationBuffer` mindestens eine Null enthält, die nicht durch den Inhalt von `publication` überschrieben wird.

Zugehörige Konzepte

„Beispiel 2: Verwalteter MQ-Subskribent" auf Seite 863
Der verwaltete MQ-Subskribent ist das am häufigsten verwendete Subskribentenmuster. Bei einer verwalteten Subskription wird die Subskription von IBM MQ abgewickelt, ebenso werden Sie automatisch registriert und erneut registriert. Dieses Beispiel setzt *keine* administrative Definition der Warteschlangen, Themen oder Subskriptionen voraus.

„Beispiel 3: Nicht verwalteter MQ-Subskribent" auf Seite 869
Der nicht verwaltete Subskribent ist eine wichtige Subskribentenanwendungsklasse. In ihm vereinen sich die Vorteile von Publish/Subscribe und die *Steuerungsmöglichkeiten*, die Warteschlangen und die Verarbeitung von Veröffentlichungen bieten. Bei einer nicht verwalteten Subskription ist die Anwendung verantwortlich. zur Angabe der Warteschlange, in der die Subskriptionen gespeichert sind. Dieses Beispiel veranschaulicht verschiedene Arten der Kombination von Subskriptionen und Warteschlangen.

„Veröffentlichungsanwendungen erstellen" auf Seite 852
Sehen Sie sich zwei Beispiele an, bevor Sie Veröffentlichungsanwendungen erstellen. Das erste Beispiel wurde möglichst getreu nach einer Punkt-zu-Punkt-Anwendung modelliert, die Nachrichten in eine Warteschlange einreiht, das zweite Beispiel und geläufigere Muster für Veröffentlichungsanwendungen veranschaulicht, wie Themen dynamisch erstellt werden.

Beispiel 2: Verwalteter MQ-Subskribent
Der verwaltete MQ-Subskribent ist das am häufigsten verwendete Subskribentenmuster. Bei einer verwalteten Subskription wird die Subskription von IBM MQ abgewickelt, ebenso werden Sie automatisch registriert und erneut registriert. Dieses Beispiel setzt *keine* administrative Definition der Warteschlangen, Themen oder Subskriptionen voraus.

Die einfachste Art des verwalteten Subskribenten verwendet in der Regel eine *nicht permanente* Subskription. In diesem Beispiel wird daher auch eine nicht permanente Subskription behandelt. Die Subskription besteht nur für die Dauer der Laufzeit der Subskriptionskennung von MQSUB. Alle Veröffentlichungen, die während der Lebensdauer der Subskription mit der Themenzeichenfolge übereinstimmen, werden an die Subskriptionswarteschlange gesendet (möglicherweise gilt dies auch für eine ständige Veröffentlichung, sofern das Flag MQSO_NEW_PUBLICATIONS_ONLY nicht gesetzt bzw. seine Standardeinstellung übernommen wurde, sofern eine frühere, mit der Themenzeichenfolge übereinstimmende Veröffentlichung aufbewahrt wurde und sofern es sich um eine permanente Veröffentlichung handelt oder der Warteschlangenmanager seit der Erstellung der Veröffentlichung nicht beendet wurde).

Dieses Muster eignet sich auch für *permanente* Subskriptionen. Eine verwaltete permanente Subskription wird in der Regel aus Gründen der Zuverlässigkeit verwendet, nicht um eine Subskription einzurichten, die, sofern keine Fehler auftreten, den Subskribenten überlebt. Weitere Informationen zu den unterschiedlichen Lebenszyklen von verwalteten, nicht verwalteten, permanenten und nicht permanenten Subskriptionen finden Sie in den entsprechenden Abschnitten.

Permanente Subskriptionen sind meist permanenten Veröffentlichungen und nicht permanente Subskriptionen sind meist nicht permanenten Veröffentlichungen zugeordnet, es besteht jedoch keine zwingende

Beziehung zwischen der Lebensdauer einer Subskription und der Persistenz einer Veröffentlichung. Es sind alle vier Kombinationen aus Lebensdauer und Persistenz möglich.

Im Falle der besprochenen verwalteten, nicht permanenten Subskription erstellt der Warteschlangenmanager eine Subskriptionswarteschlange, die beim Schließen der Warteschlange geleert und gelöscht wird. Die Veröffentlichungen werden aus der Warteschlange entfernt, sobald die nicht permanente Subskription geschlossen wird.

Nachfolgend finden Sie die wichtigsten Facetten des verwalteten, nicht permanenten Musters, das durch den Code auf dieser Seite dargestellt wird:

1. On-demand-Subskription: Die Themenzeichenfolge der Subskription ist dynamisch. Sie wird von der Anwendung bereitgestellt, so lange sie aktiv ist.
2. Sich selbst verwaltende Warteschlange: Die Subskriptionswarteschlange definiert und verwaltet sich selbst.
3. Sich selbst verwaltender Subskriptionslebenszyklus: *Nicht permanente* Subskriptionen bestehen nur für die Dauer der Subskribentenanwendung.
   - Für eine *permanente* verwaltete Subskription wird eine permanente Subskriptionswarteschlange erstellt und die Veröffentlichungen verbleiben in der Warteschlange, selbst wenn keine Subskribentenprogramme aktiv sind. Der Warteschlangenmanager löscht die Warteschlange nur (und entfernt daraus alle noch nicht abgerufenen Veröffentlichungen), wenn das Löschen der Subskription durch die Anwendung oder den Administrator angefordert wurde. Die Subskription kann mittels eines administrativen Befehls oder durch Schließen der Subskription mit der Option `MQCO_REMOVE_SUB` gelöscht werden.
   - Eventuell empfiehlt es sich bei permanenten Subskriptionen, `SubExpiry` so einzustellen, dass keine weiteren Veröffentlichungen mehr an die Warteschlange gesendet werden und der Subskribent die verbleibenden Veröffentlichungen konsumieren kann, bevor die Subskription entfernt wird und die Warteschlange sowie die darin verbliebenen Veröffentlichungen durch den Warteschlangenmanager gelöscht werden.
4. Flexible Bereitstellung der Themenzeichenfolge: Die Verwaltung der Subskriptionsthemen vereinfacht sich durch Definition der Stammkomponente der Subskription mittels eines administrativ definierten Themas. Dadurch bleibt die Stammkomponente der Themenstruktur in der Anwendung verborgen. Durch Verbergen der Stammkomponente kann eine Anwendung implementiert werden, ohne dass die Anwendung versehentlich eine Themenstruktur erstellt, die sich mit einer von einer anderen Instanz oder Anwendung erstellten Themenstruktur überschneidet.
5. Verwaltete Themen: Wenn Sie eine Themenzeichenfolge verwenden, deren erster Teil mit einem administrativ definierten Themenobjekt übereinstimmt, werden die Veröffentlichungen entsprechend der Attribute des Themenobjekts verwaltet.
   - Wenn beispielsweise der erste Teil der Themenzeichenfolge mit einer Themenzeichenfolge eines zu einem Cluster gehörigen Themenobjekts übereinstimmt, kann die Subskription auch die Veröffentlichungen der anderen Mitglieder des Clusters empfangen.
   - Durch den selektiven Abgleich administrativ definierter Themenobjekte mit den vom Programm definierten Subskriptionen können Sie die Vorteile beider Konzepte vereinen. Der Administrator stellt die Attribute für Themen bereit, und der Programmierer definiert die untergeordneten Themen dynamisch, ohne sich Gedanken über die Verwaltung der Themen machen zu müssen.
   - Zur Ermittlung des Themenobjekts, das die einem Thema zugeordneten Attribute bereitstellt, wird die daraus resultierende Themenzeichenfolge verwendet und nicht notwendigerweise das in `sd.Objectname` angegebene Themenobjekt, obwohl beide in der Regel übereinstimmen. Siehe „Beispiel 2: Veröffentlichungsanwendung für ein variables Thema” auf Seite 856.

Dadurch, dass die Subskription im nachfolgenden Beispiel permanent ist, werden an die Subskriptionswarteschlange auch dann noch Veröffentlichungen gesendet, wenn der Subskribent die Subskription mit der Option `MQCO_KEEP_SUB` geschlossen hat. Die Warteschlange erhält weiterhin Veröffentlichungen, selbst wenn der Subskribent nicht aktiv ist. Dieses Verhalten können Sie außer Kraft setzen, indem Sie die Subskription mit der Option `MQSO_PUBLICATIONS_ON_REQUEST` erstellen und `MQSUBRQ` zum Anfordern der ständigen Veröffentlichung verwenden.

Durch Öffnen der Subskription mit der Option MQCO_RESUME kann die Subskription später wieder aufgenommen werden.

Die von MQSUB zurückgegebene Warteschlangenkennung Hobj kann auf verschiedene Weisen verwendet werden. Im Beispiel wird die Warteschlangenkennung zum Abfragen des Namens der Subskriptionswarteschlange verwendet. Verwaltete Warteschlangen werden mit der Standard-Modellwarteschlange SYSTEM.NDURABLE.MODEL.QUEUE oder SYSTEM.DURABLE.MODEL.QUEUE geöffnet. Diese Standardeinstellung können Sie auf Themenbasis überschreiben, indem Sie eigene permanente oder nicht permanente Modellwarteschlangen als Eigenschaften des zur Subskription gehörigen Themenobjekts bereitstellen.

Unabhängig von den aus den Modellwarteschlangen übernommenen Attributen können Sie die Kennung einer verwalteten Warteschlange nicht zur Erstellung einer weiteren Subskription verwenden. Ebenso wenig können Sie eine weitere Kennung für die verwaltete Warteschlange abrufen, indem Sie die verwaltete Warteschlange ein zweites Mal mit dem zurückgegebenen Warteschlangennamen öffnen. Die Warteschlange würde sich in diesem Fall verhalten, als ob sie für die ausschließliche Eingabe geöffnet worden wäre.

Nicht verwaltete Warteschlangen sind flexibler als verwaltete Warteschlangen. Nicht verwaltete Warteschlangen können zum Beispiel gemeinsam genutzt werden, d. h., es können mehrere Subskriptionen für dieselbe Warteschlange definiert werden. Im nächsten Beispiel wird gezeigt, wie Subskriptionen mit einer nicht verwalteten Subskriptionswarteschlange kombiniert werden.

Anmerkung: Der kompakte Codierungsstil wurde aus Gründen der besseren Lesbarkeit verwendet, ist aber nicht für die Übernahme in die Produktion geeignet.

Die Ergebnisse werden in Abbildung 73 auf Seite 867 gezeigt.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault     = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";                 /* Use default queue manager     */
    MQCHAR48 qName = "";                  /* Allocate to query queue name  */
    char     publicationBuffer[101];      /* Allocate to receive messages  */
    char     resTopicStrBuffer[151];   /* Allocate to resolve topic string */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle             */
    MQHOBJ   Hobj = MQHO_NONE;            /* publication queue handle      */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle           */
    MQLONG   CompCode = MQCC_OK;          /* completion code               */
    MQLONG   Reason = MQRC_NONE;          /* reason code                   */
    MQLONG   messlen = 0;
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor       */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor            */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options           */

    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){               /* Replace defaults with args if provided */
        default:
          topicString = argv[2];
        case(2):
          if (strcmp(argv[1],"/"))       /* "/" invalid = No topic object  */
             topicName = argv[1];
          else
             *topicName = '\0';
        case(1):
             printf("Optional parameters: topicName, topicString\nValues \"%s\"  \"%s\"\n",
                   topicName, topicString);
}
```

Abbildung 71. Verwalteter MQ-Subskribent - Teil 1: Deklarationen und Handhabung der Parameter

Zu den Deklarationen in diesem Beispiel sind einige Anmerkungen erforderlich:

MQHOBJ Hobj = MQHO_NONE;
Eine verwaltete, nicht permanente Subskriptionswarteschlange können Sie nicht explizit öffnen, um die darin enthaltenen Veröffentlichungen abzurufen; jedoch müssen Sie für die Objektkennung, die der Warteschlangenmanager zurückgibt, wenn er die Warteschlange für Sie öffnet, Speicher reservieren. Es ist wichtig, die Kennung mit MQHO_OBJECT zu initialisieren. Dadurch wird der Warteschlangenmanager angewiesen, eine Warteschlangenkennung in die Subskriptionswarteschlange zurückzugeben.

MQSD sd = {MQSD_DEFAULT};
Der neue, in MQSUB verwendete Subskriptionsdeskriptor.

MQCHAR48 qName;
Auch wenn die Subskriptionswarteschlange in diesem Beispiel nicht bekannt sein muss, wird der Name der Subskriptionswarteschlange abgefragt - die MQINQ-Bindung ist in der Programmiersprache C ein wenig seltsam; Sie werden diesen Teil des Beispiels daher recht interessant finden.

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING ;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\"\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publicationBuffer, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer-1),
                publication, &messlen, &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
    return;
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs];                /* Array of integer attributes  */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
          &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}
```

Abbildung 72. Verwalteter MQ-Subskribent - Teil 2: Hauptteil des Codes

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from "SYSTEM.MANAGED.NDURAB-
LE.48A0AC7403300020    "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values ""  "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from "SYSTEM.MANAGED.NDURAB-
LE.48A0AC7403310020    "
Received publication "150"
Completion code 0 and Return code 0
```

Abbildung 73. MQ-Subskribent

Zum Code in diesem Beispiel sind einige Anmerkungen erforderlich:

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
Wenn der Themenname (topicName) null oder leer ist (*Standard*), wird der Themenname nicht zur Ermittlung der aufgelösten Themenzeichenfolge verwendet.

sd.ObjectString.VSPtr = topicString;
Statt nur ein vordefiniertes Themenobjekt zu verwenden, werden in diesem Beispiel ein Themenobjekt und eine Themenzeichenfolge bereitgestellt, die durch MQSUB kombiniert werden. Wie Sie sehen, handelt es sich bei der Themenzeichenfolge um eine MQCHARV-Struktur.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Eine Alternative zur Einstellung der Länge eines MQCHARV-Felds.

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
Nach der Definition der Themenzeichenfolge erfordern die sd.Options-Flags die größte Aufmerksamkeit. Für diese Flags gibt es die verschiedensten Optionen, wobei in diesem Beispiel nur die am häufigsten verwendeten angegeben wurden. Für alle anderen Optionen werden die Standardeinstellungen übernommen.

1. Da es sich um eine *nicht permanente* Subskription handelt, die Subskription also nur so lange besteht, wie Sie in der Anwendung geöffnet ist, müssen Sie das Flag MQSO_CREATE setzen. Zur besseren Lesbarkeit können Sie das Flag MQSO_NON_DURABLE auch auf *(default)* setzen.
2. MQSO_CREATE wird durch MQSO_RESUME ergänzt. Beide Flags können gleichzeitig gesetzt werden. Der Warteschlangenmanager erstellt dann entweder eine neue Subskription oder er nimmt eine vorhandene Subskription wieder auf (was immer zutreffend ist). Wenn Sie aber MQSO_RESUME angeben, müssen Sie auch die MQCHARV-Struktur für sd.SubName initialisieren, selbst wenn es keine wieder aufzunehmende Subskription gibt. Fehlt die Initialisierung von SubName, so gibt MQSUB den Rückkehrcode 2440: MQRC_SUB_NAME_ERROR aus.

   Anmerkung: Bei einer verwalteten, nicht permanenten Subskription wird MQSO_RESUME immer ignoriert. Dennoch würde dessen Angabe ohne Initialisierung der MQCHARV-Struktur für sd.SubName zu diesem Fehler führen.
3. Neben diesen beiden Flags steuert ein drittes Flag, das Flag MQSO_ALTER, wie die Subskription geöffnet wird. Sofern die entsprechenden Berechtigungen vorliegen, passt dieses Flag die Eigenschaften einer wieder aufgenommenen Subskription an die in MQSUB angegebenen Attribute an.

   Anmerkung: Mindestens eines der drei Flags MQSO_CREATE, MQSO_RESUME und MQSO_ALTER muss angegeben sein. Siehe auch Optionen (MQLONG). Im Abschnitt „Beispiel 3: Nicht verwalteter MQ-Subskribent" auf Seite 869 finden Sie Beispiele, in denen alle drei Flags verwendet werden.
4. Setzen Sie das Flag MQSO_MANAGED, wenn der Warteschlangenmanager die Subskription automatisch verwalten soll.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Optional können Sie die Längenangabe für MQCHARV bei Zeichenfolgen mit Nullabschluss auch weglassen und stattdessen das Flag MQVS_NULL_TERMINATED verwenden.

sd.ResObjectString.VSPtr = resTopicStr;
Die sich ergebende Themenzeichenfolge wird im ersten printf des Programms zurückgemeldet. Richten Sie MQCHARV ResObjectString so ein, dass IBM MQ die aufgelöste Zeichenfolge zurück an das Programm meldet.

Anmerkung: Im Beispiel wurde resTopicStringBuffer in memset(resTopicStr, 0, sizeof(resTopicStrBuffer)) mit Nullen initialisiert. Die zurückgegebenen Themenzeichenfolgen enden nicht mit einer abschließenden Null.

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
Stellen Sie die Puffergröße von sd.ResObjectString um 1 kleiner als seine tatsächliche Größe ein. Dadurch verhindern Sie, dass der bereitgestellte Nullabschluss in dem Fall, dass die aufgelöste Themenzeichenfolge den gesamten Puffer füllt, überschrieben wird.

Anmerkung: Es wird kein Fehler zurückgegeben, wenn die Themenzeichenfolge länger als sizeof(resTopicStrBuffer)-1 ist. Selbst wenn VSLength > VSBufSiz ist, ist die in sd.ResObjectString.VSLength zurückgegebene Länge die Länge der vollständigen Zeichenfolge

und nicht unbedingt die Länge der zurückgegebenen Zeichenfolge. Testen Sie `sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz` , um sicherzustellen, dass die Themenzeichenfolge vollständig ist.

`MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);`
Die Funktion MQSUB erstellt eine Subskription. Bei einer nicht permanenten Subskription müssen Sie den Namen nicht unbedingt wissen, Sie können allerdings ihren Status in IBM MQ Explorer überprüfen. Damit Sie in diesem Fall wissen, nach welchem Namen Sie suchen müssen, können Sie den Parameter `sd.SubName` als Eingabe bereitstellen. Achten Sie dabei darauf, dass Sie keine Namenskonflikte mit anderen Subskriptionen herbeiführen.

`MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);`
Optional können Sie beim Schließen der Subskription auch die Subskriptionswarteschlange schließen. Im Beispiel wird nur die Subskription, nicht aber die Subskriptionswarteschlange geschlossen. Die Option MQCLOSE `MQCO_REMOVE_SUB` ist in diesem Fall ohnehin die Standardeinstellung, da die Subskription nicht permanent ist. Die Verwendung von `MQCO_KEEP_SUB` führt zu einem Fehler.

Anmerkung: Die *Subskriptionswarteschlange* wird von MQSUB nicht geschlossen und ihre Kennung `Hobj` bleibt so lange gültig, bis die Warteschlange durch MQCLOSE oder MQDISC geschlossen wird. Wenn die Anwendung unplanmäßig beendet wird, werden die Warteschlange und die Subskription nach Ablauf einer gewissen Zeit vom Warteschlangenmanager bereinigt.

Zugehörige Konzepte

„Beispiel 1: Konsument einer MQ-Veröffentlichung” auf Seite 860
Der MQ-Veröffentlichungskonsument ist ein IBM MQ-Nachrichtenkonsument, der nicht selbst Themen subskribiert.

„Beispiel 3: Nicht verwalteter MQ-Subskribent” auf Seite 869
Der nicht verwaltete Subskribent ist eine wichtige Subskribentenanwendungsklasse. In ihm vereinen sich die Vorteile von Publish/Subscribe und die *Steuerungsmöglichkeiten*, die Warteschlangen und die Verarbeitung von Veröffentlichungen bieten. Bei einer nicht verwalteten Subskription ist die Anwendung verantwortlich. zur Angabe der Warteschlange, in der die Subskriptionen gespeichert sind. Dieses Beispiel veranschaulicht verschiedene Arten der Kombination von Subskriptionen und Warteschlangen.

„Veröffentlichungsanwendungen erstellen” auf Seite 852
Sehen Sie sich zwei Beispiele an, bevor Sie Veröffentlichungsanwendungen erstellen. Das erste Beispiel wurde möglichst getreu nach einer Punkt-zu-Punkt-Anwendung modelliert, die Nachrichten in eine Warteschlange einreiht, das zweite Beispiel und geläufigere Muster für Veröffentlichungsanwendungen veranschaulicht, wie Themen dynamisch erstellt werden.

Beispiel 3: Nicht verwalteter MQ-Subskribent
Der nicht verwaltete Subskribent ist eine wichtige Subskribentenanwendungsklasse. In ihm vereinen sich die Vorteile von Publish/Subscribe und die *Steuerungsmöglichkeiten*, die Warteschlangen und die Verarbeitung von Veröffentlichungen bieten. Bei einer nicht verwalteten Subskription ist die Anwendung verantwortlich. zur Angabe der Warteschlange, in der die Subskriptionen gespeichert sind. Dieses Beispiel veranschaulicht verschiedene Arten der Kombination von Subskriptionen und Warteschlangen.

Das nicht verwaltete Muster wird eher mit *permanenten* Subskriptionen als mit *nicht permanenten* Subskriptionen in Verbindung gebracht. In der Regel ist der Lebenszyklus einer Subskription, die von einem nicht verwalteten Subskribenten erstellt wurde, unabhängig vom Lebenszyklus der Subskribentenanwendung. So empfängt eine permanente Subskription auch dann noch Veröffentlichungen, wenn die Subskribentenanwendung nicht mehr aktiv ist.

Mit permanenten *verwalteten* Subskriptionen können Sie zwar das gleiche Ergebnis erzielen, für einige Anwendungen sind jedoch eine größere Flexibilität und mehr Steuerungsmöglichkeiten über Warteschlangen und Nachrichten erforderlich, als sie mit einer verwalteten Subskription erreicht werden können. Bei einer permanenten verwalteten Subskription erstellt der Warteschlangenmanager eine permanente Warteschlange für die mit dem Subskriptionsthema übereinstimmenden Veröffentlichungen. Wenn die Subskription gelöscht wird, löscht der Warteschlangenmanager auch die Warteschlange und die darin enthaltenen Veröffentlichungen.

Permanente *verwaltete* Subskriptionen werden in der Regel verwendet, wenn die Lebenszyklen der Anwendung und der Subskription voraussichtlich identisch, jedoch schwer vorherzusagen sind. Indem Sie eine permanente Subskription erstellen und die Veröffentlichungsanwendung so einrichten, dass sie permanente Veröffentlichungen erstellt, stellen Sie sicher, dass auch dann keine Nachrichten verloren gehen, wenn der Warteschlangenmanager oder der Subskribent vorzeitig beendet wird und wiederhergestellt werden muss.

Bei Nicht-JMS-Anwendungen oder JMS-Anwendungen, die keine gemeinsame Subskription verwenden, wird der Warteschlangenmanager die permanente verwaltete Subskriptionswarteschlange für einen Subskribenten implizit so öffnen, dass eine gemeinsame Verarbeitung der Warteschlange nicht möglich ist. Zudem können Sie pro verwalteter Warteschlange nur eine Subskription erstellen, es sei denn Ihre Anwendung verwendet gemeinsame JMS-Subskriptionen. Und die Verwaltung der Warteschlangen wird Ihnen vermutlich mehr Schwierigkeiten bereiten, da Sie weniger Kontrolle über die Namen der Warteschlangen haben. Aus diesem Grund sollten Sie sich gut überlegen, ob sich der *nicht verwaltete* MQ-Subskribent für Anwendungen, die permanente Subskriptionen erfordern, eventuell besser eignet als der *verwaltete* MQ-Subskribent.

Der Code in Abbildung 76 auf Seite 876 zeigt ein nicht verwaltetes permanentes Subskriptionsmuster. Zur Veranschaulichung erstellt der Code auch nicht verwaltete, nicht permanente Subskriptionen. Das Beispiel veranschaulicht die folgenden Musterfacetten:

- On-demand-Subskriptionen: Die Themenzeichenfolgen der Subskriptionen sind dynamisch. Sie werden von der Anwendung bereitgestellt, so lange sie aktiv ist.
- Einfachere Verwaltung der Subskriptionsthemen: Die Verwaltung der Subskriptionsthemen vereinfacht sich durch Definition der Stammkomponente der Subskriptionsthemenzeichenfolge mittels eines administrativ definierten Themas. Dadurch bleibt die Stammkomponente der Themenstruktur in der Anwendung verborgen. Durch Verbergen der Stammkomponente kann ein Subskribent in verschiedenen Themenstrukturen implementiert werden.
- Flexiblere Subskriptionsverwaltung: Eine Subskription kann administrativ, aber auch bei Bedarf in einem Subskribentenprogramm definiert werden. Administrativ und programmgesteuert erstellte Subskriptionen unterscheiden sich lediglich durch ein Attribut, das angibt, wie die Subskription erstellt wurde. Außerdem gibt es einen dritten Subskriptionstyp; diese Subskription wird automatisch vom Warteschlangenmanager zur Verteilung von Subskriptionen erstellt. Alle Subskriptionen werden in IBM MQ Explorer angezeigt.
- Flexiblere Zuordnung von Subskriptionen zu Warteschlangen: Einer Subskription wird mittels der Funktion MQSUB eine vordefinierte lokale Warteschlange zugeordnet. Die Zuordnung von Subskriptionen zu Warteschlangen mittels MQSUB kann auf verschiedene Weisen erfolgen:
  - Zuordnen einer Subskription zu einer Warteschlange, der noch *keine* Subskriptionen zugeordnet sind: `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Zuordnen einer *neuen* Subskription zu einer Warteschlange, der bereits Subskriptionen zugeordnet sind: `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Verschieben einer vorhandenen Subskription in eine andere Warteschlange: `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Wiederaufnehmen einer vorhandenen Subskription, die einer vorhandenen Warteschlange zugeordnet ist: `MQSO_RESUME + (Hobj = MQHO_NONE)` oder `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`.
  - Durch Kombination von `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` in verschiedenen Zusammenstellungen können Sie verschiedene Eingabestatus der Subskription und der Warteschlange bedienen, ohne mehrere Versionen von MQSUB mit verschiedenen `sd.Options`-Werten codieren zu müssen.
  - Wenn Sie andernfalls eine bestimmte Kombination von `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` codieren und die Status der als Eingabe in MQSUB bereitgestellten Subskription und Warteschlange nicht mit dem Wert von `sd.Options` übereinstimmen, gibt der Warteschlangenmanager einen Fehler zurück (Tabelle 123 auf Seite 872). Abbildung 82 auf Seite 879 zeigt das Ergebnis, wenn MQSUB für Subskription X mit unterschiedlichen Einstellungen des `sd.Options`-Flags ausgegeben wird und ihm drei verschiedene Objektkennungen übergeben werden.

Probieren Sie für das in Abbildung 75 auf Seite 875 gezeigte Beispielprogramm verschiedene Eingaben aus, um sich mit diesen unterschiedlichen Fehlerarten vertraut zu machen. Ein häufiger Fehler (RC = 2440), der nicht in die Fallbeispiele der Tabelle aufgenommen wurde, ist ein Subskriptionsnamensfehler. Er wird häufig durch die Übergabe eines ungültigen oder eines Nullsubskriptionsnamens in MQSO_RESUME oder MQSO_ALTER verursacht.

- Multiprocessing: Der Arbeitsaufwand für das Lesen der Veröffentlichungen kann auf mehrere Konsumenten verteilt werden. Die Veröffentlichungen werden sämtlichst in die Warteschlange eingereiht, die dem Subskriptionsthema zugeordnet ist. Die Konsumenten haben die Wahl, die Warteschlange direkt mittels MQOPEN zu öffnen oder die Subskription mittels MQSUB wieder aufzunehmen.
- Subskriptionskonzentration: Mehrere Subskriptionen können in der gleichen Warteschlange erstellt werden. Bei Verwendung dieser Funktion müssen Sie äußerst sorgfältig vorgehen, da sie zu einer Überlappung von Subskriptionen und der mehrfachen Zustellung gleicher Veröffentlichungen führen kann. Allerdings können mit der Option MQSO_GROUP_SUB durch überlappende Subskriptionen mehrfach zugestellte, gleiche Veröffentlichungen verhindert werden.
- Trennung von Subskribenten und Konsumenten: Neben den drei in den Beispielen illustrierten Konsumentenmodellen gibt es ein weiteres Modell, in dem der Konsument vom Subskribent getrennt wird. Es handelt sich hier um eine Variante des nicht verwalteten MQ-Subskribenten, statt aber MQOPEN und MQSUB im gleichen Programm auszugeben, subskribiert in diesem Modell ein Programm die Veröffentlichungen und ein anderes Programm konsumiert sie. Der Subskribent ist beispielsweise Teil eines Publish/Subscribe-Clusters, der Konsument ist aber einem Warteschlangenmanager außerhalb des Warteschlangenmanagerclusters zugeordnet. Der Konsument empfängt die Veröffentlichungen über die standardmäßige verteilte Steuerung von Warteschlangen, indem die Subskriptionswarteschlange als ferne Warteschlangendefinition definiert wird.

Sie sollten sich unbedingt mit dem Verhalten von MQSO_CREATE | MQSO_RESUME | MQSO_ALTER vertraut machen, besonders wenn Sie Ihren Code durch Kombinationen dieser drei Optionen vereinfachen wollen. Sehen Sie sich daher unbedingt die Tabelle Tabelle 123 auf Seite 872 an, die die Ergebnisse zeigt, wenn MQSUB verschiedene Warteschlangenkennungen übergeben werden; sehen Sie sich außerdem die Ergebnisse der Ausführung des Beispielprogramms in Abbildung 77 auf Seite 877 bis Abbildung 82 auf Seite 879 an.

Der Tabelle liegt ein Szenario mit der Subskription X und den beiden Warteschlangen A und B zugrunde. Der Parameter für den Subskriptionsnamen (sd.SubName) ist auf X gesetzt; dies ist der Name einer Subskription, die der Warteschlange A zugeordnet ist. Der Warteschlange B ist keine Subskription zugeordnet.

In Tabelle 123 auf Seite 872wird MQSUB die Subskription X und die Warteschlangenkennung an die Warteschlange Aübergeben. Dies führt zu folgenden Ergebnissen der Subskriptionsoptionen:

- MQSO_CREATE schlägt fehl, weil die Warteschlangenkennung zu Warteschlange A gehört, der bereits die Subskription X zugeordnet ist. Vergleichen Sie dieses Verhalten mit einem erfolgreichen Aufruf. Dieser ist erfolgreich, weil Warteschlange B nicht die Subskription X zugeordnet ist.
- MQSO_RESUME ist erfolgreich, weil die Warteschlangenkennung zu Warteschlange A gehört, der bereits die Subskription X zugeordnet ist. Der Aufruf schlägt jedoch fehl, wenn die Subskription X noch nicht der Warteschlange A zugeordnet ist.
- MQSO_ALTER verhält sich hinsichtlich des Öffnens der Subskription und der Warteschlange genauso wie MQSO_RESUME. Wenn sich allerdings die Attribute im Subskriptionsdeskriptor, der an MQSUB übergeben wird, von den Attributen der Subskription unterscheiden, schlägt MQSO_RESUME fehl, während MQSO_ALTER erfolgreich ausgeführt wird, sofern die Programminstanz über die Berechtigung zum Ändern der Attribute verfügt. Sie können die Themenzeichenfolge einer Subskription nicht ändern. Statt jedoch einen Fehler zurückzugeben, ignoriert MQSUB den Themennamen und die Themenzeichenfolge im Subskriptionsdeskriptor und verwendet stattdessen die Werte der vorhandenen Subskription.

Betrachten Sie als Nächstes Tabelle 123 auf Seite 872, in der Subskription X und die Warteschlangenkennung für Warteschlange B an MQSUB übergeben werden. Dies führt zu folgenden Ergebnissen der Subskriptionsoptionen:

- MQSO_CREATE wird erfolgreich ausgeführt und erstellt Subskription X in Warteschlange B, da diese Subskription in Warteschlange B noch nicht vorhanden ist.

- MQSO_RESUME schlägt fehl. MQSUB sucht in Warteschlange B nach Subskription X und findet sie dort nicht; statt aber *RC = 2428 - subscription X does not exist* zurückzugeben, gibt der Aufruf *RC = 2019 - Subscription queue does not match queue object handle* zurück. Das Verhalten der dritten Option MQSO_ALTER gibt einen Hinweis auf den Grund dieses unerwarteten Fehlers. MQSUB erwartet, dass die Warteschlangenkennung auf eine Warteschlange mit einer Subskription verweist. Diese Voraussetzung wird zuerst überprüft, bevor untersucht wird, ob die in `sd.SubName` genannte Subskription tatsächlich vorhanden ist.
- MQSO_ALTER wird erfolgreich ausgeführt, d. h., die Subskription wird aus Warteschlange A in Warteschlange B verschoben.

Ein weiterer möglicher Fall ist in der Tabelle nicht aufgeführt, nämlich wenn der Subskriptionsname der Subskription in Warteschlange A nicht mit dem in `sd.SubName` angegebenen Subskriptionsnamen übereinstimmt. In diesem Fall würde der Aufruf mit dem Fehler *RC = 2428 - subscription X does not exist on Queue A* fehlschlagen.

Tabelle 123. MQSUB-Fehler bei verschiedenen Warteschlangenkennungen und Subskriptionskombinationen

Warteschlangenkennungen	**Warteschlange A** **Subscription X** **Warteschlange B** **Keine Subskription**	**Warteschlange A** **Keine Subskription** **Warteschlange B** **Keine Subskription**
Hobj für *Warteschlange A* an MQSUB übergeben	**MQSO_CREATE** RC = 2432 - Subskription X in Warteschlange A bereits vorhanden **MQSO_RESUME** Subskription X in Warteschlange A wird wiederaufgenommen **MQSO_ALTER** Subskription X in Warteschlange A wird wiederaufgenommen und erlaubte Änderungen werden vorgenommen	**MQSO_CREATE** Subskription X wird in Warteschlange A erstellt **MQSO_RESUME** RC = 2428 - Subskription X in Warteschlange A nicht vorhanden **MQSO_ALTER** RC = 2428 - Subskription X in Warteschlange A nicht vorhanden
Hobj für *Warteschlange B* an MQSUB übergeben	**MQSO_CREATE** Neue Subskription X wird in Warteschlange B erstellt **MQSO_RESUME** RC = 2019 - Subskriptionswarteschlange entspricht nicht der Warteschlangenobjektkennung **MQSO_ALTER** Subskription X wird aus Warteschlange A in Warteschlange B verschoben	**MQSO_CREATE** Neue Subskription X wird in Warteschlange B erstellt **MQSO_RESUME** RC = 2428 - Subskription X in Warteschlange B nicht vorhanden **MQSO_ALTER** RC = 2428 - Subskription X in Warteschlange B nicht vorhanden

Tabelle 123. MQSUB-Fehler bei verschiedenen Warteschlangenkennungen und Subskriptionskombinationen (Forts.)

Warteschlangenkennungen	**Warteschlange A** **Subscription X** **Warteschlange B** **Keine Subskription**	**Warteschlange A** **Keine Subskription** **Warteschlange B** **Keine Subskription**
MQHO_NONE an MQSUB übergeben	**MQSO_CREATE** RC = 2019 - Bad object handle. Stellen Sie das MQSO_MANAGED-Flag so ein, dass eine verwaltete Subskription erstellt wird und erstellen Sie eine verwaltete Warteschlange **MQSO_RESUME** Subskription X in Warteschlange A wird wieder aufgenommen und Hobj wird an Warteschlange A zurückgegeben **MQSO_ALTER** Subskription X in Warteschlange A wird wieder aufgenommen, Hobj wird an Warteschlange A zurückgegeben und erlaubte Änderungen werden vorgenommen	**MQSO_CREATE** RC = 2019 - Bad object handle. Stellen Sie das MQSO_MANAGED-Flag so ein, dass eine verwaltete Subskription erstellt wird und erstellen Sie eine verwaltete Warteschlange **MQSO_RESUME** RC = 2428 - No subscription X **MQSO_ALTER** RC = 2019 - Bad object handle. Keine Warteschlange A oder B

Anmerkung: Der kompakte Codierungsstil wurde aus Gründen der besseren Lesbarkeit verwendet, ist aber nicht für die Übernahme in die Produktion geeignet.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIE
SCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

Abbildung 74. Nicht verwalteter MQ-Subskribent - Teil 1: Deklarationen

```
    switch(argc){                   /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esu□
me)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

Abbildung 75. Nicht verwalteter MQ-Subskribent - Teil 2: Handhabung der Parameter

Zusätzliche Kommentare zur Parameterbehandlung in diesem Beispiel:

switch((argv[5][0]))
Sie haben die Möglichkeit, A lter | C reate | R esume in Parameter 5 einzugeben, um zu testen, wie sich das Überschreiben eines Teils der Optionseinstellung MQSUB , die im Beispiel standardmäßig verwendet wird, auswirkt. Die in diesem Beispiel verwendete Standardeinstellung lautet MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

Anmerkung: Die Einstellung MQSO_ALTER oder MQSO_RESUME ohne gleichzeitige Verwendung von MQSO_DURABLE führt zu einem Fehler. Außerdem muss sd.SubName eingestellt sein und auf eine Subskription verweisen, die wieder aufgenommen oder geändert werden kann.

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
Wenn die Standardsubskriptionswarteschlange STOCKTICKER durch eine Nullzeichenfolge ersetzt wird, aber MQSO_CREATE eingestellt ist, wird im Beispiel das Flag MQSO_MANAGED gesetzt und eine

dynamische Subskriptionswarteschlange erstellt. Wenn im fünften Parameter `Alter or Resume` eingestellt ist, richtet sich das Verhalten des Beispiels nach dem Wert von `subscriptionName`.

`*subscriptionName = '\0';`
`sdOptions = sdOptions - MQSO_DURABLE;`

Wenn die Standardsubskription `IBMSTOCKPRICESUB` durch eine Nullzeichenfolge ersetzt wird, wird im Beispiel das Flag `MQSO_DURABLE` entfernt. Wenn Sie das Beispiel unter Beibehaltung aller anderen Standardwerte ausführen, wird für `STOCKTICKER` eine weitere temporäre Subskription erstellt, die die gleichen Veröffentlichungen mehrmals erhält. Wenn Sie das Beispiel das nächste Mal ohne Parameter ausführen, erhalten Sie nur wieder eine Veröffentlichung.

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        if (strlen(subscriptionQueue)) {
            strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
            MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
                  &Hobj, &CompCode, &Reason);
            if (CompCode != MQCC_OK) break;
        }
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.SubName.VSPtr = subscriptionName;
        sd.SubName.VSLength = MQVS_NULL_TERMINATED;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
        memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
                gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen, &CompC□
ode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_NONE, &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);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors    */
    MQLONG intAttrs[_intAttrs];            /* Array of integer attributes           */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName, &CompC□
ode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

Abbildung 76. Nicht verwalteter MQ-Subskribent - Teil 3: Hauptteil des Codes

Zusätzliche Kommentare zum Code in diesem Beispiel:

if (strlen(subscriptionQueue))
Wenn kein Name für die Subskriptionswarteschlange angegeben ist, wird im Beispiel MQHO_NONE als Wert von Hobj verwendet.

MQOPEN(...);
Die Subskriptionswarteschlange wird geöffnet und die Warteschlangenkennung wird in Hobj gespeichert.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
Die Subskription wird mit Hilfe des von MQOPEN (oder von MQHO_NONE, wenn kein Subskriptionswarteschlangenname angegeben ist) übergebenen Hobj geöffnet. Eine nicht verwaltete Warteschlange kann auch wieder aufgenommen werden, ohne sie explizit mit MQOPEN zu öffnen.

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
Die Subskription wird mit Hilfe der Subskriptionskennung geschlossen. Je nachdem, ob es sich um eine permanente oder um eine nicht permanente Subskription handelt, wird die Subskription mit einem impliziten MQCO_KEEP_SUB oder MQCO_REMOVE_SUB geschlossen. Sie können eine permanente Subskription mit MQCO_REMOVE_SUB schließen, aber Sie können eine nicht permanente Subskription nicht mit MQCO_KEEP_SUB schließen. MQCO_REMOVE_SUB entfernt die Subskription, wodurch keine weiteren Veröffentlichungen mehr in die Subskriptionswarteschlange eingereiht werden.

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
Bei einer nicht verwalteten Subskription wird keine weitere Aktion vorgenommen. Wenn die Warteschlange aber verwaltet ist und die Subskription mit einem expliziten oder impliziten MQCO_REMOVE_SUB geschlossen wird, werden alle Veröffentlichungen aus der Warteschlange entfernt und schließlich auch die Warteschlange gelöscht.

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
Stellen Sie sicher, dass die empfangenen Nachrichten zu der entsprechenden Subskription gehören.

Ergebnisse des Beispiels veranschaulichen Aspekte von Publish/Subscribe:

In Abbildung 77 auf Seite 877 beginnt das Beispiel mit der Veröffentlichung von 130 unter dem Thema NYSE/IBM/PRICE.

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

Abbildung 77. Veröffentlichung von 130 unter NYSE/IBM/PRICE

In Abbildung 78 auf Seite 877 wird durch die Ausführung des Beispiels unter Beibehaltung der Standardparameter die ständige Veröffentlichung 130 empfangen. Das bereitgestellte Themenobjekt und die bereitgestellte Themenzeichenfolge werden ignoriert, wie in Abbildung 82 auf Seite 879 gezeigt. Themenobjekt und Themenzeichenfolge werden immer aus dem Subskriptionsobjekt übernommen, sofern dieses bereitgestellt wird, und die Themenzeichenfolge kann nicht geändert werden. Das tatsächliche Verhalten des Beispiels richtet sich nach der verwendeten Kombination von MQSO_CREATE, MQSO_RESUME und MQSO_ALTER. In diesem Beispiel wurde MQSO_RESUME ausgewählt.

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(rea-
te)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

Abbildung 78. Ständige Veröffentlichung empfangen

In Abbildung 79 auf Seite 878 werden keine Veröffentlichungen empfangen, da die permanente Subskription bereits die ständige Veröffentlichung erhalten hat. In diesem Beispiel wird die Subskription allein durch Bereitstellung des Subskriptionsnamens ohne Angabe des Warteschlangennamens wieder aufgenommen. Wenn der Warteschlangennamen angegeben wäre, würde die Warteschlange zuerst geöffnet und die Kennung an MQSUB übergeben werden.

Anmerkung: Fehler 2038 aus MQINQ wurde deswegen ausgegeben, weil im impliziten MQOPEN von STOCKTICKER durch MQSUB die Option MQOO_INQUIRE fehlt. Der 2038-Rückkehrcode aus MQINQ ließe sich durch explizites Öffnen der Warteschlange vermeiden.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(rea-
te)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

Abbildung 79. Subskription wieder aufnehmen

In Abbildung 80 auf Seite 878 erstellt das Beispiel eine nicht permanente, nicht verwaltete Subskription mit dem Ziel STOCKTICKER. Da es sich um eine neue Subskription handelt, erhält sie die ständige Veröffentlichung.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(rea-
te)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

Abbildung 80. Ständige Veröffentlichung bei neuer, nicht verwalteter, nicht permanenter Subskription empfangen

Zur Veranschaulichung sich überschneidender Subskriptionen wird in Abbildung 81 auf Seite 878 eine andere Veröffentlichung gesendet und dadurch die ständige Veröffentlichung geändert. Als Nächstes wird eine nicht permanente und nicht verwaltete Subskription erstellt, indem kein Subskriptionsname angegeben wird. Die ständige Veröffentlichung wird zweimal empfangen, einmal für die neue Subskription und einmal für die permanente Subskription IBMSTOCKPRICESUB, die nach wie vor in der Warteschlange STOCKTICKER aktiv ist. Das Beispiel verdeutlicht, dass Subskriptionen der Warteschlange zugeordnet sind, nicht der Anwendung. Obwohl in diesem Aufruf der Anwendung nicht auf die Subskription IBMSTOCKPRICESUB verwiesen wird, erhält die Anwendung die Veröffentlichung zweimal: einmal aus der administrativ erstellten permanenten Subskription und einmal aus der nicht permanenten Subskription, die von der Anwendung selbst erstellt wurde.

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(rea-
te)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

Abbildung 81. Überlappende Subskriptionen

Abbildung 82 auf Seite 879 zeigt, dass sich die Subskription in diesem Beispiel durch die Bereitstellung einer neuen Themenzeichenfolge und einer bestehenden Subskription nicht ändert.

1. Im ersten Beispiel nimmt `Resume` die vorhandene Subskription wieder auf, wie Sie es vermutlich erwartet haben, und ignoriert die geänderte Themenzeichenfolge.
2. Im zweiten Beispiel führt `Alter` zum Fehler `RC = 2510, Topic not alterable`.
3. Im dritten Fall führt `Create` zum Fehler `RC = 2432, Sub already exists`.

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

Abbildung 82. Subskriptionsthemen können nicht geändert werden

Zugehörige Konzepte

„Beispiel 1: Konsument einer MQ-Veröffentlichung” auf Seite 860
Der MQ-Veröffentlichungskonsument ist ein IBM MQ-Nachrichtenkonsument, der nicht selbst Themen subskribiert.

„Beispiel 2: Verwalteter MQ-Subskribent” auf Seite 863
Der verwaltete MQ-Subskribent ist das am häufigsten verwendete Subskribentenmuster. Bei einer verwalteten Subskription wird die Subskription von IBM MQ abgewickelt, ebenso werden Sie automatisch registriert und erneut registriert. Dieses Beispiel setzt *keine* administrative Definition der Warteschlangen, Themen oder Subskriptionen voraus.

„Veröffentlichungsanwendungen erstellen” auf Seite 852
Sehen Sie sich zwei Beispiele an, bevor Sie Veröffentlichungsanwendungen erstellen. Das erste Beispiel wurde möglichst getreu nach einer Punkt-zu-Punkt-Anwendung modelliert, die Nachrichten in eine Warteschlange einreiht, das zweite Beispiel und geläufigere Muster für Veröffentlichungsanwendungen veranschaulicht, wie Themen dynamisch erstellt werden.

Publish/Subscribe-Lebenszyklen

Berücksichtigen Sie bei der Entwicklung von Publish/Subscribe-Anwendungen die Lebenszyklen von Themen, Subskriptionen, Subskribenten, Veröffentlichungen, Veröffentlichungsanwendungen und Warteschlangen.

Der Lebenszyklus eines Objekts, einer Subskription zum Beispiel, beginnt mit seiner Erstellung und endet mit seiner Löschung. Zwischen diesen beiden Punkten kann es verschiedene Zustände und Änderungen durchlaufen, es kann zum Beispiel vorübergehend ausgesetzt werden, über- und untergeordnete Themen haben und vor dem Löschen ablaufen.

Bislang wurden IBM MQ-Objekte wie Warteschlangen administrativ oder durch Verwaltungsprogramme erstellt, die das Programmable Command Format (PCF) verwenden. Publish/Subscribe unterscheidet sich hiervon durch die Bereitstellung der API-Verben `MQSUB` und `MQCLOSE` zum Erstellen und Löschen von Subskriptionen. Es geht nach dem Konzept verwalteter Subskriptionen vor, durch das nicht nur Warteschlangen erstellt und gelöscht, sondern auch nicht konsumierte Nachrichten entfernt werden, und bei dem Verbindungen zwischen administrativ erstellten Themenobjekten und programmgesteuert oder administrativ erstellten Themenzeichenfolgen bestehen.

Diese funktionale Reichhaltigkeit bedient ein weites Spektrum an Publish/Subscribe-Anforderungen und vereinfacht zudem die Entwicklung einiger häufiger Publish/Subscribe-Anwendungsmuster. So vereinfachen verwaltete Subskriptionen sowohl die Programmierung als auch die Verwaltung von Subskriptionen, die nur so lange bestehen sollen wie das Programm, durch die sie erstellt wurden. Nicht verwaltete Subskriptionen vereinfachen hingegen die Programmierung, wenn eine losere Verbindung zwischen der Subskription und dem Konsum von Veröffentlichungen besteht. Zentral erstellte Subskriptionen sind nützlich, wenn der Veröffentlichungsdatenverkehr auf Basis eines zentralen Steuerungsmodells an die

Konsumenten weitergeleitet werden soll, wenn also beispielsweise Fluginformationen an automatische Gates gesendet werden sollen. Programmgesteuert erstellte Subskriptionen sind hingegen praktisch, wenn die Mitarbeiter am Gate dafür zuständig sind, die Passagierdatensätze für den jeweiligen Flug selbst zu subskribieren, indem sie am Gate eine Flugnummer eingeben.

Für das letzte Beispiel wäre eine verwaltete permanente Subskription geeignet: verwaltet, da die Subskriptionen sehr häufig erstellt werden und einen eindeutigen Endpunkt haben, dann nämlich, wenn das Gate schließt und die Subskription programmgesteuert entfernt werden kann; permanent, um zu verhindern, dass Passagierdatensätze verloren gehen, wenn das Subskribentenprogramm am Gate aus dem einen oder anderen Grund ausfällt.[8]Um die Veröffentlichung der Passagierdatensätze am Gate zu initiieren, könnte die Anwendung am Gate sowohl die Passagierdatensätze mittels der Gate-Nummer subskribieren als auch das Ereignis der Gate-Öffnung mittels der Gate-Nummer veröffentlichen. Das Veröffentlichungsprogramm reagiert auf das Ereignis der Gate-Öffnung, indem es die Passagierdatensätze veröffentlicht - diese können dann auch an andere interessierte Stellen weitergeleitet werden, beispielsweise an die Rechnungsstelle, um diese zu informieren, dass der Flug stattfindet, oder an den Kundendienst, damit dieser an die Mobiltelefone der Passagiere eine Nachricht mit der Gate-Nummer senden kann.

Der zentral verwalteten Subskription kann ein nicht verwaltetes, permanentes Modell zugrunde liegen, das über eine für jedes Gate vordefinierte Warteschlange Passagierlisten an das Gate weiterleitet.

Die folgenden drei Beispiele zu Publish/Subscribe-Lebenszyklen veranschaulichen, wie verwaltete nicht permanente, verwaltete permanente und nicht verwaltete permanente Subskribenten mit Subskriptionen, Themen, Warteschlangen, Veröffentlichungsanwendungen und dem Warteschlangenmanager interagieren und wie die Zuständigkeiten zwischen der Verwaltung und den Subskribentenprogrammen aufgeteilt werden können.

Verwalteter, nicht permanenter Subskribent

Abbildung 83 auf Seite 881 zeigt eine Anwendung, die eine verwaltete, nicht permanente Subskription erstellt, zwei Nachrichten abruft, die unter dem in der Subskription angegebenen Thema veröffentlicht sind und anschließend beendet wird. Die durch graue Kursivschrift und gestrichelte Richtungslinien dargestellten Interaktionen sind implizit.

Zu diesem Beispiel gibt es folgende Anmerkungen:

1. Die Anwendung erstellt eine Subskription zu einem Thema, zu dem bereits zwei Veröffentlichungen vorliegen. Beim Empfang seiner ersten Veröffentlichung erhält der Subskribent die *zweite* Veröffentlichung, da dies die aktuelle ständige Veröffentlichung ist.
2. Der Warteschlangenmanager erstellt sowohl eine temporäre Subskriptionswarteschlange als auch eine Subskription zum Thema.
3. Die Subskription verfügt über einen Ablaufzeitpunkt. Nach Ablauf der Subskription werden an diese Subskription zu diesem Thema keine weiteren Veröffentlichungen mehr gesendet, der Subskribent erhält aber nach wie vor Nachrichten, die vor Ablauf der Subskription veröffentlicht wurden. Der Ablauf einer Veröffentlichung wird durch den Subskriptionsablauf nicht beeinflusst.
4. Die vierte Veröffentlichung wird nicht in die Subskriptionswarteschlange eingereiht, das letzte `MQGET` gibt daher keine Veröffentlichung zurück.
5. Der Subskribent schließt zwar seine Subskription, er schließt jedoch nicht seine Verbindung zur Warteschlange bzw. zum Warteschlangenmanager.
6. Der Warteschlangenmanager nimmt kurz nach der Beendigung der Anwendung eine Bereinigung vor. Da die Subskription verwaltet und nicht permanent ist, wird die Subskriptionswarteschlange gelöscht.

[8] Natürlich muss auch das Veröffentlichungsprogramm die Passagierdatensätze als persistente Nachrichten senden, um anderen möglichen Ausfällen vorzubeugen.

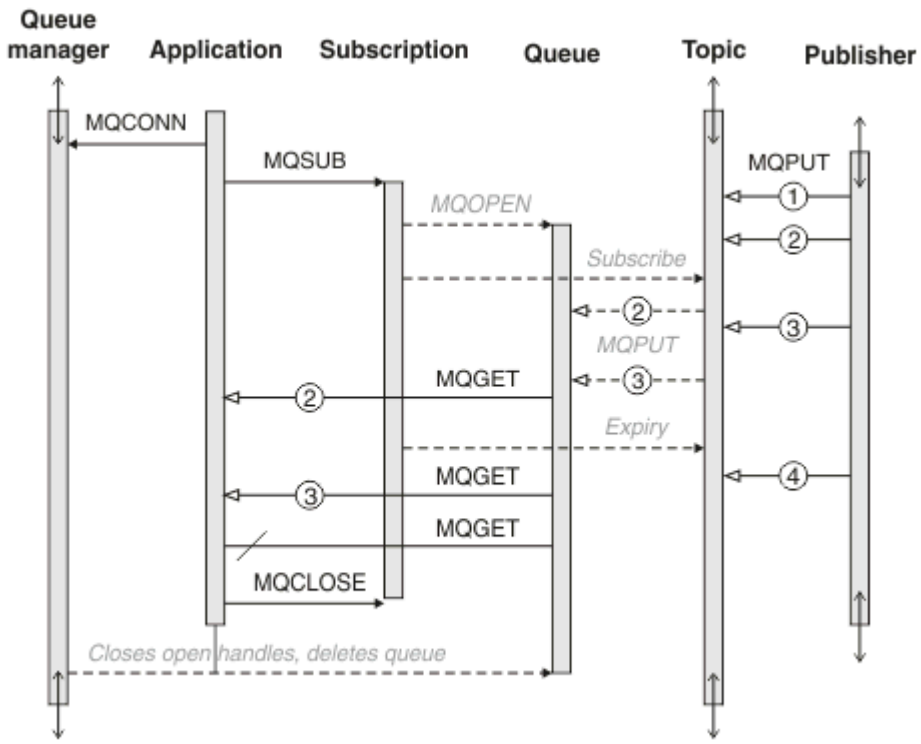


Abbildung 83. Lebenslinien eines verwalteten, nicht permanenten Subskribenten

Verwalteter, permanenter Subskribent

Dieses Beispiel eines verwalteten, permanenten Subskribenten geht noch einen Schritt weiter wie das vorangegangene Beispiel. Es zeigt eine verwaltete Subskription, die trotz Beendigung der Subskribentenanwendung fortbesteht und nach dem Neustart der Anwendung weiterhin vorhanden ist.

Zu diesem Beispiel gibt es einige zusätzliche Anmerkungen:

1. In diesem Beispiel war das Veröffentlichungsthema nicht wie beim ersten Beispiel bereits vorhanden. Es wurde erst durch die Subskription erstellt.
2. Bei der ersten Beendigung des Subskribenten schließt er die Subskription mit der Option MQCO_KEEP_SUB. Dies ist das Standardverhalten für das implizite Schließen einer verwalteten, permanenten Subskription.
3. Bei der Wideraufnahme der Subskription durch den Subskribenten wird die Subskriptionswarteschlange erneut geöffnet.
4. Die neue Veröffentlichung (2), die in die Warteschlange eingereiht wurde, bevor diese erneut geöffnet wurde, steht MQGET selbst dann noch zur Verfügung, nachdem die Subskription entfernt wurde.

   Obwohl die Subskription permanent ist, erhält der Subskribent die von der Veröffentlichungsanwendung gesendeten Nachrichten nur dann zuverlässig, wenn die Subskription *und* die Nachrichten permanent sind. Die Nachrichtenpersistenz hängt von der Einstellung des Felds Persistent im MQMD der von der Veröffentlichungsanwendung gesendeten Nachricht ab. Der Subskribent hat darüber keine Kontrolle.
5. Durch das Schließen der Subskription mit dem Flag MQCO_REMOVE_SUB wird die Subskription entfernt und folglich werden auch keine Veröffentlichungen mehr in die Subskriptionswarteschlange eingereiht. Beim Schließen der Subskriptionswarteschlange entfernt der Warteschlangenmanager die noch nicht gelesene Veröffentlichung (3) und löscht anschließend die Warteschlange. Dieser Vorgang ist vergleichbar mit einer administrativen Löschung der Subskription.

Anmerkung: Löschen Sie die Warteschlange nicht manuell und geben Sie MQCLOSE nicht mit der Option MQCO_DELETE oder MQCO_PURGE_DELETE aus. Die sichtbare Implementierung einer verwalteten Subskription ist nicht Teil der unterstützten IBM MQ-Schnittstelle. Der Warteschlangenmanager kann eine Subskription nur dann zuverlässig verwalten, wenn er die vollständige Kontrolle hat.

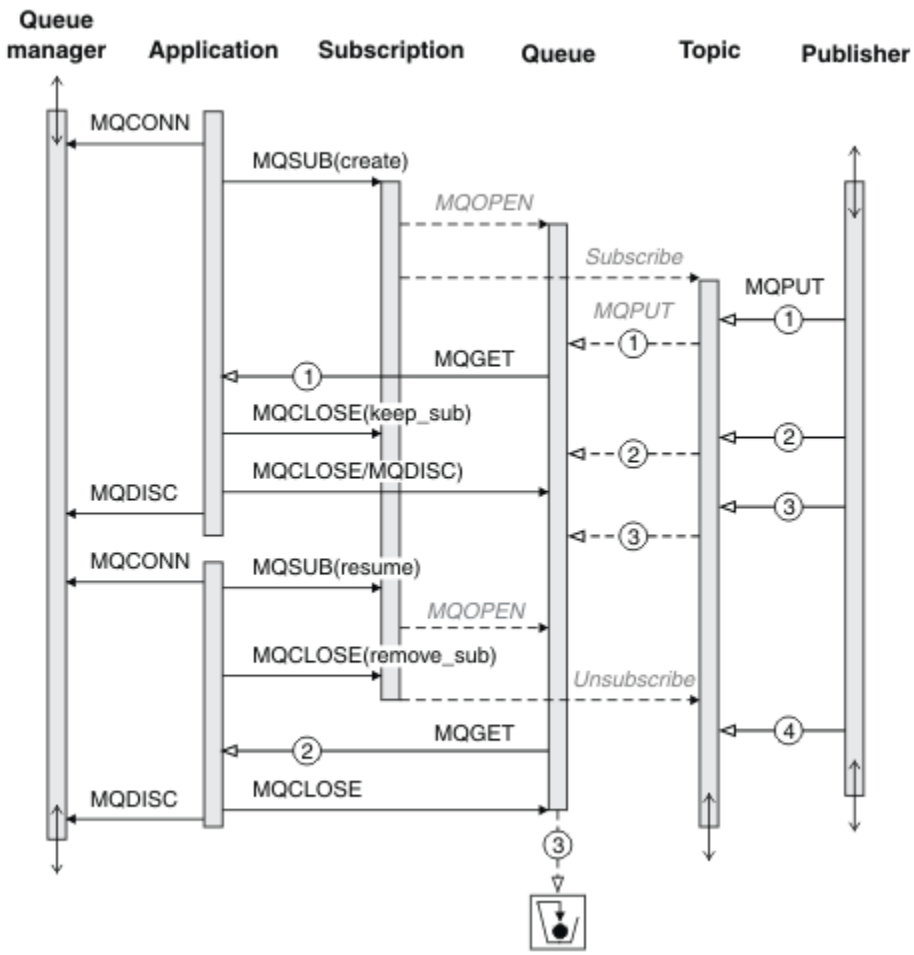


Abbildung 84. Lebenslinien eines verwalteten, permanenten Subskribenten

Nicht verwalteter, permanenter Subskribent

Im dritten Beispiel des nicht verwalteten, permanenten Subskribenten kommt ein Administrator hinzu. Anhand dieses Beispiels lässt sich hervorragend zeigen, wie ein Administrator mit einer Publish/Subscribe-Anwendung interagieren kann.

Auch zu diesem Beispiel gibt es einige Anmerkungen:

1. Die Veröffentlichungsanwendung erstellt eine Nachricht (1) für ein Thema, das später mit dem für die Subskription verwendeten Themenobjekt verbunden wird. Das Themenobjekt definiert eine Themenzeichenfolge, die aufgrund der Platzhalterzeichen mit dem Thema übereinstimmt, dem die Veröffentlichung zugeordnet wurde.
2. Das Thema enthält eine ständige Veröffentlichung.
3. Der Administrator erstellt ein Themenobjekt, eine Warteschlange und eine Subskription. Das Themenobjekt und die Warteschlange müssen vor der Erstellung der Subskription definiert werden.
4. Die Anwendung öffnet die mit der Subskription verbundene Warteschlange und übergibt MQSUB die Kennung der Warteschlange. Alternativ könnte sie auch lediglich die Subskription öffnen und ihr die Warteschlangenkennung MQHO_NONE übergeben. Umgekehrt kann sie allerdings keine Subskription

durch die alleinige Übergabe der Warteschlangenkennung ohne die Angabe des Subskriptionsnamens wiederaufnehmen, da eine Warteschlange mehrere Subskriptionen enthalten kann.

5. Die Anwendung öffnet die Subskription mit der Option MQSO_RESUME, obwohl sie die Subskription damit zum ersten Mal öffnet. Sie nimmt damit eine administrativ erstellte Subskription wieder auf.
6. Der Subskribent empfängt die ständige Veröffentlichung 1. Veröffentlichung 2, die zwar vor dem Empfang von Veröffentlichungen durch den Subskribenten, jedoch erst nach dem Start der Subskription veröffentlicht wurde, ist die zweite Veröffentlichung in der Subskriptionswarteschlange.

   Anmerkung: Wenn die ständige Veröffentlichung nicht als persistente Nachricht veröffentlicht wird, geht sie nach einem Neustart des Warteschlangenmanagers verloren.
7. In diesem Beispiel ist die Subskription permanent. Ein Programm kann zwar auch eine nicht verwaltete, nicht permanente Subskription erstellen, dem Administrator ist dies jedoch nicht möglich.
8. Durch die Option MQCO_REMOVE_SUB beim Schließen der Subskription wird die Subskription genauso entfernt, als ob der Administrator sie gelöscht hätte. Durch das Entfernen der Subskription werden an die Warteschlange keine weiteren Veröffentlichungen mehr gesendet, allerdings wirkt sich dies im Gegensatz zu einer *verwalteten*, permanenten Subskription nicht auf Veröffentlichungen aus, die sich bereits in der Warteschlange befinden, selbst wenn die Warteschlange geschlossen wird.
9. Der Administrator löscht die verbleibende Nachricht 3 später und löscht anschließend die Warteschlange.

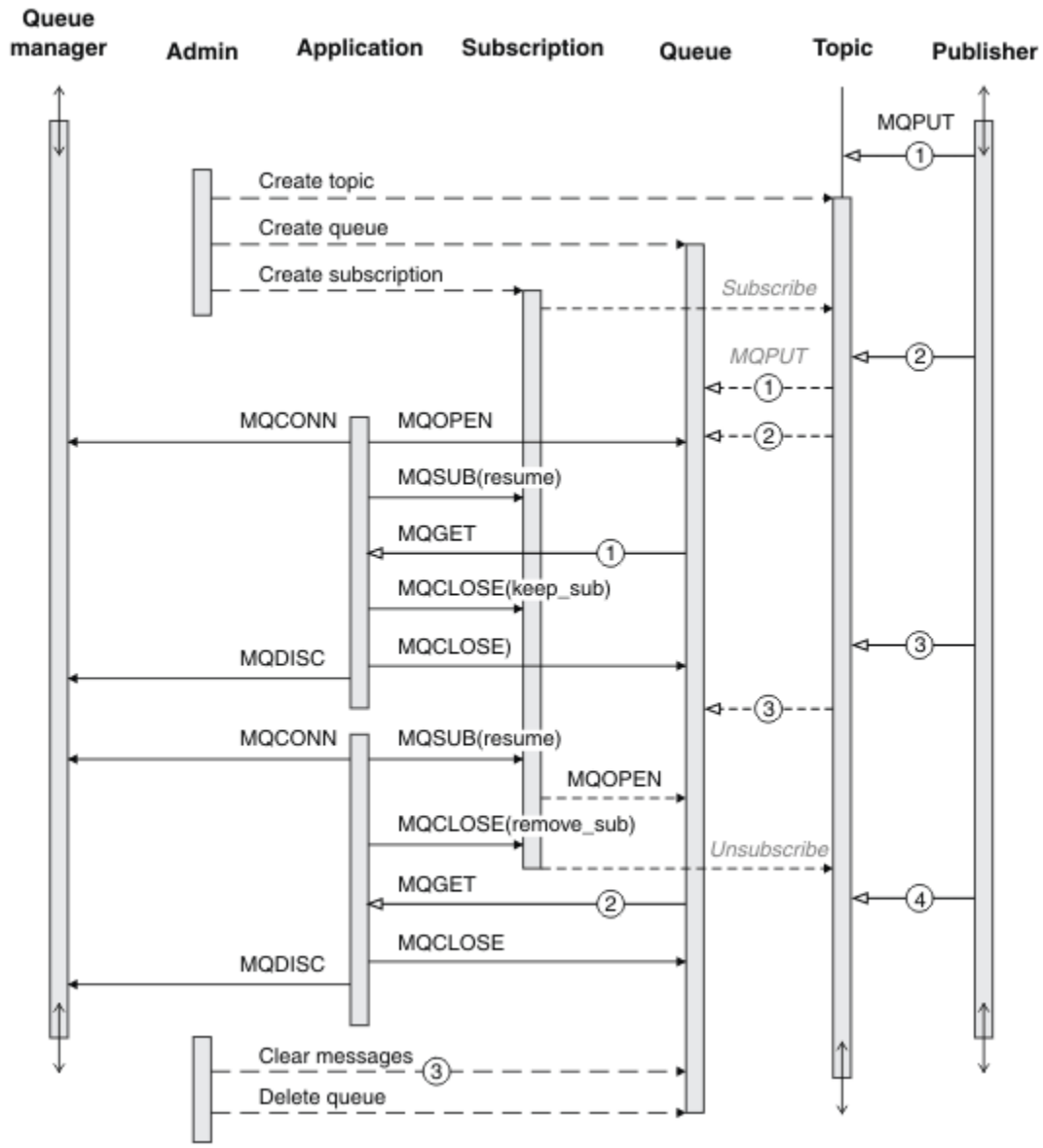


Abbildung 85. Lebenslinien eines nicht verwalteten, permanenten Subskribenten

Bei einer nicht verwalteten Subskription ist normalerweise der Administrator für die Verwaltung der Warteschlange und der Subskription zuständig. In der Regel wird bei dieser Art der Subskription nicht versucht, das Verhalten eines verwalteten Subskribenten zu emulieren und die Warteschlangen und Subskriptionen programmgesteuert durch den Anwendungscode zu bereinigen. Wenn Sie eine Verwaltungslogik entwickeln müssen, stellen Sie sich die Frage, ob Sie das gleiche Ergebnis auch mittels einer verwalteten Lösung erreichen können. Es ist nicht einfach, gut synchronisierten, in jeder Hinsicht zuverlässigen Verwaltungscode zu schreiben. Einfacher ist es, Nachrichten, Subskriptionen und Warteschlangen später, manuell oder durch ein automatisches Verwaltungsprogramm, zu löschen, wenn Sie sicher sind, dass diese unabhängig von ihrem Status nicht mehr benötigt werden.

Publish/Subscribe-Nachrichteneigenschaften

Zahlreiche Nachrichteneigenschaften betreffen das IBM MQ-Publish/Subscribe-Messaging.

PubAccountingToken

Dies ist der Wert, der in dem Feld 'AccountingToken' des Nachrichtendeskriptors (MQMD) aller Veröffentlichungsnachrichten enthalten ist, die mit dieser Subskription übereinstimmen. 'AccountingToken' ist ein Teil des Identitätskontexts der Nachricht. Weitere Informationen zum Nachrichtenkontext finden Sie im Abschnitt „Nachrichtenkontext” auf Seite 50. Weitere Informationen zum Feld 'AccountingToken' im Nachrichtendeskriptor finden Sie im Abschnitt AccountingToken.

PubApplIdentityData

Dies ist der Wert, der in dem Feld 'ApplIdentityData' des Nachrichtendeskriptors (MQMD) aller Veröffentlichungsnachrichten enthalten ist, die mit dieser Subskription übereinstimmen. 'ApplIdentityData' ist ein Teil des Identitätskontexts der Nachricht. Weitere Informationen zum Nachrichtenkontext finden Sie im Abschnitt „Nachrichtenkontext” auf Seite 50. Weitere Informationen zum Feld 'ApplIdentityData' im Nachrichtendeskriptor finden Sie im Abschnitt ApplIdentityData.

Falls die Option MQSO_SET_IDENTITY_CONTEXT nicht angegeben wird, werden für das Element 'ApplIdentityData', das in jeder Nachricht, die für diese Subskription veröffentlicht wird, enthalten ist, Leerzeichen als Standard-Kontextinformationen eingefügt.

Falls die Option MQSO_SET_IDENTITY_CONTEXT angegeben wird, wird das Element 'PubApplIdentityData' durch den Benutzer erzeugt. Dieses Feld ist dann ein Eingabefeld, das das Element 'ApplIdentityData-'enthält, das in jeder Veröffentlichung für diese Subskription angegebenen wird.

PubPriority

Dies ist der Wert, der in dem Feld 'Priority' des Nachrichtendeskriptors (MQMD) aller Veröffentlichungsnachrichten enthalten ist, die mit dieser Subskription übereinstimmen. Weitere Informationen zum Feld 'Priority' im Nachrichtendeskriptor finden Sie im Abschnitt Priority.

Der Wert muss größer oder gleich Null sein. Null steht für die niedrigste Priorität. Die folgenden besonderen Werte können ebenfalls verwendet werden:

- MQPRI_PRIORITY_AS_Q_DEF - Wenn eine Subskriptionswarteschlange im Feld 'Hobj' des Aufrufs MQSUB angegeben wird, und es sich nicht um eine verwaltete Kennung handelt, dann wird die Priorität der Nachricht von dem Attribut 'DefPriority' dieser Warteschlange übernommen. Wenn es sich bei dieser angegebenen Warteschlange um eine Clusterwarteschlange handelt oder es mehrere Definitionen im Auflösungspfad des Warteschlangennamens gibt, wird die Priorität bestimmt, wenn die Veröffentlichungsnachricht, wie für Priority im Nachrichtendeskriptor beschrieben, in die Warteschlange eingereiht wird. Falls der Aufruf MQSUB eine verwaltete Kennung verwendet, wird die Priorität für die Nachricht von dem Attribut 'DefPriority' übernommen, und zwar von der Modellwarteschlange, die dem subskribierten Thema zugeordnet ist.
- MQPRI_PRIORITY_AS_PUBLISHED - Die Priorität der Nachricht entspricht der Priorität der ursprünglichen Veröffentlichung. Dies ist der Anfangswert dieses Felds.

SubCorrelId

Achtung: Eine Korrelations-ID kann nur zwischen Warteschlangenmanagern in einem Publish/Subscribe-Cluster übergeben werden, nicht in einer Hierarchie.

Alle Veröffentlichungen, die zum Abgleich mit dieser Subskription versandt werden, enthalten diese Korrelations-ID im Nachrichtendeskriptor. Falls mehrere Subskriptionen ihre Veröffentlichungen aus derselben Warteschlange abrufen, ist es möglich, mit MQGET und der Angabe der Korrelations-ID ausschließlich die Veröffentlichungen für eine bestimmte Subskription abzurufen. Diese Korrelations-ID kann entweder vom Warteschlangenmanager oder vom Benutzer generiert werden.

Falls die Option MQSO_SET_CORREL_ID nicht angegeben wurde, wird die Korrelations-ID durch den Warteschlangenmanager erzeugt. Dieses Feld ist dann ein Ausgabefeld, das die Korrelations-ID enthält. Sie ist in jeder Nachricht enthalten, die für diese Subskription veröffentlicht wird.

Falls die Option MQSO_SET_CORREL_ID angegeben wird, wird die Korrelations-ID durch den Benutzer angegeben. Dieses Feld ist dann ein Eingabefeld, das die Korrelations-ID enthält, die in jeder Veröffentlichung für diese Subskription angegebenen wird. In diesem Fall, falls das Feld MQCI_NONE enthält, wird die Korrelations-ID in jeder Nachricht, die für diese Subskription veröffentlicht wird, diejenige sein, die bei der ursprünglichen Einreihung der Nachricht erstellt wurde.

Wenn die Option MQSO_GROUP_SUB angegeben ist und die angegebene Korrelations-ID mit einer vorhandenen gruppierten Subskription übereinstimmt, die dieselbe Warteschlange und eine überlappende Themenzeichenfolge verwendet, erhält nur die höchstwertige Subskription in der Gruppe eine Kopie der Veröffentlichung.

SubUserData

Dies sind die Subskriptions-Benutzerdaten. Die Daten, die bei Subskription in diesem Feld angegeben werden, sind als Nachrichteneigenschaft 'MQSubUserData' jeder Veröffentlichung enthalten, die an diese Subskription gesendet wird.

Veröffentlichungseigenschaften

In Tabelle 124 auf Seite 886 sind die Veröffentlichungseigenschaften aufgeführt, die in einer Veröffentlichungsnachricht enthalten sind.

Aus dem Ordner **MQRFH2** können Sie direkt auf diese Eigenschaften zugreifen oder sie mittels MQINQMP abrufen. MQINQMP akzeptiert entweder den Eigenschaftsname oder den **MQRFH2**-Namen als Namen der abzufragenden Eigenschaft.

Tabelle 124. Veröffentlichungseigenschaften

Eigenschaftsname	MQRFH2-Name	Typ	Beschreibung
MQTopicString	mqps.Top	MQTYPE_STRING	Themenzeichenfolge
MQSubUserData	mqps.Sud	MQTYPE_STRING	Subskribentbenutzerdaten
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	Ständige Veröffentlichung
MQPubOptions	mqps.Pub	MQTYPE_INT32	Veröffentlichungsoptionen
MQPubLevel	mqps.Pbl	MQTYPE_INT32	Veröffentlichungsebene
MQPubTime	mqpse.Pts	MQTYPE_STRING	Zeitpunkt der Veröffentlichung
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	Veröffentlichungsfolgenummer
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	Vom Publisher hinzugefügte Zeichenfolge/Ganzzahl-Daten
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	Nachrichtenformat: MQRFH1 MQRFH2 PCF

Nachrichtenreihenfolge bestimmen

Innerhalb eines Themas werden die Nachrichten vom Warteschlangenmanager in derselben Reihenfolge veröffentlicht, wie sie von den veröffentlichendenAnwendungen empfangen werden (eine Änderung dieser Reihenfolge erfolgt nur aufgrund der Nachrichtenpriorität).

Durch die Beibehaltung der Reihenfolge des Nachrichteneingangs empfängt jeder Subskribent normalerweise die Nachrichten von einem bestimmten Warteschlangenmanager zu einem bestimmten Thema von einer bestimmten Veröffentlichungsanwendung in derselben Reihenfolge, wie sie von der Veröffentlichungsanwendung veröffentlicht wurden.

Wie bei allen IBM MQ-Nachrichten ist es jedoch auch möglich, dass Nachrichten gelegentlich außerhalb dieser Reihenfolge zugestellt werden. Dies kann in folgenden Situationen geschehen:

- Eine Verbindung im Netz wird unterbrochen und nachfolgende Nachrichten werden über eine andere Verbindung umgeleitet.
- Eine Warteschlange ist vorübergehend voll oder gesperrt, so dass eine Nachricht in die Warteschlange für nicht zustellbare Nachrichten eingereiht und deshalb mit Verzögerung zugestellt wird, während nachfolgende Nachrichten auf direktem Wege gesendet werden.
- Ein Administrator löscht einen Warteschlangenmanager, obwohl noch Veröffentlichungsanwendungen und Subskribenten aktiv sind, so dass Nachrichten, die sich noch in Warteschlangen befinden, in die Warteschlange für nicht zustellbare Nachrichten eingereiht und Subskriptionen unterbrochen werden.

Sofern diese Umstände nicht eintreten können, werden Veröffentlichungen immer in der richtigen Reihenfolge zugestellt.

Anmerkung: Gruppierte und segmentierte Nachrichten können mit Publish/Subscribe nicht verwendet werden.

Veröffentlichungen abfangen

Sie können eine Veröffentlichung abfangen, bearbeiten und erneut veröffentlichen, bevor sie einen anderen Subskribenten erreicht.

Eventuell müssen Sie eine Veröffentlichung vor Erreichen eines Subskribenten abfangen, um folgende Aktionen durchzuführen:

- Der Nachricht weitere Informationen hinzufügen
- Die Nachricht blockieren
- Die Nachricht transformieren

Sie können den gleichen Vorgang für jede Nachricht durchführen oder das Vorgehen variieren, abhängig von der Subskription, der Nachricht oder dem Nachrichtenheader.

Zugehörige Verweise

Veröffentlichungsexit - MQ_PUBLISH_EXIT

Subskriptionsebenen

Durch die Subskriptionsebene einer Subskription können Sie eine Veröffentlichung abfangen, bevor sie ihre endgültigen Subskribenten erreicht. Ein abfangender Subskribent subskribiert auf einer höheren Subskriptionsebene und veröffentlicht die Subskription erneut auf einer niedrigeren Subskriptionsebene. Durch eine Kette abfangender Subskribenten können Sie die Nachrichten einer Veröffentlichung verarbeiten, bevor sie dem endgültigen Subskribenten zugestellt wird.

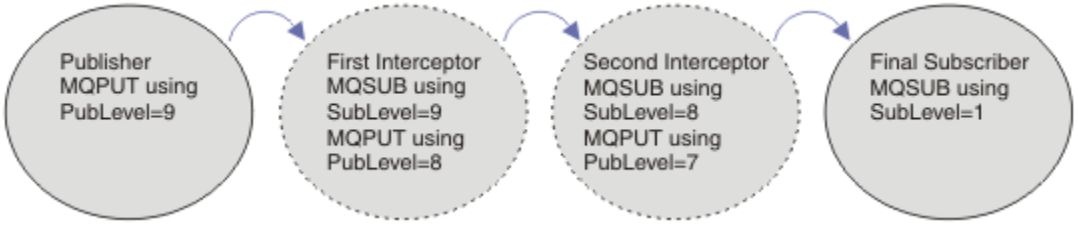


Abbildung 86. Abfolge abfangender Subskribenten

Zum Abfangen einer Veröffentlichung verwenden Sie das Subskriptionsattribut **MQSD** SubLevel. Eine abgefangene Nachricht kann transformiert und durch Änderung des Attributs **MQPMO** PubLevel auf einer niedrigeren Veröffentlichungsebene erneut veröffentlicht werden. Die Nachricht wird dann dem

endgültigen Subskribenten zugestellt oder erneut von einem dazwischenliegenden Subskribenten auf einer niedrigeren Subskriptionsebene abgefangen.

Der abfangende Subskribent transformiert die Nachricht normalerweise, bevor er sie erneut veröffentlicht. Eine Abfolge mehrerer abfangender Subskribenten bildet einen Nachrichtenfluss. Alternativ kann die Wiederveröffentlichung mit dem Abfangen der Nachricht auch abgebrochen werden. Die Subskribenten auf niedrigeren Veröffentlichungsebenen erhalten die Nachricht dann nicht.

Stellen Sie sicher, die abfangende Anwendung die Veröffentlichungen vor den Subskribenten erhält. Dazu muss die Subskriptionsebene der abfangenden Anwendung höher als die der Subskribenten sein. Standardmäßig, haben Subskribenten den `SubLevel` 1. Der höchste Wert ist 9. Eine Veröffentlichung muss mit einem `PubLevel` beginnen, der mindestens so hoch ist wie der `SubLevel`. Verwenden Sie für Veröffentlichungen zunächst die Standardeinstellung von `PubLevel` (9).

- Bei einem einzelnen abfangenden Subskribenten für ein Thema sollten Sie `SubLevel` auf 9 setzen.
- Bei mehreren abfangenden Anwendungen für ein Thema sollten Sie `SubLevel` für jeden nachfolgenden abfangenden Subskribenten jeweils um 1 herabsetzen.
- Es können maximal 8 abfangende Anwendungen implementiert werden (mit Subskriptionsebenen von 9 bis einschließlich 2). Der letzte Empfänger der Nachricht hat dann den `SubLevel` 1.

Die abfangende Anwendung mit der höchsten Subskriptionsebene kleiner oder gleich dem `PubLevel` der Veröffentlichung erhält die Veröffentlichung zuerst. Pro Subskriptionsebene darf für ein Thema nur ein abfangender Subskribent konfiguriert werden. Andernfalls, bei mehreren abfangenden Subskribenten pro Subskriptionsebene, erhält der endgültige Satz der subskribierenden Anwendungen mehrere Kopien der Veröffentlichung.

Ein Subskribent mit dem `SubLevel` `0` dient als Sammelplatz. Er erhält die Veröffentlichung, wenn kein endgültiger Subskribent für die Nachricht vorhanden ist. Ein solcher Subskribent mit dem `SubLevel` `0` kann daher zur Überwachung der Veröffentlichungen verwendet werden, die bei keinem anderen Subskribenten eingehen.

Abfangenden Subskribenten programmieren

Hierzu verwenden Sie die im Abschnitt Tabelle 125 auf Seite 888 beschriebenen Subskriptionsoptionen.

Tabelle 125. Subskriptionsoptionen für abfangende Subskribenten

Subskriptionsoption	**Anmerkungen**
`MQSO_SET_CORREL_ID` und `SubCorrelId` gleich `MQCI_NONE`	Die `CorrelId` der abgefangenen Veröffentlichung ändert sich gegenüber der ursprünglichen Veröffentlichung nicht. **Anmerkung:** In einer Hierarchie können Sie die Korrelations-ID einer Veröffentlichung nicht übergeben. Das Feld wird vom Warteschlangenmanager verwendet.
`PubPriority` gleich `MQPRI_PRIORITY_AS_PUBLISHED`	Die Priorität der abgefangenen Veröffentlichung ändert sich gegenüber der ursprünglichen Veröffentlichung nicht.

Die Optionen in Tabelle 125 auf Seite 888 müssen von allen abfangenden Subskribenten verwendet werden. Die Korrelations-ID und die Nachrichtenpriorität ändern sich dann nicht gegenüber den Einstellungen der ursprünglichen Veröffentlichungsanwendung.

Nachdem der abfangende Subskribent die Veröffentlichung bearbeitet hat, veröffentlicht er die Nachricht erneut unter dem gleichen Thema, allerdings ein `PubLevel` unter dem `SubLevel` seiner eigenen Subskription. War der `SubLevel` des abfangenden Subskribenten also 9, so veröffentlicht er die Nachricht unter dem `PubLevel` 8.

Für die korrekte Wiederveröffentlichung der Nachricht sind verschiedene Informationen der ursprünglichen Veröffentlichung erforderlich. So muss die **MQMD** der Originalnachricht verwendet werden. Außerdem muss MQPMO_PASS_ALL_CONTEXT gesetzt werden, um sicherzustellen, dass alle Informationen der **MQMD** an den nächsten Subskribenten weitergeleitet werden. Kopieren Sie hierzu die Werte der in Tabelle 126 auf Seite 889 aufgeführten Nachrichteneigenschaften in die entsprechenden Felder der wiederveröffentlichten Nachricht. Diese Werte können vom abfangenden Subskribenten geändert werden. Mit dem Operator OR können Sie dem **MQPMO**-Feld Options (Optionen) weitere Werte hinzufügen, um Optionen zum Einreihen der Nachricht zu kombinieren.

Statt eine verwaltete Veröffentlichungswarteschlange zu verwenden, müssen Sie die Veröffentlichungswarteschlange explizit öffnen. MQSO_SET_CORREL_ID kann für eine verwaltete Warteschlange nicht gesetzt werden. Ebenso kann MQOO_SAVE_ALL_CONTEXT nicht für eine verwaltete Warteschlange gesetzt werden. Sehen Sie sich hierzu auch die Codefragmente im Abschnitt „Beispiele” auf Seite 889 an.

Tabelle 126. MQPUT-Werte für erneut veröffentlichte Nachrichten

Erneutes Veröffentlichen von Nachrichten mit MQPUT	**Angaben in Veröffentlichungsnachricht**
MQOD. ObjectString	Nachrichteneigenschaft MQTopicString
MQPMO. Options	Nachrichteneigenschaft MQPubOptions

Der letzte Subskribent hat die Möglichkeit, seine Subskriptionseinstellungen anders zu setzen. Zum Beispiel kann er die Veröffentlichungspriorität statt auf MQPRI_PRIORITY_AS_PUBLISHED explizit festlegen. Die Einstellungen des letzten Subskribenten wirken sich nur auf die Veröffentlichung des letzten abfangenden Subskribenten der Kette aus.

Ständige Veröffentlichungen

Eine ständige Veröffentlichung muss nach dem Abfangen beibehalten bleiben, indem die ursprünglichen Optionen der Nachrichteneinreihung in die erneut veröffentlichte Nachricht kopiert werden.

Die Option MQPMO_RETAIN wird durch den Publisher gesetzt. Jeder abfangende Subskribent muss die MQPubOptions an die Optionen der Nachrichteneinreihung der erneut veröffentlichten Nachricht weiterleiten, wie im Abschnitt Tabelle 126 auf Seite 889 gezeigt. Durch Kopieren der Optionen der Nachrichteneinreihung bleiben die vom ursprünglichen Publisher gesetzten Optionen einschließlich der Einstellung, ob es sich um eine ständige Veröffentlichung handelt, erhalten.

Nachdem eine Veröffentlichung ihren Weg durch die Kette der abfangenden Subskribenten abgeschlossen hat und den letzten Subskribenten zugestellt wurde, wird sie endgültig aufbewahrt. Weitere Subskribenten auf SubLevel 1, die diese ständige Veröffentlichung nun anfordern, erhalten sie ohne weitere Abfangvorgänge. Subskribenten auf einem SubLevel über 1 erhalten die ständige Veröffentlichung gar nicht. Eine Änderung der ständigen Veröffentlichung in einer zweiten Runde abfangender Subskribenten ist daher nicht mehr möglich.

Beispiele

Bei den nachfolgenden Beispielen handelt es sich um Codefragmente, die zu einem abfangenden Subskribenten kombiniert werden können. Zugunsten der Kürze entsprechen diese Codesbeispiele eher nicht einer Produktionsqualität.

In den in Abbildung 87 auf Seite 890 beschriebenen Vorprozessoranweisungen werden die beiden Eigenschaften definiert, die im MQI-Aufruf MQINQMP erforderlich sind und daher aus der Veröffentlichungsnachricht extrahiert werden müssen.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define      MQPUBOPTIONS        (MQPTR)(char*) "MQPubOptions",\
                                 0,\
                                 12,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
#define      MQTOPICSTRING       (MQPTR)(char*) "MQTopicString",\
                                 0,\
                                 13,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
```

Abbildung 87. Vorprozessoranweisungen

Abbildung 88 auf Seite 890 listet die in den Codefragmenten verwendeten Deklarationen auf. Mit Ausnahme der hervorgehobenen Stellen handelt es sich hier um die Standarddeklarationen einer IBM MQ-Anwendung.

Durch die hervorgehobenen Put- und Get-Optionen wird der gesamte Kontext weitergeleitet. Die hervorgehobenen MQTOPICSTRING und MQPUBOPTIONS sind MQCHARV-Initialisierungen für die in den Vorprozessoranweisungen definierten Eigenschaftsnamen. Die Namen werden an MQINQMP weitergeleitet.

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

Abbildung 88. Deklarationen

Initialisierungen, die in den Deklarationen nicht leicht durchführbar sind, werden im Abschnitt Abbildung 89 auf Seite 891 gezeigt. Die hervorgehobenen Stellen müssen erläutert werden.

SYSTEM.NDURABLE.MODEL.QUEUE
: In diesem Beispiel öffnet MQSUB keine verwaltete nicht ständige Subskription. Stattdessen erstellt SYSTEM.NDURABLE.MODEL.QUEUE eine temporäre dynamische Warteschlange. Ihre Kennung wird

an MQSUB weitergeleitet. Durch das direkte Öffnen der Warteschlange kann der gesamte Nachrichtenkontext gespeichert werden und die Subskriptionsoption MQSO_SET_CORREL_ID kann festgelegt werden.

MQGMO_CURRENT_VERSION
Für die meisten IBM MQ-Strukturen sollte möglichst immer die aktuelle Version verwendet werden. Felder wie gmo.MsgHandle stehen nur in der neuesten Version der Steuerstrukturen zur Verfügung.

MQGMO_PROPERTIES_IN_HANDLE
Die Themenzeichenfolge und die Optionen der Nachrichteneinreihung der ursprünglichen Veröffentlichung müssen vom abfangenden Subskribenten mit Nachrichteneigenschaften abgerufen werden. Eine Alternative wäre das direkte Einlesen der Struktur **MQRFH2** in der Nachricht selbst.

MQSO_SET_CORREL_ID
Verwenden Sie MQSO_SET_CORREL_ID in Kombination mit

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

Durch diese Optionen wird die Korrelations-ID weitergeleitet. Die vom ursprünglichen Publisher festgelegte Korrelations-ID wird in das Korrelations-ID-Feld der vom abfangenden Subskribenten erhaltenen Veröffentlichung eingefügt. Jeder abfangende Subskribent gibt die gleiche Korrelations-ID weiter. Auf diese Weise erhält selbst der letzte Subskribent noch die gleiche Korrelations-ID.

Anmerkung: Innerhalb einer Publish/Subscribe-Hierarchie bleibt die Korrelations-ID einer Veröffentlichung niemals erhalten.

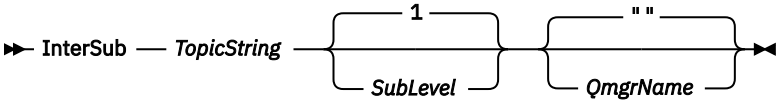
MQPRI_PRIORITY_AS_PUBLISHED
Die Veröffentlichung wird mit ihrer ursprünglichen, bei der Veröffentlichung zugewiesenen Nachrichtenpriorität in die Veröffentlichungswarteschlange eingereiht.

```
strncpy(getOD.ObjectName,    "SYSTEM.NDURABLE.MODEL.QUEUE",
        sizeof(getOD.ObjectName));
gmo.Version                = MQGMO_VERSION_4;
gmo.Options                = MQGMO_WAIT
                           | MQGMO_PROPERTIES_IN_HANDLE
                           | MQGMO_CONVERT;
gmo.WaitInterval           = 30000;
sd.Options                 = MQSO_CREATE
                           | MQSO_FAIL_IF_QUIESCING
                           | MQSO_SET_CORREL_ID;
sd.PubPriority             = MQPRI_PRIORITY_AS_PUBLISHED;
sd.Version                 = MQSD_VERSION_1;
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
putOD.ObjectType             = MQOT_TOPIC;
putOD.ObjectString.VSPtr     = &TopStrBuf;
putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
putOD.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
putOD.ObjectString.VSCCSID   = MQCCSI_APPL;
putOD.Version                = MQOD_VERSION_4;
pmo.Version                = MQPMO_VERSION_3;
```

Abbildung 89. Initialisierungen

Abbildung 90 auf Seite 892 zeigt ein Codefragment, mit dem Befehlszeilenparameter gelesen, die Initialisierung abgeschlossen und die abfangende Subskription erstellt wird.

Führen Sie das Programm mit folgendem Befehl aus:



Damit die Fehlerbehandlung möglichst nicht stört, wird der Ursachencode jedes MQI-Aufrufs in einem anderen Feldgruppenelement gespeichert. Nach jedem Aufruf wird der Beendigungscode geprüft, und falls dessen Wert MQCC_FAIL ist, wird der Codeblock do { } while(0) beendet.

Die beiden folgenden Codezeilen sind hier besonders hervorzuheben:

pmo.PubLevel = sd.SubLevel - 1;
Setzt die Veröffentlichungsebene der erneut veröffentlichten Nachricht auf einen Wert, der um eins kleiner ist als die Subskriptionsebene des abfangenden Subskribenten.

gmo.MsgHandle = Hmsg;
Stellt einen Nachrichtenhandle für MQGET für die Rückgabe der Nachrichteneigenschaften bereit.

```
do {
   printf("Intercepting subscriber start\n");
   if (argc < 2) {
       printf("Required parameter missing - topic string\n");
       exit(99);
   } else {
       sd.ObjectString.VSPtr    = argv[1];
       sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
       printf("TopicString = %s\n", sd.ObjectString.VSPtr);
   }
   if (argc > 2) {
       sd.SubLevel  = atoi(argv[2]);
       pmo.PubLevel = sd.SubLevel - 1;
       printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
   }
   if (argc > 3)
       strncpy(QMName, argv[3], sizeof(QMName));
   MQCONN(QMName, &Hcon, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   gmo.MsgHandle = Hmsg;
```

Abbildung 90. Abfangen von Veröffentlichungen vorbereiten

Das Hauptcodefragment, Abbildung 91 auf Seite 893 ruft Nachrichten aus der Veröffentlichungswarteschlange ab. Es fragt die Nachrichteneigenschaften ab und veröffentlicht die Nachrichten mit der Themenzeichenfolge und den ursprünglichen**MQPMO**. option-Eigenschaften der Veröffentlichung erneut.

In diesem Beispiel wird die Veröffentlichung nicht transformiert. Die Themenzeichenfolge der erneut veröffentlichten Veröffentlichung stimmt immer mit der Themenzeichenfolge überein, die der abfangende Subskribent subskribiert hat. Wenn der abfangende Subskribent mehrere an die gleiche Veröffentlichungswarteschlange gesendete Subskriptionen abfangen muss, muss er unter Umständen die Themenzeichenfolge abfragen, damit die zu verschiedenen Subskriptionen passenden Veröffentlichungen unterschieden werden können.

Die MQINQMP-Aufrufe sind hervorgehoben. Die Themenzeichenfolge und die Nachrichteneinreihungsoptionen der Veröffentlichung werden direkt in die Ausgabesteuerungsstrukturen geschrieben. Der einzige Grund, das Feld für die MQCHARV-Länge von putOD.ObjectString in eine auf null endende Zeichenfolge zu ändern, ist die Verwendung von printf zur Ausgabe der Zeichenfolge.

```
while (CompCode != MQCC_FAILED) {
   memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
   md.Encoding       = MQENC_NATIVE;
   md.CodedCharSetId = MQCCSI_Q_MGR;
   printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
   MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   buffer[messlen] = '\0';
   MQINQMP(Hcon, Hmsg,    &InqPropOpts, &TopStrProp, &PropDesc, &Type,
       putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
       &(putOD.ObjectString.VSLength), &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
       + putOD.ObjectString.VSLength),'\0',1);
   putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
   MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
       sizeof(pmo.Options), &(pmo.Options), &DataLength,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   printf("Republish message <%s> on topic <%s> with options %d\n",
       buffer, putOD.ObjectString.VSPtr, pmo.Options);
   MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
   break;
}
```

Abbildung 91. Veröffentlichung abfangen und erneut veröffentlichen

Das abschließende Codefragment wird im Abschnitt Abbildung 92 auf Seite 893 gezeigt.

```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

Abbildung 92. Beendigung

Veröffentlichungen und verteiltes Publish/Subscribe abfangen
Bei der Implementierung von abfangenden Subskribenten oder Publish-Exits in einer Publish/Subscribe-Topologie können Sie einem einfachen Muster folgen. Abfangende Subskribenten implementieren Sie auf den gleichen Warteschlangenmanagern wie Publisher und Publish-Exits auf den gleichen Warteschlangenmanagern wie die endgültigen Subskribenten.

Abbildung 93 auf Seite 894 zeigt zwei Warteschlangenmanager. die in einem Publish/Subscribe-Cluster verbunden sind. Ein Publisher erstellt eine Veröffentlichung auf PubLevel 9. Die nummerierten Pfeile zeigen die Reihenfolge der Schritte, die von der Veröffentlichung hin zu den Subskribenten des Clusterthemas durchlaufen werden. Die Veröffentlichung wird vom Subskribenten mit SubLevel 9 abgefangen und erneut mit PubLevel 8 veröffentlicht. Auf SubLevel 8 wird sie wieder von einem Subskribenten abgefangen. Der Subskribent veröffentlicht sie erneut auf PubLevel 7. Der vom Warteschlangenmanager bereitgestellte Proxy-Subskribent leitet die Veröffentlichung an den Warteschlangenmanager B weiter, bei dem zusätzlich zu einem letzten Subskribenten ein Veröffentlichungsexit implementiert wurde. Die Veröffentlichung wird vom Veröffentlichungsexit verarbeitet, bevor sie schließlich vom letzten Subskribenten auf SubLevel 1 empfangen wird. Die abfangenden Subskribenten und der Veröffentlichungsexit sind mit gestrichelten Umrissen dargestellt.

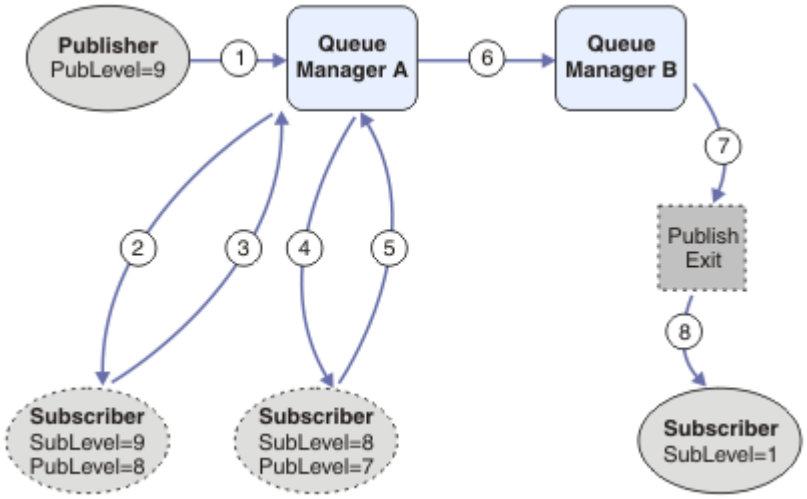


Abbildung 93. Abfangende Subskribenten und Publish-Exit in einem Cluster

Ziel dieses einfachen Musters ist es, dass alle Subskribenten, die eine Veröffentlichung erhalten, eine Veröffentlichung mit identischem Inhalt erhalten. Unabhängig vom Verbindungspunkt des Subskribenten durchläuft die Veröffentlichung die gleiche Reihenfolge an Transformationen. Schließlich möchten Sie ganz bestimmt nicht, dass die Reihenfolge der Transformationen je nach Verbindungspunkt des Publishers oder des endgültigen Subskribenten variiert. Sinnvoll wäre eine Ausnahme hiervon höchstens, wenn die Veröffentlichung den einzelnen Subskribenten angepasst werden soll. Dies erreichen Sie mit dem Publish-Exit, der die Veröffentlichung der Warteschlange anpasst, der die endgültige Veröffentlichung zugestellt wird.

In einer verteilten Publish/Subscribe-Topologie sollten Sie sorgfältig planen, wo abfangende Subskribenten und Publish-Exits implementiert werden. In unserem einfachen Muster werden die abfangenden Subskribenten auf dem gleichen Warteschlangenmanager implementiert wie die Publisher, und die Publish-Exits werden auf dem gleichen Warteschlangenmanager implementiert wie die endgültigen Subskribenten.

Schlechtes Musterbeispiel

Abbildung 94 auf Seite 895 zeigt, wie die Dinge schief laufen können, wenn das Muster unnötig kompliziert wird. Zur Komplizierung der Implementierung wird Warteschlangenmanager A ein endgültiger Subskribent hinzugefügt und Warteschlangenmanager B zwei weitere abfangende Subskribenten.

Die Veröffentlichung wird an Warteschlangenmanager B auf `PubLevel` 7 weitergeleitet, wo sie vom Subskribenten auf `SubLevel` 5 abgefangen wird, bevor sie vom letzten Subskribenten auf `SubLevel` 1 verarbeitet wird. Der Veröffentlichungsexit fängt die Veröffentlichung ab, bevor sie im Warteschlangenmanager B dem abfangenden Konsumenten und dem endgültigen Konsumenten übergeben wird. Die Veröffentlichung erreicht den endgültigen Subskribenten auf Warteschlangenmanager A, ohne vom Veröffentlichungsexit verarbeitet worden zu sein.

In einer Publish/Subscribe-Topologie subskribieren Proxy-Subskribenten auf `SubLevel` 1 und übergeben den vom letzten abfangenden Subskribenten festgelegten `PubLevel`. Daraus folgt in Abbildung 94 auf Seite 895, dass die Veröffentlichung nicht vom Subskribenten auf `SubLevel` 9 und dem Warteschlangenmanager B abgefangen wird.

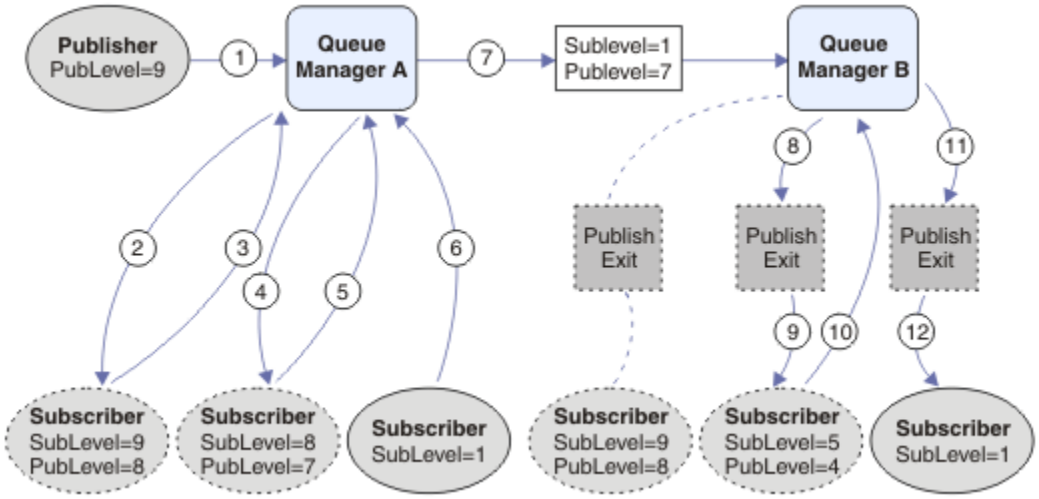


Abbildung 94. Komplexe Implementierung abfangender Subskribenten

Veröffentlichungsoptionen

Es stehen verschiedene Optionen zur Verfügung, um die Art und Weise, wie Nachrichten veröffentlicht werden, zu steuern.

Zurückhalten von Antwortinformationen für Subskribenten

Falls Sie nicht wollen, dass Subskribenten Veröffentlichungen, die sie erhalten, beantworten können, können Sie die Informationen in den Feldern ReplyToQ und ReplyToQmgr mit der Nachrichteneinreihungsoption MQPMO_SUPPRESS_REPLYTO vor MQMD zurückhalten. Falls diese Option verwendet wird, entfernt der Warteschlangenmanager diese Informationen aus MQMD, sobald er die Veröffentlichung erhält, und zwar noch bevor er diese an Subskribenten weiterleitet.

Diese Option kann nicht in Kombination mit einer Berichtsoption verwendet werden, die ein ReplyToQ-Objekt erfordert. Falls dies versucht wird, wird der Aufruf mit MQRC_MISSING_REPLY_TO_Q fehlschlagen.

Veröffentlichungsebene

Die Verwendung von Veröffentlichungsebenen ist eine Möglichkeit, um zu kontrollieren, welche Subskribenten eine Veröffentlichung erhalten. Die Veröffentlichungsebene gibt die Ebene der Subskription an, die durch die Veröffentlichung erreicht werden soll. Ausschließlich Subskriptionen mit der höchsten Subskriptionsebene, die kleiner als oder gleich der Veröffentlichungsebene der Veröffentlichung ist, erhalten die Veröffentlichung. Dieser Wert muss im Bereich von 0 bis 9 liegen. Null steht dabei für die niedrigste Veröffentlichungsebene. Der Anfangswert dieses Feldes ist 9. Eine mögliche Verwendung von Veröffentlichungs- und Subskriptionsebenen ist das Abfangen von Veröffentlichungen.

Prüfen, ob eine Veröffentlichung nicht an Subskribenten übergeben wurde

Verwenden Sie die Nachrichteneinreihungsoption MQPMO_WARN_IF_NO_SUBS_MATCHED mit dem Aufruf MQPUT, um zu prüfen, ob eine Veröffentlichung nicht an Subskribenten übergeben wurde. Wenn von der Nachrichteneinreihungsoption ein Beendigungscode von MQCC_WARNING und ein Ursachencode von MQRC_NO_SUBS_MATCHED zurückgegeben werden, wurde die Veröffentlichung an keine Subskription übermittelt. Wenn in der Operation zur Nachrichteneinreihung die Option MQPMO_RETAIN angegeben ist, wird die Nachricht beibehalten und an alle nachträglich definieren übereinstimmenden Subskriptionen übergeben. In einem verteilten Publish/Subscribe-System wird der Ursachencode MQRC_NO_SUBS_MATCHED nur zurückgegeben, wenn für das Thema im Warteschlangenmanager keine Proxy-Subskriptionen registriert sind.

Subskriptionsoptionen

Es sind mehrere Optionen verfügbar, mit deren Hilfe gesteuert werden kann, wie Nachrichtensubskriptionen gehandhabt werden.

Nachrichtenpersistenz

Warteschlangenmanager verwalten die Persistenz der Veröffentlichungen, die sie, wie von der Veröffentlichungskomponente festgelegt, an Subskribenten weiterleiten. Die Veröffentlichungskomponente legt für die Persistenz eine der folgenden Optionen fest:

0
Nicht persistent

1
Persistent

2
Persistenz als Warteschlangen-/Themendefinition

Für Publish/Subscribe löst die Veröffentlichungskomponente das Themenobjekt und die Themenzeichenfolge (**`topicString`**) in ein aufgelöstes Themenobjekt auf. Wenn die Veröffentlichungskomponente 'Persistenz als Warteschlangen-/Themendefinition' angibt, wird für die Veröffentlichung die Standardpersistenz aus dem aufgelösten Themenobjekt festgelegt.

Ständige Veröffentlichungen

Um zu steuern, wann ständige Veröffentlichungen empfangen werden, können Subskribenten zwei Subskriptionsoptionen verwenden:

Veröffentlichung nur auf Anforderung, MQSO_PUBLICATIONS_ON_REQUEST

Falls Sie Subskribenten die Möglichkeit geben wollen, zu steuern, wann Veröffentlichungen empfangen werden, können Sie die Subskriptionsoption MQSO_PUBLICATIONS_ON_REQUEST verwenden. Ein Subskribent kann steuern, ob er Veröffentlichungen empfängt, indem er den Aufruf MQSUBRQ verwendet (unter Angabe der Hsub-Kennung, die von dem ursprünglichen MQSUB-Aufruf zurückgegeben wurde), um die ständigen Veröffentlichungen eines Themas zu empfangen. Subskribenten, die die Subskriptions-Option MQSO_PUBLICATIONS_ON_REQUEST verwenden, erhalten keine temporären Veröffentlichungen.

Wenn Sie die Option MQSO_PUBLICATIONS_ON_REQUEST angeben, müssen Sie alle Veröffentlichungen mithilfe von MQSUBRQ abrufen. Wenn Sie die Option MQSO_PUBLICATIONS_ON_REQUEST nicht verwenden, erhalten Sie Nachrichten bei der Veröffentlichung.

Wenn ein Subskribent den Aufruf MQSUBRQ verwendet und Platzhalterzeichen im Thema der Subskription angibt, stimmt die Subskription möglicherweise mit mehreren Themen oder Knoten in einer Themenstruktur überein. Zu all diesen Themen mit ständigen Nachrichten (falls eine vorhanden ist) werden dann Veröffentlichungen an den Subskribenten gesendet.

Diese Optionen kann besonders dann hilfreich sein, wenn sie für ständige Subskriptionen verwendet wird, da ein Warteschlangenmanager das Senden von Veröffentlichungen an einen Subskribenten in diesem Fall auch dann fortsetzt, wenn die Anwendung des Subskribenten nicht ausgeführt wird. Dies kann dazu führen, dass sich in der Warteschlange des Subskribenten ein Rückstau an Nachrichten bildet. Dieser kann jedoch vermieden werden, wenn der Subskribent sich mit der Option MQSO_PUBLICATIONS_ON_REQUEST registriert. Alternativ können Sie temporäre Subskriptionen verwenden, falls diese für Ihre Anwendung geeignet sind, um eine Ansammlung unerwünschter Nachrichten zu vermeiden.

Falls eine Subskription permanent ist und eine Veröffentlichungsanwendung ständige Veröffentlichungen verwendet, kann die Anwendung des Subskribenten den Aufruf MQSUBRQ verwenden, um ihre Statusinformationen nach einem Neustart zu aktualisieren. In diesem Fall muss der Subskribent seinen Status in regelmäßigen Abständen mit dem Aufruf MQSUBRQ aktualisieren.

Bei Verwendung dieser Option werden nach einem MQSUB-Aufruf keine Veröffentlichungen versandt. Eine ständige Subskription, die nach einer Verbindungstrennung wiederaufgenommen wurde, verwendet die Option MQSO_PUBLICATIONS_ON_REQUEST, falls die ursprüngliche Subskription für die Verwendung dieser Option konfiguriert wurde.

Nur neue Veröffentlichungen, MQSO_NEW_PUBLICATIONS_ONLY

Falls eine ständige Veröffentlichung für ein Thema vorhanden ist, erhalten alle Subskribenten, die eine Subskription erstellen, nachdem die Veröffentlichung veröffentlicht wurde, eine Kopie von dieser. Falls ein Subskribent keine Veröffentlichungen erhalten möchte, die veröffentlicht wurden, bevor die Subskription erstellt wurde, kann er die Subskriptions-Option MQSO_NEW_PUBLICATIONS_ONLY verwenden.

Gruppierung von Subskriptionen

Sie können Subskriptionen gruppieren, wenn Sie eine neue Warteschlange für den Empfang von Veröffentlichungen eingerichtet haben und über eine Reihe von sich überschneidenden Subskriptionen verfügen, die Veröffentlichungen in die gleiche Warteschlange stellen. Diese Situation ist mit dem Beispiel in Überschneidung von Subskriptionen vergleichbar.

Sie können dem Empfang von doppelten Veröffentlichungen vermeiden, indem Sie beim Subskribieren eines Themas die Option MQSO_GROUP_SUB festlegen. Wenn nun mehrere Subskriptionen in der Gruppe mit dem Thema einer Veröffentlichung übereinstimmen, ist nur eine Subskription dafür zuständig, die Veröffentlichung in die Warteschlange einzureihen. Die anderen Subskriptionen mit dem gleichen Veröffentlichungsthema werden ignoriert.

Die Subskription, durch die die Veröffentlichung in die Warteschlange eingereiht wird, wird dadurch ausgewählt, dass sie die längste übereinstimmende Themenzeichenfolge vor dem Auftreten von Platzhalterzeichen enthält. Es handelt sich also um die Subskription mit der größten Übereinstimmung. Die zugehörigen Eigenschaften werden an die Veröffentlichung weitergegeben, einschließlich der Information, ob die Eigenschaft MQSO_NOT_OWN_PUBS enthalten ist. Wenn diese Eigenschaft festgelegt ist, wird keine Veröffentlichung an die Warteschlange übergeben, selbst wenn die Eigenschaft MQSO_NOT_OWN_PUBS in anderen übereinstimmenden Subskriptionen nicht festgelegt ist.

Sie können nicht alle Subskriptionen in eine einzelne Gruppe stellen, um doppelte Veröffentlichungen auszuschließen. Gruppierte Subskriptionen müssen die folgenden Bedingungen erfüllen:

1. Keine der Subskriptionen ist verwaltet.
2. Eine Gruppe von Subskriptionen übergibt Veröffentlichungen an die gleiche Warteschlange.
3. Jede Subskription muss über die gleiche Subskriptionsebene verfügen.
4. Die Veröffentlichungsnachricht für jede Subskription in der Gruppe verfügt über die gleiche Korrelations-ID.

   Um sicherzustellen, dass sich aus jeder Subskription eine Veröffentlichungsnachricht mit der gleichen Korrelations-ID ergibt, legen Sie die Option MQSO_SET_CORREL_ID fest, um Ihre eigene Korrelations-ID in der Veröffentlichung zu erstellen, und legen Sie in jeder Subskription im Feld **SubCorrelId** den gleichen Wert fest. Legen Sie für **SubCorrelId** nicht den Wert MQCI_NONE fest.

Weitere Informationen finden Sie im Abschnitt ../refdev/q100080_.dita#q100080_/mqso_group_sub.

Objektattribute abfragen und einstellen

Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

Sie bestimmen, wie ein Warteschlangenmanager ein Objekt verarbeitet. Die Attribute der einzelnen Arten von IBM MQ-Objekten werden in Objektattribute näher beschrieben.

Einige Attribute werden bei der Definition eines Objekts festgelegt und können nur mit den IBM MQ-Befehlen geändert werden. Ein Beispiel hierfür ist die Standardpriorität für die in Warteschlangen eingereihten Nachrichten. Andere Attribute werden durch die Funktionsweise des Warteschlangenmanagers beeinflusst und können sich im Laufe der Zeit ändern. Ein Beispiel hierfür ist die aktuelle Tiefe einer Warteschlange.

Mit dem MQINQ-Aufruf können Sie die aktuellen Werte der meisten Attribute abrufen. Mit dem MQI-Aufruf MQSET können Sie auch einige Warteschlangenattribute ändern. Die Attribute eines anderen Objekttyps lassen sich mit den MQI-Aufrufen allerdings nicht ändern. Hierzu müssten Sie eine der folgenden Ressourcen verwenden:

- ALW Die MQSC-Funktion (siehe MQSC-Befehle)
- IBM i Die CHGMQMx CL-Befehle (siehe CL-Befehlsreferenz für IBM i) oder die MQSC-Funktion.
- z/OS Die ALTER-Bedienerbefehle oder die DEFINE-Befehle mit der Option REPLACE (siehe MQSC-Befehle)

Anmerkung: Die Namen der Objektattribute werden in dieser Dokumentation in der Form dargestellt, in der sie in den MQINQ- und MQSET-Aufrufen verwendet werden. Wenn Sie die IBM MQ-Befehle zum Einstellen, Ändern oder Anzeigen der Attribute verwenden, müssen Sie die Attribute durch die in den Befehlsbeschreibungen (siehe Links) angegebenen Schlüsselwörter kennzeichnen.

Sowohl in den MQINQ- als auch in den MQSET-Aufrufen verwenden Sie Selektorfeldgruppen zur Kennzeichnung der Attribute, die Sie abfragen oder einstellen möchten. Für jedes Attribut, das Sie aufrufen oder ändern können, gibt es einen Selektor. Der Selektorname hat ein durch den Attributtyp bestimmtes Präfix:

Tabelle 127. Präfixe für Selektornamen

Präfix	**Beschreibung**
MQCA_	Diese Selektoren kennzeichnen Attribute, die Zeichendaten enthalten (z. B. den Warteschlangennamen).
MQIA_	Diese Selektoren verweisen auf Attribute, die numerische Werte (z. B. *CurrentQueueDepth*, d. h. die Anzahl der Nachrichten in einer Warteschlange) oder konstante Werte (z. B. *SyncPoint*, eine Konstante, die festlegt, ob der Warteschlangenmanager Synchronisationspunkte unterstützt) enthalten.

Vor Verwendung eines MQINQ- oder MQSET-Aufrufs muss Ihre Anwendung mit dem Warteschlangenmanager verbunden werden, und Sie müssen das Objekt, dessen Attribute abgefragt oder eingestellt werden sollen, mit dem MQOPEN-Aufruf öffnen. Eine Beschreibung dieser Operationen finden Sie in den Abschnitten „Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773 und „Objekte öffnen und schließen” auf Seite 780.

Unter den folgenden Links erhalten Sie weitere Informationen zum Abfragen und Einstellen von Objektattributen:

- „Abfragen der Attribute eines Objekts” auf Seite 899
- „Fälle, in denen der MQINQ-Aufruf fehlschlägt” auf Seite 900
- „Warteschlangenattribute einstellen” auf Seite 900

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773
Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Objekte öffnen und schließen” auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS” auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS” auf Seite 73
This information helps you to write IMS applications using IBM MQ.

Abfragen der Attribute eines Objekts
Mit dem Aufruf MQINQ können Sie die Attribute jedes IBM MQ-Objekttyps abfragen.

Als Eingabe für diesen Aufruf müssen Sie Folgendes angeben:

- Eine Verbindungskennung.
- Eine Objektkennung;
- Die Anzahl der Selektoren
- Eine Feldgruppe mit Attributselektoren, wobei jeder Selektor das Format MQCA_* oder MQIA_* aufweisen muss. Jeder Selektor stellt ein Attribut mit einem Wert dar, den Sie abfragen möchten. Außerdem muss jeder Selektor für den Objekttyp gültig sein, den die Objektkennung darstellt. Die Selektoren können in beliebiger Reihenfolge angegeben werden.
- Die Anzahl der Ganzzahlattribute, die Sie abfragen möchten. Geben Sie null an, wenn Sie keine Ganzzahlattribute abfragen.
- Die Länge des Zeichenattributpuffers in *CharAttrLength*. Dies muss mindestens die Summe der Längen sein, die für die einzelnen Zeichenfolgen der Zeichenattribute erforderlich sind. Geben Sie null an, wenn Sie keine Zeichenattribute abfragen.

Die Ausgabe von MQINQ enthält folgende Komponenten:

- Einen Satz mit den Werten der Ganzzahlattribute, kopiert in die Feldgruppe. Die Anzahl der Werte wird von *IntAttrCount* festgelegt. Wenn *IntAttrCount* oder *SelectorCount* null ist, wird dieser Parameter nicht verwendet.
- Den Puffer, in dem Zeichenattribute zurückgegeben werden. Die Länge des Puffers wird durch den Parameter **CharAttrLength** angegeben. Wenn *CharAttrLength* oder *SelectorCount* null ist, wird dieser Parameter nicht verwendet.
- Einen Beendigungscode. Wenn dieser eine Warnung enthält, wurde der Aufruf nur teilweise ausgeführt. Sie sollten in diesem Fall den Ursachencode überprüfen.
- Einen Ursachencode. Es gibt drei Bedingungen, unter denen der Aufruf nur teilweise ausgeführt wird:
  - Der Selektor passt nicht zum Warteschlangentyp
  - Für Ganzzahlattribute ist nicht ausreichend Speicher zugewiesen
  - Für Zeichenattribute ist nicht ausreichend Speicher zugewiesen

  Wenn mehrere dieser Bedingungen zutreffen, wird die zuerst vorgefundene Bedingung zurückgegeben.

Wenn Sie eine Warteschlange für die Ausgabe oder Abfrage öffnen und der Name der Warteschlange löst sich auf eine nicht lokale Clusterwarteschlange auf, können Sie nur den Namen der Warteschlange, den Warteschlangentyp und einige allgemeine Attribute abfragen. Wenn MQOO_BIND_ON_OPEN verwendet wurde, handelt es sich bei den Werten der allgemeinen Attribute um die Werte der ausgewählten Warteschlange. Dagegen handelt es sich bei diesen Werten um die Werte einer beliebigen der möglichen Clusterwarteschlangen, wenn MQOO_BIND_NOT_FIXED oder MQOO_BIND_ON_GROUP verwendet

wurde oder wenn MQOO_BIND_AS_Q_DEF verwendet wurde und das Warteschlangenattribut **DefBind** auf MQBND_BIND_NOT_FIXED gesetzt ist. Weitere Informationen hierzu finden Sie in den Abschnitten „MQOPEN und Cluster" auf Seite 934 und MQOPEN.

Anmerkung: Bei den vom Aufruf zurückgegebenen Werten handelt es sich um eine Momentaufnahme der ausgewählten Attribute. Die Attributwerte können sich ändern, noch bevor Ihr Programm auf die zurückgegebenen Werte einwirkt.

Eine genaue Beschreibung des MQINQ-Aufrufs finden Sie im Abschnitt MQINQ.

Fälle, in denen der MQINQ-Aufruf fehlschlägt

Wenn Sie beim Öffnen einen Aliasnamen eingeben, um dessen Attribute abzufragen, werden die Attribute der Aliaswarteschlange zurückgegeben (das IBM MQ-Objekt, das für den Zugriff auf eine andere Warteschlange verwendet wurde), nicht diejenigen der Basiswarteschlange.

Allerdings wird auch die Definition der Basiswarteschlange, auf die sich der Aliasname auflöst, vom Warteschlangenmanager geöffnet, und falls ein anderes Programm die Verwendung der Basiswarteschlange zwischen der Ausgabe Ihres MQOPEN- und Ihres MQINQ-Aufrufs ändert, schlägt der MQINQ-Aufruf mit dem Ursachencode MQRC_OBJECT_CHANGED fehl. Außerdem schlägt der Aufruf fehl, wenn zwischenzeitlich die Attribute des Aliaswarteschlangenobjekts geändert werden.

Ähnliches passiert, wenn Sie eine ferne Warteschlange öffnen, um deren Attribute abzufragen. Sie erhalten in diesem Fall nur die Attribute der lokalen Definition der fernen Warteschlange zurück.

Wenn Sie einen oder mehrere Selektoren angeben, die für die abgefragten Warteschlangenattribute ungültig sind, wird der MQINQ-Aufruf mit einer Warnung beendet. Als Ausgabe erhalten Sie Folgendes zurück:

- Bei ganzzahligen Attributen werden die zugehörigen Elemente von *IntAttrs* auf MQIAV_NOT_APPLICABLE gesetzt.
- Bei Zeichenattributen werden die zugehörigen Teile der Zeichenfolge *CharAttrs* durch Sternchen ersetzt.

Wenn Sie einen oder mehrere Selektoren angeben, die für die abgefragten Objektattribute ungültig sind, schlägt der MQINQ-Aufruf mit dem Ursachencode MQRC_SELECTOR_ERROR fehl.

Die Attribute einer Modellwarteschlange können Sie mit dem MQINQ-Aufruf nicht abrufen. Hierzu müssen Sie die MQSC-Funktion oder die Befehle Ihrer Plattform verwenden.

Warteschlangenattribute einstellen

In diesem Abschnitt erfahren Sie, wie Warteschlangenattribute mithilfe des MQSET-Aufrufs eingestellt werden.

Sie können nur die folgenden Warteschlangenattribute mithilfe des MQSET-Aufrufs einstellen:

- *InhibitGet* (aber nicht für ferne Warteschlangen)
- Multi *DistList*
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

Der MQSET-Aufruf hat dieselben Parameter wie der MQINQ-Aufruf. Bei MQSET sind jedoch alle Parameter Eingabeparameter außer dem Beendigungscode und dem Ursachencode. Es gibt keine Bedingungen, unter denen der Aufruf nur teilweise ausgeführt wird.

Anmerkung: Mit MQI können Sie nur die Attribute von lokal definierten IBM MQ-Warteschlangen einstellen.

Weitere Informationen zum MQSET-Aufruf finden Sie im Abschnitt MQSET.

Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen

In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

In diesem Abschnitt werden folgende Begriffe verwendet:

- Festschreiben
- Zurücksetzen
- Synchronisationspunktkoordinierung
- Synchronisationspunkt
- Arbeitseinheit
- Einphasige Festschreibung
- Zweiphasiges Commit

Wenn Sie mit diesen Fachtermini der Transaktionsverarbeitung vertraut sind, können Sie mit Abschnitt „Überlegungen zu Synchronisationspunkten in IBM MQ-Anwendungen” auf Seite 903 fortfahren.

Commit und Backout

Wenn ein Programm eine Nachricht innerhalb einer Arbeitseinheit in eine Warteschlange einreiht, wird diese Nachricht erst dann für die anderen Programme sichtbar, wenn das einreihende Programm die Arbeitseinheit mit einem Commit festschreibt. Zur Gewährleistung der Datenintegrität müssen für das Commit einer Arbeitseinheit alle Aktualisierungen erfolgreich durchgeführt worden sein. Falls das Programm dabei einen Fehler feststellt, der die Festschreibung der Put-Operation in Frage stellt, kann es die Arbeitseinheit mit einem Backout zurücksetzen. Bei einem Backout stellt IBM MQ den alten Zustand der Warteschlange wieder her, indem es die Nachrichten entfernt, die im Zuge der betreffenden Arbeitseinheit eingereiht wurden. Wie ein Programm ein Commit bzw. ein Backout durchführt, richtet sich nach der Umgebung, in der das Programm ausgeführt wird.

Ebenso verbleibt eine Nachricht, die ein Programm innerhalb einer Arbeitseinheit aus einer Warteschlange abruft, in der Warteschlange, bis das Programm die Arbeitseinheit mit einem Commit festschreibt. Allerdings kann eine bereits abgerufene Nachricht nicht mehr von anderen Programmen abgerufen werden, auch wenn die Arbeitseinheit noch nicht festgeschrieben ist. Erst wenn das Programm die Arbeitseinheit festgeschrieben hat, wird die Nachricht permanent aus der Warteschlange entfernt. Falls das Programm die Arbeitseinheit mit einem Backout zurücksetzt, stellt IBM MQ die Warteschlange wieder her, so dass die Nachrichten auch von anderen Programmen abgerufen werden können.

Synchronisationspunktkoordinierung, Synchronisationspunkte, Arbeitseinheit

Synchronisationspunktkoordinierung ist der Prozess, bei dem Arbeitseinheiten unter Gewährleistung der Datenintegrität entweder festgeschrieben oder zurückgesetzt werden.

Ob die vorgenommenen Änderungen festgeschrieben oder zurückgesetzt werden, wird im einfachsten Fall am Ende einer Transaktion entschieden. Manchmal ist es für eine Anwendung jedoch günstiger, Datenänderungen an anderen logischen Stellen einer Transaktion zu synchronisieren. Diese logischen Stellen werden als *Synchronisationspunkte* (*Syncpoints*) bezeichnet und das Verarbeitungsintervall der Aktualisierungen zwischen zwei Synchronisationspunkten als *Arbeitseinheit*. Eine einzelne Arbeitseinheit kann mehrere MQGET- und MQPUT-Aufrufe umfassen.

Die maximale Anzahl von Nachrichten innerhalb einer Arbeitseinheit kann mit dem Attribut MAXUMSGS des Befehls ALTER QMGR gesteuert werden.

Einphasige Festschreibung

Ein *einphasiges Commit* ist ein Prozess, im Zuge dessen ein Programm Änderungen an einer Warteschlange festschreiben kann, ohne die Änderungen mit anderen Ressourcenmanagern koordinieren zu müssen.

Zweiphasiges Commit

Ein *zweiphasiges Commit* ist ein Prozess, in dem Änderungen, die ein Programm an IBM MQ-Warteschlangen vorgenommen hat, mit Änderungen an anderen Ressourcen (z. B. Datenbanken unter der Kontrolle von Db2) koordiniert werden können. In einem solchen Prozess werden die Änderungen an allen Ressourcen gemeinsam festgeschrieben oder zurückgesetzt.

Für einen besseren Überblick über die Backouts in Arbeitseinheiten stellt IBM MQ das Attribut **BackoutCount** bereit. Dieses Attribut wird bei jeder Zurücksetzung einer Nachricht innerhalb einer Arbeitseinheit um eins erhöht. Wenn eine Nachricht mehrmals zu einer abnormalen Beendigung der Arbeitseinheit führt, überschreitet der Wert von *BackoutCount* irgendwann den Grenzwert *BackoutThreshold*. Dieser Wert wird bei der Definition der Warteschlange festgelegt. In einem solchen Fall kann die Anwendung die Nachricht aus der Arbeitseinheit entfernen und in eine andere Warteschlange einreihen, wie im Feld *BackoutRequeueQName* festgelegt. Nach dem Verschieben der problematischen Nachricht kann die Arbeitseinheit festgeschrieben werden.

Unter den folgenden Links erhalten Sie weitere Informationen zum Festschreiben (Commit) und Zurücksetzen (Backout) von Arbeitseinheiten:

- „Überlegungen zu Synchronisationspunkten in IBM MQ-Anwendungen” auf Seite 903
- z/OS „Syncpoints in IBM MQ for z/OS applications” auf Seite 904
- IBM i „Synchronisationspunkte in CICS for IBM i-Anwendungen” auf Seite 906
- „Synchronisationspunkte in IBM MQ for Multiplatforms” auf Seite 906
- IBM i „Schnittstellen mit dem externen Synchronisationspunktmanager von IBM i” auf Seite 911

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773
Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Objekte öffnen und schließen” auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS” auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS” auf Seite 73
This information helps you to write IMS applications using IBM MQ.

Überlegungen zu Synchronisationspunkten in IBM MQ-Anwendungen

In diesem Abschnitt erhalten Sie Informationen zur Verwendung von Synchronisationspunkten in IBM MQ-Anwendungen.

Das zweiphasige Commit wird in den folgenden Umgebungen unterstützt:

- Multi IBM MQ for Multiplatforms
- z/OS CICS Transaction Server für z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS z/OS-Stapel mit RRS
- Andere externe Koordinatoren, die die X/Open XA-Schnittstelle verwenden

Das einphasige Commit wird in den folgenden Umgebungen unterstützt:

- Multi IBM MQ for Multiplatforms
- z/OS z/OS-Stapel

Ausführliche Informationen zu externen Schnittstellen finden Sie im Abschnitt „Schnittstellen zu externen Synchronisationspunktmanagern auf Multiplatforms" auf Seite 910 und in der von der The Open Group herausgegebenen XA-Dokumentation *CAE Specification Distributed Transaction Processing: The XA Specification*. Transaktionsmanager (wie CICS, IMS, Encina und Tuxedo) können am zweiphasigen, mit anderen wiederherstellbaren Ressourcen koordinierten Commit mitwirken. Dies bedeutet, dass die Warteschlangenfunktionen von IBM MQ im Rahmen von Arbeitseinheiten von einem Transaktionsmanager verwaltet werden können.

Die mit IBM MQ bereitgestellten Beispielprogramme zeigen, wie IBM MQ XA-konforme Datenbanken koordiniert. Weitere Informationen zu diesen Beispielprogrammen finden Sie im Abschnitt „Prozedurale IBM MQ-Beispielprogramme verwenden" auf Seite 1111.

In Ihrer IBM MQ-Anwendung können Sie in jedem Put- oder Get-Aufruf angeben, ob der Aufruf unter Synchronisationspunktsteuerung stehen soll. Soll eine Put-Operation unter Synchronisationspunktsteuerung ausgeführt werden, verwenden Sie beim Aufrufen von MQPUT im Feld *Options* der MQPMO-Struktur den Wert MQPMO_SYNCPOINT. Für eine Get-Operation unter Synchronisationspunktsteuerung verwenden Sie im Feld *Options* der MQGMO-Struktur den Wert MQGMO_SYNCPOINT. Wenn Sie keine Option explizit auswählen, ist die Standardaktion plattformabhängig:

- Multi Der Standardwert für die Synchronisationspunktsteuerung ist NO.
- z/OS Der Standardwert für die Synchronisationspunktsteuerung ist YES.

Wenn ein MQPUT1-Aufruf mit MQPMO_SYNCPOINT ausgegeben wird, ändert sich das Standardverhalten, sodass die Put-Operation asynchron beendet wird. Dies kann eine Änderung des Verhaltens einiger Anwendungen verursachen, die bestimmte Felder in den MQOD- und MQMD-Strukturen benötigen, die zurückgegeben werden, aber jetzt undefinierte Werte enthalten. Eine Anwendung kann MQPMO_SYNC_RESPONSE angeben, um sicherzustellen, dass die Put-Operation synchron ausgeführt wird und dass alle zutreffenden Feldwerte abgeschlossen werden.

Wenn Ihre Anwendung auf einen MQPUT- oder MQGET-Aufruf unter Synchronisationspunktsteuerung den Ursachencode MQRC_BACKED_OUT erhält, sollte die Anwendung die aktuelle Transaktion normalerweise mit MQBACK zurücksetzen und anschließend erneut ausführen. Wenn die Anwendung MQRC_BACKED_OUT als Antwort auf einen MQCMIT- oder MQDISC-Aufruf erhält, muss sie MQBACK nicht aufrufen.

Bei jeder Zurücksetzung eines MQGET-Aufrufs wird das Feld *BackoutCount* der MQMD-Struktur der betroffenen Nachricht um eins erhöht. Ein hoher *BackoutCount*-Wert ist ein Hinweis darauf, dass die Nachricht mehrfach zurückgesetzt wurde. Das kann ein Hinweis auf ein Problem mit der Nachricht sein, das Sie untersuchen sollten. Details zu *BackoutCount*finden Sie unter BackoutCount .

Wenn ein Programm den MQDISC-Aufruf ausgibt, während nicht festgeschriebene Anforderungen vorliegen, tritt ein impliziter Synchronisationspunkt auf (außer unter z/OS Batch mit RRS). Wenn das Programm fehlerhaft endet, kommt es zu einem impliziten Backout.

z/OS Unter z/OS erfolgt ein impliziter Synchronisationspunkt auch, wenn das Programm ohne vorherigen Aufruf von MQDISC normal endet. Das Programm wird als normal beendetes Programm eingestuft, wenn der Übertragungssteuerblock, der mit MQ verbunden ist, normal beendet wird. Unter z/OS UNIX System Services and Language Environment (LE) wird bei Abbrüchen oder Signalen die standardmäßige Behandlung von Ausnahmebedingungen aufgerufen. Die LE-Bedingungshandler verarbeiten die Fehlerbedingung und der Übertragungssteuerblock wird normal beendet. Unter diesen Bedingungen schreibt MQ die Arbeitseinheit fest. Weitere Informationen enthält der Abschnitt Introduction to Language Environment Condition Handling.

z/OS In IBM MQ for z/OS-Programmen können Sie mit der Option MQGMO_MARK_SKIP_BACKOUT festlegen, dass bestimmte Nachrichten bei einem Backout nicht zurückgesetzt werden (um zum Beispiel eine *MQGET-error-backout*-Schleife zu vermeiden). Weitere Informationen zu dieser Option finden Sie im Abschnitt „Backout überspringen” auf Seite 840.

Änderungen an den Warteschlangenattributen (entweder durch den MQSET-Aufruf oder durch Befehle) werden nicht durch Festschreiben oder Zurücksetzen von Arbeitseinheiten beeinträchtigt.

z/OS *Syncpoints in IBM MQ for z/OS applications*

This topic explains how to use syncpoints in transaction manager (CICS and IMS) and batch applications.

z/OS *Syncpoints in CICS Transaction Server for z/OS applications*

In a CICS application you establish a syncpoint by using the EXEC CICS SYNCPOINT command.

To back out all changes to the previous syncpoint, you can use the EXEC CICS SYNCPOINT ROLLBACK command. For more information, see the *CICS Application Programming Reference*.

If other recoverable resources are involved in the unit of work, the queue manager (in conjunction with the CICS syncpoint manager) participates in a two-phase commit protocol; otherwise, the queue manager performs a single-phase commit process.

If a CICS application issues the MQDISC call, no implicit syncpoint is taken. If the application closes down normally, any open queues are closed and an implicit commit occurs. If the application closes down abnormally, any open queues are closed and an implicit backout occurs.

z/OS *Syncpoints in IMS applications*

In an IMS application, establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint).

To back out all changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see the IMS documentation.

The queue manager (in conjunction with the IMS syncpoint manager) participates in a two-phase commit protocol if other recoverable resources are also involved in the unit of work.

All open handles are closed by the IMS adapter at a syncpoint (except in a batch or non-message driven BMP environment). This is because a different user could initiate the next unit of work and IBM MQ security checking is performed when the MQCONN, MQCONNX, and MQOPEN calls are made, not when the MQPUT or MQGET calls are made.

However, in a Wait-for-Input (WFI) or pseudo Wait-for-Input (PWFI) environment IMS does not notify IBM MQ to close the handles until either the next message arrives or a QC status code is returned to the application. If the application is waiting in the IMS region and any of these handles belong to triggered queues, triggering will not occur because the queues are open. For this reason applications running in a WFI or PWFI environment should explicitly MQCLOSE the queue handles before doing the GU to the IOPCB for the next message.

If an IMS application (either a BMP or an MPP) issues the MQDISC call, open queues are closed but no implicit syncpoint is taken. If the application closes down normally, any open queues are closed and an implicit commit occurs. If the application closes down abnormally, any open queues are closed and an implicit backout occurs.

z/OS *Syncpoints in z/OS batch applications*

For batch applications, you can use the IBM MQ syncpoint management calls: MQCMIT and MQBACK. For compatibility with earlier versions, CSQBCMT and CSQBBAK are available as synonyms.

Note: If you need to commit or back out updates to resources managed by different resource managers, such as IBM MQ and Db2, within a single unit of work you can use RRS. For further information see "Transaction management and recoverable resource manager services" on page 905.

Committing changes using the MQCMIT call

As input, you must supply the connection handle (*Hconn*) that is returned by the MQCONN or MQCONNX call.

The output from MQCMIT is a completion code and a reason code. The call completes with a warning if the syncpoint was completed but the queue manager backed out the put and get operations since the previous syncpoint.

Successful completion of the MQCMIT call indicates to the queue manager that the application has reached a syncpoint and that all put and get operations made since the previous syncpoint have been made permanent.

Not all failure responses mean that the MQCMIT did not complete. For example, the application can receive MQRC_CONNECTION_BROKEN.

There is a description of the MQCMIT call in MQCMIT.

Backing out changes using the MQBACK call

As input, you must supply a connection handle (*Hconn*). Use the handle that is returned by the MQCONN or MQCONNX call.

The output from MQBACK is a completion code and a reason code.

The output indicates to the queue manager that the application has reached a syncpoint and that all gets and puts that have been made since the last syncpoint have been backed out.

There is a description of the MQBACK call in MQBACK.

Transaction management and recoverable resource manager services

Transaction management and recoverable resource manager services (RRS) is a z/OS facility to provide two-phase syncpoint support across participating resource managers.

An application can update recoverable resources managed by various z/OS resource managers such as IBM MQ and Db2, and then commit or back out these updates as a single unit of work. RRS provides the necessary unit-of-work status logging during normal execution, coordinates the syncpoint processing, and provides appropriate unit-of-work status information during subsystem restart.

IBM MQ for z/OS RRS participant support enables IBM MQ applications in the batch, TSO, and Db2 stored procedure environments to update both IBM MQ and non-IBM MQ resources (for example, Db2) within a single logical unit of work. For information about RRS participant support, see *z/OS MVS Programming: Resource Recovery*.

Your IBM MQ application can use either MQCMIT and MQBACK or the equivalent RRS calls, SRRCMIT and SRRBACK. See "The RRS batch adapter" on page 941 for more information.

RRS availability

If RRS is not active on your z/OS system, any IBM MQ call issued from a program linked with either RRS stub (CSQBRSTB or CSQBRRSI) returns MQRC_ENVIRONMENT_ERROR.

Db2 stored procedures

If you use Db2 stored procedures with RRS, be aware of the following:

- Db2 stored procedures that use RRS must be managed by workload manager (WLM-managed).
- If a Db2-managed stored procedure contains IBM MQ calls, and it is linked with either RRS stub (CSQBRSTB or CSQBRRSI), the MQCONN or MQCONNX call returns MQRC_ENVIRONMENT_ERROR.
- If a WLM-managed stored procedure contains IBM MQ calls, and is linked with a non-RRS stub, the MQCONN or MQCONNX call returns MQRC_ENVIRONMENT_ERROR, unless it is the first IBM MQ call executed since the stored procedure address space started.
- If your Db2 stored procedure contains IBM MQ calls and is linked with a non-RRS stub, IBM MQ resources updated in that stored procedure are not committed until the stored procedure address space ends, or until a subsequent stored procedure does an MQCMIT (using an IBM MQ Batch/TSO stub).
- Multiple copies of the same stored procedure can execute concurrently in the same address space. Ensure that your program is coded in a reentrant manner if you want Db2 to use a single copy of your stored procedure. Otherwise you might receive MQRC_HCONN_ERROR on any IBM MQ call in your program.
- Do not code MQCMIT or MQBACK in a WLM-managed Db2 stored procedure.
- Design all programs to run in Language Environment (LE).

IBM i *Synchronisationspunkte in CICS for IBM i-Anwendungen*

IBM MQ for IBM i kann in CICS for IBM i-Arbeitseinheiten mitwirken. Mit der Message Queue Interface (MQI - Schnittstelle für Nachrichtenwarteschlangen) innerhalb einer CICS for IBM i-Anwendung können Sie Nachrichten in die aktuelle Arbeitseinheit einreihen und daraus abrufen.

Mit dem Befehl EXEC CICS SYNCPOINT können Sie einen Synchronisationspunkt einrichten, der auch IBM MQ for IBM i-Operationen einschließt. Um alle Änderungen bis zum vorherigen Synchronisationspunkt zurückzusetzen, können Sie den Befehl EXEC CICS SYNCPOINT ROLLBACK verwenden.

Wenn Sie MQPUT, MQPUT1 oder MQGET mit der Option MQPMO_SYNCPOINT oder MQGMO_SYNCPOINT in einer CICS for IBM i-Anwendung verwenden, können Sie CICS for IBM i erst abmelden, wenn IBM MQ for IBM i seine Registrierung als API-Commitressource entfernt hat. Vor der Trennung vom Warteschlangenmanager müssen Sie alle ausstehenden Put- oder Get-Operationen festschreiben oder zurücksetzen. Nur dann können Sie CICS for IBM i abmelden.

Multi *Synchronisationspunkte in IBM MQ for Multiplatforms*

Die Synchronisationspunktunterstützung gilt für lokale und globale Arbeitseinheiten.

Eine *lokale* Arbeitseinheit ist eine Arbeitseinheit, in der nur die Ressourcen des IBM MQ-Warteschlangenmanagers aktualisiert werden. Hier wird die Synchronisationspunktkoordinierung durch den Warteschlangenmanager mittels einer einphasigen Commitprozedur bereitgestellt.

Eine *globale* Arbeitseinheit ist eine Arbeitseinheit, in der auch die Ressourcen anderer Ressourcenmanager, beispielsweise Datenbanken, aktualisiert werden. IBM MQ kann solche Arbeitseinheiten selbst koordinieren. Sie können auch von einem externen Commit-Controller koordiniert werden. For example:

- Ein anderer Transaktionsmanager
- IBM i Commit-Controller von IBM i

Zur Gewährleistung der Datenintegrität sollten Sie eine zweiphasige Commitprozedur verwenden. Zweiphasiges Commit kann von XA-konformen Transaktionsmanagern und Datenbanken bereitgestellt werden. For example:

- TXSeries

- Universal Database
- IBM i Commit-Controller von IBM i

ALW IBM MQ-Produkte können globale Arbeitseinheiten mit einem zweiphasigen Commitprozess koordinieren.

IBM i IBM MQ for IBM i kann als Ressourcenmanager für globale Arbeitseinheiten in einer WebSphere Application Server-Umgebung fungieren, jedoch nicht als Transaktionsmanager.

Impliziter Synchronisationspunkt

Beim Einreihen persistenter Nachrichten ist IBM MQ für das Einreihen von persistenten Nachrichten unter Synchronisationspunkt optimiert. Mehrere Anwendungen, die persistente Nachrichten in dieselbe Warteschlange einreihen, zeigen eine bessere Leistung, wenn sie mit Synchronisationspunkten arbeiten. Grund dafür ist, dass es weniger Konflikte für die Warteschlange gibt, wenn zum Einreihen persistenter Nachrichten ein Synchronisationspunkt verwendet wird.

ImplSyncOpenOutput fügt einen impliziten Synchronisationspunkt hinzu, wenn Anwendungen persistente Nachrichten außerhalb des Synchronisationspunkts einreihen. Dies führt zu einer Leistungsverbesserung, ohne dass Anwendungen etwas von dem impliziten Synchronisationspunkt mitbekommen.

Ein impliziter Synchronisationspunkt sorgt nur dann für eine Leistungssteigerung, wenn mehrere Anwendungen Nachrichten in die Warteschlange einreihen, denn dadurch wird die Konkurrenz für die Warteschlange reduziert. **ImplSyncOpenOutput** gibt also die Mindestanzahl von Anwendungen an, die eine Warteschlange für Ausgaben geöffnet haben, bevor ein impliziter Synchronisationspunkt hinzugefügt wird. Der Standardwert ist 2. Er bedeutet Folgendes: Wenn Sie **ImplSyncOpenOutput** nicht angeben, wird ein impliziter Synchronisationspunkt nur dann hinzugefügt, wenn mehrere Anwendungen Nachrichten in die Warteschlange einreihen.

Weitere Informationen finden Sie im Abschnitt Parameter optimieren.

Lokale Arbeitseinheiten unter Multiplatforms
Arbeitseinheiten, an denen nur der Warteschlangenmanager beteiligt ist, werden als *lokale* Arbeitseinheiten bezeichnet. Hier wird die Synchronisationspunktkoordinierung durch den Warteschlangenmanager selbst (interne Koordination) mittels eines einphasigen Commitprozesses bereitgestellt.

Zum Starten einer lokalen Arbeitseinheit gibt die Anwendung einen MQGET-, MQPUT- oder MQPUT1-Aufruf mit der zutreffenden Synchronisationspunktoption aus. Die Arbeitseinheit wird mit MQCMIT festgeschrieben oder mit MQBACK zurückgesetzt. Ebenso endet die Arbeitseinheit, wenn die Verbindung zwischen der Anwendung und dem Warteschlangenmanager absichtlich oder unabsichtlich getrennt wird.

Wenn eine Anwendung die Verbindung zum Warteschlangenmanager mit einem MQDISC trennt, obwohl noch eine globale, von IBM MQ koordinierte Arbeitseinheit aktiv ist, wird eine Festschreibung der Arbeitseinheit versucht. Wird die Anwendung hingegen ohne Verbindungstrennung beendet, so wird die Arbeitseinheit zurückgesetzt, da davon ausgegangen wird, dass die Anwendung fälschlicherweise beendet wurde.

ALW *Globale Arbeitseinheiten unter AIX, Linux, and Windows*
Verwenden Sie globale Arbeitseinheiten, wenn Sie auch Änderungen an Ressourcen einbeziehen müssen, die zu anderen Ressourcenmanagern gehören. Die Koordination kann intern oder außerhalb des Warteschlangenmanagers erfolgen.

Interne Synchronisationspunktkoordination

Die Koordination globaler Arbeitseinheiten durch den Warteschlangenmanager wird in einer IBM MQ MQI client -Umgebung nicht unterstützt.

Hier übernimmt IBM MQ die Koordination. Zum Starten einer globalen Arbeitseinheit gibt die Anwendung einen MQBEGIN-Aufruf aus.

Als Eingabe für den MQBEGIN-Aufruf ist die Verbindungskennung (*Hconn*) erforderlich, die durch den MQCONN- oder MQCONNX-Aufruf zurückgegeben wird. Diese Kennung stellt die Verbindung zum IBM MQ-Warteschlangenmanager dar.

Die Anwendung gibt einen MQGET-, MQPUT- oder MQPUT1-Aufruf mit der zutreffenden Synchronisationspunktoption aus. Das bedeutet, dass Sie MQBEGIN verwenden können, um eine globale Arbeitseinheit auszulösen, die lokale Ressourcen, Ressourcen, die zu anderen Ressourcenmanagern gehören, oder beides aktualisieren. Updates bei Ressourcen, die zu anderen Ressourcenmanagern gehören, erfolgen mithilfe des API des jeweiligen Ressourcenmanagers. Sie können allerdings die MQI nicht verwenden, um Warteschlangen zu aktualisieren, die zu anderen Warteschlangenmanagern gehören. Geben Sie MQCMIT oder MQBACK aus, bevor Sie weitere Arbeitseinheiten starten (lokal oder global).

Die globale Arbeitseinheit wird mithilfe von MQCMIT festgeschrieben; dadurch wir ein zweiphasiges Commit aller Ressourcenmanager ausgelöst, die an der Arbeitseinheit beteiligt sind. Während des zweiphasigen Commitprozesses werden die Ressourcenmanager (z. B. XA-konforme Datenbankmanager wie Db2, Oracle und Sybase) zunächst zur Vorbereitung auf das Commit aufgefordert. Nur wenn alle vorbereitet sind, werden sie zur Durchführung des Commit aufgefordert. Wenn ein Ressourcenmanager signalisiert, dass er kein Commit durchführen kann, werden alle gefragt, ob sie stattdessen ein Backout ausführen können. Wahlweise können Sie MQBACK verwenden, um die Updates aller Ressourcenmanager rückgängig zu machen.

Wenn eine Anwendung die Verbindung trennt (MQDISC), obwohl noch eine globale Arbeitseinheit aktiv ist, wird die Arbeitseinheit festgeschrieben. Wird die Anwendung hingegen ohne Verbindungstrennung beendet, so wird die Arbeitseinheit zurückgesetzt, da davon ausgegangen wird, dass die Anwendung fälschlicherweise beendet wurde.

Die Ausgabe von MQBEGIN ist ein Beendigungscode und ein Ursachencode.

Wenn Sie MQBEGIN für den Start einer globalen Arbeitseinheit verwenden, werden alle externen Ressourcenmanager einbezogen, die mit dem Warteschlangenmanager konfiguriert wurden. Der Aufruf startet jedoch eine Arbeitseinheit, wird aber mit einer Warnung beendet, falls:

- Es keine teilnehmenden Ressourcenmanager gibt (das bedeutet, dass keine Ressourcenmanager mit dem Warteschlangenmanager konfiguriert wurden)

oder

- Ein oder mehrere Ressourcenmanager nicht verfügbar sind

In solchen Fällen muss die Arbeitseinheit Updates nur für die Ressourcenmanager beinhalten, die beim Start der Arbeitseinheit verfügbar waren.

Wenn einer der Ressourcenmanager seine Updates nicht festschreiben kann, werden alle Ressourcenmanager angewiesen, Ihre Updates rückgängig zu machen und MQCMIT schließt mit einer Warnung ab. Unter ungewöhnlichen Umständen (in der Regel Bedienereingriff) kann ein MQCMIT-Aufruf fehlschlagen, wenn einige Ressourcenmanager Ihre Updates festschreiben und andere sie hingegen zurücksetzen; die Arbeit gilt dann als abgeschlossen mit einem *gemischten* Ergebnis. Derartige Vorkommnisse werden im Fehlerprotokoll des Warteschlangenmanagers diagnostiziert, sodass Korrekturmaßnahmen ergriffen werden können.

Ein MQCMIT einer globalen Arbeitseinheit ist erfolgreich, wenn alle beteiligten Ressourcenmanager Ihre Updates festschreiben.

Eine Beschreibung des MQBEGIN-Aufrufs finden Sie im Abschnitt MQBEGIN.

Externe Synchronisationspunktkoordinierung

Die Koordinierung erfolgt extern, wenn statt IBM MQ ein anderer Synchronisationspunktkoordinator ausgewählt wurde (z. B. CICS, Encina oder Tuxedo).

In dieser Situation registrieren IBM MQ for AIX, Linux, and Windows-Systeme ihr Interesse am Ergebnis der Arbeitseinheit bei dem externen Synchronisationspunktkoordinator, sodass sie nicht festgeschriebene Get- oder Put-Operationen nach Bedarf festschreiben oder einen entsprechenden Rollback durchführen

können. Der externe Synchronisationspunktkoordinator legt fest, ob Protokolle für ein ein- oder zweiphasiges Commit bereitgestellt werden.

Wenn Sie einen externen Koordinator verwenden, können MQCMIT, MQBACK und MQBEGIN ausgegeben werden. Aufrufe zu diesen Funktionen schlagen mit dem Ursachencode MQRC_ENVIRONMENT_ERROR fehl.

Die Art, in der eine extern koordinierte Arbeitseinheit gestartet wird, hängt von der Programmierschnittstelle ab, die vom Synchronisationspunktkoordinator bereitgestellt wird. Ein expliziter Aufruf ist möglicherweise erforderlich. Wenn ein expliziter Aufruf erforderlich ist und Sie einen MQPUT-Aufruf ausgeben, der die Option MQPMO_SYNCPOINT angibt, wenn eine Arbeitseinheit nicht gestartet wird, wird der Beendigungscode MQRC_SYNCPOINT_NOT_AVAILABLE zurückgegeben.

Der Umfang der Arbeitseinheit wird durch den Synchronisationspunktkoordinator bestimmt. Der Status der Verbindung zwischen der Anwendung und dem Warteschlangenmanager beeinträchtigt den Erfolg oder Misserfolg von MQI-Aufrufen, die eine Anwendung ausgibt, aber nicht den Status der Arbeitseinheit. Eine Anwendung kann beispielsweise eine Verbindung zu einem Warteschlangenmanager während einer aktiven Arbeitseinheit trennen und wiederherstellen und weitere MQGET- und MQPUT-Operationen innerhalb derselben Arbeitseinheit ausführen. Dies wird als anstehende Verbindungsunterbrechung bezeichnet.

Sie können IBM MQ -API-Aufrufe in CICS -Programmen verwenden, unabhängig davon, ob Sie sich für die Verwendung der XA-Funktionalität von CICSentscheiden. Wenn Sie kein XA verwenden, werden die Nachrichteneinreihungs- und -abrufoperationen in und aus Warteschlangen nicht innerhalb der atomaren Arbeitseinheiten von CICS verwaltet. Diese Methode wählen Sie in der Regel nur dann, wenn die Konsistenz einer Arbeitseinheit für Sie eher sekundär ist.

Wenn Ihnen die Datenintegrität Ihrer Arbeitseinheiten wichtig ist, müssen Sie XA verwenden. Bei Verwendung von XA verwendet CICS ein Protokoll für zweiphasiges Commit, das gewährleistet, dass alle Ressourcen innerhalb einer Arbeitseinheit gemeinsam geändert werden.

Weitere Informationen zur Einrichtung der Transaktionsunterstützung finden Sie im Abschnitt Szenarios zur Transaktionsunterstützung sowie in der Dokumentation zu TXSeries CICS, zum Beispiel *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

Multi *Impliziter Synchronisationspunkt unter Multiplatforms*

Die Unterstützung des impliziten Synchronisationspunkts ermöglicht das Einreihen persistenter Nachrichten außerhalb des Synchronisationspunkts.

Beim Einreihen persistenter Nachrichten ist IBM MQ für das Einreihen von persistenten Nachrichten unter Synchronisationspunkt optimiert. Mehrere Anwendungen, die gleichzeitig persistente Nachrichten in dieselbe Warteschlange einreihen, zeigen üblicherweise eine bessere Leistung, wenn sie mit Synchronisationspunkten arbeiten. Dies liegt daran, dass die Sperrstrategie von IBM MQ effizienter ist, wenn beim Einreihen persistenter Nachrichten ein Synchronisationspunkt verwendet wird.

Mit dem Parameter **ImplSyncOpenOutput** in der Datei `qm.ini` wird gesteuert, ob ein impliziter Synchronisationspunkt hinzugefügt werden kann, wenn Anwendungen persistente Nachrichten außerhalb des Synchronisationspunkts einreihen. Dies kann zu einer Leistungsverbesserung führen, ohne dass Anwendungen etwas von dem impliziten Synchronisationspunkt mitbekommen.

Ein impliziter Synchronisationspunkt sorgt nur dann für eine Leistungssteigerung, wenn mehrere Anwendungen gleichzeitig Nachrichten in die Warteschlange einreihen, weil dadurch Sperrenkonflikte reduziert werden. **ImplSyncOpenOutput** gibt die Mindestanzahl von Anwendungen an, die eine Warteschlange für Ausgaben geöffnet haben, bevor ein impliziter Synchronisationspunkt hinzugefügt werden kann. Der Standardwert ist *2*. Das heißt: Wenn Sie **ImplSyncOpenOutput** nicht explizit angeben, wird ein impliziter Synchronisationspunkt nur hinzugefügt, wenn mehrere Anwendungen Nachrichten in die Warteschlange einreihen.

Wenn Sie einen impliziten Synchronisationspunkt hinzufügen, wird dieses Ereignis in der Statistik abgebildet und Sie sehen möglicherweise eine Transaktionsausgabe von **runmqsc display conn**.

Geben Sie **ImplSyncOpenOutput**=*OFF* an, wenn nie ein impliziter Synchronisationspunkt hinzugefügt werden soll.

Weitere Informationen finden Sie im Abschnitt Parameter optimieren.

Schnittstellen zu externen Synchronisationspunktmanagern auf Multiplatforms
IBM MQ for Multiplatforms unterstützt die Transaktionskoordination durch externe Synchronisationspunktmanager, die die X/Open XA-Schnittstelle verwenden.

Einige XA-Transaktionsmanager (TXSeries) setzen voraus, dass die XA-Ressourcenmanager ihre Namen bereitstellen. Diese erfolgt in der XA-Switch-Struktur über die Zeichenfolge `name`.

- ALW Der Ressourcenmanager für IBM MQ unter AIX, Linux, and Windows heißt 'MQSeries_XA_RMI'.
- IBM i Unter IBM i heißt der Ressourcenmanager 'MQSeries XA RMI'.

Ausführliche Informationen zu XA-Schnittstellen finden Sie in der von The Open Group herausgegebenen XA-Dokumentation *CAE Specification Distributed Transaction Processing: The XA Specification*.

In einer XA-Konfiguration nimmt IBM MQ for Multiplatforms die Rolle eines XA-Ressourcenmanagers ein. Ein XA-Synchronisationspunktkoordinator kann einen Satz XA-Ressourcenmanager verwalten und die Festschreibung bzw. Zurücksetzung von Transaktionen zwischen diesen Ressourcenmanagern synchronisieren. Und so funktioniert dies für einen statisch registrierten Ressourcenmanager:

1. Eine Anwendung teilt dem Synchronisationspunktkoordinator mit, dass es eine Transaktion starten will.
2. Der Synchronisationspunktkoordinator gibt einen Aufruf an alle Ressourcenmanager aus, die ihm bekannt sind, um sie über die aktuelle Transaktion zu informieren.
3. Die Anwendung gibt Aufrufe aus, um die Ressourcen zu aktualisieren, die von den der aktuellen Transaktion zugeordneten Ressourcenmanagern verwaltet werden.
4. Die Anwendung fordert von dem Synchronisationspunktkoordinator ein Commit oder ein Backout der Transaktion an.
5. Der Synchronisationspunktkoordinator gibt über ein Protokoll für ein zweiphasiges Commit an jeden Ressourcenmanager Aufrufe zur Ausführung der angeforderten Transaktion aus.

Die XA-Spezifikation verlangt von jedem Ressourcenmanager eine als XA-Switch bezeichnete Struktur. Diese Struktur deklariert das Leistungsspektrum des Ressourcenmanagers sowie die Funktionen, die vom Synchronisationspunktkoordinator aufgerufen werden können.

Diese Struktur liegt in zwei Versionen vor:

Tabelle 128. Versionen des XA-Switch

Version	**Beschreibung**
MQRMIXASwitch	Statisches XA-Ressourcenmanagement
MQRMIXASwitchDynamic	Dynamisches XA-Ressourcenmanagement

Eine Liste der Bibliotheken mit dieser Struktur finden Sie unter XA-Switchstruktur von IBM MQ.

Die Methode, mit der diese Strukturen mit einem XA-Synchronisationspunktkoordinator verknüpft werden, wird durch den Koordinator bestimmt. Informieren Sie sich in der vom Koordinator bereitgestellten Dokumentation, was Sie tun müssen, damit IBM MQ mit Ihrem XA-Synchronisationspunktkoordinator zusammenarbeitet.

Die Struktur *xa_info*, die mit jedem *xa_open*-Aufruf vom Synchronisationspunktkoordinator übergeben wird, kann der Name des zu verwaltenden Warteschlangenmanagers sein. Sie hat die gleiche Form wie der an MQCONN oder MQCONNX übergebene Warteschlangenmanagername und kann leer sein, wenn der Standardwarteschlangenmanager verwendet werden soll. Allerdings können Sie zusätzlich auch die beiden Parameter TPM und AXLIB verwenden.

Mit TPM können Sie IBM MQ den Namen des Transaktionsmanagers angeben, zum Beispiel CICS. Mit AXLIB können Sie den Namen der Bibliothek im Transaktionsmanager angeben, in der sich die XA-AX-Eingangspunkte befinden.

Wenn Sie einen dieser Parameter oder einen nicht standardgemäßen Warteschlangenmanager verwenden, müssen Sie den Namen des Warteschlangenmanagers mit dem Parameter QMNAME angeben. Weitere Informationen finden Sie im Abschnitt The CHANNEL, TRPTYPE, CONNAME, and QMNAME parameters of the xa_open string.

Einschränkungen

1. Globale Arbeitseinheiten sind bei einer gemeinsam genutzten Hconn nicht erlaubt (siehe „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX" auf Seite 777).
2. IBM i IBM MQ for IBM i unterstützt keine dynamische Registrierung von XA-Ressourcenmanagern.

   Als Transaktionsmanager wird nur WebSphere Application Server unterstützt.
3. Windows Auf Windows-Systemen werden alle im XA-Switch deklarierten Funktionen als _cdecl-Funktionen deklariert.
4. Ein externer Synchronisationspunktkoordinator kann jeweils nur einen Warteschlangenmanager verwalten. Der Koordinator hat nämlich zu jedem Warteschlangenmanager eine effektive Verbindung und unterliegt somit der Regel, dass zur selben Zeit jeweils nur eine Verbindung erlaubt ist.

   Anmerkung: Hinweis: Für eine JMS-Clientanwendung (CLIENT JEE-Anwendung) auf einem JEE-Server gilt diese Einschränkung nicht. Hier kann eine einzelne von einem JEE-Server verwaltete Transaktion mehrere Warteschlangenmanager koordinieren. Dennoch gilt für eine JMS-Serveranwendung im Bindungsmodus nach wie vor die Regel, dass zur selben Zeit jeweils nur eine Verbindung erlaubt ist.
5. Alle Anwendungen, die über den Synchronisationspunktkoordinator ausgeführt werden, können nur eine Verbindung zu dem vom Koordinator verwalteten Warteschlangenmanager herstellen, da sie bereits effektiv mit diesem Warteschlangenmanager verbunden sind. Sie müssen sich mit MQCONN oder MQCONNX eine Verbindungskennung beschaffen und vor dem Beenden MQDISC ausgeben. Für TXSeries CICS kann alternativ auch der Exit UE014015 verwendet werden.

IBM i *Schnittstellen mit dem externen Synchronisationspunktmanager von IBM i*

IBM MQ for IBM i kann die native Commitsteuerung von IBM i als externen Synchronisationspunktkoordinator verwenden.

Threadunabhängige (gemeinsam genutzte) Verbindungen sind bei Commitsteuerung nicht zulässig. Weitere Informationen zu den Commitsteuerungsfunktionen von IBM ifinden Sie im Handbuch *IBM i Programming: Backup and Recovery Guide, SC21-8079* .

Verwenden Sie den Systembefehl STRCMTCTL, um die Funktionen der IBM i-COMMIT-Steuerung zu starten. Zum Beenden der COMMIT-Steuerung verwenden Sie den Systembefehl ENDCMTCTL.

Anmerkung: Der Standardwert des *COMMIT-Definitionsbereichs* lautet *ACTGRP. Dieser muss als *JOB für IBM MQ for IBM i definiert werden. For example:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i kann auch lokale Arbeitseinheiten ausführen, die nur Änderungen an IBM MQ-Ressourcen enthalten. Die Entscheidung zwischen lokalen Arbeitseinheiten und der Teilnahme an globalen, durch IBM i koordinierten Arbeitseinheiten wird in jeder Anwendung bei Ausführung eines MQPUT-, MQPUT1- oder MQGET-Aufrufs mit der Option MQPMO_SYNCPOINT oder MQGMO_SYNCPOINT, oder bei Ausführung eines MQBEGIN-Aufrufs getroffen. Wenn die Commitsteuerung bei Ausgabe des ersten dieser Befehle nicht aktiv ist, startet IBM MQ eine lokale Arbeitseinheit, und alle weiteren Arbeitseinheiten für diese Verbindung mit IBM MQ verwenden dann ebenso lokale Arbeitseinheiten, unabhängig davon, ob die Commitsteuerung noch gestartet wird. Verwenden Sie MQCMIT, um eine lokale Arbeitseinheit festzuschreiben. Verwenden Sie MQBACK, um eine lokale Arbeitseinheit zurückzusetzen. Die Commit-

und Rollback-Aufrufe von IBM i, beispielsweise der CL-Befehl COMMIT, haben keine Auswirkung auf lokale Arbeitseinheiten von IBM MQ.

Wenn Sie IBM MQ for IBM i mit IBM i-eigener Commitsteuerung als externen Synchronisationspunktkoordinator verwenden möchten, müssen Sie sicherstellen, dass ein Job mit Commitsteuerung aktiv ist und dass Sie IBM MQ in einem Einzelthread-Job verwenden. Wenn Sie beim Aufruf von MQPUT, MQPUT1 oder MQGET in einem Multithread-Job, bei dem die Commitsteuerung gestartet wurde, MQPMO_SYNCPOINT oder MQGMO_SYNCPOINT angeben, schlägt der Aufruf mit dem Ursachencode MQRC_SYNCPOINT_NOT_AVAILABLE fehl.

Es besteht die Möglichkeit, lokale Arbeitseinheiten und die MQCMIT- und MQBACK-Aufrufe in einem Multithread-Job zu verwenden.

Wenn Sie MQPUT, MQPUT1 oder MQGET nach dem Start der Commitsteuerung mit der Option MQPMO_SYNCPOINT oder MQGMO_SYNCPOINT aufrufen, fügt sich IBM MQ for IBM i selbst als API-Commitressource zur Commitdefinition hinzu. Das ist in der Regel der erste Aufruf dieser Art in einem Job. Obwohl unter einer bestimmten Commitdefinition alle API-COMMIT-Ressourcen registriert sind, können Sie die Commitsteuerung für diese Definition nicht beenden.

Bei einer Trennung vom Warteschlangenmanager entfernt IBM MQ for IBM i seine Registrierung als API-Commitressource, wenn die aktuelle Arbeitseinheit keine ausstehenden MQI-Operationen enthält.

Wenn Sie die Verbindung zum Warteschlangenmanager trennen und die aktuelle Arbeitseinheit noch anstehende MQPUT-, MQPUT1- oder MQGET-Operationen enthält, so bleibt die Registrierung von IBM MQ for IBM i als API-COMMIT-Ressource weiterhin erhalten, sodass es über die nächste Festschreibung bzw. den nächsten Rollback benachrichtigt wird. Bei Erreichen des nächsten Synchronisationspunktes schreibt IBM MQ for IBM i die Änderungen wie angefordert fest oder setzt diese zurück. Eine Anwendung kann die Verbindung zu einem Warteschlangenmanager während einer aktiven Arbeitseinheit trennen und wiederherstellen und weitere MQGET- und MQPUT-Operationen innerhalb derselben Arbeitseinheit ausführen (das ist eine anstehende Verbindungstrennung).

Wenn Sie versuchen, für diese COMMIT-Definition den Systembefehl ENDCMTCTL auszugeben, wird die Nachricht CPF8355 ausgegeben; diese gibt an, dass anstehende Änderungen vorhanden waren. Diese Nachricht erscheint auch im Jobprotokoll, wenn der Job endet. Um diese Nachricht zu vermeiden, sollten Sie alle anstehenden IBM MQ for IBM i-Operationen festschreiben oder zurücksetzen und die Anwendung dann vom Warteschlangenmanager trennen. Somit wird mithilfe der Befehle COMMIT oder ROLLBACK vor ENDCMTCTL eine Endcommitsteuerung für einen erfolgreichen Abschluss ermöglicht.

Wenn Sie IBM i-Commitsteuerung als externen Synchronisationspunktkoordinator verwenden, können Sie keine MQCMIT-, MQBACK- und MQBEGIN-Aufrufe ausgeben. Aufrufe zu diesen Funktionen schlagen mit dem Ursachencode MQRC_ENVIRONMENT_ERROR fehl.

Wenn Sie Ihre Arbeitseinheit festschreiben oder zurücksetzen möchten, verwenden Sie eine der Programmiersprachen, die Commitsteuerung unterstützen. For example:

- CL-Befehle: COMMIT und ROLLBACK
- ILE C-Programmierungsfunktionen: _Rcommit und _Rrollback
- ILE RPG: COMMIT und ROLBK
- COBOL/400: COMMIT und ROLLBACK

Wenn Sie die IBM i-Commitsteuerung in IBM MQ for IBM i als externen Synchronisationspunktkoordinator verwenden, führt IBM i ein zweiphasiges Commitprotokoll durch, an dem IBM MQ beteiligt ist. Weil jede Arbeitseinheit in zwei Phasen festgeschrieben wird, ist der Warteschlangenmanager für die zweite Phase möglicherweise nicht verfügbar, nachdem in der ersten Phase ein Commit beschlossen wurde. Dies kann beispielsweise der Fall sein, wenn die internen Jobs des Warteschlangenmanagers beendet werden. In dieser Situation enthält das Jobprotokoll, das das Commit ausführt, die Nachricht CPF835F, die angibt, dass eine Commit- oder Rollback-Operation fehlgeschlagen ist. Die Nachrichten, die dem vorausgehen, zeigen die Ursache des Problems an, ob es während einer Commit- oder Rollback-Operation aufgetreten ist und außerdem die ID der logischen Arbeitseinheit (LUWID) für die fehlgeschlagene Arbeitseinheit.

Wenn das Problem während des Commits oder Rollbacks einer vorbereiteten Arbeitseinheit durch einen Fehler der Commitressource der IBM MQ-API verursacht wurde, können Sie die Operation mit dem Befehl

WRKMQMTRN abschließen und die Integrität der Transaktion wiederherstellen. Für den Befehl müssen Sie die ID der logischen Arbeitseinheit kennen, die festgeschrieben bzw. zurückgesetzt werden soll.

IBM MQ-Anwendungen durch Auslöser starten

In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

Einige IBM MQ-Anwendungen, die Warteschlangen bedienen, werden unterbrechungsfrei ausgeführt, so dass sie immer bereitstehen, um die in Warteschlangen eintreffenden Nachrichten abzurufen. Wenn jedoch nicht vorhersehbar ist, wie viele Nachrichten in den Warteschlangen ankommen werden, ist dies möglicherweise nicht wünschenswert. In diesem Fall würden Anwendungen auch dann Systemressourcen verbrauchen, wenn gar keine Nachrichten abzurufen sind.

IBM MQ bietet eine Funktion, mit der eine Anwendung automatisch gestartet werden kann, wenn Nachrichten zum Abrufen bereitstehen. Diese Funktion wird als *Auslösefunktion* bezeichnet.

Informationen zur Kanalauslösung finden Sie im Abschnitt Kanalauslösung.

Das Prinzip der Auslösefunktion

Der Warteschlangenmanager definiert bestimmte Bedingungen als konstituierende *Auslöserereignisse*.

Wenn für eine Warteschlange die Auslösefunktion aktiviert ist und ein Auslöserereignis eintritt, sendet der Warteschlangenmanager eine *Auslösenachricht* an die sogenannte *Initialisierungswarteschlange*. Das Vorhandensein der Auslösenachricht in der Initialisierungswarteschlange zeigt an, dass ein Auslöserereignis aufgetreten ist.

Vom Warteschlangenmanager generierte Auslösenachrichten sind keine persistenten Nachrichten. Dies bedeutet einen geringeren Protokollierungsaufwand (was sich in verbesserter Leistung zeigt) sowie ein Minimum an Duplikaten bei einem Neustart und damit eine verbesserte Neustartzeit.

Die Initialisierungswarteschlange wird von einer sogenannten *Auslösemonitoranwendung* verarbeitet, deren Aufgabe es ist, die Auslösenachricht zu lesen und unter Berücksichtigung der darin enthaltenen Angaben die entsprechende Maßnahme zu ergreifen. In der Regel besteht diese Maßnahme darin, eine andere Anwendung zur Verarbeitung der Warteschlange zu starten, welche die Auslösenachricht generiert hat. Für den Warteschlangenmanager ist die Auslösemonitoranwendung eine ganz gewöhnliche Anwendung, die Nachrichten aus einer Warteschlange (der Initialisierungswarteschlange) liest.

Wenn für eine Warteschlange die Auslösefunktion aktiviert ist, können Sie ein zugehöriges *Prozessdefinitionsobjekt* erstellen. Dieses Objekt enthält Informationen zu der Anwendung, welche die Nachricht verarbeitet, die das Auslöserereignis verursacht hat. Ist das Prozessdefinitionsobjekt erstellt, extrahiert der Warteschlangenmanager diese Informationen und stellt sie in die Auslösenachricht, damit sie von der Auslösemonitoranwendung verwendet werden kann. Der Name der Prozessdefinition einer Warteschlange wird durch das Attribut `ProcessName` der lokalen Warteschlange festgelegt. Jede Warteschlange kann eine eigene Prozessdefinition angeben, oder mehrere Warteschlangen können eine gemeinsame Prozessdefinition teilen.

Um einen Kanalstart auszulösen, muss kein Prozessdefinitionsobjekt definiert werden. Stattdessen wird die Übertragungswarteschlangendefinition verwendet.

Triggering wird von IBM MQ-Clients unterstützt, die unter AIX, Linux, and Windows ausgeführt werden. Eine Anwendung, die in einer Clientumgebung ausgeführt wird, unterscheidet sich lediglich darin von einer Anwendung in einer vollständigen IBM MQ-Umgebung, dass sie mit den Clientbibliotheken verknüpft wird. Allerdings müssen der Auslösemonitor und die zu startende Anwendung derselben Umgebung angehören.

Die Auslösefunktion umfasst folgende Komponenten:

Anwendungswarteschlange
: Eine *Anwendungswarteschlange* ist eine lokale Warteschlange, die bei aktivierter Auslösefunktion erfordert, dass Auslösenachrichten geschrieben werden, wenn die entsprechenden Bedingungen erfüllt sind.

Prozessdefinition
Einer Anwendungswarteschlange kann ein *Prozessdefinitionsobjekt* zugeordnet sein, in dem Einzelheiten zu der Anwendung gespeichert sind, welche Nachrichten aus der Anwendungswarteschlange abrufen soll. (Eine Liste der Attribute finden Sie im Abschnitt Attribute für Prozessdefinitionen.)

Beachten Sie bitte, dass zum Auslösen eines Kanalstarts kein Prozessdefinitionsobjekt definiert werden muss.

Übertragungswarteschlange
Um einen Kanalstart auszulösen, wird eine Übertragungswarteschlange benötigt.

Auf allen Plattformen mit Ausnahme von Linux kann im Attribut *TriggerData* der Übertragungswarteschlange der Name des Kanals angegeben werden, der gestartet werden soll. Diese Angabe kann die Prozessdefinition für die Kanalauslösung ersetzen, wird jedoch nur verwendet, wenn keine Prozessdefinition erstellt ist.

Auslöserereignis
Ein *Auslöserereignis* bewirkt, dass vom Warteschlangenmanager eine Auslösenachricht generiert wird. In der Regel ist dies eine Nachricht in einer Anwendungswarteschlange, jedoch kann diese auch zu anderen Zeiten ausgegeben werden. Ein Beispiel hierzu finden Sie im Abschnitt „Bedingungen für ein Auslöserereignis" auf Seite 919.

IBM MQ bietet verschiedene Optionen zur Steuerung der Bedingungen, die ein Auslöserereignis bewirken (siehe „Auslöserereignisse steuern" auf Seite 924).

Auslösenachricht
Der Warteschlangenmanager erstellt eine *Auslösenachricht*, sobald er ein Auslöserereignis erkennt. In die Auslösenachricht kopiert er Informationen zu der zu startenden Anwendung. Diese Informationen stammen aus der Anwendungswarteschlange und dem zugehörigen Prozessdefinitionsobjekt.

Auslösenachrichten haben ein festes Format (siehe „Format von Auslösenachrichten" auf Seite 932).

Initialisierungswarteschlange
Eine *Initialisierungswarteschlange* ist eine lokale Warteschlange, in die der Warteschlangenmanager Auslösenachricht stellt. Die Initialisierungswarteschlange darf keine Alias- oder Modellwarteschlange sein.

Warteschlangenmanager können über mehrere Initialisierungswarteschlangen verfügen, die jeweils einer oder mehreren Anwendungswarteschlangen zugeordnet sind.

z/OS In IBM MQ for z/OS kann eine gemeinsam genutzte Warteschlange, d. h. eine lokale Warteschlange, auf die die Warteschlangenmanager einer Gruppe mit gemeinsamer Warteschlange zugreifen können, eine Initialisierungswarteschlange sein.

Auslösemonitor
Ein *Auslösemonitor* ist ein ständig aktives Programm, das für eine oder mehrere Initialisierungswarteschlangen zuständig ist. Sobald eine Auslösenachricht in einer Initialisierungswarteschlange ankommt, wird sie vom Auslösemonitor abgerufen. Der Auslösemonitor verwendet die Informationen aus der Auslösenachricht. Er setzt einen Befehl zum Start der Anwendung ab, welche die in der Anwendungswarteschlange eingehenden Nachrichten abrufen soll, und übergibt dieser Anwendung Informationen aus dem Header der Auslösenachricht, darunter auch den Namen der Anwendungswarteschlange.

Auf allen Plattformen ist für den Kanalstart ein spezieller Auslösemonitor, der sogenannte Kanalinitiator zuständig.

z/OS Unter z/OS wird der Kanalinitiator in der Regel manuell gestartet, der Start kann aber auch automatisch beim Starten eines Warteschlangenmanagers erfolgen, wenn der Wert für CSQINP2 in der JCL für den Start des Warteschlangenmanagers entsprechend geändert wird.

Multi Auf Multiplatforms-Systemen wird der Kanalinitiator automatisch beim Start des Warteschlangenmanagers gestartet. Der Start kann aber auch manuell mit dem Befehl **runmqchi** erfolgen.

Weitere Informationen finden Sie unter „Initialisierungswarteschlangenverarbeitung durch Auslösemonitore" auf Seite 928.

Sehen Sie sich zum besseren Verständnis des Auslösemechanismus auch das Beispiel im Abschnitt Abbildung 95 auf Seite 915 an, das den Auslösertyp FIRST (MQTT_FIRST) veranschaulicht.

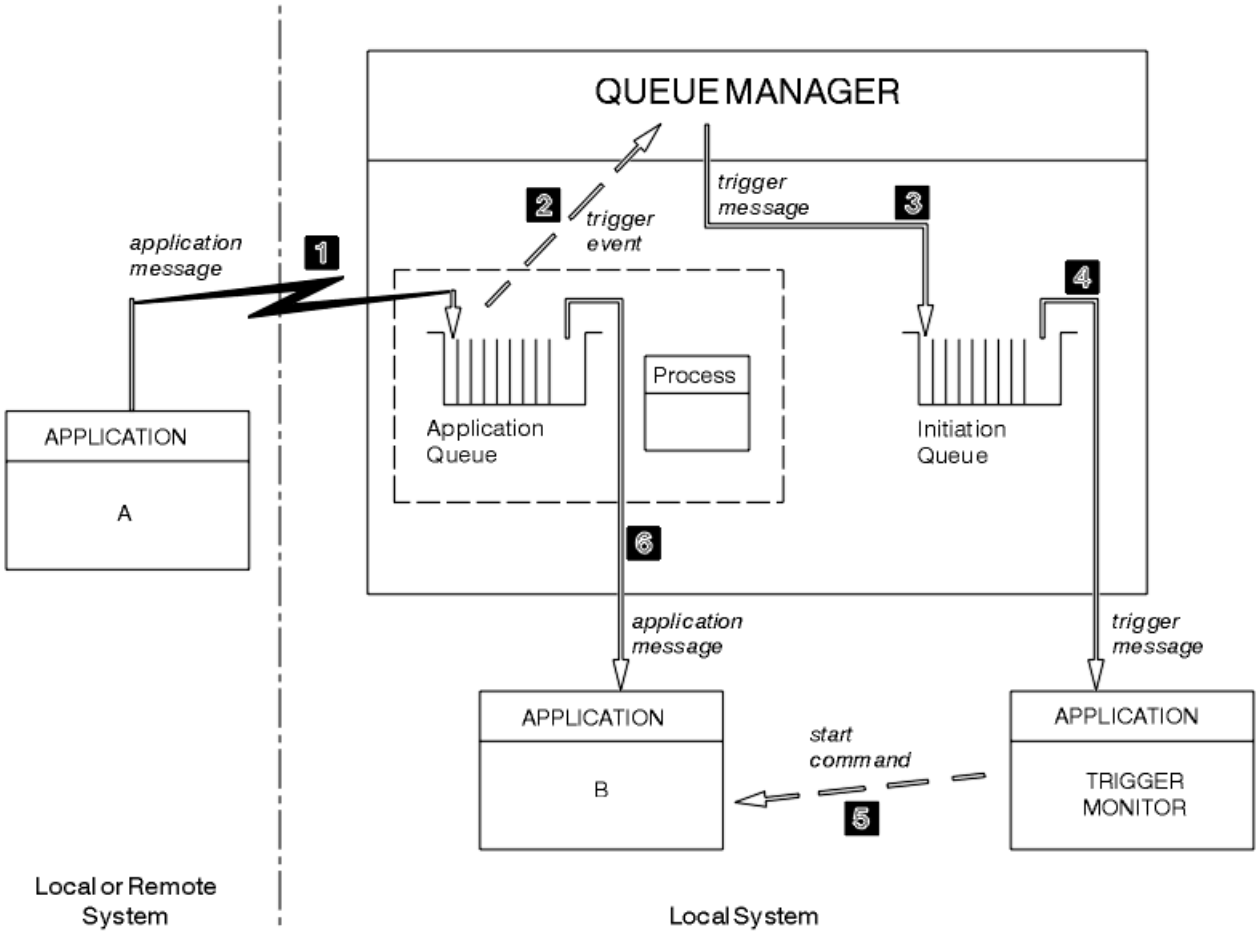


Abbildung 95. Ablauf von Anwendungs- und Auslösenachrichten

In Abbildung 95 auf Seite 915 gilt folgende Ereignisfolge:

1. Anwendung A, die sich auf demselben System wie der Warteschlangenmanager oder auf einem fernen System befinden kann, reiht eine Nachricht in die Anwendungswarteschlange ein. Diese Warteschlange ist von keiner Anwendung zur Eingabe geöffnet. Dies ist jedoch nur für die Auslösertypen FIRST und DEPTH relevant.
2. Der Warteschlangenmanager prüft, ob die Bedingungen für ein Auslöserereignis erfüllt sind. Dies ist der Fall, somit wird ein Auslöserereignis generiert. Beim Erstellen der Auslösenachricht werden die im zugehörigen Prozessdefinitionsobjekt gespeicherten Informationen verwendet.
3. Der Warteschlangenmanager erstellt eine Auslösenachricht und reiht sie in die der betreffenden Anwendungswarteschlange zugeordnete Initialisierungswarteschlange ein, allerdings nur, wenn die Initialisierungswarteschlange von einer Anwendung (einem Auslösemonitor) zur Eingabe geöffnet ist.
4. Der Auslösemonitor ruft die Auslösenachricht aus der Initialisierungswarteschlange ab.
5. Der Auslösemonitor ruft einen Befehl zum Starten von Anwendung B (Serveranwendung) auf.
6. Anwendung B öffnet die Anwendungswarteschlange und ruft die Nachricht ab.

Anmerkung:

1. Ist die Anwendungswarteschlange von einem Programm zur Eingabe geöffnet und ist als Auslösertyp FIRST oder DEPTH definiert, kommt es zu keinem Auslöserereignis, da die Warteschlange bereits bedient wird.

2. Ist die Initialisierungswarteschlange nicht zur Eingabe geöffnet, generiert der Warteschlangenmanager keine Auslösenachrichten sondern wartet, bis die Initialisierungswarteschlange von einer Anwendung zur Eingabe geöffnet wird.
3. Verwenden Sie für Kanäle den Auslösertyp FIRST oder DEPTH.
4. Ausgelöste Anwendungen werden unter der Benutzer-ID und Gruppe des Benutzers ausgeführt, der das Auslöseüberwachungsprogramm gestartet hat, des CICS-Benutzers oder des Benutzers, der den Warteschlangenmanager gestartet hat.

Bei den bisherigen Beispielen bestand zwischen den Warteschlangen im Zusammenhang mit der Auslösefunktion lediglich eine Eins-zu-Eins-Beziehung. Betrachten Sie nun Abbildung 96 auf Seite 916.

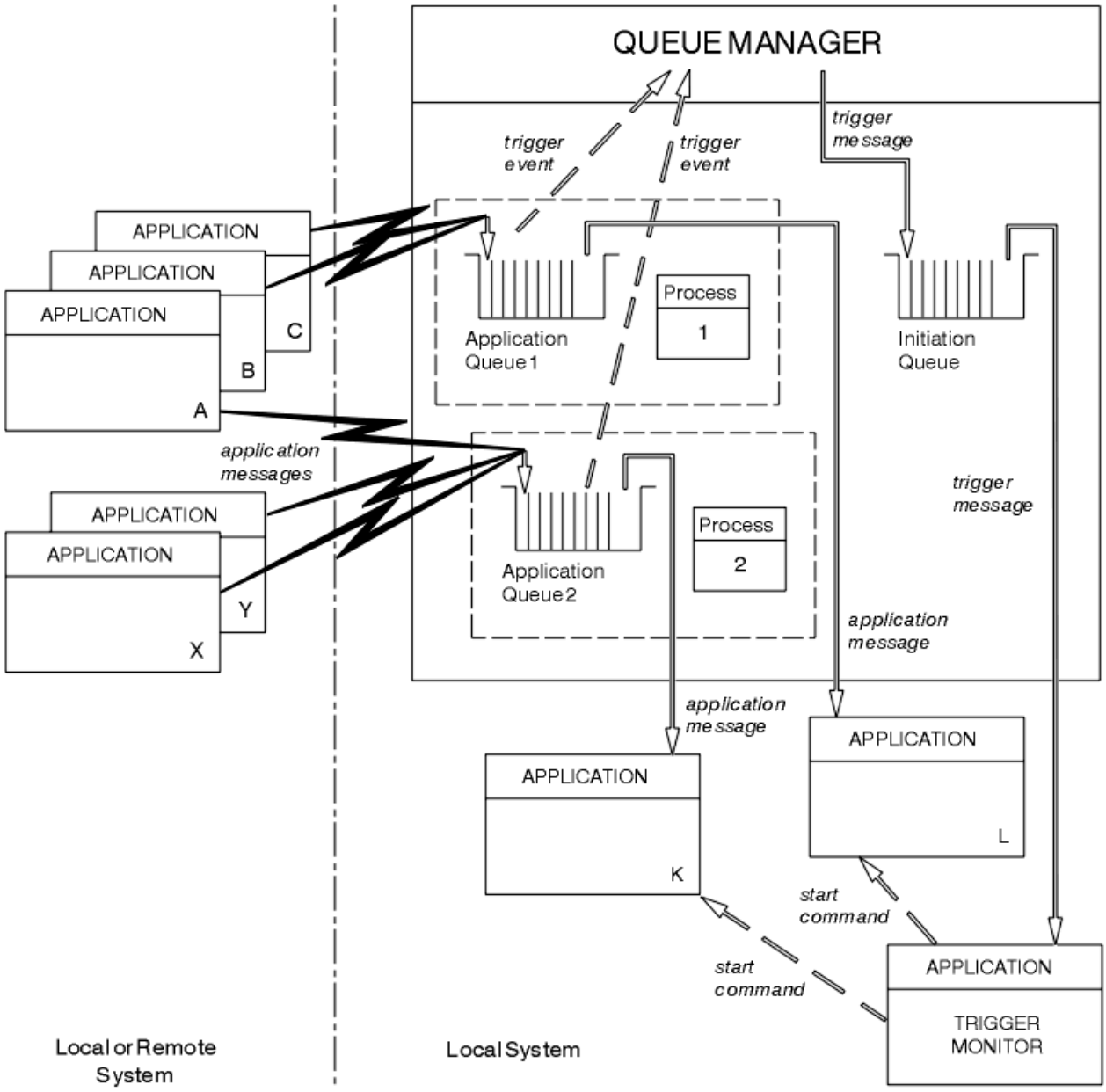


Abbildung 96. Warteschlangenbeziehungen im Zusammenhang mit der Auslösefunktion

Einer Anwendungswarteschlange ist ein Prozessdefinitionsobjekt zugeordnet, in dem Einzelheiten zu der Anwendung gespeichert sind, welche die Nachricht verarbeiten soll. Der Warteschlangenmanager stellt die Informationen in die Auslösenachricht, somit ist nur eine Initialisierungswarteschlange erforderlich. Der Auslösemonitor extrahiert diese Informationen aus der Auslösenachricht und startet die entsprechende Anwendung, welche die Nachricht in der jeweiligen Anwendungswarteschlange bearbeiten soll.

Wie bereits erwähnt, muss zum Auslösen eines Kanalstarts kein Prozessdefinitionsobjekt definiert werden. Welcher Kanal ausgelöst werden soll, kann anhand der Übertragungswarteschlangendefinition bestimmt werden.

Unter den folgenden Links erhalten Sie weitere Informationen zum Starten von IBM MQ-Anwendungen mit Auslösern:

- „Voraussetzungen für die Auslösung" auf Seite 917
- „Bedingungen für ein Auslöserereignis" auf Seite 919
- „Auslöserereignisse steuern" auf Seite 924
- „Anwendung entwickeln, die ausgelöste Warteschlangen verwendet" auf Seite 926
- „Initialisierungswarteschlangenverarbeitung durch Auslösemonitore" auf Seite 928
- „Eigenschaften von Auslösenachrichten" auf Seite 931
- „Wenn die Auslösung nicht funktioniert" auf Seite 933

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht" auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen" auf Seite 773
Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Objekte öffnen und schließen" auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen" auf Seite 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen" auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen" auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen" auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„Mit MQI und Clustern arbeiten" auf Seite 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Using and writing applications on IBM MQ for z/OS" auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS" auf Seite 73
This information helps you to write IMS applications using IBM MQ.

Voraussetzungen für die Auslösung

In diesem Abschnitt erfahren Sie schrittweise, wie der Auslösemechanismus verwendet wird.

Bevor Ihre Anwendung die Auslösefunktion verwenden kann, führen Sie folgende Schritte durch:

1. Entweder:
   a. Erstellen Sie eine Initialisierungswarteschlange für Ihre Anwendungswarteschlange. For example:

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   oder

b. Ermitteln Sie den Namen einer lokalen Warteschlange, die vorhanden ist und von Ihrer Anwendung verwendet werden kann (in der Regel ist deren Name SYSTEM.DEFAULT.INITIATION.QUEUE oder, wenn Sie Kanäle mit Auslösern starten, SYSTEM.CHANNEL.INITQ), und geben Sie diesen Namen im Feld *InitiationQName* der Anwendungswarteschlange ein.

2. Ordnen Sie der Anwendungswarteschlange die Initialisierungswarteschlange zu. Ein Warteschlangenmanager kann Eigner von mindestens einer Initialisierungswarteschlange sein. Möglicherweise wünschen Sie, dass einige Ihrer Anwendungswarteschlangen von verschiedenen Programmen bedient werden. In diesem Fall können Sie die Initialisierungswarteschlange für alle Bereitstellungsprogramme verwenden, aber Sie müssen nicht. Hier ist ein Beispiel, wie eine Anwendungswarteschlange erstellt wird:

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i Hier ein Auszug aus einem CL-Programm für IBM MQ for IBM i, in dem eine Initialisierungswarteschlange erstellt wird:

```
/*   Queue used by AMQSINQA                                      */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                 +
                        QTYPE(*LCL)  REPLACE(*YES)                 +
                        MQMNAME                                    +
                        TEXT('queue for AMQSINQA')                 +
                        SHARE(*YES)                /* Shareable   */+
                        DFTMSGPST(*YES)/* Persistent messages OK  */+
                                                                   +
                        TRGENBL(*YES)  /* Trigger control on      */+
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.INQPROCESS')        +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. Wenn Sie eine Anwendung auslösen, erstellen Sie ein Prozessdefinitionsobjekt, um Informationen bezüglich der Anwendung aufzunehmen, die Ihrer Anwendungswarteschlange dienen sollen. Hier ein Beispiel, in dem der Start einer CICS-Lohnbuchhaltungstransaktion namens PAYRF ausgelöst wird:

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i Hier ein Auszug aus einem CL-Programm für IBM MQ for IBM i, in dem ein Prozessdefinitionsobjekt erstellt wird:

```
/*   Process definition                                          */
            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')      +
                        REPLACE(*YES)                            +
                        MQMNAME                                  +
                        TEXT('trigger process for AMQSINQA')     +
                        ENVDATA('JOBPTY(3)') /* Submit parameter */+
                        APPID('AMQSINQA')    /* Program name    */
```

Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, kopiert er Informationen von den Attributen des Prozessdefinitionsobjekts in die Auslösenachricht.

Plattform	Zum Erstellen eines Prozessdefinitionsobjekts
AIX, Linux, and Windows-Systeme	Verwenden Sie DEFINE PROCESS oder verwenden Sie SYSTEM.DEFAULT.PROCESS und ändern Sie es mit ALTER PROCESS
z/OS z/OS z/OS	Verwenden Sie DEFINE PROCESS (siehe Beispielcode in Schritt „3" auf Seite 918) oder verwenden Sie die Operationen und Bedienfelder.
IBM i IBM i IBM i	Verwenden Sie ein CL-Programm, das Code wie in Schritt „3" auf Seite 918 enthält.

4. Optional: Erstellen Sie die Definition einer Übertragungswarteschlange, wobei Sie für das Attribut **ProcessName** Leerzeichen verwenden.

   Das Attribut **TrigData** kann den Namen des Kanals enthalten, der ausgelöst werden soll, oder leer gelassen werden; außer unter IBM MQ for z/OS, wenn es leer bleibt, durchsucht der Kanalinitiator die Kanaldefinitionsdateien, bis er einen Kanal findet, der der angegebenen Übertragungswarteschlange zugeordnet ist. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, kopiert er Informationen aus dem Attribut **TrigData** der Definition der Übertragungswarteschlange in die Auslösenachricht.

5. Wenn Sie ein Prozessdefinitionsobjekt erstellt haben, um Eigenschaften der Anwendung anzugeben, die Ihre Anwendungswarteschlange bedienen soll, ordnen Sie das Prozessobjekt Ihrer Anwendungswarteschlange zu, indem Sie den Namen des Objekts im Attribut **ProcessName** der Warteschlange angeben.

Plattform	Befehle
AIX, Linux, and Windows-Systeme	ALTER QLOCAL
z/OS z/OS z/OS	ALTER QLOCAL
IBM i IBM i IBM i	CHGMQMQ

6. Starten Sie Instanzen der Auslösemonitore IBM i (bzw. in IBM MQ for IBM i Auslöseserver), die die von Ihnen definierten Initialisierungswarteschlangen bedienen sollen. Weitere Informationen finden Sie im Abschnitt „Initialisierungswarteschlangenverarbeitung durch Auslösemonitore" auf Seite 928.

Wenn Sie Kenntnis über nicht zugestellte Auslösenachrichten erhalten möchten, stellen Sie sicher, dass Ihr Warteschlangenmanager eine Warteschlange für nicht zustellbare Nachrichten definiert hat. Geben Sie den Namen der Warteschlange im Feld *DeadLetterQName* des Warteschlangenmanagers an.

Sie können die erforderlichen Auslöserbedingungen mithilfe der Attribute des Warteschlangenobjekts einstellen, das Ihre Anwendungswarteschlange definiert. Weitere Informationen finden Sie unter „Auslösereignisse steuern" auf Seite 924.

Bedingungen für ein Auslöserereignis

Der Warteschlangenmanager erstellt eine Auslösenachricht, wenn bestimmte Bedingungen erfüllt sind.

Die folgenden Bedingungen bewirken, dass der Warteschlangenmanager eine Auslösenachricht erzeugt:

1. Eine Nachricht wird in eine Warteschlange *eingereiht*.
2. Die Nachricht hat eine höhere oder gleiche Priorität wie der Auslöser bei Erreichen eines Schwellenwerts der Warteschlange. Diese Priorität wird durch das Attribut **TriggerMsgPriority** der lokalen Warteschlange festgelegt; bei null gilt jede Nachricht.

3. Die Anzahl der Nachrichten in der Warteschlange mit einer höheren oder gleichen Priorität wie *TriggerMsgPriority* war zuvor vom *TriggerType* abhängig:
   - Null (für Auslösertyp MQTT_FIRST)
   - Beliebige Anzahl (für Auslösertyp MQTT_EVERY)
   - *TriggerDepth* minus 1 (für Auslösertyp MQTT_DEPTH)

   Anmerkung:

   - Bei nicht gemeinsam genutzten Warteschlangen zählt der Warteschlangenmanager sowohl festgeschriebene als auch nicht festgeschriebene Nachrichten, wenn er beurteilt, ob die Bedingungen für ein Auslöserereignis erfüllt werden. Folglich kann eine Anwendung gestartet werden, wenn es keine Nachrichten gibt, die sie abrufen kann, weil die Nachrichten in der Warteschlange nicht festgeschrieben wurden. In dieser Situation kann gegebenenfalls mit der Warteoption mit einem geeigneten *WaitInterval* erreicht werden, dass die Anwendung wartet, bis ihre Nachrichten ankommen.
   - z/OS Bei lokalen gemeinsam genutzten Warteschlangen zählt der Warteschlangenmanager nur festgeschriebene Nachrichten.
4. Bei Verwendung der Auslösefunktion des Typs FIRST oder DEPTH hält kein Programm die Anwendungswarteschlange zum Entfernen von Nachrichten offen (d. h., das Attribut **OpenInputCount** der lokalen Warteschlange ist null).

   Anmerkung: z/OS

   - Bei gemeinsam genutzten Warteschlangen gelten besondere Bedingungen, wenn bei mehreren Warteschlangenmanagern Auslösemonitore auf einer Warteschlange ausgeführt werden. Wenn in dieser Situation ein oder mehrere Warteschlangenmanager die Warteschlange für gemeinsame Eingaben geöffnet haben, werden die Auslöserkriterien der anderen Warteschlangenmanager als *TriggerType* MQTT_FIRST und *TriggerMsgPriority* gleich null behandelt. Wenn alle Warteschlangenmanager die Warteschlange für die Eingabe schließen werden die Auslöserbedingungen auf die in der Warteschlangendefinition angegebenen Bedingungen zurückgesetzt.

     Ein Beispielszenario, auf das diese Bedingung zutrifft, sind die drei Warteschlangenmanager QM1, QM2 und QM3 mit dem Auslösemonitor für die Anwendungswarteschlange A. In diesem Beispiel trifft eine Nachricht in Warteschlange A ein, die die Bedingungen für eine Auslösung erfüllt, wodurch in der Initialisierungswarteschlange eine Auslösenachricht generiert wird. Der Auslösemonitor in QM1 ruft die Auslösenachricht ab und löst eine Anwendung aus. Die ausgelöste Anwendung öffnet die Anwendungswarteschlange für gemeinsam genutzte Eingaben. Ab diesem Punkt werden die Auslöserbedingungen für Anwendungswarteschlange A als *TriggerType* MQTT_FIRST und *TriggerMsgPriority* gleich null auf den Warteschlangenmanagern QM2 und QM3 ausgewertet, bis QM1 die Anwendungswarteschlange schließt.
   - Bei gemeinsam genutzten Warteschlangen wird diese Bedingung bei jedem Warteschlangenmanager angewendet. Das bedeutet also, dass ein Warteschlangenmanager nur dann eine Auslösenachricht für die Warteschlange erstellt, wenn sein *OpenInputCount*-Wert für diese Warteschlange null ist. Wenn jedoch ein Warteschlangenmanager in der Gruppe mit gemeinsamer Warteschlange die Warteschlange mithilfe der Option MQOO_INPUT_EXCLUSIVE geöffnet hat, wird für diese Warteschlange von keinem der Warteschlangenmanager in der Gruppe mit gemeinsamer Warteschlange eine Auslösenachricht generiert.

     Die Bewertung der Auslöserbedingungen ändert sich, wenn die ausgelöste Anwendung die Warteschlange für die Eingabe öffnet. In Szenarios, in denen nur ein Auslösemonitor ausgeführt wird, können auch andere Anwendungen diese Wirkung haben, da sie ebenso die Anwendungswarteschlange für Eingaben öffnen. Es ist dabei völlig unerheblich, ob die Anwendungswarteschlange durch eine Anwendung geöffnet wurde, die durch einen Auslösemonitor gestartet wurde, oder durch eine andere Anwendung. Einzig der Fakt, dass die Warteschlange offen für Eingaben über einen anderen Warteschlangenmanager ist, bewirkt die Änderung der Auslösekriterien.
5. z/OS Wenn in IBM MQ for z/OS die Anwendungswarteschlange ein **Usage**-Attribut mit dem Wert MQUS_NORMAL aufweist, werden Abrufanforderungen für die Warteschlange nicht unterdrückt

(d. h., das Warteschlangenattribut **InhibitGet** ist MQQA_GET_ALLOWED). Wenn die ausgelöste Anwendungswarteschlange ein **Usage**-Attribut mit dem Wert MQUS_XMITQ aufweist, werden Abrufanforderungen ebenfalls nicht unterdrückt.

6. Entweder:
   - Das lokale Warteschlangenattribut **ProcessName** für die Warteschlange ist nicht leer und das durch dieses Attribut angegebene Prozessdefinitionsobjekt wurde erstellt, oder
   - Das lokale Warteschlangenattribut **ProcessName** für die Warteschlange ist leer, bei der Warteschlange handelt es sich aber um eine Übertragungswarteschlange. Da die Prozessdefinition optional ist, kann das Attribut **TriggerData** auch den Namen des zu startenden Kanals enthalten. In diesem Fall enthält die Auslösernachricht Attribute mit den folgenden Werten:
     - **QName**: Warteschlangenname
     - **ProcessName**: Leerzeichen
     - **TriggerData**: Auslöserdaten
     - **ApplType**: MQAT_UNKNOWN
     - **ApplId**: Leerzeichen
     - **EnvData**: Leerzeichen
     - **UserData**: Leerzeichen
7. Es wurde eine Initialisierungswarteschlange erstellt und im Attribut **InitiationQName** der lokalen Warteschlange angegeben. Ebenso trifft Folgendes zu:
   - Abrufanforderungen werden für die Initialisierungswarteschlange nicht unterdrückt (d. h., das Warteschlangenattribut **InhibitGet** hat den Wert MQQA_GET_ALLOWED).
   - Einreihungsanforderungen dürfen für die Initialisierungswarteschlange nicht unterdrückt werden (d. h., das Warteschlangenattribut **InhibitPut** muss den Wert MQQA_PUT_ALLOWED haben).
   - Das Attribut **Usage** der Initialisierungswarteschlange muss den Wert MQUS_NORMAL haben.
   - In Umgebungen, bei denen dynamische Warteschlangen unterstützt werden, darf die Initialisierungswarteschlange keine dynamische Warteschlange sein, die als logisch gelöscht gekennzeichnet wurde.
8. Ein Auslösemonitor hat die Initialisierungswarteschlange zum Entfernen von Nachrichten geöffnet (d. h., das Attribut **OpenInputCount** der lokalen Warteschlange ist größer als null).
9. Die Auslösersteuerung (Attribut **TriggerControl** der lokalen Warteschlange) für die Anwendungswarteschlange wird auf MQTC_ON gesetzt. Setzen Sie zu diesem Zweck das Attribut **trigger** bei der Definition Ihrer Warteschlange oder verwenden Sie den Befehl ALTER QLOCAL.
10. Der Auslösertyp (Attribut **TriggerType** der lokalen Warteschlange) ist nicht MQTT_NONE.

    Wenn alle erforderlichen Bedingungen erfüllt sind und die Nachricht, die die Auslöserbedingung verursacht hat, als Teil einer Arbeitseinheit eingereiht wird, wird die Auslösenachricht erst für den Abruf durch die Auslösemonitoranwendung verfügbar, wenn die Arbeitseinheit abgeschlossen ist, egal ob die Arbeitseinheit festgeschrieben oder, für Auslösertyp MQTT_FIRST oder MQTT_DEPTH, zurückgesetzt wird.
11. Für den **TriggerType** MQTT_FIRST oder MQTT_DEPTH wird eine geeignete Nachricht in die Warteschlange gestellt, wenn die Warteschlange
    - vorher nicht leer war (MQTT_FIRST) oder
    - **TriggerDepth** oder mehr Nachrichten enthielt (MQTT_DEPTH)

    und die Bedingungen „2” auf Seite 919 bis „10” auf Seite 921 (mit Ausnahme von „3” auf Seite 920) erfüllt sind, wenn im Fall von MQTT_FIRST seit der Ausgabe der letzten Auslösenachricht für diese Warteschlange ein ausreichender Zeitraum (Warteschlangenmanagerattribut **TriggerInterval**) vergangen ist.

Dadurch wird es einem Warteschlangenserver ermöglicht, vor Verarbeitung aller Nachrichten in der Warteschlange zu beenden. Der Zweck des Auslöseintervalls ist es, die Anzahl der erstellten duplizierten Auslösenachrichten zu reduzieren.

Anmerkung: Wenn Sie den Warteschlangenmanager stoppen und erneut starten, wird der Zeitgeber **TriggerInterval** zurückgesetzt. Es gibt ein kleines Fenster, währenddessen es möglich ist, zwei Auslösenachrichten zu erzeugen. Dieses Fenster entsteht, wenn das Auslöserattribut der Warteschlange aktiviert ist und zur selben Zeit eine Nachricht eintrifft, und die Warteschlange zuvor nicht leer war MQTT_FIRST) oder **TriggerDepth** oder mehr Nachrichten enthielt (MQTT_DEPTH).

12. Die einzige Anwendung, die eine Warteschlange bedient, gibt beim **TriggerType** MQTT_FIRST oder MQTT_DEPTH den Aufruf MQCLOSE aus, und die Warteschlange enthält mindestens:

    - eine (MQTT_FIRST) oder
    - **TriggerDepth** (MQTT_DEPTH)

    Nachrichten mit ausreichender Priorität (Bedingung „2" auf Seite 919), und die Bedingungen „6" auf Seite 921 bis „10" auf Seite 921 sind ebenfalls erfüllt.

    Dadurch wird einem Warteschlangenserver, der einen MQGET-Aufruf ausgibt, ermöglicht, zu enden, wenn er eine leere Warteschlange vorfindet; allerdings trifft im Intervall zwischen den MQGET- und MQCLOSE-Aufrufen mindestens eine Nachricht ein.

    Anmerkung:

    a. Wenn das Programm, das die Anwendungswarteschlange bedient, nicht alle Nachrichten abruft, kann eine geschlossene Schleife entstehen. Immer, wenn das Programm die Warteschlange schließt, erstellt der Warteschlangenmanager eine weitere Auslösenachricht, durch die der Auslösemonitor das Serverprogramm erneut ausführt.

    b. Wenn das Programm, dass die Anwendungswarteschlange bereitstellt, seine Abrufanforderung zurücksetzt (oder wenn das Programm abstürzt), bevor es die Warteschlange schließt, passiert dasselbe. Wenn das Programm die Warteschlange jedoch vor dem Zurücksetzen schließt und die Warteschlange ansonsten leer ist, wird keine Auslösenachricht erstellt.

    c. Verhindern lässt sich eine solche Schleife durch das MQMD-Feld *BackoutCount*, mit dessen Hilfe mehrfach zurückgesetzte Nachrichten erkannt werden. Weitere Informationen finden Sie unter „Zurückgesetzte Nachrichten" auf Seite 49.

13. Die folgenden Bedingungen werden mithilfe von MQSET oder durch einen Befehl erfüllt:

    a. • **TriggerControl** wird in MQTC_ON geändert oder
    • **TriggerControl** ist bereits MQTC_ON und der Wert von **TriggerType**, **TriggerMsgPriority** oder **TriggerDepth** (falls relevant) wird geändert

    und die Warteschlange enthält mindestens:

    - eine (MQTT_FIRST oder MQTT_EVERY) oder
    - **TriggerDepth** (MQTT_DEPTH)

    Nachrichten mit ausreichender Priorität (Bedingung „2" auf Seite 919), und die Bedingungen „4" auf Seite 920 bis „10" auf Seite 921 (mit Ausnahme von „8" auf Seite 921) sind ebenfalls erfüllt.

    Dadurch wird es einer Anwendung oder einem Operator ermöglicht, die Auslösekriterien zu ändern, wenn die Bedingungen für das Auftreten eines Auslösers bereits erfüllt sind.

    b. Das Warteschlangenattribut **InhibitPut** einer Initialisierungswarteschlange wird von MQQA_PUT_INHIBITED in MQQA_PUT_ALLOWED geändert und eine der Warteschlangen, für die diese Warteschlange die Initialisierungswarteschlange ist, enthält mindestens:

    - eine (MQTT_FIRST oder MQTT_EVERY) oder
    - **TriggerDepth** (MQTT_DEPTH)

    Nachrichten mit ausreichender Priorität (Bedingung „2" auf Seite 919), und die Bedingungen „4" auf Seite 920 bis „10" auf Seite 921 sind ebenfalls erfüllt. (Für jede dieser Warteschlangen, die die Bedingungen erfüllen wird eine Auslösenachricht erstellt.)

Dadurch werden Auslösenachrichten nicht wegen der Bedingung MQQA_PUT_INHIBITED in der Initialisierungswarteschlange erstellt, aber diese Bedingung wurde nun geändert.

c. Das Warteschlangenattribut **InhibitGet** einer Anwendungswarteschlange wird von MQQA_GET_INHIBITED in MQQA_GET_ALLOWED geändert und die Warteschlange enthält mindestens:

- eine (MQTT_FIRST oder MQTT_EVERY) oder
- **TriggerDepth** (MQTT_DEPTH)

Nachrichten mit ausreichender Priorität (Bedingung „2" auf Seite 919) in der Warteschlange und die Bedingungen „4" auf Seite 920 bis „10" auf Seite 921, ohne „5" auf Seite 920, sind ebenfalls erfüllt.

Dadurch können Anwendungen nur ausgelöst werden, wenn sie Nachrichten von der Anwendungswarteschlange abrufen können.

d. Eine Auslösemonitoranwendung gibt einen MQOPEN-Aufruf für eine Eingabe aus einer Initialisierungswarteschlange aus und eine der Anwendungswarteschlangen, für die diese Warteschlange die Initialisierungswarteschlange ist, enthält mindestens:

- eine (MQTT_FIRST oder MQTT_EVERY) oder
- **TriggerDepth** (MQTT_DEPTH)

Nachrichten mit ausreichender Priorität (Bedingung „2" auf Seite 919), und die Bedingungen „4" auf Seite 920 bis „10" auf Seite 921 (mit Ausnahme von „8" auf Seite 921) sind ebenfalls erfüllt, und keine andere Anwendung hat die Initialisierungswarteschlange für Eingaben geöffnet (für jede Warteschlange, die diese Bedingungen erfüllt, wird eine Auslösenachricht erstellt).

Dadurch können Nachrichten in der Warteschlange eingehen, obwohl der Auslösemonitor nicht ausgeführt wird, und der Warteschlangenmanager kann neu gestartet werden und (nicht persistente) Auslösenachrichten sind nicht mehr vorhanden.

14. MSGDLVSQ ist ordnungsgemäß eingestellt. Wenn Sie MSGDLVSQ=FIFO einstellen, werden Nachrichten nach dem Prinzip "First In First Out" der Warteschlange zugestellt. Die Priorität der Nachricht wird ignoriert und der Nachricht wird die Standardpriorität der Warteschlange zugewiesen. Wenn **TriggerMsgPriority** auf einen höheren Wert als die Standardpriorität der Warteschlange gesetzt ist, werden keine Nachrichten ausgelöst. Wenn **TriggerMsgPriority** kleiner oder gleich der Standardpriorität der Warteschlange ist, findet Auslösen für die Typen FIRST, EVERY und DEPTH statt. Informationen zu diesen Typen finden Sie in der Beschreibung des Felds **TriggerType** im Abschnitt „Auslöserereignisse steuern" auf Seite 924.

Wenn MSGDLVSQ=PRIORITY festgelegt ist, werden Nachrichten nur dann zu einem Auslöserereignis hochgezählt, wenn die Nachrichtenpriorität größer oder gleich dem Wert des Felds *TriggerMsgPriority* ist. In diesem Fall erfolgt das Auslösen für die Typen FIRST, EVERY und DEPTH. Werden beispielsweise 100 Nachrichten mit einer niedrigeren Priorität als **TriggerMsgPriority** eingereiht, ist die für eine Auslösung geltende effektive Warteschlangenlänge nach wie vor null. Wird nun eine Nachricht mit einer Priorität größer oder gleich der **TriggerMsgPriority** eingereiht, erhöht sich die effektive Warteschlangenlänge von null auf eins, d. h., die Bedingung für **TriggerType** FIRST ist erfüllt.

Anmerkungen:

1. Ab Schritt „12" auf Seite 922 (wo Auslösenachrichten in Folge eines Ereignisses ausgelöst werden - außer durch das Eintreffen einer Nachricht in der Anwendungswarteschlange), wird die Auslösenachricht nicht als Teil der Arbeitseinheit eingereiht. Außerdem wird, wenn **TriggerType** MQTT_EVERY ist und die Anwendungswarteschlange eine oder mehrere Nachrichten enthält, nur eine Auslösenachricht generiert.
2. Wenn IBM MQ eine Nachricht während des MQPUT segmentiert, wird ein Auslöserereignis erst dann verarbeitet, nachdem alle Segmente erfolgreich in die Warteschlange eingereiht wurden. Sobald sich aber Nachrichtensegmente in der Warteschlange befinden, behandelt IBM MQ diese hinsichtlich der Auslösefunktion als Einzelnachrichten. Wenn z. B. eine einzelne logische Nachricht in drei Teile aufgeteilt wird, wird nur ein Auslöserereignis verarbeitet, wenn es das erste MQPUT und segmentiert ist.

Jedes der drei Segmente bewirkt jedoch während seiner Übertragung im IBM MQ-Netz sein eigenes Auslöserereignis.

3. z/OS Wenn für IBM MQ for z/OSeine gemeinsam genutzte Warteschlange für das Auslösen konfiguriert ist und die Verbindung zu der Coupling-Facility, die die gemeinsam genutzte Warteschlange hostet, verloren geht, wird möglicherweise ein Auslöserereignis generiert und eine Nachricht in die Initialisierungswarteschlange eingereiht. Dies kann auch auftreten, wenn keine Nachricht zur Auslösung in die ursprüngliche Konfiguration der gemeinsam genutzten Warteschlange eingereiht wurde. Dies wird durch die Überanzeige von Bits durch das Makro IXLVECTR verursacht, wie in The List Notification Vectordokumentiert.

Auslöserereignisse steuern

Auslöserereignisse werden über einige der Attribute gesteuert, die eine Anwendungswarteschlange definieren. Die folgenden Informationen enthalten auch Beispiele für die Verwendung der Auslösertypen EVERY, FIRST und DEPTH.

Sie können die Auslösefunktion aktivieren und inaktivieren und Sie können die Anzahl oder Priorität der Nachrichten auswählen, die für ein Auslöserereignis gezählt werden. Eine ausführliche Beschreibung dieser Attribute finden Sie im Abschnitt Objektattribute.

Folgende Attribute sind relevant:

TriggerControl
Verwenden Sie dieses Attribut zum Aktivieren und Inaktivieren der Auslösefunktion für eine Anwendungswarteschlange.

TriggerMsgPriority
Die Mindestpriorität, die eine Nachricht haben muss, damit sie für ein Auslöserereignis gezählt wird. Wenn eine Nachricht mit einer niedrigeren Priorität als *TriggerMsgPriority* in der Anwendungswarteschlange eintrifft, wird diese vom Warteschlangenmanager bei der Ermittlung, ob eine Auslösenachricht erstellt werden soll, ignoriert. Wenn *TriggerMsgPriority* auf null gesetzt ist, werden alle Nachrichten für ein Auslöserereignis gezählt.

TriggerType
Zusätzlich zum Auslösertyp NONE (der die Auslösefunktion ebenso inaktiviert wie das Setzen von *TriggerControl* auf OFF), können Sie die Sensitivität einer Warteschlange für Auslöserereignisse mithilfe folgender Auslösertypen einstellen:

EVERY
Ein Auslöserereignis tritt immer dann ein, wenn eine Nachricht in der Anwendungswarteschlange eintrifft. Verwenden Sie diesen Auslösertyp, wenn mehrere Instanzen einer Anwendung gestartet werden sollen.

FIRST
Ein Auslöserereignis tritt nur dann ein, wenn sich die Anzahl der Nachrichten in der Anwendungswarteschlange von 0 nach eins ändert. Verwenden Sie diesen Auslösertyp, wenn ein Bereitstellungsprogramm gestartet werden soll, sobald die erste Nachricht in der Warteschlange eintrifft. Setzen Sie den Vorgang so lange fort, bis keine Nachrichten mehr zu verarbeiten sind. Sie müssen die Warteschlangen immer so lange verarbeiten, bis sie leer ist. Weitere Informationen hierzu finden Sie im Abschnitt „Sonderfall des Auslösertyps FIRST” auf Seite 925.

DEPTH
Ein Auslöserereignis tritt nur dann ein, wenn die Anzahl der Nachrichten in der Anwendungswarteschlange den Wert des Attributs **TriggerDepth** erreicht. Eine typische Verwendung dieses Auslösertyps besteht darin, ein Programm zu starten, nachdem alle Antworten für eine bestimmte Gruppe von Anfragen empfangen wurden.

Auslösen abhängig von der Länge: Wenn das Auslösen abhängig von der Länge erfolgt, inaktiviert der Warteschlangenmanager das Auslösen (mit dem Attribut *TriggerControl*) nach der Erstellung einer Auslösenachricht. Nach einem solchen Vorgang muss Ihre Anwendung die Auslösefunktion selbst erneut aktivieren (mit dem Aufruf MQSET).

Die Aktion der Inaktivierung der Auslösefunktion erfolgt nicht unter Synchronisationspunktsteuerung, d. h., die Auslösefunktion kann nicht erneut aktiviert werden, indem eine Arbeitseinheit zu-

rückgesetzt wird. Wenn ein Programm eine PUT-Anforderung zurücksetzt, die ein Auslöserereignis verursacht hat, oder wenn das Programm abnormal beendet wird, müssen Sie die Auslösefunktion mit dem Aufruf MQSET oder dem Befehl ALTER QLOCAL erneut aktivieren.

TriggerDepth
Die Anzahl der Nachrichten in einer Warteschlange, die ein Auslöserereignis verursacht, wenn das Auslösen abhängig von der Länge erfolgt.

Die Bedingungen, die erfüllt sein müssen, damit ein Warteschlangenmanager eine Auslösenachricht erstellt, werden im Abschnitt „Bedingungen für ein Auslöserereignis" auf Seite 919 beschrieben.

Beispiel für Verwendung des Auslösertyps EVERY

Das Beispiel geht von einer Anwendung aus, die Anfragen nach einer Autoversicherung generiert. Die Anwendung kann Anfragenachrichten an eine Reihe von Versicherungsgesellschaften senden, in denen jedesmal dieselbe Warteschlange für zu beantwortende Nachrichten angegeben ist. Sie kann für diese Empfangswarteschlange für Antworten den Auslösertyp EVERY festlegen, sodass bei jedem Eintreffen einer Antwort eine Instanz des Servers aufgefordert wird, die Antwort zu verarbeiten.

Beispiel für Verwendung des Auslösertyps FIRST

In diesem Beispiel wird ein Unternehmen mit mehreren Filialen angenommen, von denen jede Einzeldaten des Tagesgeschäfts an die Zentrale überträgt. Dies geschieht immer zur selben Zeit, am Ende des Geschäftstages, und in der Zentrale gibt es eine Anwendung, die die Einzeldaten von allen Filialen verarbeitet. Die erste Nachricht, die in der Zentrale eintrifft, kann ein Auslöserereignis verursachen, das diese Anwendung startet. Die Anwendung setzt die Verarbeitung dann so lange fort, bis keine Nachrichten mehr in der Warteschlange stehen.

Beispiel für Verwendung des Auslösertyps DEPTH

In diesem Beispiel gibt es eine Reisebüroanwendung, die eine einzelne Anforderung zur Bestätigung einer Flugreservierung, einer Hotelzimmerreservierung, eines Mietwagens und zur Bestellung einiger Reiseschecks erstellt. Die Anwendung kann diese Punkte auf vier einzelne Anforderungsnachrichten aufteilen und jede an eine andere Zieladresse senden. Sie kann für ihre Empfangswarteschlange für Antworten den Auslösertyp DEPTH festlegen (wobei die Länge auf den Wert 4 gesetzt wird), sodass sie nur erneut gestartet wird, wenn alle vier Antworten eingetroffen sind.

Falls eine andere Nachricht (möglicherweise zu einer anderen Anforderung) vor der letzten der vier Antworten in der Empfangswarteschlange für Antworten eintrifft, wird die anfordernde Anwendung früher ausgelöst. Um dies bei Verwendung des Auslösertyps DEPTH zum Sammeln mehrerer Antworten zu einer Anforderung zu vermeiden, sollte für jede Anforderung eine neue Empfangswarteschlange für Antworten verwendet werden.

Sonderfall des Auslösertyps FIRST

Falls bei Verwendung des Auslösertyps FIRST bereits eine Nachricht in der Anwendungswarteschlange steht, wenn eine weitere Nachricht eintrifft, erstellt der Warteschlangenmanager normalerweise keine weitere Auslösenachricht.

Die Anwendung, die für die Warteschlange zuständig ist, öffnet die Warteschlange aber in Wirklichkeit möglicherweise gar nicht (z. B. weil sie wegen eines Systemfehlers beendet wurde). Wenn im Prozessdefinitionsobjekt ein falscher Anwendungsname angegeben wurde, holt die für die Warteschlange zuständige Anwendung keine der Nachrichten ab. In solchen Situationen ist beim Eintreffen einer weiteren Nachricht in der Anwendungswarteschlange kein Server aktiv, der diese Nachricht (und alle weiteren Nachrichten in der Warteschlange) verarbeitet.

Um dies abzufangen, erstellt der Warteschlangenmanager unter folgenden Bedingungen weitere Auslösenachrichten:

- Wenn eine weitere Nachricht in der Anwendungswarteschlange eintrifft, aber nur wenn ein vordefiniertes Zeitintervall abgelaufen ist, seitdem der Warteschlangenmanager die letzte Auslösenachricht für

die Warteschlange erstellt hat. Dieses Zeitintervall wird mit dem Attribut *TriggerInterval* des Warteschlangenmanagers eingestellt. Der Standardwert sind 999 999 999 Millisekunden.

- z/OS Unter IBM MQ for z/OS werden Anwendungswarteschlangen, die sich auf eine offene Initialisierungswarteschlange beziehen, regelmäßig gescannt. Wenn seit dem letzten Senden einer Auslösenachricht *TRIGINT* Millisekunden vergangen sind, die Warteschlange die Bedingungen für ein Auslöserereignis erfüllt und CURDEPTH größer als null ist, wird eine Auslösenachricht generiert. Dieser Prozess wird als Backstop-Triggering bezeichnet.

Beachten Sie die folgenden Punkte, wenn Sie für Ihre Anwendung einen Wert für das Auslöseintervall festlegen:

- Wenn Sie *TriggerInterval* auf einen niedrigen Wert setzen und die Nachrichten der Anwendungswarteschlange von keiner Anwendung verarbeitet werden, hat Auslösertyp FIRST möglicherweise dieselbe Wirkung wie Auslösertyp EVERY. Dies hängt davon ab, in welchen Abständen Nachrichten in die Anwendungswarteschlange eingereiht werden, was wiederum von anderen Systemaktivitäten abhängig sein kann. Das hat folgende Ursache: Wenn das Auslöseintervall sehr kurz ist, wird bei jedem Einreihen einer Nachricht in die Anwendungswarteschlange eine weitere Auslösenachricht erstellt, auch wenn der Auslösertyp FIRST und nicht EVERY ist. (Auslösertyp FIRST mit einem Auslöseintervall null hat die gleiche Wirkung wie Auslösertyp EVERY.)
- z/OS Wenn Sie unter IBM MQ for z/OS den Wert *TRIGINT* auf einen niedrigen Wert setzen und keine Anwendung die Anwendungswarteschlange des Auslösertyps FIRST bedient, generiert die Backstop-Auslösung bei jedem regelmäßigen Scan von Anwendungswarteschlangen, die offene Initialisierungswarteschlangen benennen, eine Auslösenachricht.
- Wenn eine Arbeitseinheit zurückgesetzt wird (siehe Auslösenachrichten und Arbeitseinheiten) und das Auslöseintervall hoch bzw. auf den Standardwert eingestellt ist, wird beim Zurücksetzen der Arbeitseinheit eine Auslösenachricht erstellt. Wenn Sie das Auslöseintervall jedoch auf einen niedrigen Wert oder auf null (d. h. Auslösertyp FIRST verhält sich wie Auslösertyp EVERY) gesetzt haben, können viele Auslösenachrichten generiert werden. Wird die Arbeitseinheit zurückgesetzt, werden trotzdem alle Auslösenachrichten zur Verfügung gestellt. Wie viele Auslösenachrichten generiert werden, ist vom Auslöseintervall abhängig. Ist das Auslöseintervall auf 0 gesetzt, wird die maximale Anzahl Nachrichten generiert.

Anwendung entwickeln, die ausgelöste Warteschlangen verwendet

Sie haben gesehen, wie das Einrichten, Steuern und Auslösen bei Ihren Anwendungen funktioniert. Nun folgen ein paar Tipps, was Sie bei der Entwicklung Ihrer Anwendung beachten sollten.

Auslösenachrichten und Arbeitseinheiten

Auslösenachrichten, die durch Auslöserereignisse erstellt werden, die kein Teil einer Arbeitseinheit sind, werden in eine Initialisierungswarteschlange außerhalb der Arbeitseinheiten eingereiht, unabhängig von anderen Nachrichten, und sind für den Abruf durch den Auslösemonitor sofort verfügbar.

Auslösenachrichten, die durch Auslöserereignisse erstellt werden, die Teil einer Arbeitseinheit sind, werden in der Initialisierungswarteschlange zur Verfügung gestellt, wenn die Arbeitseinheit aufgelöst wird, egal ob die Arbeitseinheit festgeschrieben oder zurückgesetzt wird.

Wenn der Versuch des Warteschlangenmanagers, eine Auslösenachricht in eine Initialisierungswarteschlange einzureihen, fehlschlägt, wird sie in eine Warteschlange für nicht zustellbare Nachrichten eingereiht.

Anmerkung:

1. Der Warteschlangenmanager zählt sowohl festgeschriebene als auch nicht festgeschriebene Nachrichten, wenn er beurteilt, ob die Bedingungen für ein Auslöserereignis erfüllt werden.

   Durch das Auslösen von Typ FIRST oder DEPTH werden Auslösenachrichten zur Verfügung gestellt, sogar wenn die Arbeitseinheit zurückgesetzt wird, sodass eine Auslösenachricht immer verfügbar ist, wenn die erforderlichen Bedingungen erfüllt werden. Beispiel: Angenommen eine Put-Anforderung innerhalb eine Arbeitseinheit wird mit Auslösertyp FIRST ausgelöst. Dadurch wird der Warteschlan-

genmanager veranlasst, eine Auslösenachricht zu erstellen. Wenn eine weitere Put-Anforderung von einer anderen Arbeitseinheit erfolgt, wird dadurch kein weiteres Auslöserereignis veranlasst, weil die Anzahl der Nachrichten in der Anwendungswarteschlange sich nun von einer auf zwei Nachrichten geändert hat, wodurch die Bedingungen für ein Auslöserereignis nicht erfüllt werden. Wird nun die erste Arbeitseinheit zurückgesetzt, die zweite hingegen festgeschrieben, wird dennoch eine Auslösenachricht erstellt.

Dies bedeutet jedoch, dass Auslösenachrichten manchmal erstellt werden, wenn die Bedingungen für ein Auslöserereignis nicht erfüllt werden. Anwendungen, die das Auslösen verwenden, müssen auf diese Situation vorbereitet sein, um damit umgehen zu können. Es wird empfohlen, den MQGET-Aufruf mit Angabe der Option für Warten zu verwenden und im Feld *WaitInterval* einen geeigneten Wert festzulegen.

Erstellte Auslösenachrichten werden stets zur Verfügung gestellt, egal ob die Arbeitseinheit zurückgesetzt oder festgeschrieben wird.

2. Bei lokalen gemeinsam genutzten Warteschlangen (das heißt, gemeinsam genutzte Warteschlangen in einer Gruppe mit gemeinsamer Warteschlange) zählt der Warteschlangenmanager nur festgeschriebene Nachrichten.

Abrufen von Nachrichten aus einer ausgelösten Warteschlange

Bedenken Sie beim Entwickeln von Anwendungen, die mit Auslösevorgängen arbeiten, dass zwischen dem Start durch einen Auslösemonitor und der Verfügbarkeit der Nachrichten in einer Anwendungswarteschlange eine Verzögerung auftreten kann. Das kann passieren, wenn die Nachricht, die das Auslöserereignis veranlasst, vor den anderen festgeschrieben wird.

Verwenden Sie stets die Option für Warten, wenn Sie den MQGET-Aufruf zum Löschen von Nachrichten aus einer Warteschlange verwenden, für die die Auslöserbedingungen eingestellt sind, um den Nachrichten Zeit bis zum Eintreffen einzuräumen. Der Wert *WaitInterval* sollte so dimensioniert sein, dass er für die längste in angemessenem Rahmen anzunehmende Zeit zwischen der Einreihung einer Nachricht und der Festschreibung dieses Put-Aufrufs ausreicht. Wenn die Nachricht von einem fernen Warteschlangenmanager eintrifft, wird diese Zeit durch Folgendes beeinträchtigt:

- Die Anzahl der Nachrichten, die vor der Festschreibung eingereiht werden
- Die Geschwindigkeit und Verfügbarkeit der Kommunikationsverbindung
- Die Größe der Nachrichten

Beachten Sie dasselbe Beispiel, das wir für die Beschreibung der Arbeitseinheit verwendeten, ebenfalls für eine Situation, in der Sie den MQGET-Aufruf mit der Option für Warten verwenden. Dies war eine Put-Anforderung innerhalb einer Arbeitseinheit für eine Warteschlange, die mit dem Auslösertyp FIRST ausgelöst wird. Durch dieses Ereignis wird der Warteschlangenmanager veranlasst, eine Auslösenachricht zu erstellen. Wenn eine weitere Put-Anforderung von einer anderen Arbeitseinheit erfolgt, wird dadurch kein weiteres Auslöserereignis veranlasst, weil sich die Anzahl der Nachrichten in der Anwendungswarteschlange nicht von Null auf Eins geändert hat. Wird nun die erste Arbeitseinheit zurückgesetzt, die zweite hingegen festgeschrieben, wird dennoch eine Auslösenachricht erstellt. Daher wird die Auslösenachricht in dem Moment erstellt, in dem die erste Arbeitseinheit zurückgesetzt wird. Wenn es eine bedeutende Verzögerung gibt, bevor die zweite Nachricht festgeschrieben wird, muss die ausgelöste Anwendung möglicherweise darauf warten.

Beim Auslösen des Typs DEPTH kann eine Verzögerung auftreten, sogar wenn alle relevanten Nachrichten schließlich festgeschrieben werden. Angenommen, das Warteschlangenattribut **TriggerDepth** hat den Wert 2. Wenn in diesem Fall zwei Nachrichten in der Warteschlange eintreffen, bewirkt die zweite Nachricht die Generierung einer Auslösenachricht. Wenn jedoch die zweite Nachricht die erste Nachricht ist, die festgeschrieben werden soll, ist das der Moment, in der die Auslösenachricht wieder verfügbar ist. Der Auslösemonitor startet das Serverprogramm, aber das Programm kann die zweite Nachricht nur abrufen, bis die erste festgeschrieben wird. Daher muss das Programm möglicherweise warten, bis die erste Nachricht zur Verfügung gestellt wird.

Entwickeln Sie Ihre Anwendung so, dass sie beendet wird, wenn keine Nachrichten zum Abruf verfügbar sind, wenn Ihr Warteintervall abläuft. Wenn eine oder mehrere Nachrichten später eintreffen, können

Sie sich darauf verlassen, dass auf Ihre Anwendung erneut ausgelöst wird, um sie zu verarbeiten. Durch dieses Verfahren wird eine Inaktivität der Anwendung und eine unnötige Verwendung von Ressourcen vermieden.

Initialisierungswarteschlangenverarbeitung durch Auslösemonitore

Für einen Warteschlangenmanager ist ein Auslösemonitor eine Anwendung, die wie jede andere Anwendung eine Warteschlange bedient. Ein Auslösemonitor bedient jedoch Initialisierungswarteschlangen.

Ein Auslösemonitor ist normalerweise ein kontinuierlich ausgeführtes Programm. Wenn eine Auslösenachricht eine Initialisierungswarteschlange erreicht, erhält der Auslösemonitor diese Nachricht. Er verwendet Informationen in der Nachricht, um einen Befehl zum Starten einer Anwendung abzusetzen, die die Nachrichten in der Anwendungswarteschlange verarbeiten soll.

Der Auslösemonitor muss dem Programm, das die Anwendung startet, ausreichende Informationen übergeben, damit das Programm auf der Anwendungswarteschlange die richtigen Aktionen ausführen kann.

Ein Kanalinitiator ist ein Beispiel für einen Sondertyp des Auslösemonitors für Nachrichtenkanalagenten. In dieser Situation müssen Sie jedoch entweder den Auslösertyp FIRST oder den Auslösertyp DEPTH verwenden.

ALW *Auslösemonitore auf AIX, Linux, and Windows-Systemen*

Dieser Abschnitt enthält Informationen zu den auf AIX, Linux, and Windows-Systemen bereitgestellten Auslösemonitoren.

Die folgenden Auslösemonitore werden in der Serverumgebung bereitgestellt:

amqstrg0
: Dies ist ein Beispielauslösemonitor, der eine Teilmenge der von **runmqtrm** bereitgestellten Funktionen bietet. Weitere Informationen zu amqstrg0 finden Sie im Abschnitt „Beispielprogramme plattformübergreifend verwenden” auf Seite 1112.

runmqtrm
: Die Syntax dieses Befehls lautet **runmqtrm** [*-m QMgrName*] [*-q InitQ*], wobei QMgrName der Warteschlangenmanager und InitQ die Initialisierungswarteschlange ist. Die Standardwarteschlange ist die Warteschlange SYSTEM.DEFAULT.INITIATION.QUEUE auf dem Standardwarteschlangenmanager. Der Befehl ruft Programme für die geeigneten Auslösenachrichten auf. Dieser Auslösemonitor unterstützt den Standardanwendungstyp.

  Die vom Auslösemonitor zum Betriebssystem übergebene Befehlsfolge wird folgendermaßen erstellt:

  1. Die *ApplId* aus der relevanten PROCESS-Definition (sofern erstellt)
  2. Die MQTMC2-Struktur, eingeschlossen in Anführungszeichen
  3. Die *EnvData* aus der relevanten PROCESS-Definition (sofern erstellt)

  Dabei ist *ApplId* der Name des auszuführenden Programms, wie er in der Befehlszeile eingegeben werden würde.

  Der übergebene Parameter ist die MQTMC2-Zeichenstruktur. Eine Befehlsfolge mit exakt der dargestellten Zeichenfolge wird in doppelten Anführungszeichen aufgerufen, damit der Systembefehl diese als einen Parameter akzeptiert.

  Erst nach Abschluss der eben gestarteten Anwendung durchsucht der Auslösemonitor die Initialisierungswarteschlange wieder nach einer weiteren Nachricht. Falls die Anwendungsverarbeitung lange dauert, kommt der Auslösemonitor daher unter Umständen den in der Warteschlange auflaufenden Auslösenachrichten nicht mehr nach. Dem können Sie wie folgt vorbeugen:

  - Richten Sie weitere Auslösemonitore ein.
  - Führen Sie die gestarteten Anwendungen im Hintergrund aus.

Bei mehreren Auslösemonitoren können Sie die maximale Anzahl der gleichzeitig ausführbaren Anwendungen eingrenzen. Bei einer Hintergrundausführung grenzt IBM MQ die Anzahl der ausführbaren Anwendungen hingegen nicht ein.

Linux AIX Um die gestartete Anwendung im Hintergrund unter AIX and Linuxauszuführen, fügen Sie ein & am Ende der *EnvData* der PROCESS-Definition ein.

Zur Ausführung der gestarteten Anwendung im Hintergrund auf Windows-Systemen geben Sie im Feld *ApplId* als Präfix zum Anwendungsnamen einen START-Befehl ein. Beispiel:

```
START ?B AMQSECHA
```

Anmerkung: Windows Wenn ein Windows-Pfad Leerzeichen enthält, sollten Sie diesen in Anführungszeichen (") einschließen, um sicherzustellen, dass dieser als zusammengehöriges Argument behandelt wird. Beispiel: "`C:\Program Files\Application Directory\Application.exe`".

Nachfolgend sehen Sie ein Beispiel für eine APPLICID-Zeichenfolge, in der der Dateipfad Leerzeichen enthält:

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

Die Syntax des Windows-Befehls START in diesem Beispiel enthält eine leere, in Anführungszeichen eingeschlossene Zeichenfolge. Im Befehl START gibt das erste in Anführungszeichen eingeschlossene Argument den Titel des neuen Befehls an. Um sicherzustellen, dass Windows den Anwendungspfad nicht als 'title'-Argument fehlinterpretiert, sollten Sie dem Anwendungsnamen in diesem Befehl eine 'title'-Zeichenfolge in doppelten Anführungszeichen voranstellen.

Die folgenden Auslösemonitore werden für den IBM MQ-Client bereitgestellt:

runmqtmc
Dieser Monitor ist identisch zu runmqtrm, mit der Ausnahme, dass er eine Verknüpfung zu den IBM MQ MQI client-Bibliotheken herstellt.

ALW *Auslösemonitor für CICS*

Der Auslösemonitor amqltmc0 wird für CICS bereitgestellt. Er funktioniert genauso wie der Standardauslösemonitor runmqtrm, jedoch starten Sie ihn auf eine andere Weise, und er löst CICS-Transaktionen aus.

Dieser Abschnitt gilt nur für Windows-, AIX and Linux x86-64 -Systeme.

Der Auslösemonitor wird als CICS-Programm bereitgestellt. Sie müssen ihn mit einem vierstelligen Transaktionsnamen definieren. Zum Starten geben Sie einen vierstelligen Namen ein. Es verwendet den Standardwarteschlangenmanager (wie in der Datei qm.ini oder unter IBM MQ for Windowsin der Registry angegeben) und SYSTEM.CICS.INITIATION.QUEUE.

Wenn Sie einen anderen Warteschlangenmanager bzw. eine andere Warteschlange verwenden möchten, erstellen Sie die Auslösemonitorstruktur MQTMC2. Dazu müssen Sie ein Programm mit dem Aufruf EXEC CICS START schreiben, da die Struktur zu lang ist, um sie als Parameter hinzuzufügen. Diese MQTMC2-Struktur übergeben Sie dann als Daten an die START-Anforderung für den Auslösemonitor.

Bei Verwendung der MQTMC2-Struktur müssen Sie dem Auslösemonitor nur die Parameter *StrucId*, *Version*, *QName* und **QMgrName** übergeben, da er keine anderen Felder referenziert.

Die Nachrichten werden mit EXEC CICS START aus der Initialisierungswarteschlange eingelesen und zum Starten von CICS-Transaktionen verwendet (vorausgesetzt, APPL_TYPE in der Auslösenachricht ist MQAT_CICS). Das Einlesen der Nachrichten aus der Initialisierungswarteschlange erfolgt unter der Synchronisationspunktsteuerung von CICS.

Beim Starten und Stoppen des Monitors sowie bei Fehlern werden Nachrichten generiert. Diese Nachrichten werden an die CSMT-Warteschlange mit transienten Daten gesendet.

Tabelle 129. Verfügbare Versionen des Auslösemonitors.

Eine Tabelle mit zwei Spalten. In der ersten Spalte sind die verfügbaren Versionen des Auslösemonitors aufgeführt und in der zweiten Spalte ist angegeben, für welche Plattformen die einzelnen Versionen verwendet werden.

Version	**Verwenden Sie**
amqltmc0	TXSeries für: • AIX AIX • Linux Linux x86-64-Systeme
amqltmc4	Windows TXSeries für Windows 5.1
amqltmcc	Client-gebundene Version des CICS-Auslösemonitors
V 9.4.0 V 9.4.0 amqltmc064	64-Bit-TXSeries für Linux x86-64 -Systeme
V 9.4.0 V 9.4.0 amqltmcc64	Clientversion von amqltmc064

Wenn Sie einen Auslösemonitor für andere Umgebungen benötigen, müssen Sie ein Programm schreiben, das die vom Warteschlangenmanager in die Initialisierungswarteschlangen eingereihten Auslösenachrichten verarbeiten kann. Ein solches Programm sollte die folgenden Aktionen ausführen können:

1. Mit dem MQGET-Aufruf darauf warten, dass eine Nachricht in der Initialisierungswarteschlange eintrifft.
2. Die Felder der MQTM-Struktur der Auslösenachricht nach dem Namen der zu startenden Anwendung und der Umgebung, in der diese ausgeführt werden soll, durchsuchen.
3. Einen umgebungsspezifischen Startbefehl ausgeben.

   z/OS In einem z/OS-Stapel beispielsweise einen Job an den internen Leser übergeben.
4. Die MQTM-Struktur bei Bedarf in die MQTMC2-Struktur konvertieren.
5. Entweder die MQTMC2- oder die MQTM-Struktur an die gestartete Anwendung übergeben. Diese kann Benutzerdaten enthalten.
6. Der Anwendungswarteschlange die Anwendung zuordnen, die diese Warteschlange bedienen soll. Dazu geben Sie das Prozessdefinitionsobjekt (sofern erstellt) im Attribut **ProcessName** der Warteschlange an. Sie können das Prozessdefinitionsobjekt mit dem Befehl **DEFINE QLOCAL** oder **ALTER QLOCAL** benennen.

   IBM i Unter IBM i können Sie auch CRTMQMQ oder CHGMQMQ verwenden.

Weitere Informationen zur Auslösemonitorschnittstelle (TMI) finden Sie im Abschnitt MQTMC2.

IBM i *Auslösemonitore unter IBM i*

Verwenden Sie unter IBM ianstelle des Steuerbefehls **runmqtrm** den IBM MQ for IBM i CL-Befehl **STRMQMTRM**.

Verwenden Sie den Befehl STRMQMTRM wie folgt:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

Die Details sind identisch mit denen des Befehls runmqtrm.

Folgende Beispielprogramme sind verfügbar. Diese können Sie als Modelle für die Entwicklung eigener Auslösemonitore verwenden:

AMQSTRG4
Dies ist ein Auslösemonitor, der für den zu startenden Prozess einen IBM i-Job übergibt. Dies bedeutet jedoch für jede Auslösenachricht einen zusätzlichen Verarbeitungsaufwand.

AMQSERV4
Dies ist ein Auslöseserver. Dieser Server führt für jede Auslösenachricht den Befehl für den in seinem eigenen Job vorliegenden Prozess aus, wobei er auch CICS-Transaktionen aufrufen kann.

Sowohl der Auslösemonitor als auch der Auslöseserver übergeben den von ihnen gestarteten Programmen eine MQTMC2-Struktur. Eine Beschreibung dieser Struktur finden Sie im Abschnitt MQTMC2. Beide Beispiele werden sowohl als Quellcode als auch in ausführbarer Form bereitgestellt.

Das diese Auslösemonitore nur native IBM i-Programme aufrufen können, können sie Java-Programme nicht direkt aufrufen, denn die Java-Klassen befinden sich im IFS. Sehr wohl ist es allerdings möglich, Java-Programme indirekt durch Auslösen eines CL-Programms auszulösen, das dann wiederum das Java-Programm aufruft, wobei die Übergabe über die TMC2-Struktur erfolgt. Die Mindestgröße einer TMC2-Struktur ist 732 Byte.

Hier die Quelle des Beispiel-CLP:

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

Für den IBM MQ MQI client wird folgendes Auslösemonitorprogramm bereitgestellt: RUNMQTMC

Rufen Sie RUNMQTMC wie folgt auf:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

Eigenschaften von Auslösenachrichten

Im folgenden Abschnitt werden weitere Eigenschaften von Auslösenachrichten beschrieben.

- „Persistenz und Priorität von Auslösenachrichten” auf Seite 931
- „Neustart des Warteschlangenmanagers und Auslösenachrichten” auf Seite 931
- „Auslösenachrichten und Änderungen bei Objektattributen” auf Seite 932
- „Format von Auslösenachrichten” auf Seite 932

Persistenz und Priorität von Auslösenachrichten

Auslösenachrichten sind nicht persistent, weil es für sie nicht erforderlich ist.

Die Bedingungen für die Erstellung von Auslöserereignissen bleiben jedoch bestehen, sodass Auslösenachrichten erstellt werden, wenn diese Bedingungen erfüllt sind. Wenn eine Auslösenachricht verloren geht, garantiert das Fortbestehen der Anwendungsnachricht in der Anwendungswarteschlange, dass der Warteschlangenmanager eine Auslösenachricht erstellt, sobald alle Bedingungen erfüllt sind.

Wenn eine Arbeitseinheit zurückgesetzt wird, werden alle von ihr erstellten Auslösenachrichten stets zugestellt.

Auslösenachrichten übernehmen die Standardpriorität der Initialisierungswarteschlange.

Neustart des Warteschlangenmanagers und Auslösenachrichten

Wenn nach dem Neustart eines Warteschlangenmanagers die nächste Initialisierungswarteschlange für die Eingabe geöffnet wird, kann eine Auslösenachricht in diese Initialisierungswarteschlange eingereiht

werden, wenn es bei einer zugehörigen Anwendungswarteschlange Nachrichten gibt und sie für das Auslösen definiert ist.

Auslösenachrichten und Änderungen bei Objektattributen

Auslösenachrichten werden entsprechend der Werte der Auslöserattribute erstellt, die zum Zeitpunkt des Auslöserereignisses bestehen.

Wenn die Auslösenachricht nicht bis zu einem späteren Zeitpunkt verfügbar ist (weil die Nachricht, die sie ausgelöst hat, innerhalb einer Arbeitseinheit eingereiht wurde), haben alle zwischenzeitlichen Änderungen an den Auslöserattributen keine Auswirkungen auf die Auslösenachricht. Insbesondere die Inaktivierung der Auslösefunktion verhindert nicht, dass die Auslösenachricht zur Verfügung gestellt wird, sobald sie erstellt wurde. Außerdem ist die Anwendungswarteschlange möglicherweise nicht mehr in dem Moment vorhanden, in dem die Auslösenachricht zur Verfügung gestellt wird.

Format von Auslösenachrichten

Das Format von Auslösenachrichten wird durch die MQTM-Struktur definiert.

Dort gibt es folgende Felder, die der Warteschlangenmanager ausfüllt, wenn er die Auslösenachricht mithilfe der Informationen in den Objektdefinitionen der Anwendungswarteschlange und des Prozesses, der der Warteschlange zugeordnet ist, erstellt:

StrucId
Die Struktur-ID.

Version
Die Version der Struktur.

QName
Der Name der Anwendungswarteschlange, in der das Auslöserereignis aufgetreten ist. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, gibt er in diesem Feld das Attribut **QName** aus der Anwendungswarteschlange ein.

ProcessName
Der Name des Prozessdefinitionsobjekts, der der Anwendungswarteschlange zugeordnet ist. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, gibt er in diesem Feld das Attribut **ProcessName** aus der Anwendungswarteschlange ein.

TriggerData
Ein Feld mit freiem Format für die Verwendung durch den Auslösemonitor. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, gibt er in diesem Feld das Attribut **TriggerData** aus der Anwendungswarteschlange ein. Unter IBM MQ for Multiplatformskann in diesem Feld der Name des auszulösenden Kanals angegeben werden.

ApplType
Der Typ der Anwendung, die der Auslösemonitor starten soll. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, gibt er in diesem Feld das Attribut **ApplType** des Prozessdefinitionsobjekts ein, das in *ProcessName* angegeben wird.

ApplId
Eine Zeichenfolge, die die Anwendung angibt, die der Auslösemonitor starten soll. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, gibt er in diesem Feld das Attribut **ApplId** des Prozessdefinitionsobjekts ein, das in *ProcessName* angegeben wird.

Bei Verwendung des mit CICS bereitgestellten Auslösemonitors CKTI ist das Attribut **ApplId** des Prozessdefinitionsobjekts eine CICS-Transaktions-ID.

z/OS Wenn Sie CSQQTRMN verwenden, das von IBM MQ for z/OSbereitgestellt wird, ist das Attribut **ApplId** des Prozessdefinitionsobjekts eine IMS -Transaktions-ID.

EnvData
Ein Zeichenfeld, das umgebungsbezogene Daten für die Verwendung durch den Auslösemonitor enthält. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, gibt er in diesem Feld das

Attribut **EnvData** des Prozessdefinitionsobjekts ein, das in *ProcessName* angegeben wird. Der von CICSbereitgestellte Auslösemonitor (CKTI) oder der von IBM MQ for z/OSbereitgestellte Auslösemonitor (CSQQTRMN) verwendet dieses Feld nicht, aber andere Auslösemonitore verwenden es möglicherweise.

UserData
Ein Zeichenfeld, das Benutzerdaten enthält, die vom Auslösemonitor verwendet werden können. Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, gibt er in diesem Feld das Attribut **UserData** des Prozessdefinitionsobjekts ein, das in *ProcessName* angegeben wird. In diesem Feld kann der Name des Kanals angegeben werden, der ausgelöst werden soll.

Eine ausführliche Beschreibung der Auslösenachrichtenstruktur finden Sie im Abschnitt MQTM.

Wenn die Auslösung nicht funktioniert

Ein Programm wird nicht ausgelöst, wenn der Auslösemonitor das Programm nicht starten kann oder der Warteschlangenmanager die Auslösenachricht nicht übermitteln kann. So muss beispielsweise die Anwendungs-ID im Prozessobjekt angeben, dass das Programm im Hintergrund gestartet werden soll; andernfalls kann der Auslösemonitor das Programm nicht starten.

Wenn eine Auslösenachricht erstellt wird, aber nicht in die Initialisierungswarteschlange eingereiht werden kann (z. B., weil die Warteschlange voll ist oder die Auslösenachricht größer ist als die für die Initialisierungswarteschlange maximal erlaubte Nachrichtenlänge), wird die Auslösenachricht stattdessen in die Warteschlange für nicht zustellbare Nachrichten eingereiht.

Wenn die Einreihung in die Warteschlange für nicht zustellbare Nachrichten nicht erfolgreich abgeschlossen werden kann, wird die Auslösenachricht verworfen. Zudem wird eine Warnung an z/OS die z/OS-Konsole oder den Systembediener gesendet bzw. in das Fehlerprotokoll geschrieben.

Durch das Einreihen einer Auslösenachricht in die Warteschlange für nicht zustellbare Nachrichten kann auch für diese Warteschlange eine Auslösenachricht erstellt werden. Diese zweite Auslösenachricht wird verworfen, wenn auch sie eine Nachricht in die Warteschlange für nicht zustellbare Nachrichten einfügt.

Wenn das Programm erfolgreich ausgelöst wird, aber vor dem Eintreffen der Nachricht aus der Warteschlange beendet wird, können Sie die Ursache dieses Fehlers mit einem Trace-Programm ermitteln (z. B. mit CICS AUXTRACE, wenn das Programm unter CICS ausgeführt wird).

Mit MQI und Clustern arbeiten

In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

Unter den folgenden Links erhalten Sie weitere Informationen zu den Optionen, die in Verbindung mit Clustern für Aufrufe und Rückkehrcodes zur Verfügung stehen:

- „MQOPEN und Cluster” auf Seite 934
- „MQPUT, MQPUT1 und Cluster” auf Seite 935
- „MQINQ und Cluster” auf Seite 936
- „MQSET und Cluster” auf Seite 937
- „Rückkehrcodes” auf Seite 937

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 773
Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

„Objekte öffnen und schließen” auf Seite 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 792

In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Using and writing applications on IBM MQ for z/OS” auf Seite 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

„IMS and IMS bridge applications on IBM MQ for z/OS” auf Seite 73
This information helps you to write IMS applications using IBM MQ.

MQOPEN und Cluster

In welche Warteschlange eine Nachricht beim Öffnen einer Clusterwarteschlange eingereiht bzw. aus welcher Warteschlange sie abgerufen wird, hängt vom MQOPEN-Aufruf ab.

Zielwarteschlange auswählen

Wenn Sie im Objektdeskriptor MQOD keinen Warteschlangenmanagernamen angeben, wählt der Warteschlangenmanager den Warteschlangenmanager aus, an den eine Nachricht gesendet wird. Geben Sie hingegen im Objektdeskriptor den Namen eines Warteschlangenmanagers an, werden die Nachrichten immer an diesen gesendet.

Die Auswahl des Warteschlangenmanagers bezüglich des Zielwarteschlangenmanagers hängt von den Bindungsoptionen MQOO_BIND_* sowie davon ab, ob eine lokale Warteschlange vorhanden ist. Sofern eine lokale Instanz der Warteschlange vorhanden ist, wird diese gegenüber einer fernen Instanz bevorzugt geöffnet, es sei denn, das Attribut CLWLUSEQ ist auf ANY gesetzt. Andernfalls richtet sich die Auswahl nach den Bindungsoptionen. Bei Verwendung von Nachrichtengruppen mit Clustern muss entweder MQOO_BIND_ON_OPEN oder MQOO_BIND_ON_GROUP angegeben werden, um sicherzustellen, dass alle Nachrichten in der Gruppe an demselben Ziel verarbeitet werden.

Wenn der Warteschlangenmanager den Zielwarteschlangenmanager auswählt, macht er dies unter Verwendung des Auslastungsmanagementalgorithmus auf Round-Robin-Weise (Umlaufverfahren). Informationen hierzu finden Sie auch im Abschnitt Lastausgleich in Clustern.

Wann der Auslastungsmanagementalgorithmus verwendet wird, richtet sich danach, wie die Clusterwarteschlange geöffnet wird:

- MQOO_BIND_ON_OPEN - der Algorithmus wird nur einmal beim Öffnen der Warteschlange durch die Anwendung verwendet.
- MQOO_BIND_NOT_FIXED - der Algorithmus wird für jede in die Warteschlange eingereihte Nachricht verwendet.
- MQOO_BIND_ON_GROUP - der Algorithmus wird nur einmal beim Starten jeder Nachrichtengruppe verwendet.

MQOO_BIND_ON_OPEN
Die Option MQOO_BIND_ON_OPEN des MQOPEN-Aufrufs gibt an, dass der Zielwarteschlangenmanager festgelegt sein muss. Geben Sie MQOO_BIND_ON_OPEN an, wenn ein Cluster mehrere Instanzen der gleichen Warteschlange enthält. Alle Nachrichten, die mit der vom MQOPEN-Aufruf zurückgegebenen Objektkennung in die Warteschlange eingereiht werden, werden an den gleichen Warteschlangenmanager weitergeleitet.

- Verwenden Sie die Option MQOO_BIND_ON_OPEN, wenn Nachricht Affinitäten haben. Soll beispielsweise ein Nachrichtenstapel komplett von einem Warteschlangenmanager verarbeitet werden, geben Sie beim Öffnen der Warteschlange MQOO_BIND_ON_OPEN an. IBM MQ legt sowohl den Warteschlangenmanager fest als auch die Route, über die alle in diese Warteschlange eingereihten Nachrichten übertragen werden.
- Wenn die Option MQOO_BIND_ON_OPEN angegeben ist, muss die Warteschlange zur Auswahl einer neuen Instanz der Warteschlange neu geöffnet werden.

MQOO_BIND_NOT_FIXED
Die Option MQOO_BIND_NOT_FIXED des MQOPEN-Aufrufs gibt an, dass der Zielwarteschlangenmanager nicht festgelegt ist. Nachrichten, die in die Warteschlange geschrieben werden und die Objektkennung angeben, die vom MQOPEN -Aufruf zurückgegeben wird, werden zur MQPUT -Zeit auf Nachrichtenbasis an einen Warteschlangenmanager weitergeleitet. Verwenden Sie MQOO_BIND_NOT_FIXED, wenn Sie nicht möchten, dass alle Nachrichten an das gleiche Ziel übertragen werden.

- Die Optionen MQOO_BIND_NOT_FIXED und MQMF_SEGMENTATION_ALLOWED dürfen nicht gleichzeitig angegeben werden. Andernfalls werden die Segmente Ihrer Nachricht vermutlich verschiedenen, im Cluster verstreuten Warteschlangenmanagern zugestellt.

MQOO_BIND_ON_GROUP
Ermöglicht es einer Anwendung, zu fordern, dass eine Gruppe von Nachrichten derselben Zielinstanz zugeordnet wird. Diese Option ist nur für Warteschlangen gültig und betrifft nur Clusterwarteschlangen. Die Option wird ignoriert, wenn sie für eine Warteschlange angegeben wird, die keine Clusterwarteschlange ist.

- Gruppen werden nur dann dem gleichen Ziel zugestellt, wenn im MQPUT-Aufruf MQPMO_LOGICAL_ORDER angegeben ist. Wenn MQOO_BIND_ON_GROUP angegeben ist, aber eine Nachricht nicht Teil einer logischen Gruppe ist, wird stattdessen das Verhalten von BIND_NOT_FIXED angewendet.

MQOO_BIND_AS_Q_DEF
Wenn weder MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED noch MQOO_BIND_ON_GROUP angegeben ist, gilt die Standardoption MQOO_BIND_AS_Q_DEF. Bei Verwendung von MQOO_BIND_AS_Q_DEF wird die Bindung, die für die Warteschlangenkennung verwendet wird, aus dem Warteschlangenattribut DefBind übernommen.

Relevanz der MQOPEN-Optionen

Für die MQOPEN-Optionen MQOO_BROWSE, MQOO_INPUT_* und MQOO_SET ist eine lokale Instanz der Clusterwarteschlange erforderlich, damit der MQOPEN-Aufruf erfolgreich durchgeführt werden kann.

Für die MQOPEN-Optionen MQOO_OUTPUT, MQOO_BIND_* und MQOO_INQUIRE hingegen ist eine solche lokale Instanz der Clusterwarteschlange nicht erforderlich.

Aufgelöster Name des Warteschlangenmanagers

Wenn ein Warteschlangenmanagername beim Aufruf von MQOPEN aufgelöst wird, wird der aufgelöste Name an die Anwendung zurückgegeben. Verwendet die Anwendung diesen Namen beim nächsten MQOPEN-Aufruf, so kann sich herausstellen, dass die Anwendung über keine Berechtigung für diesen Namen verfügt.

MQPUT, MQPUT1 und Cluster

Bei Angabe von MQOO_BIND_NOT_FIXED im Aufruf MQOPEN entscheidet die Routine zur Auslastungsverwaltung, welches Ziel MQPUT oder MQPUT1 nehmen.

Bei Angabe von MQOO_BIND_NOT_FIXED im Aufruf MQOPEN ruft jeder nachfolgende MQPUT-Aufruf die Routine für Auslastungsverwaltung auf, die dann entscheidet, an welchen Warteschlangenmanager die Nachricht gesendet wird. Die Zieladresse und die Übertragungsroute wird für jede einzelne Nachricht individuell ermittelt. Nachdem eine Nachricht eingereiht wurde, können sich bei Änderungen der Netzbedingungen auch die Zieladresse und die Route ändern. Der Aufruf MQPUT1 wird immer so ausgeführt,

als ob die Option MQOO_BIND_NOT_FIXED angegeben wäre, d. h., bei jedem Aufruf wird die Routine zur Auslastungsverwaltung aufgerufen.

Nachdem von der Routine zur Auslastungsverwaltung ein Warteschlangenmanager ausgewählt wurde, führt der lokale Warteschlangenmanager den PUT-Vorgang zu Ende. Die Nachricht kann in verschiedene Warteschlangen eingereiht werden:

1. Wenn das Ziel die lokale Instanz der Warteschlange ist, wird die Nachricht in die lokale Warteschlange eingereiht.
2. Wenn das Ziel ein Warteschlangenmanager in einem Cluster ist, wird die Nachricht in eine Clusterübertragungswarteschlange eingereiht.
3. Wenn das Ziel ein Warteschlangenmanager außerhalb eines Clusters ist, wird die Nachricht in eine Übertragungswarteschlange mit dem Namen des Zielwarteschlangenmanagers eingereiht.

Bei Angabe von MQOO_BIND_ON_OPEN im MQOPEN-Aufruf wird durch nachfolgende MQPUT-Aufrufe die Routine zur Auslastungsverwaltung nicht aufgerufen, da Ziel und Route bereits ausgewählt sind.

MQINQ und Cluster

Welche Clusterwarteschlange abgefragt wird, ist abhängig von den Optionen, die Sie mit MQOO_INQUIRE kombinieren.

Bevor Sie eine Warteschlange abfragen, müssen Sie sie mithilfe des MQOPEN-Aufrufs öffnen und MQOO_INQUIRE angeben.

Zum Abfragen einer Clusterwarteschlange verwenden Sie den MQOPEN-Aufruf und kombinieren andere Optionen mit MQOO_INQUIRE. Welche Attribute abgefragt werden können, hängt davon ab, ob eine lokale Instanz der Clusterwarteschlange vorhanden ist, und davon, wie die Warteschlange geöffnet wird:

- Das Kombinieren von MQOO_BROWSE, MQOO_INPUT_* oder MQOO_SET mit MQOO_INQUIRE erfordert eine lokale Instanz der Clusterwarteschlange, damit das Öffnen erfolgreich ist. In diesem Fall können Sie alle Attribute abfragen, die für lokale Warteschlangen gültig sind.
- Wenn MQOO_OUTPUT mit MQOO_INQUIRE kombiniert und keine der vorhergehenden Optionen angegeben wird, wird eine der folgenden Instanzen geöffnet:
  - Die Instanz auf dem lokalen Warteschlangenmanager, falls vorhanden. In diesem Fall können Sie alle Attribute abfragen, die für lokale Warteschlangen gültig sind.
  - Eine Instanz an einem anderen Ort im Cluster, wenn es keine lokale Warteschlangenmanagerinstanz gibt. In diesem Fall nur können die folgenden Attribute abgefragt werden. Das Attribut QType hat hierbei den Wert MQQT_CLUSTER.

    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QType

Zum Abfragen des DefBind-Attributs einer Clusterwarteschlange verwenden Sie den MQINQ-Aufruf mit dem Selektor MQIA_DEF_BIND. Der zurückgegebene Wert lautet entweder MQBND_BIND_ON_OPEN oder MQBND_BIND_NOT_FIXED oder MQBND_BIND_ON_GROUP. EntwederMQBND_BIND_ON_OPEN oderMQBND_BIND_ON_GROUP muss bei der Verwendung von Gruppen mit Clustern angegeben werden.

Zum Abfragen der Attribute CLUSTER und CLUSNL der lokalen Instanz einer Warteschlange verwenden Sie den MQINQ-Aufruf mit dem Selektor MQCA_CLUSTER_NAME oder mit dem Selektor MQCA_CLUSTER_NAMELIST.

Anmerkung: Wenn Sie eine Clusterwarteschlange öffnen, ohne die Warteschlange zu korrigieren, an die sich der MQOPEN-Aufruf gebunden hat, könnten aufeinanderfolgende MQINQ-Aufrufe verschiedene Instanzen der Clusterwarteschlange abfragen.

Zugehörige Konzepte

„MQOPEN-Option für Clusterwarteschlangen” auf Seite 787

Die Bindung für die Warteschlangenkennung wird aus dem Warteschlangenattribut **DefBind** übernommen, das den Wert MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED oder MQBND_BIND_ON_GROUP haben kann.

MQSET und Cluster

Für die MQOPEN-Option MQOO_SET muss eine lokale Instanz der Clusterwarteschlange vorhanden sein, damit MQSET ausgeführt werden kann.

Mit dem Aufruf MQSET können Sie keine Attribute einer fernen Warteschlange im Cluster festlegen.

Allerdings können Sie einen lokalen Alias oder eine ferne Warteschlange, die durch das Clusterattribut definiert wurden, öffnen und dann den MQSET-Aufruf verwenden, um deren Attribute einzustellen. Dabei ist es unerheblich, ob die Zielwarteschlange eine auf einem anderen Warteschlangenmanager definierte Clusterwarteschlange ist.

Rückkehrcodes

Clusterspezifische Rückgabecodes

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

Zum Öffnen einer Clusterwarteschlange bzw. zum Einreihen einer Nachricht in eine Clusterwarteschlange wurde ein MQOPEN-, ein MQPUT- oder ein MQPUT1-Aufruf ausgegeben. Dabei schlägt der vom Attribut ClusterWorkloadExit eines Warteschlangenmanagers definierte Exit für Clusterauslastung unerwartet fehl oder reagiert nicht rechtzeitig.

z/OS Unter IBM MQ for z/OS wird eine Nachricht mit weiteren Informationen zu diesem Fehler in das Systemprotokoll geschrieben.

Nachfolgende MQOPEN-, MQPUT- und MQPUT1-Aufrufe für diese Warteschlangenkennung werden so ausgeführt, als ob für das Attribut ClusterWorkloadExit keine Angabe erfolgt wäre.

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

z/OS Unter z/OS kann der Exit für Clusterauslastung nicht geladen werden.

Eine Nachricht wird in das Systemprotokoll geschrieben und die Verarbeitung wird fortgesetzt, als ob für das Attribut ClusterWorkloadExit keine Angabe erfolgt wäre.

Multi Auf Multiplatforms-Systemen wird der Aufruf MQCONN oder MQCONNX ausgegeben, um einer Verbindung zu einem Warteschlangenmanager herzustellen. Der Aufruf schlägt fehl, weil der vom Attribut ClusterWorkloadExit eines Warteschlangenmanagers definierte Exit für Clusterauslastung nicht geladen werden kann.

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

Für eine Clusterwarteschlange wird ein MQOPEN-Aufruf mit den Optionen MQOO_OUTPUT und MQOO_BIND_ON_OPEN ausgegeben. Jedoch sind alle Instanzen der Warteschlange im Cluster für PUT-Vorgänge gesperrt, da das Attribut InhibitPut auf MQQA_PUT_INHIBITED gesetzt ist. Da keine Warteschlangeninstanzen für den Empfang von Nachrichten verfügbar sind, schlägt der MQOPEN-Aufruf fehl.

Dieser Ursachencode wird nur zurückgegeben, wenn die folgenden beiden Aussagen wahr sind:

- Es ist keine lokale Instanz der Warteschlange vorhanden. Ist eine lokale Instanz vorhanden, verläuft die Ausführung des MQOPEN-Aufrufs erfolgreich, selbst wenn die lokale Instanz für PUT-Vorgänge gesperrt ist.
- Für die Warteschlange ist kein Exit für Clusterauslastung vorhanden, oder der Exit ist zwar vorhanden, er wählt jedoch keine Warteschlangeninstanz aus. (Wenn der Exit für Clusterauslastung eine Warteschlangeninstanz ausgewählt, verläuft der MQOPEN-Aufruf erfolgreich, selbst wenn die Instanz für PUT-Vorgänge gesperrt ist.)

Ist im Aufruf MQOPEN die Option MQOO_BIND_NOT_FIXED angegeben, kann der Aufruf erfolgreich verlaufen, selbst wenn alle Warteschlangen im Cluster für PUT-Vorgänge gesperrt sind. Jedoch kann ein nachfolgender MQPUT-Aufruf fehlschlagen, wenn die Warteschlangen zum Zeitpunkt des Aufrufs noch immer für PUT-Vorgänge gesperrt sind.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. Zum Öffnen einer Clusterwarteschlange bzw. zum Einreihen einer Nachricht in eine Clusterwarteschlange wurde ein MQOPEN-, ein MQPUT- oder ein MQPUT1-Aufruf ausgegeben. Die Warteschlangendefinition kann nicht korrekt aufgelöst werden, da vom Warteschlangenmanager mit vollständigem Repository eine Antwort benötigt wird, die nicht eintrifft.
2. Ein Aufruf MQOPEN, MQPUT, MQPUT1 oder MQSUB wird für ein Themenobjekt ausgegeben, das PUBSCOPE (ALL) bzw. SUBSCOPE (ALL) angibt. Die Clusterthemendefinition kann nicht ordnungsgemäß aufgelöst werden, weil eine Antwort vom Warteschlangenmanager des vollständigen Repositorys erforderlich ist, aber keine verfügbar ist.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Für eine Clusterwarteschlange wurde ein MQOPEN-, MQPUT- oder MQPUT1-Aufruf ausgegeben. Beim Versuch, eine für das Clustering erforderliche Ressource zu verwenden, tritt ein Fehler auf.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Zum Einreihen einer Nachricht in eine Clusterwarteschlange wurde ein MQPUT- oder MQPUT1-Aufruf ausgegeben. Zum Zeitpunkt des Aufrufs befinden sich im Cluster jedoch keine Instanzen dieser Warteschlange mehr. Der MQPUT-Aufruf schlägt fehl, und die Nachricht wird nicht gesendet.

Dieser Fehler kann auftreten, wenn im Aufruf MQOO_BIND_NOT_FIXED zum Öffnen der Warteschlange die Option MQOPEN angegeben wurde, oder die Nachricht mit dem Aufruf MQPUT1 eingereiht wird.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Zum Öffnen einer Clusterwarteschlange bzw. zum Einreihen einer Nachricht in eine Clusterwarteschlange wurde ein MQOPEN-, MQPUT- oder MQPUT1-Aufruf ausgegeben. Der Aufruf wurde aber vom Exit für Clusterauslastung zurückgewiesen.

z/OS Using and writing applications on IBM MQ for z/OS

IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

This information explains the IBM MQ facilities available to programs running in each of the supported environments. In addition,

- For information about using the IBM MQ-CICS bridge, see Using IBM MQ with CICS .
- For information about using IMS and the IMS bridge, see “IMS and IMS bridge applications on IBM MQ for z/OS” on page 73.

Use the following links to find out more about using and writing applications on IBM MQ for z/OS:

- “Environment-dependent IBM MQ for z/OS functions” on page 939
- “Debugging facilities, syncpoint support, and recovery support” on page 940
- “The IBM MQ for z/OS interface with the application environment” on page 940
- “Writing z/OS UNIX System Services applications” on page 942
- “Application programming with shared queues” on page 945

Related concepts

“Message Queue Interface (MQI) - Übersicht” on page 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

“Verbindung zu einem Warteschlangenmanager herstellen und trennen” on page 773

Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

“Objekte öffnen und schließen” on page 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

“Nachrichten in eine Warteschlange einreihen” on page 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

“Nachrichten aus einer Warteschlange abrufen” on page 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

“Objektattribute abfragen und einstellen” on page 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

“Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” on page 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

“IBM MQ-Anwendungen durch Auslöser starten” on page 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

“Mit MQI und Clustern arbeiten” on page 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

“IMS and IMS bridge applications on IBM MQ for z/OS” on page 73
This information helps you to write IMS applications using IBM MQ.

z/OS *Environment-dependent IBM MQ for z/OS functions*

Use this information when considering IBM MQ for z/OS functions.

The main differences to be considered between IBM MQ functions in the environments in which IBM MQ for z/OS runs are:

- IBM MQ for z/OS supplies the following trigger monitors:
  - CKTI for use in the CICS environment
  - CSQQTRMN for use in the IMS environment

  You must write your own module to start applications in other environments.
- Syncpointing using two-phase commit is supported in the CICS and IMS environments. It is also supported in the z/OS batch environment using transaction management and recoverable resource manager services (RRS). Single-phase commit is supported in the z/OS environment by IBM MQ itself.
- For the batch and IMS environments, the MQI provides calls to connect programs to, and to disconnect them from, a queue manager. Programs can connect to more than one queue manager.
- A CICS system can connect to only one queue manager. This can be made to happen when CICS is initiated if the subsystem name is defined in the CICS system startup job. The MQI connect and disconnect calls are tolerated, but have no effect, in the CICS environment.
- The API-crossing exit allows a program to intervene in the processing of all MQI calls. This exit is available in the CICS environment only.
- In CICS on multiprocessor systems, some performance advantage is gained because MQI calls can be executed under multiple z/OS TCBs. For more information, see the Planung für z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

These features are summarized in Table 130 on page 939.

Table 130. z/OS environmental features

	CICS	**IMS**	**Batch/TSO**
Trigger monitor supplied	Yes	Yes	No

Table 130. z/OS environmental features (continued)			
	CICS	**IMS**	**Batch/TSO**
Two-phase commit	Yes	Yes	Yes
Single-phase commit	Yes	No	Yes
Connect/disconnect MQI calls	Tolerated	Yes	Yes
API-crossing exit	Yes	No	No

Note: Two-phase commit is supported in the Batch/TSO environment using RRS.

z/OS *Debugging facilities, syncpoint support, and recovery support*

Use this information to learn about program debugging facilities, syncpoint support, and recovery support.

Program debugging facilities

IBM MQ for z/OS provides a trace facility that you can use to debug your programs in all environments.

Additionally, in the CICS environment you can use:

- The CICS Execution Diagnostic Facility (CEDF)
- The CICS Trace Control Transaction (CETR)
- The IBM MQ for z/OS API-crossing exit

On the z/OS platform, you can use any available interactive debugging tool that is supported by the programming language that you are using.

Syncpoint support

Synchronizing the start and end of units of work is necessary in a transaction processing environment so that transaction processing can be used safely.

This is fully supported by IBM MQ for z/OS in the CICS and IMS environments. Full support means cooperation between resource managers so that units of work can be committed or backed out in unison, under control of CICS or IMS. Examples of resource managers are Db2, CICS File Control, IMS, and IBM MQ for z/OS.

z/OS batch applications can use IBM MQ for z/OS calls to give a single-phase commit facility. This means that an application-defined set of queue operations can be committed, or backed out, without reference to other resource managers.

Two-phase commit is also supported in the z/OS batch environment using transaction management and recoverable resource manager services (RRS). For further information see Syncpoints in z/OS batch applications.

Recovery support

If the connection between a queue manager and a CICS or IMS system is broken during a transaction, some units of work might not be backed out successfully.

However, these units of work are resolved by the queue manager (under the control of the syncpoint manager) when its connection with the CICS or IMS system is reestablished.

z/OS *The IBM MQ for z/OS interface with the application environment*

To allow applications running in different environments to send and receive messages through a message queuing network, IBM MQ for z/OS provides an *adapter* for each of the environments it supports.

These adapters are the interface between application programs and IBM MQ for z/OS subsystems. They allow the programs to use the MQI.

z/OS *The batch adapter*

Use this information to learn about the batch adapter and the commit protocol it supports.

The *batch adapter* provides access to IBM MQ for z/OS resources for programs running in:

- Task (TCB) mode
- Problem or supervisor state
- Primary address space control mode

The programs must not be in cross-memory mode.

Connections between application programs and IBM MQ for z/OS are at the task level. The adapter provides a single connection thread from an application task control block (TCB) to IBM MQ for z/OS.

The adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ for z/OS ; it does not support multiphase-commit protocols.

z/OS *The RRS batch adapter*

Use this information to learn about the RRS batch adapter and the two RRS batch adapters provided by IBM MQ.

The transaction management and recoverable resource manager services (RRS) adapter:

- Uses z/OS RRS for commit control.
- Supports simultaneous connections to multiple IBM MQ subsystems running on a single z/OS instance from a single task.
- Provides z/OS-wide coordinated commitment control (using z/OS RRS) for recoverable resources accessed through z/OS RRS-compliant recoverable managers for:
  - Applications that connect to IBM MQ using the RRS batch adapter.
  - Db2-stored procedures executing in a Db2-stored procedures address space that is managed by a workload manager (WLM) on z/OS.
- Supports the ability to switch an IBM MQ batch thread between TCBs.

IBM MQ for z/OS provides two RRS batch adapters:

CSQBRSTB
: This adapter requires you to change any MQCMIT statement to SRRCMIT and any MQBACK statement to SRRBACK in your IBM MQ application. (If you code MQCMIT or MQBACK in an application linked with CSQBRSTB, you receive MQRC_ENVIRONMENT_ERROR.)

CSQBRRSI
: This adapter allows your IBM MQ application to use either MQCMIT and MQBACK or SRRCMIT and SRRBACK.

Note: CSQBRSTB and CSQBRRSI are shipped with linkage attributes AMODE(31) RMODE(ANY). If your application loads either stub below the 16 MB line, first relink the stub with RMODE(24).

Migration

You can migrate existing Batch/TSO IBM MQ applications to use RRS coordination with few or no changes.

If you link-edit your IBM MQ application with the CSQBRRSI adapter, MQCMIT and MQBACK syncpoint your unit of work across IBM MQ and all other RRS-enabled resource managers. If you link-edit your IBM MQ application with the CSQBRSTB adapter, change MQCMIT to SRRCMIT and MQBACK to SRRBACK. The latter approach is preferable; it clearly indicates that the syncpoint is not restricted to IBM MQ resources only.

z/OS *The IMS adapter*

If you are using the IMS adapter from an IBM MQ for z/OS system, ensure that IMS can obtain sufficient storage to accommodate messages up to 100 MB long.

Note to users

The *IMS adapter* provides access to IBM MQ for z/OS resources for:

- Online message processing programs (MPPs)
- Interactive fast path programs (IFPs)
- Batch message processing programs (BMPs)

To use these resources, the programs must be running in task (TCB) mode and problem state; they must not be in cross-memory mode or 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 for z/OS, with IMS acting as the syncpoint coordinator.

The adapter also provides a trigger monitor program that can start programs automatically when certain trigger conditions on a queue are met. For more information, see "IBM MQ-Anwendungen durch Auslöser starten" on page 913.

If you are writing batch DL/I programs, follow the guidance given in this topic for z/OS batch programs.

z/OS *Writing z/OS UNIX System Services applications*

The batch adapter supports queue manager connections from batch and TSO address spaces.

For a batch address space, the adapter supports connections from multiple TCBs within that address space as follows:

- Each TCB can connect to multiple queue managers using the MQCONN or MQCONNX call (but a TCB can only have one instance of a connection to a particular queue manager at any one time).
- Multiple TCBs can connect to the same queue manager (but the queue manager handle returned on any MQCONN or MQCONNX call is bound to the issuing TCB and cannot be used by any other TCB).

z/OS UNIX System Services supports two types of pthread_create call:

1. Heavyweight threads, run one for each TCB, that are ATTACHed and DETACHed at thread start and end by z/OS.
2. Medium-weight threads, run one for each TCB, but the TCB can be one of a pool of long-running TCBs. The application must perform all necessary application cleanup, because, if it is connected to a server, the default thread termination that might be provided by the server at task (TCB) termination, is **not** always driven.

Lightweight threads are not supported. (If an application creates permanent threads that dispatch their own work requests, the ***application*** is responsible for cleaning up any resources before starting the next work request.)

IBM MQ for z/OS supports z/OS UNIX System Services threads using the Batch Adapter as follows:

1. Heavyweight threads are fully supported as batch connections. Each thread runs in its own TCB, which is attached and detached at thread start and end. Should the thread end before issuing an MQDISC call, IBM MQ for z/OS performs its standard task cleanup, which includes committing any outstanding unit of work if the thread terminated normally, or backing it out if the thread terminated abnormally.
2. Medium-weight threads are fully supported, but if the TCB is going to be reused by another thread, the application must ensure that an MQDISC call, preceded by either MQCMIT or MQBACK, is issued before the next thread start. This implies that if the application has established a Program Interrupt Handler, and the application then abends, the Interrupt Handler must issue MQCMIT and MQDISC calls before reusing the TCB for another thread.

Note: Threading models do **not** support access to common IBM MQ resources from multiple threads.

z/OS *The API-crossing exit for z/OS*

This topic contains product-sensitive programming interface information.

An exit is a point in IBM-supplied code where you can run your own code. IBM MQ for z/OS provides an *API-crossing exit* that you can use to intercept calls to the MQI, and to monitor or modify the function of the MQI calls. This section describes how to use the API-crossing exit, and describes the sample exit program that is supplied with IBM MQ for z/OS.

This section is applicable only for users of CICS TS V3.1 and earlier. Users of CICS TS V3.2 and later should refer to the section CICS Integration with IBM MQ in the CICS product documentation.

Note

The API-crossing exit is invoked only by the CICS adapter of IBM MQ for z/OS. The exit program runs in the CICS address space.

z/OS *Writing your own exit program*

You can use the sample API-crossing exit program (CSQCAPX) that is supplied with IBM MQ for z/OS as a framework for your own program.

This is described in "The sample API-crossing exit program, CSQCAPX" on page 944.

When writing an exit program, to find the name of an MQI call issued by an application, examine the *ExitCommand* field of the MQXP structure. To find the number of parameters on the call, examine the *ExitParmCount* field. You can use the 16-byte *ExitUserArea* field to store the address of any dynamic storage that the application obtains. This field is retained across invocations of the exit and has the same lifetime as a CICS task.

If you are using CICS Transaction Server V3.2, you must write your exit program to be threadsafe and declare your exit program as threadsafe. If you are using earlier CICS releases, you are also recommended to write and declare your exit programs as threadsafe to be ready for migrating to CICS Transaction Server V3.2.

Your exit program can suppress execution of an MQI call by returning MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION in the *ExitResponse* field. To allow the call to be executed (and the exit program to be reinvoked after the call has completed), your exit program must return MQXCC_OK.

When invoked after an MQI call, an exit program can inspect and modify the completion and reason codes set by the call.

Usage notes

Here are some general points to consider when writing your exit program:

- For performance reasons, write your program in assembler-language. If you write it in any of the other languages supported by IBM MQ for z/OS, you must provide your own data definition file.
- Link-edit your program as AMODE(31) and RMODE(ANY).
- To define the exit parameter block to your program, use the assembler-language macro, CMQXPA.
- Specify CONCURRENCY(THREADSAFE) when you define your exit program and any programs that your exit program calls.
- If you are using the CICS Transaction Server for z/OS storage protection feature, your program must run in CICS execution key. That is, you must specify EXECKEY(CICS) when defining both your exit program and any programs to which it passes control. For information about CICS exit programs and the CICS storage protection facility, see the *CICS Customization Guide*.
- Your program can use all the APIs (for example, IMS, Db2, and CICS) that a CICS task-related user exit program can use. It can also use any of the MQI calls except MQCONN, MQCONNX, and MQDISC. However, any MQI calls within the exit program do not invoke the exit program a second time.
- Your program can issue EXEC CICS SYNCPOINT or EXEC CICS SYNCPOINT ROLLBACK commands. However, these commands commit or roll back ***all*** the updates done by the task up to the point that the exit was used, and so their use is not recommended.
- Your program must end by issuing an EXEC CICS RETURN command. It must not transfer control with an XCTL command.

- Exits are written as extensions to the IBM MQ for z/OS code. Ensure that your exit does not disrupt any IBM MQ for z/OS programs or transactions that use the MQI. These are typically indicated with a prefix of CSQ or CK.
- If CSQCAPX is defined to CICS, the CICS system attempts to load the exit program when CICS connects to IBM MQ for z/OS. If this attempt is successful, message CSQC301I is sent to the CKQC panel or to the system console. If the load is unsuccessful (for example, if the load module does not exist in any of the libraries in the DFHRPL concatenation), message CSQC315 is sent to the CKQC panel or to the system console.
- Because the parameters in the communication area are addresses, the exit program must be defined as local to the CICS system (that is, not as a remote program).

z/OS *The sample API-crossing exit program, CSQCAPX*

The sample exit program is supplied as an assembler-language program. The source file (CSQCAPX) is supplied in the library **thlqual**.SCSQASMS (where **thlqual** is the high-level qualifier used by your installation). This source file includes pseudocode that describes the program logic.

The sample program contains initialization code and a layout that you can use when writing your own exit programs.

The sample shows how to:

- Set up the exit parameter block
- Address the call and exit parameter blocks
- Determine for which MQI call the exit is being invoked
- Determine whether the exit is being invoked before or after processing of the MQI call
- Put a message on a CICS temporary storage queue
- Use the macro DFHEIENT for dynamic storage acquisition to maintain reentrancy
- Use DFHEIBLK for the CICS exec interface control block
- Trap error conditions
- Return control to the caller

Design of the sample exit program

The sample exit program writes messages to a CICS temporary storage queue (CSQ1EXIT) to show the operation of the exit.

The messages show whether the exit is being invoked before or after the MQI call. If the exit is invoked after the call, the message contains the completion code and reason code returned by the call. The sample uses named constants from the CMQXPA macro to check on the type of entry (that is, before or after the call).

The sample does not perform any monitoring function, but simply places time-stamped messages into a CICS queue indicating the type of call it is processing. This provides an indication of the performance of the MQI, as well as the correct functioning of the exit program.

Note: The sample exit program issues six EXEC CICS calls for each MQI call that is made while the program is running. If you use this exit program, IBM MQ for z/OS performance is degraded.

z/OS *Preparing and using the API-crossing exit*

The sample exit is supplied in source form only.

To use the sample exit, or an exit program that you have written, create a load library, as you would for any other CICS program, as described in “Building CICS applications in z/OS” on page 1080.

- For CICS Transaction Server for z/OS and CICS for MVS™/ESA, when you update the CICS system definition (CSD) data set, the definitions you need are in the member **thlqual**.SCSQPROC(CSQ4B100).

Note: The definitions use a suffix of MQ. If this suffix is already used in your enterprise, this must be changed before the assembly stage.

If you use the default CICS program definitions supplied, the exit program CSQCAPX is installed in a ***disabled*** state. This is because using the exit program can produce a significant reduction in performance.

To activate the API-crossing exit temporarily:

1. Issue the command **CEMT S PROGRAM(CSQCAPX) ENABLED** from the CICS master terminal.
2. Run the CKQC transaction, and use option 3 in the Connection pull-down to alter the status of the API-crossing exit to **Enabled**.

If you want to run IBM MQ for z/OS with the API-crossing exit permanently enabled, with CICS Transaction Server for z/OS and CICS for MVS/ESA, do one of the following:

- Alter the CSQCAPX definition in member CSQ4B100, changing STATUS(DISABLED) to STATUS(ENABLED). You can update the CICS CSD definition using the CICS-supplied batch program DFHCSDUP.
- Alter the CSQCAPX definition in the CSQCAT1 group by changing the status from DISABLED to ENABLED.

In both cases, you must reinstall the group. You can do this by cold-starting your CICS system or by using the CICS CEDA transaction to reinstall the group while CICS is running.

Note: Using CEDA might cause an error if any of the entries in the group are currently in use.

End of product-sensitive programming interface information.

z/OS *Application programming with shared queues*

This topic provides information on some of the factors that you need to take into account when designing new applications to use shared queues, and when migrating existing applications to the shared-queue environment.

z/OS *Serializing your applications*

Certain types of applications might have to ensure that messages are retrieved from a queue in exactly the same order as they arrived on the queue.

For example, if IBM MQ is being used to shadow database updates on to a remote system, a message describing the update to a record must be processed after a message describing the insert of that record. In a local queuing environment, this is often achieved by the application that is getting the messages opening the queue with the MQOO_INPUT_EXCLUSIVE option, thus preventing any other getting application from processing the queue at the same time.

IBM MQ allows applications to open shared queues exclusively in the same way. However, if the application is working from a partition of a queue (for example, all database updates are on the same queue, but those for table A have a correlation identifier of A, and those for table B a correlation identifier of B), and applications want to get messages for table A updates and table B updates concurrently, the simple mechanism of opening the queue exclusively is not possible.

If this type of application is to take advantage of the high availability of shared queues, you might decide that another instance of the application that accesses the same shared queues, running on a secondary queue manager, should take over if the primary getting application or queue manager fails.

If the primary queue manager fails, two things happen:

- Shared queue peer recovery ensures that any incomplete updates from the primary application are completed or backed out.
- The secondary application takes over processing the queue.

The secondary application might start before all the incomplete units of work have been dealt with, which could lead to the secondary application retrieving the messages out of sequence. To solve this type of problem, the application can choose to be a *serialized application*.

A serialized application uses the MQCONNX call to connect to the queue manager, specifying a connection tag when it connects that is unique to that application. Any units of work performed by the application are marked with the connection tag. IBM MQ ensures that units of work within the queue sharing group with the same connection tag are serialized (according to the serialization options on the MQCONNX call).

This means that, if the primary application uses the MQCONNX call with a connection tag of `Database shadow retriever`, and the secondary takeover application attempts to use the MQCONNX call with an identical connection tag, the secondary application cannot connect to the second IBM MQ until any outstanding primary units of work have been completed, in this case by peer recovery.

Consider using the serialized application technique for applications that depend on the exact sequence of messages on a queue. In particular:

- Applications that must not restart after an application or queue manager failure until all commit and backout operations for the previous execution of the application are complete.

  In this case, the serialized application technique is only applicable if the application works in syncpoint.
- Applications that must not start while another instance of the same application is already running.

  In this case, the serialized application technique is only required if the application cannot open the queue for exclusive input.

Note: IBM MQ only guarantees to preserve the sequence of messages when certain criteria are met. These are described in the description of MQGET.

z/OS *Applications that are not suitable for use with shared queues*

Some features of IBM MQ are not supported when you are using shared queues, so applications that use these features are not suitable for the shared queue environment.

Consider the following points when designing your shared-queue applications:

- Queue indexing is limited for shared queues. If you want to use the message identifier or correlation identifier to select the message that you want to get from the queue, the queue should be indexed with the correct value. If you are selecting messages by message identifier alone, the queue needs an index type of MQIT_MSG_ID (although you can also use MQIT_NONE). If you are selecting messages by correlation identifier alone, the queue must have an index type of MQIT_CORREL_ID.
- You cannot use temporary dynamic queues as shared queues. However, you can use permanent dynamic queues. The models for shared dynamic queues have a DEFTYPE of SHAREDYN (shared dynamic) although they are created and destroyed in the same way as PERMDYN (permanent dynamic) queues.

z/OS *Deciding whether to share non-application queues*

Use this information when considering sharing non-application queues.

There are queues other than application queues that you might want to consider sharing:

Initiation queues

If you define a shared initiation queue, you do not need to have a trigger monitor running on every queue manager in the queue sharing group, as long as there is at least one trigger monitor running. (You can also use a shared initiation queue even if there is a trigger monitor running on each queue manager in the queue sharing group.)

If you have a shared application queue and use the trigger type of EVERY (or a trigger type of FIRST with a small trigger interval, which behaves like a trigger type of EVERY) your initiation queue must always be a shared queue. For more information about when to use a shared initiation queue, see Table 131 on page 947.

SYSTEM.* queues

You can define the SYSTEM.ADMIN.* queues used to hold event messages as shared queues. This can be useful to check load balancing if an exception occurs. Each event message created by IBM MQ contains a correlation identifier indicating which queue manager produced it.

You must define the SYSTEM.QSG.* queues used for shared channels and intra-group queuing as shared queues.

You can also change the definitions of the SYSTEM.DEFAULT.LOCAL.QUEUE to be shared, or define your own default shared queue definition. See Defining system objects for IBM MQ for z/OS for more information.

You cannot define any other SYSTEM.* queues as shared queues.

z/OS *Migrating your existing applications to use shared queues*

Reason codes, triggering, and the MQINQ API call can work differently in a shared queue environment.

See Migrating non-shared queues to shared queues for information on migrating your existing queues to shared queues.

When you migrate your existing applications, consider the following things, which might work in a different way in the shared queue environment:

Reason codes

When you migrate your existing applications to use shared queues, check for the new reason codes that can be issued.

Triggering

If you are using a shared application queue, triggering works on committed messages only (on a non-shared application queue, triggering works on all messages).

If you use triggering to start applications, you might want to use a shared initiation queue. Table 131 on page 947 describes what you need to consider when deciding which type of initiation queue to use.

Table 131. When to use a shared-initiation queue

	Non-shared application queue	**Shared application queue**
Non-shared initiation queue	As for previous releases.	If you use a trigger type of FIRST or DEPTH, you can use a non-shared initiation queue with a shared application queue. Extra trigger messages might be generated, but this setup is good for triggering long-running applications (like the CICS bridge) and provides high availability. For trigger type FIRST or DEPTH, a trigger message triggers an instance of the application on every queue manager that is running a trigger monitor and that does not already have the application queue open for input. One trigger message is generated for every queue manager; if there is more than one trigger monitor running against the non-shared local initiation queue, on a particular queue manager, they will compete to process the message.

Table 131. When to use a shared-initiation queue (continued)		
	Non-shared application queue	**Shared application queue**
Shared initiation queue	Do not use a shared initiation queue with a non-shared application queue.	For trigger type EVERY, when an application puts a message to a shared application queue, the putting queue manager determines which queue managers have an interest in the trigger-every event and sends a notification to one of those queue managers. On the notified queue manager, the resulting action is to generate a trigger message to the initiation queue., **Note:** If you have a shared application queue that has a trigger type of EVERY, use a shared initiation queue, or you might lose trigger messages in certain circumstances; for example, a queue manager failing. For trigger type FIRST or DEPTH, one trigger message is generated by each queue manager that has the named initiation queue open for input. **Note:** For trigger type FIRST or DEPTH, if one trigger monitor instance is busy, this leaves the potential for less busy trigger monitors to process more than one trigger message from the shared initiation queue. Hence, multiple instances of the server application may be started against a given queue manager. Note that these multiple instances are started as a result of processing multiple trigger messages. Ordinarily, for trigger type FIRST or DEPTH, if an application instance is already serving an application queue, another trigger message will not be generated by the queue manager that the application is connected to.

MQINQ

When you use the MQINQ call to display information about a shared queue, the values of the number of MQOPEN calls that have the queue open for input and output relate only to the queue manager that issued the call. No information is produced about other queue managers in the queue sharing group that have the queue open.

z/OS IMS and IMS bridge applications on IBM MQ for z/OS

This information helps you to write IMS applications using IBM MQ.

- To use syncpoints and MQI calls in IMS applications, see “Writing IMS applications using IBM MQ” on page 74.
- To write applications that use the IBM MQ - IMS bridge, see “Writing IMS bridge applications” on page 78.

Use the following links to find out more about IMS and IMS bridge applications on IBM MQ for z/OS:

- “Writing IMS applications using IBM MQ” on page 74
- “Writing IMS bridge applications” on page 78

Related concepts

“Message Queue Interface (MQI) - Übersicht” on page 759
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

“Verbindung zu einem Warteschlangenmanager herstellen und trennen” on page 773

Damit IBM MQ-Programmierservices verwendet werden können, muss ein Programm mit einem Warteschlangenmanager verbunden sein. Dieser Abschnitt enthält Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager.

“Objekte öffnen und schließen” on page 780
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

“Nachrichten in eine Warteschlange einreihen” on page 792
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

“Nachrichten aus einer Warteschlange abrufen” on page 808
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

“Objektattribute abfragen und einstellen” on page 897
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

“Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” on page 901
In diesem Abschnitt erfahren Sie, wie wiederherstellbare Get- und Put-Operationen, die innerhalb einer Arbeitseinheit ausgeführt wurden, per Commit festgeschrieben bzw. per Backout zurückgesetzt werden.

“IBM MQ-Anwendungen durch Auslöser starten” on page 913
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

“Mit MQI und Clustern arbeiten” on page 933
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

“Using and writing applications on IBM MQ for z/OS” on page 938
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

z/OS *Writing IMS applications using IBM MQ*

There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

Use the following links to find out more about writing IMS applications on IBM MQ for z/OS:

- “Syncpoints in IMS applications” on page 74
- “MQI calls in IMS applications” on page 75

Restrictions

There are restrictions on which IBM MQ API calls can used by an application using the IMS adapter.

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

Related concepts

“Writing IMS bridge applications” on page 78
This topic contains information about writing applications to use the IBM MQ - IMS bridge.

z/OS *Syncpoints in IMS applications*

In an IMS application, you establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint).

To back out all changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see ROLB call in the IMS documentation.

The queue manager is a participant in a two-phase commit protocol; the IMS syncpoint manager is the coordinator.

All open handles are closed by the IMS adapter at a syncpoint (except in a batch or non-message driven BMP environment). This is because a different user could initiate the next unit of work and IBM MQ security checking is performed when the MQCONN, MQCONNX, and MQOPEN calls are made, not when the MQPUT or MQGET calls are made.

However, in a Wait-for-Input (WFI) or pseudo Wait-for-Input (PWFI) environment IMS does not notify IBM MQ to close the handles until either the next message arrives or a QC status code is returned to the application. If the application is waiting in the IMS region and any of these handles belong to triggered queues, triggering will not occur because the queues are open. For this reason, applications running in a WFI or PWFI environment should explicitly MQCLOSE the queue handles before doing the GU to the IOPCB for the next message.

If an IMS application (either a BMP or an MPP) issues the MQDISC call, open queues are closed but no implicit syncpoint is taken. If the application ends normally, any open queues are closed and an implicit commit occurs. If the application ends abnormally, any open queues are closed and an implicit backout occurs.

z/OS *MQI calls in IMS applications*

Use this information to learn about the use of MQI calls on Server applications and Enquiry applications.

This section covers the use of MQI calls in the following types of IMS applications:

- “Server applications” on page 950
- “Inquiry applications” on page 952

Server applications

Here is an outline of the MQI server application model:

```
Initialize/Connect
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

Sample program CSQ4ICB3 shows the implementation, in C/370, of a BMP using this model. The program establishes communication with IMS first, and then with IBM MQ:

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

The IMS initialization determines whether the program has been called as a message-driven or a batch-oriented BMP and controls IBM MQ queue manager connection and queue handles accordingly:

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

The IBM MQ initialization connects to the queue manager and opens the queues. In a message-driven BMP this is called after each IMS syncpoint is taken; in a batch-oriented BMP, this is called only during program startup:

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

The implementation of the server model in an MPP is influenced by the fact that the MPP processes a single unit of work per invocation. This is because, when a syncpoint (GU) is taken, the connection and queue handles are closed and the next IMS message is delivered. This limitation can be partially overcome by one of the following:

- **Processing many messages within a single unit-of-work**

  This involves:

  – Reading a message
  – Processing the required updates

– Putting the reply

in a loop until all messages have been processed or until a set maximum number of messages has been processed, at which time a syncpoint is taken.

Only certain types of application (for example, a simple database update or inquiry) can be approached in this way. Although the MQI reply messages can be put with the authority of the originator of the MQI message being handled, the security implications of any IMS resource updates need to be addressed carefully.

- **Processing one message per invocation of the MPP and ensuring multiple scheduling of the MPP to process all available messages.**

  Use the IBM MQ IMS trigger monitor program (CSQQTRMN) to schedule the MPP transaction when there are messages on the IBM MQ queue and no applications serving it.

  If trigger monitor starts the MPP, the queue manager name and queue name are passed to the program, as shown in the following COBOL code extract:

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

The server model, which is expected to be a long running task, is better supported in a batch processing region, although the BMP cannot be triggered using CSQQTRMN.

Inquiry applications

A typical IBM MQ application initiating an inquiry or update works as follows:

- Gather data from the user
- Put one or more IBM MQ messages
- Get the reply messages (you might have to wait for them)
- Provide a response to the user

Because messages put on to IBM MQ queues do not become available to other IBM MQ applications until they are committed, they must either be put out of syncpoint, or the IMS application must be split into two transactions.

If the inquiry involves putting a single message, you can use the *no syncpoint* option; however, if the inquiry is more complex, or resource updates are involved, you might get consistency problems if failure occurs and you do not use syncpointing.

To overcome this, you can split IMS MPP transactions using MQI calls using a program-to-program message switch; see *IMS Intersystem Communication (ISC)* for information about this. This allows an inquiry program to be implemented in an MPP:

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS *Writing IMS bridge applications*

This topic contains information about writing applications to use the IBM MQ - IMS bridge.

For information about the IBM MQ - IMS bridge, see The IMS bridge.

Use the following links to find out more about writing IMS bridge applications on IBM MQ for z/OS:

- “How the IMS bridge deals with messages” on page 78
- “Writing IMS transaction programs through IBM MQ” on page 960

Related concepts

“Writing IMS applications using IBM MQ” on page 74
There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

z/OS *How the IMS bridge deals with messages*

When you use the IBM MQ - IMS bridge to send messages to an IMS application, you need to construct your messages in a special format.

You must also put your messages on IBM MQ queues that have been defined with a storage class that specifies the XCF group and member name of the target IMS system. These are known as MQ-IMS bridge queues, or simply **bridge** queues.

The IBM MQ-IMS bridge requires exclusive input access (MQOO_INPUT_EXCLUSIVE) to the bridge queue if it is defined with QSGDISP(QMGR), or if it is defined with QSGDISP(SHARED) together with the NOSHARE option.

A user does not need to sign on to IMS before sending messages to an IMS application. The user ID in the *UserIdentifier* field of the MQMD structure is used for security checking. The level of checking is determined when IBM MQ connects to IMS, and is described in Application access control for the IMS bridge. This enables a pseudo signon to be implemented.

The IBM MQ - IMS bridge accepts the following types of message:

- Messages containing IMS transaction data and an MQIIH structure (described in MQIIH):

```
MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
```

Note:

1. The square brackets, [ ], represent optional multi-segments.
2. Set the *Format* field of the MQMD structure to MQFMT_IMS to use the MQIIH structure.

- Messages containing IMS transaction data but no MQIIH structure:

```
LLZZ<trancode><data> \
[LLZZ<data>][LLZZ<data>]
```

IBM MQ validates the message data to ensure that the sum of the LL bytes plus the length of the MQIIH (if it is present) is equal to the message length.

When the IBM MQ - IMS bridge gets messages from the bridge queues, it processes them as follows:

- If the message contains an MQIIH structure, the bridge verifies the MQIIH (see MQIIH), builds the OTMA headers, and sends the message to IMS. The transaction code is specified in the input message. If this is an LTERM, IMS replies with a DFS1288E message. If the transaction code represents a command, IMS executes the command; otherwise the message is queued in IMS for the transaction.
- If the message contains IMS transaction data, but no MQIIH structure, the IMS bridge makes the following assumptions:
  - The transaction code is in bytes 5 through 12 of the user data
  - The transaction is in nonconversational mode
  - The transaction is in commit mode 0 (commit-then-send)
  - The *Format* in the MQMD is used as the *MFSMapName* (on input)
  - The security mode is MQISS_CHECK

  The reply message is also built without an MQIIH structure, taking the *Format* for the MQMD from the *MFSMapName* of the IMS output.

The IBM MQ - IMS bridge uses one or two Tpipes for each IBM MQ queue:

- A synchronized Tpipe is used for all messages using Commit mode 0 (COMMIT_THEN_SEND) (these show with SYN in the status field of the IMS /DIS TMEMBER client TPIPE xxxx command)
- A non-synchronized Tpipe is used for all messages using Commit mode 1 (SEND_THEN_COMMIT)

The Tpipes are created by IBM MQ when they are first used. A non-synchronized Tpipe exists until IMS is restarted. Synchronized Tpipes exist until IMS is cold started. You cannot delete these Tpipes yourself.

See the following topics for more information about how the IBM MQ - IMS bridge deals with messages:

- “Mapping IBM MQ messages to IMS transaction types” on page 80
- “If the message cannot be put to the IMS queue” on page 80
- “IMS bridge feedback codes” on page 80
- “The MQMD fields in messages from the IMS bridge” on page 81
- “The MQIIH fields in messages from the IMS bridge” on page 82
- “Reply messages from IMS” on page 83
- “Using alternate response PCBs in IMS transactions” on page 83
- “Sending unsolicited messages from IMS” on page 83
- “Message segmentation” on page 83
- “Data conversion for messages to and from the IMS bridge” on page 84

Related concepts

“Writing IMS transaction programs through IBM MQ” on page 960
The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

z/OS *Mapping IBM MQ messages to IMS transaction types*

A table describing the mapping of IBM MQ messages to IMS transaction types.

Table 132. How IBM MQ messages map to IMS transaction types

IBM MQ message type	Commit-then-send (mode 0) - uses synchronized IMS Tpipes	Send-then-commit (mode 1) - uses non-synchronized IMS Tpipes
Persistent IBM MQ messages	• Recoverable full function transactions • Unrecoverable transactions are rejected by IMS	• Fastpath transactions • Conversational transactions • Full function transactions
Nonpersistent IBM MQ messages	• Unrecoverable full function transactions • Recoverable transactions are permitted with IMS V8 and APAR PQ61404 and all later versions of IMS	• Fastpath transactions • Conversational transactions • Full function transactions

Note: IMS commands cannot use persistent IBM MQ messages with commit mode 0. See Commit mode (commitMode) for more information.

z/OS *If the message cannot be put to the IMS queue*

Learn about actions to take if the message cannot be put to the IMS queue.

If the message cannot be put to the IMS queue, the following action is taken by IBM MQ:

- If a message cannot be put to IMS because the message is invalid, the message is put to the dead-letter queue, and a message is sent to the system console.
- If the message is valid, but is rejected by IMS, IBM MQ sends an error message to the system console, the message includes the IMS sense code, and the IBM MQ message is put to the dead-letter queue. If the IMS sense code is 001A, IMS sends an IBM MQ message containing the reason for the failure to the reply-to queue.

  Note: In the circumstances listed previously, if IBM MQ cannot put the message to the dead-letter queue for any reason, the message is returned to the originating IBM MQ queue. An error message is sent to the system console, and no further messages are sent from that queue.

  To resend the messages, do ***one*** of the following:

  – Stop and restart the Tpipes in IMS corresponding to the queue
  – Alter the queue to GET(DISABLED), and again to GET(ENABLED)
  – Stop and restart IMS or the OTMA
  – Stop and restart your IBM MQ subsystem
- If the message is rejected by IMS for anything other than a message error, the IBM MQ message is returned to the originating queue, IBM MQ stops processing the queue, and an error message is sent to the system console.

  If an exception report message is required, the bridge puts it to the reply-to queue with the authority of the originator. If the message cannot be put to the queue, the report message is put to the dead-letter queue with the authority of the bridge. If it cannot be put to the DLQ, it is discarded.

z/OS *IMS bridge feedback codes*

IMS sense codes are typically output in hexadecimal format in IBM MQ console messages such as CSQ2001I (for example, sense code 0x001F). IBM MQ feedback codes as seen in the dead-letter header of messages put to the dead-letter queue are decimal numbers.

The IMS bridge feedback codes are in the range 301 through 399, or 600 through 855 for NACK sense code 0x001A. They are mapped from the IMS-OTMA sense codes as follows:

1. The IMS-OTMA sense code is converted from a hexadecimal number to a decimal number.
2. 300 is added to the number resulting from the calculation in 1, giving the IBM MQ *Feedback* code.
3. The IMS-OTMA sense code 0x001A, decimal 26 is a special case. A *Feedback* code in the range 600-855 is generated.
   a. The IMS-OTMA reason code is converted from a hexadecimal number to a decimal number.
   b. 600 is added to the number resulting from the calculation in a, giving the IBM MQ *Feedback* code.

For information about IMS-OTMA sense codes, see OTMA sense codes for NAK messages.

z/OS *The MQMD fields in messages from the IMS bridge*

Learn about the MQMD fields in messages from the IMS bridge.

The MQMD of the originating message is carried by IMS in the User Data section of the OTMA headers. If the message originates in IMS, this is built by the IMS Destination Resolution Exit. The MQMD of a message received from IMS is built as follows:

StrucID
"MD "

Version
MQMD_VERSION_1

Report
MQRO_NONE

MsgType
MQMT_REPLY

Expiry
If MQIIH_PASS_EXPIRATION is set in the Flags field of the MQIIH, this field contains the remaining expiry time, else it is set to MQEI_UNLIMITED

Feedback
MQFB_NONE

Encoding
MQENC.Native (the encoding of the z/OS system)

CodedCharSetId
MQCCSI_Q_MGR (the CodedCharSetID of the z/OS system)

Format
MQFMT_IMS if the MQMD.Format of the input message is MQFMT_IMS, otherwise IOPCB.MODNAME

Priority
MQMD.Priority of the input message

Persistence
Depends on commit mode: MQMD.Persistence of the input message if CM-1; persistence matches recoverability of the IMS message if CM-0

MsgId
MQMD.MsgId if MQRO_PASS_MSG_ID, otherwise New MsgId (the default)

CorrelId
MQMD.CorrelId from the input message if MQRO_PASS_CORREL_ID, otherwise MQMD.MsgId from the input message (the default)

BackoutCount
0

ReplyToQ
Blanks

ReplyToQMgr
Blanks (set to local qmgr name by the queue manager during the MQPUT)

UserIdentifier
MQMD.UserIdentifier of the input message

AccountingToken
MQMD.AccountingToken of the input message

ApplIdentityData
MQMD.ApplIdentityData of the input message

PutApplType
MQAT_XCF if no error, otherwise MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
Date when message was put

PutTime
Time when message was put

ApplOriginData
Blanks

z/OS *The MQIIH fields in messages from the IMS bridge*

Learn about the MQIIH fields in messages from the IMS bridge.

The MQIIH of a message received from IMS is built as follows:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat of the input message if MQIIH.ReplyToFormat is not blank, otherwise IOPCB.MODNAME

Flags
0

LTermOverride
LTERM name (Tpipe) from OTMA header

MFSMapName
Map name from OTMA header

ReplyToFormat
Blanks

Authenticator
MQIIH.Authenticator of the input message if the reply message is being put to an MQ-IMS bridge queue, otherwise blanks.

TranInstanceId
Conversation ID / Server Token from OTMA header if in conversation. In versions of IMS prior to V14, this field is always nulls if not in conversation. From IMS V14 onwards, this field may be set by IMS even if not in conversation.

TranState
"C" if in conversation, otherwise blank

CommitMode
Commit mode from OTMA header ("0" or "1")

SecurityScope
Blank

Reserved
Blank

z/OS *Reply messages from IMS*

When an IMS transaction ISRTs to its IOPCB, the message is routed back to the originating LTERM or TPIPE.

These are seen in IBM MQ as reply messages. Reply messages from IMS are put onto the reply-to queue specified in the original message. If the message cannot be put onto the reply-to queue, it is put onto the dead-letter queue using the authority of the bridge. If the message cannot be put onto the dead-letter queue, a negative acknowledgment is sent to IMS to say that the message cannot be received. Responsibility for the message is then returned to IMS. If you are using commit mode 0, messages from that Tpipe are not sent to the bridge, and remain on the IMS queue; that is, no further messages are sent until restart. If you are using commit mode 1, other work can continue.

If the reply has an MQIIH structure, its format type is MQFMT_IMS; if not, its format type is specified by the IMS MOD name used when inserting the message.

z/OS *Using alternate response PCBs in IMS transactions*

When an IMS transaction uses alternate response PCBs (ISRTs to the ALTPCB, or issues a CHNG call to a modifiable PCB), the pre-routing exit (DFSYPRX0) is invoked to determine if the message should be rerouted.

If the message is to be rerouted, the destination resolution exit (DFSYDRU0) is invoked to confirm the destination and prepare the header information See Using OTMA exits in IMS and The pre-routing exit DFSYPRX0 for information about these exit programs.

Unless action is taken in the exits, all output from IMS transactions initiated from an IBM MQ queue manager, whether to the IOPCB or to an ALTPCB, will be returned to the same queue manager.

z/OS *Sending unsolicited messages from IMS*

To send messages from IMS to an IBM MQ queue, you need to invoke an IMS transaction that ISRTs to an ALTPCB.

You need to write pre-routing and destination resolution exits to route unsolicited messages from IMS and build the OTMA user data, so that the MQMD of the message can be built correctly. See The pre-routing exit DFSYPRX0 and The destination resolution user exit for information about these exit programs.

Note: The IBM MQ - IMS bridge does not know whether a message that it receives is a reply or an unsolicited message. It handles the message the same way in each case, building the MQMD and MQIIH of the reply based on the OTMA UserData that arrived with the message

Unsolicited messages can create new Tpipes. For example, if an existing IMS transaction switched to a new LTERM (for example PRINT01), but the implementation requires that the output be delivered through OTMA, a new Tpipe (called PRINT01 in this example) is created. By default, this is a non-synchronized Tpipe. If the implementation requires the message to be recoverable, set the destination resolution exit output flag. See the *IMS Customization Guide* for more information.

z/OS *Message segmentation*

You can define IMS transactions as expecting single- or multi-segment input.

The originating IBM MQ application must construct the user input following the MQIIH structure as one or more LLZZ-data segments. All segments of an IMS message must be contained in a single IBM MQ message sent with a single MQPUT.

The maximum length of an LLZZ-data segment is defined by IMS/OTMA (32767 bytes). The total IBM MQ message length is the sum of the LL bytes, plus the length of the MQIIH structure.

All the segments of the reply are contained in a single IBM MQ message.

There is a further restriction on the 32 KB limitation on messages with format MQFMT_IMS_VAR_STRING. When the data in an ASCII-mixed CCSID message is converted to an EBCDIC-mixed CCSID message, a shift-in byte or a shift-out byte is added every time that there is a transition between SBCS and DBCS characters. The 32 KB restriction applies to the maximum size of the message. That is, because the LL field in the message cannot exceed 32 KB, the message must not exceed 32 KB including all shift-in and shift-out characters. The application building the message must allow for this.

z/OS *Data conversion for messages to and from the IMS bridge*

The data conversion is performed by either the distributed queuing facility (which may call any necessary exits) or by the intra group queuing agent (which does not support the use of exits) when it puts a message to a destination queue that has XCF information defined for its storage class. The data conversion does not occur when a message is delivered to a queue by publish/subscribe.

Any exits needed must be available to the distributed queuing facility in the data set referenced by the CSQXLIB DD statement. This means that you can send messages to an IMS application using the IBM MQ - IMS bridge from any IBM MQ platform.

If there are conversion errors, the message is put to the queue unconverted; this results eventually in it being treated as an error by the IBM MQ - IMS bridge, because the bridge cannot recognize the header format. If a conversion error occurs, an error message is sent to the z/OS console.

See “Datenkonvertierungsexits schreiben” on page 1035 for detailed information about data conversion in general.

Sending messages to the IBM MQ - IMS bridge

To ensure that conversion is performed correctly, you must tell the queue manager what the format of the message is.

If the message has an MQIIH structure, the *Format* in the MQMD must be set to the built-in format MQFMT_IMS, and the *Format* in the MQIIH must be set to the name of the format that describes your message data. If there is no MQIIH, set the *Format* in the MQMD to your format name.

If your data (other than the LLZZs) is all character data (MQCHAR), use as your format name (in the MQIIH or MQMD, as appropriate) the built-in format MQFMT_IMS_VAR_STRING. Otherwise, use your own format name, in which case you must also provide a data-conversion exit for your format. The exit must handle the conversion of the LLZZs in your message, in addition to the data itself (but it does not have to handle any MQIIH at the start of the message).

If your application uses *MFSMapName*, you can use messages with the MQFMT_IMS instead, and define the map name passed to the IMS transaction in the MFSMapName field of the MQIIH.

Receiving messages from the IBM MQ - IMS bridge

If an MQIIH structure is present on the original message that you are sending to IMS, one is also present on the reply message.

To ensure that your reply is converted correctly:

- If you have an MQIIH structure on your original message, specify the format that you want for your reply message in the MQIIH *ReplytoFormat* field of the original message. This value is placed in the MQIIH *Format* field of the reply message. This is particularly useful if all your output data is of the form LLZZ<character data>.

- If you do not have an MQIIH structure on your original message, specify the format that you want for the reply message as the MFS MOD name in the IMS application's ISRT to the IOPCB.

z/OS *Writing IMS transaction programs through IBM MQ*

The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

When an IMS transaction is started from a 3270 screen, the message passes through IMS Message Format Services. This can remove all terminal dependency from the data stream seen by the transaction. When a transaction is started through OTMA, MFS is not involved. If application logic is implemented in MFS, this must be re-created in the new application.

In some IMS transactions, the end-user application can modify certain 3270 screen behavior, for example, highlighting a field that has had invalid data entered. This type of information is communicated by adding a two-byte attribute field to the IMS message for each screen field that needs to be modified by the program.

Thus, if you are coding an application to mimic a 3270, you need to take account of these fields when building or receiving messages.

You might need to code information in your program to process:

- Which key is pressed (for example, Enter and PF1)
- Where the cursor is when the message is passed to your application
- Whether the attribute fields have been set by the IMS application
  - High, normal, or zero intensity
  - Color
  - Whether IMS is expecting the field back the next time that Enter is pressed
- Whether the IMS application has used null characters (X'3F') in any fields.

If your IMS message contains only character data (apart from the LLZZ-data segment), and you are using an MQIIH structure, set the MQMD format to MQFMT_IMS and the MQIIH format to MQFMT_IMS_VAR_STRING.

If your IMS message contains only character data (apart from the LLZZ-data segment), and you are **not** using an MQIIH structure, set the MQMD format to MQFMT_IMS_VAR_STRING and ensure that your IMS application specifies MODname MQFMT_IMS_VAR_STRING when replying. If a problem occurs (for example, user not authorized to use the transaction) and IMS sends an error message, this has an MODname of the form DFSMOx, where x is a number in the range 1 through 5. This is put in the MQMD.Format.

If your IMS message contains binary, packed, or floating point data (apart from the LLZZ-data segment), code your own data-conversion routines. Refer to *IMS/ESA Application Programming: Transaction Manager* for information about IMS screen formatting.

Consider the following topics when writing code to handle IMS transactions through IBM MQ.

- “Writing IBM MQ applications to invoke IMS conversational transactions” on page 960
- “Writing programs containing IMS commands” on page 961
- “Triggering” on page 961

Writing IBM MQ applications to invoke IMS conversational transactions

Use this information as a guide for considerations when writing IBM MQ application to invoke IMS conversational transactions.

When you write an application that invokes an IMS conversation, consider the following:

- Include an MQIIH structure with your application message.
- Set the *CommitMode* in MQIIH to MQICM_SEND_THEN_COMMIT.

- To invoke a new conversation, set *TranState* in MQIIH to MQITS_NOT_IN_CONVERSATION.
- To invoke second and subsequent steps of a conversation, set *TranState* to MQITS_IN_CONVERSATION, and set *TranInstanceId* to the value of that field returned in the previous step of the conversation.
- There is no easy way in IMS to find the value of a *TranInstanceId*, should you lose the original message sent from IMS.
- The application must check the *TranState* of messages from IMS to check whether the IMS transaction has terminated the conversation.
- You can use /EXIT to end a conversation. You must also quote the *TranInstanceId*, set *TranState* to MQITS_IN_CONVERSATION, and use the IBM MQ queue on which the conversation is being carried out.
- You cannot use /HOLD or /REL to hold or release a conversation.
- Conversations invoked through the IBM MQ - IMS bridge are terminated if IMS is restarted.

Writing programs containing IMS commands

An application program can build an IBM MQ message of the form LLZZ*command*, instead of a transaction, where *command* is of the form /DIS TRAN PART or /DIS POOL ALL.

Most IMS commands can be issued in this way; see *IMS V11 Communications and Connections* for details. The command output is received in the IBM MQ reply message in the text form as would be sent to a 3270 terminal for display.

OTMA has implemented a special form of the IMS display transaction command, which returns an architected form of the output. The exact format is defined in *IMS V11 Communications and Connections*. To invoke this form from an IBM MQ message, build the message data as before, for example /DIS TRAN PART, and set the TranState field in the MQIIH to MQITS_ARCHITECTED. IMS processes the command, and returns the reply in the architected form. An architected response contains all the information that could be found in the text form of the output, and one additional piece of information: whether the transaction is defined as recoverable or non-recoverable.

Triggering

The IBM MQ - IMS bridge does not support trigger messages.

If you define an initiation queue that uses a storage class with XCF parameters, messages put to that queue are rejected when they get to the bridge.

Prozedurale Clientanwendungen schreiben

Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

Anwendungen können in der IBM MQ-Clientumgebung erstellt und ausgeführt werden. Die Anwendung muss mit dem verwendeten IBM MQ MQI client erstellt und verknüpft werden. Die Vorgehensweise hängt dabei von der Plattform und der Programmiersprache ab. Informationen zum Schreiben von Clientanwendungen finden Sie im Abschnitt „Anwendungen für IBM MQ MQI clients entwickeln” auf Seite 968.

Sie können eine IBM MQ-Anwendung sowohl in einer vollständigen IBM MQ-Umgebung als auch in einer IBM MQ MQI client-Umgebung ausführen, ohne Ihren Code ändern zu müssen. Allerdings müssen hierfür bestimmte Voraussetzungen erfüllt sein. Weitere Informationen zur Ausführung Ihrer Anwendungen in der IBM MQ-Clientumgebung finden Sie im Abschnitt „Anwendungen in der IBM MQ MQI client-Umgebung ausführen” auf Seite 970.

Wenn Sie die Message Queue Interface (MQI) zum Schreiben von Anwendungen verwenden, die in einer IBM MQ MQI client-Umgebung ausgeführt werden sollen, müssen Sie während des MQI-Aufrufs einige zusätzliche Steuerelemente implementieren, um sicherzustellen, dass die IBM MQ-Anwendungsverarbeitung nicht unterbrochen wird. Weitere Informationen zu diesen Steuerelementen finden Sie im Abschnitt „MQI in einer Clientanwendung verwenden” auf Seite 962.

In den folgenden Abschnitten finden Sie Informationen zum Vorbereiten und Ausführen anderer Anwendungstypen als Clientanwendungen:

- „CICS- und Tuxedo-Anwendungen vorbereiten und ausführen” auf Seite 983
- „Microsoft Transaction Server-Anwendungen vorbereiten und ausführen” auf Seite 52
- „IBM MQ JMS-Anwendungen vorbereiten und ausführen” auf Seite 986

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

„Anwendungen für IBM MQ entwickeln” auf Seite 5
Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

„Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 52
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

„Prozedurale Anwendungen für die Warteschlangensteuerung erstellen” auf Seite 758
Dieser Abschnitt enthält Informationen zum Erstellen von Anwendungen für die Warteschlangensteuerung, zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager und zum Öffnen und Schließen von Objekten.

„Publish/Subscribe-Anwendungen schreiben” auf Seite 851
Beginnen Sie nun, Ihre eigenen IBM MQ-Publish/Subscribe-Anwendungen zu schreiben.

„Prozedurale Anwendung erstellen” auf Seite 1052
Sie können eine IBM MQ-Anwendung in einer von mehreren prozeduralen Sprachen schreiben und die Anwendung auf mehreren unterschiedlichen Plattformen ausführen.

„Prozedurale Programmfehler handhaben” auf Seite 1090
In diesen Informationen werden Fehler erläutert, die den MQI-Aufrufen Ihrer Anwendungen beim Ausgeben eines Aufrufs oder bei der Übergabe einer Nachricht an den Zielort zugeordnet werden.

Zugehörige Tasks

„Prozedurale IBM MQ-Beispielprogramme verwenden” auf Seite 1111
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

MQI in einer Clientanwendung verwenden

In der nachfolgenden Themensammlung werden die Unterschiede zwischen der Entwicklung von IBM MQ-Anwendungen behandelt, die in einer MQI-Clientumgebung ausgeführt werden sollen, und solchen, die in einer vollständigen IBM MQ-Warteschlangenmanagerumgebung ausgeführt werden sollen.

Berücksichtigen Sie beim Entwickeln einer Anwendung die Steuerelemente, die Sie während des MQI-Aufrufs zusätzlich implementieren müssen, um sicherzustellen, dass die IBM MQ-Anwendungsverarbeitung nicht unterbrochen wird.

Bevor Sie Anwendungen ausführen können, die mit der MQI arbeiten, müssen Sie bestimmte IBM MQ-Objekte erstellen. Weitere Informationen finden Sie im Abschnitt Application programs using the MQI.

Nachrichtengröße in Clientanwendungen einschränken

Ein Warteschlangenmanager hat eine maximale Nachrichtenlänge, jedoch wird die maximale Nachrichtengröße, die Sie aus einer Clientanwendung übertragen können, durch die Kanaldefinition beschränkt.

Das Attribut 'MaxMsgLength' eines Warteschlangenmanagers legt die maximale durch diesen Warteschlangenmanager verarbeitbare Nachrichtenlänge fest.

Multi Auf Multiplatforms können Sie die maximale Nachrichtenlänge eines Warteschlangenmanagers heraufsetzen. Weitere Informationen finden Sie in ALTER QMGR .

Den Wert des Attributs 'MaxMsgLength' eines Warteschlangenmanagers können Sie mit einem MQINQ-Aufruf ermitteln.

Bei einer Änderung von 'MaxMsgLength' wird nicht überprüft, ob nicht bereits Warteschlangen oder gar Nachrichten vorhanden sind, deren Größe den neuen Wert übersteigt. Starten Sie nach einer Änderung dieses Attributs Ihre Anwendungen und Kanäle neu, um sicherzustellen, dass die Änderung wirksam wird. Danach können keine neuen Nachrichten mehr generiert werden, deren Größe 'MaxMsgLength' des Warteschlangenmanagers oder der Warteschlange übersteigt (es sei denn, eine Segmentierung durch den Warteschlangenmanager ist erlaubt).

Die maximale Nachrichtenlänge einer Kanaldefinition beschränkt die Größe einer Nachricht, die über eine Clientverbindung übertragen werden kann. Wenn eine IBM MQ-Anwendung einen MQPUT- oder MQGET-Aufruf mit einer größeren Nachricht ausgibt, erhält die Anwendung einen Fehlercode zurück. Die maximale Nachrichtenlänge der Kanaldefinition wirkt sich nicht auf die maximale Nachrichtenlänge aus, die bei Verwendung von MQCB über eine Clientverbindung verarbeitet werden kann.

Zugehörige Konzepte

„MQCONNX verwenden” auf Seite 967
Mit dem Aufruf MQCONNX können Sie eine Struktur für die Kanaldefinition (MQCD) in der MQCNO-Struktur angeben.

Zugehörige Verweise

Maximale Nachrichtenlänge (MAXMSGL)

ALTER CHANNEL

2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

Client- oder Server-CCSID auswählen

Verwenden Sie die lokale ID des codierten Zeichensatzes (CCSID) für den Client. Der Warteschlangenmanager führt die erforderliche Konvertierung durch. Sie können die Umgebungsvariable **MQCCSID** verwenden, um die CCSID zu überschreiben. Wenn Ihre Anwendung mehrere PUT-Operationen ausführt, können die CCSID und die Codierungsfelder der MQMD-Struktur nach Abschluss des ersten PUT-Vorgangs überschrieben werden.

Die von der Anwendung über die MQI (Message Queue Interface; Schnittstelle für Nachrichtenwarteschlangen) an den Client-Stub übertragenen Daten müssen in der lokalen CCSID, codiert für den IBM MQ MQI client, vorliegen. Falls für den verbundenen Warteschlangenmanager eine Konvertierung der Daten erforderlich ist, wird die Konvertierung vom Clientunterstützungscode des Warteschlangenmanagers ausgeführt.

Ab IBM WebSphere MQ 7.0 kann der Java-Client die Konvertierung auch selbst durchführen, wenn der Warteschlangenmanager hierzu nicht in der Lage ist. Siehe „IBM MQ classes for Java-Clientverbindungen” auf Seite 391.

Der Clientcode geht davon aus, dass die auf dem Client über die MQI übertragenen Zeichendaten im CCSID dieser Workstation vorliegen. Wenn diese CCSID eine nicht unterstützte CCSID ist oder nicht die erforderliche CCSID ist, kann sie mit einem der folgenden Befehle mit der Umgebungsvariablen **MQCCSID** überschrieben werden:

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

Wenn dieser Parameter im Profil festgelegt ist, wird davon ausgegangen, dass alle MQI-Daten in der Codepage 850 vorliegen.

Anmerkung: Diese Erwartung hinsichtlich der Codepage 850 gilt nicht für die Anwendungsdaten in der Nachricht.

Wenn Ihre Anwendung mehrere PUT-Operationen ausführt, die nach dem Nachrichtendeskriptor (MQMD) IBM MQ-Header enthalten, sollte Ihnen bewusst sein, dass die CCSID und die Codierungsfelder der MQMD-Struktur nach Abschluss des ersten PUT-Vorgangs überschrieben werden.

Nach dem ersten PUT-Vorgang enthalten diese Felder den vom verbundenen Warteschlangenmanager zur Konvertierung der IBM MQ-Header verwendeten Wert. Stellen Sie sicher, dass Ihre Anwendung die Werte auf diejenigen Werte zurücksetzt, die sie benötigt.

MQINQ in einer Clientanwendung verwenden

Einige durch MQINQ abgefragten Werte werden vom Clientcode geändert.

CCSID
wird auf die CCSID des Clients, nicht auf die des Warteschlangenmanagers gesetzt.

MaxMsgLength
wird herabgesetzt, wenn durch die Kanaldefinition beschränkt. Es wird der niedrigere Wert der folgenden Werte verwendet:

- Der in der Warteschlangendefinition festgelegte Wert
- Der in der Kanaldefinition festgelegte Wert

Weitere Informationen finden Sie im Abschnitt MQINQ.

Synchronisationspunktkoordinierung in einer Clientanwendung verwenden

Eine auf dem Basisclient ausgeführte Anwendung kann die Aufrufe MQCMIT und MQBACK ausgeben, die Synchronisationspunktsteuerung beschränkt sich jedoch auf die MQI-Ressourcen. Bei einem erweiterten transaktionsorientierten Client können Sie einen externen Transaktionsmanager verwenden.

Innerhalb von IBM MQ besteht eine der Aufgaben des Warteschlangenmanagers auch in der Synchronisationspunktsteuerung innerhalb der Anwendung. Wenn eine Anwendung auf einem IBM MQ-Basisclient ausgeführt wird, kann sie die Aufrufe MQCMIT und MQBACK ausgeben, wobei sich die Synchronisationspunktsteuerung jedoch auf die MQI-Ressourcen beschränkt. Der IBM MQ-Aufruf MQBEGIN ist in einer Basisclientumgebung ungültig.

Anwendungen, die in der vollständigen Warteschlangenmanagerumgebung auf dem Server ausgeführt werden, können mehrere Ressourcen (z. B. Datenbanken) über einem Transaktionsmonitor koordinieren. Auf dem Server können Sie den mit IBM MQ-Produkten bereitgestellten Transaktionsmonitor oder auch Transaktionsmonitor wie CICS verwenden. Bei Basisclientanwendungen können Sie keinen Transaktionsmonitor verwenden.

Bei einem erweiterten transaktionsorientierten IBM MQ-Client können Sie einen externen Transaktionsmanager verwenden. Siehe auch Was ist ein erweiterter transaktionsorientierter Client? .

Vorauslesen in einer Clientanwendung verwenden

Sie können Vorauslesen auf einem Client verwenden, damit nicht persistente Nachrichten an einen Client gesendet werden können, ohne dass die Clientanwendung die Nachrichten anfordern muss.

Wenn ein Client eine Nachricht von einem Server benötigt, sendet er eine Anforderung an den Server. Für jede von ihm verarbeitete Nachricht sendet der Client eine eigene Anforderung. Zur Verbesserung

der Leistung eines Clients, der nicht persistente Nachrichten verarbeitet, kann er für die Verwendung von Vorauslesen konfiguriert und es so vermieden werden, dass er diese Anforderungsnachrichten senden muss. Mit Vorauslesen können Nachrichten an einen Client gesendet werden, ohne dass eine Anwendung sie anfordern muss.

Durch die Verwendung von Vorauslesen kann die Leistung verbessert werden, wenn nicht persistente Nachrichten von einer Clientanwendung verarbeitet werden. Diese Leistungsverbesserung betrifft sowohl MQI- als auch JMS-Anwendungen. Clientanwendungen, die MQGET oder asynchrone Verarbeitung verwenden, profitieren von den Leistungsverbesserungen, wenn sie nicht persistente Nachrichten verarbeiten.

Bei einem MQOPEN-Aufruf mit MQOO_READ_AHEAD aktiviert der IBM MQ-Client die Vorausleseoption nur, wenn bestimmte Bedingungen erfüllt sind. Zu diesen Bedingungen gehören:

- Die Clientanwendung muss kompiliert und mit den IBM MQ MQ-Client-Thread-Bibliotheken verknüpft sein.
- Der Clientkanal muss das TCP/IP-Protokoll verwenden.
- In den Client- und Serverkanaldefinitionen muss für den Kanal ein Wert ungleich null für den Parameter SHARECNV (gemeinsame Dialognutzung) angegeben sein.

Wenn Vorauslesen aktiviert ist, werden die Nachrichten in einen Hauptspeicherpuffer auf dem Client gesendet, der als Vorauslesepuffer bezeichnet wird. Der Client besitzt für jede geöffnete Warteschlange mit aktiviertem Vorauslesen einen Vorauslesepuffer. Die Nachrichten im Vorauslesepuffer sind nicht als persistent definiert. Der Client sendet dem Server regelmäßig aktuelle Informationen über das von ihm verarbeitete Datenvolumen.

Nicht alle Clientanwendungen sind von ihrem Design her für die Verwendung von Vorauslesen geeignet, da nicht die Verwendung aller Optionen unterstützt wird. Wenn Vorauslesen aktiviert ist, müssen einige Optionen zwischen MQGET-Aufrufen konsistent sein. Wenn ein Client seine Auswahlkriterien zwischen MQGET-Aufrufen ändert, werden im Vorauslesepuffer gespeicherte Nachrichten im Vorauslesepuffer des Clients zurückgelassen. Weitere Informationen finden Sie unter „Leistung nicht persistenter Nachrichten verbessern” auf Seite 827

Die Vorauslesekonfiguration besteht aus den drei Attributen MaximumSize, PurgeTime und UpdatePercentage in der Zeilengruppe MessageBuffer der Clientkonfigurationsdatei von IBM MQ.

Asynchrone Put-Operationen in einer Clientanwendung verwenden

Mit einer asynchronen Put-Operation kann eine Anwendung eine Nachricht in eine Warteschlange einreihen, ohne die Antwort des Warteschlangenmanagers abzuwarten. Damit lässt sich in manchen Situationen die Messagingleistung verbessern.

Normalerweise muss eine Anwendung, wenn Sie Nachrichten mittels MQPUT oder MQPUT1 in eine Warteschlange einreiht, warten, bis der Warteschlangenmanager die Verarbeitung der MQI-Anforderung bestätigt. Die Messagingleistung lässt sich jedoch durch Verwendung der asynchronen Nachrichteneinreihung verbessern, insbesondere bei Anwendungen, die Clientbindung verwenden oder sehr viele kleine Nachrichten in eine Warteschlange einreihen. Wenn eine Anwendung eine Nachricht asynchron einreiht, meldet der Warteschlangenmanager nicht für jeden einzelnen Aufruf einen Erfolg oder Fehler, Sie können jedoch stattdessen laufend prüfen, ob Fehler aufgetreten sind.

Wenn Sie eine Nachricht asynchron in eine Warteschlange einreihen möchten, geben Sie die Option MQPMO_ASYNC_RESPONSE im Feld *Options* der MQPMO-Struktur an.

Falls sich eine Nachricht nicht für die asynchrone Einreihung eignet, wird sie synchron in die Warteschlange eingereiht.

Bei Anforderung einer asynchronen PUT-Antwort für MQPUT oder MQPUT1 bedeuten ein Beendigungscode MQCC_OK und ein Ursachencode MQRC_NONE nicht notwendigerweise, dass die Nachricht erfolgreich in eine Warteschlange eingereiht wurde. Auch wenn der Erfolg oder Misserfolg der einzelnen MQPUT- und MQPUT1-Aufrufe nicht sofort zurückgegeben wird, kann der zuerst unter einem asynchronen Aufruf aufgetretene Fehler später mittels eines MQSTAT-Aufrufs festgestellt werden.

Weitere Informationen zu MQPMO_ASYNC_RESPONSE finden Sie im Abschnitt MQPMO-Optionen.

Einige der verfügbaren Funktionen werden durch das Beispielprogramm für die asynchrone Einreihung veranschaulicht. Details zu den Funktionen, zum Aufbau des Programms und zu dessen Ausführung finden Sie im Abschnitt „Das Asynchronous Put-Beispielprogramm” auf Seite 1133.

Gemeinsamer Datenaustausch in einer Clientanwendung

In einer Umgebung, in der mehrere gemeinsame Datenaustauschvorgänge zulässig sind, kann für mehrere Datenaustauschvorgänge eine einzige Instanz eines MQI-Kanals gemeinsam genutzt werden.

Die gemeinsame Nutzung einer Kanalinstanz für Datenübertragungen wird über zwei Felder desselben Namens (SharingConversations) gesteuert; ein Feld gehört zur MQCD-Struktur (Kanaldefinition), das andere zur MQCXP-Struktur (Kanalexitparameter). Im Feld 'SharingConversations' in der MQCD-Struktur wird eine Ganzzahl angegeben, die die Anzahl an Datenaustauschvorgängen angibt, die maximal die Kanalinstanz des Kanals gemeinsam nutzen können. Im Feld 'SharingConversations' in der MQCXP-Struktur wird ein boolescher Wert angegeben, der angibt, ob die Kanalinstanz momentan gemeinsam genutzt wird.

In Umgebungen, in denen diese gemeinsame Nutzung nicht zulässig ist, können neue Clientverbindungen mit identischen MQCDs keine Kanalinstanz gemeinsam nutzen.

Eine neue Clientanwendungsverbindung kann die Kanalinstanz gemeinsam nutzen, wenn Folgendes zutrifft:

- Sowohl die Clientverbindungs- als auch die Serververbindungsseite der Kanalinstanz ist für die gemeinsame Nutzung der Kanalinstanz für den Datenaustausch konfiguriert und diese Werte werden nicht von Kanalexits überschrieben.
- Der MQCD-Wert (aus dem MQCONNX-Clientaufruf oder der Clientkanal-Definitionstabelle) ist mit dem MQCD-Wert identisch, der zu dem Zeitpunkt, zu dem die Kanalinstanz eingerichtet wurde, aus dem MQCONNX-Clientaufruf oder der Definitionstabelle bereitgestellt wurde. Der ursprüngliche MQCD-Wert kann zwischenzeitlich durch Exits oder Kanalvereinbarungen geändert worden sein; der Vergleich wird jedoch mit Wert vorgenommen, wie er dem Clientsystem vor diesen Änderungen übergeben wurde.
- Die maximal mögliche Anzahl an Datenaustauschvorgängen auf der Serverseite wird nicht überschritten.

Erfüllt eine neue Clientanwendungsverbindung die Bedingungen, die die gemeinsame Nutzung einer Kanalinstanz für andere Datenaustauschvorgängen erlaubt, wird diese Entscheidung getroffen, noch bevor ein Exit im Rahmen dieses Datenaustauschs aufgerufen wird. Exits in einem solchen Datenaustausch können den Umstand, dass die Kanalinstanz von anderen Datenaustauschvorgängen gemeinsam genutzt wird, nicht ändern. Sind keine Kanalinstanzen vorhanden, die der neuen Kanaldefinition entsprechen, wird eine neue Kanalinstanz verbunden.

Eine Kanalvereinbarung wird nur beim ersten Datenaustausch über eine Kanalinstanz durchgeführt; zu diesem Zeitpunkt werden die vereinbarten Werte für die Kanalinstanz festgelegt; diese Werte können von nachfolgenden Datenaustauschvorgängen nicht mehr geändert werden. Auch die TLS-Authentifizierung wird nur beim ersten Datenaustausch durchgeführt.

Wird der Wert für 'SharingConversations' in der MQCD-Struktur bei der Initialisierung eines Sicherheits-, Sende- oder Empfangsexits für den ersten Datenaustausch am Socket auf der Clientverbindungs- oder Serververbindungsseite der Kanalinstanz geändert, wird nach der Initialisierung alle dieser Exits anhand dieses Werts der Wert von 'SharingConversations' für die Kanalinstanz festgelegt (der niedrigere Wert hat Vorrang).

Ist der vereinbarte Wert für 'SharingConversations' gleich null, wird die Kanalinstanz nie gemeinsam genutzt. Alle weiteren Exitprogramme, die dieses Feld auf null setzen, werden ebenfalls über eine eigene Kanalinstanz ausgeführt.

Ist der vereinbarte Wert für 'SharingConversations' größer als null, wird das Feld 'SharingConversations' in der MQCXP-Struktur für nachfolgende Exitaufrufe auf TRUE gesetzt; dies bedeutet, dass andere Exitprogramme über diese Kanalinstanz gleichzeitig mit diesem Exitprogramm ausgeführt werden können.

Beim Erstellen eines Kanalexitprogramms sollten Sie sich überlegen, ob das Programm über eine Kanalinstanz ausgeführt werden soll, die für andere Datenaustauschvorgänge gemeinsam genutzt werden kann. Wenn die Kanalinstanz für mehrere Datenaustauschvorgänge gemeinsam genutzt werden kann, sollten Sie die Auswirkungen von Änderungen an MQCD-Feldern auf andere Instanzen des Kanalexits

bedenken; die Werte aller MQCD-Felder sind für alle gemeinsamen Datenaustauschvorgänge einheitlich. Wenn eine Kanalinstanz eingerichtet wurde und ein Exitprogramm versucht, die MQCD-Felder zu ändern, kann dies unter Umständen zu Problemen führen, da möglicherweise andere Exitprogramme, die dieselbe Kanalinstanz verwenden, dieselben Felder zu dieser Zeit ebenfalls ändern. Wenn diese Möglichkeit für Ihre Exitprogramme gegeben ist, müssen Sie den Zugriff auf die MQCD-Struktur in Ihrem Exit-Code serialisieren.

Wenn Sie mit einem Kanal arbeiten, für eine gemeinsame Nutzung für mehrere Datenaustauschvorgänge aktiviert ist, die gemeinsame Nutzung jedoch für eine bestimmte Kanalinstanz unterdrückt werden soll, müssen Sie den Wert von 'SharingConversations' in der MQCD-Struktur beim Initialisieren eines Kanalexits im ersten Datenaustausch über diese Kanalinstanz auf 1 oder 0 setzen. Eine Erläuterung der Werte für 'SharingConversations' finden Sie im Abschnitt SharingConversations.

Beispiel

Der gemeinsame Datenaustausch ist aktiviert.

Sie verwenden die Definition eines Clientverbindungskanals, in der ein Exitprogramm angegeben ist.

Beim ersten Start dieses Kanals werden vom Exitprogramm bei dessen Initialisierung einige MQCD-Parameter geändert. Diese geänderten Werte werden vom Kanal übernommen, sodass der Kanal jetzt mit einer anderen als der ursprünglich bereitgestellten Definition arbeitet. Der MQCXP-Parameter 'SharingConversations' wird auf TRUE gesetzt.

Wenn die Anwendung das nächste Mal eine Verbindung über diesen Kanal herstellt, erfolgt der Datenaustausch über die Kanalinstanz, die zuvor gestartet wurde, da sie die ursprüngliche Kanaldefinition verwendet. Beim zweiten Mal stellt die Anwendung erneut eine Verbindung zu der Kanalinstanz her, zu der die erste Verbindung hergestellt wurde. Folglich verwendet sie die Definitionen, die vom Exitprogramm geändert wurden. Wird das Exitprogramm für den zweiten Datenaustausch initialisiert, kann es die MQCD-Felder zwar ändern, diese Änderungen werden jedoch nicht vom Kanal übernommen. Dies gilt auch für alle weiteren Datenaustauschvorgänge, die die Kanalinstanz gemeinsam nutzen.

MQCONNX verwenden

Mit dem Aufruf MQCONNX können Sie eine Struktur für die Kanaldefinition (MQCD) in der MQCNO-Struktur angeben.

Damit kann die aufrufende Clientanwendung die Definition des Clientverbindungskanals während der Ausführung angeben. Weitere Informationen finden Sie unter Clientverbindungskanal auf dem IBM MQ MQI client mit MQCNO erstellen. Wenn Sie MQCONNX verwenden, hängt der am Server ausgegebene Aufruf von der Version des Servers und der Konfiguration des Empfangsprogramms ab.

Wenn Sie MQCONNX von einem Client aus verwenden, werden die folgenden Optionen ignoriert:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

Die MQCD-Struktur, die Sie verwenden können, hängt von der MQCD-Versionsnummer ab, die Sie verwenden. Informationen zu MQCD-Versionen (MQCD_VERSION) finden Sie im Abschnitt MQCD-Version. Sie können die MQCD-Struktur zum Beispiel verwenden, um Kanalexitprogramme an den Server zu übergeben. Wenn Sie MQCD Version 3 oder höher verwenden, können Sie mit der Struktur ein Array mit Exits an den Server übergeben. Sie können mit dieser Funktion mehrere Operationen für dieselbe Nachricht ausführen, zum Beispiel Verschlüsselung und Komprimierung. Dazu fügen Sie einen Exit für jede Operation hinzu, anstatt einen vorhandenen Exit zu ändern. Wenn Sie kein Array in der MQCD-Struktur angeben, werden die einzelnen Exitfelder geprüft. Weitere Informationen zu Kanalexitprogrammen finden Sie im Abschnitt „Kanalexitprogramme für Messaging-Kanäle” auf Seite 1013.

Gemeinsam genutzte Verbindungskennungen in MQCONNX

Wenn Sie gemeinsam genutzte Verbindungskennungen verwenden, können Sie Kennungen zwischen den einzelnen Threads des gleichen Prozesses austauschen.

Bei Angabe einer gemeinsam genutzten Verbindungskennung kann die vom MQCONNX-Aufruf zurückgegebene Verbindungskennung in nachfolgenden MQI-Aufrufen für jeden Thread des gleichen Prozesses weitergegeben werden.

Anmerkung: Mit einer gemeinsam genutzten Verbindungskennung können Sie auf einem IBM MQ MQI client auch eine Verbindung zu einem Warteschlangenmanager herstellen, der keine gemeinsam genutzten Verbindungskennungen unterstützt.

Anwendungen für IBM MQ MQI clients entwickeln

Anwendungen können in einer IBM MQ MQI client-Umgebung erstellt und ausgeführt werden. Die Anwendung muss mit dem verwendeten IBM MQ MQI client erstellt und verknüpft werden. Die Vorgehensweise hängt dabei von der Plattform und der Programmiersprache ab.

Soll eine Anwendung in einer Clientanwendung zum Einsatz kommen, kann sie mit einer der in der folgenden Tabelle aufgeführten Programmiersprachen geschrieben werden:

Tabelle 133. In Clientumgebungen unterstützte Programmiersprachen

Clientplattform	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX AIX	Ja	Ja	Ja			
IBM i IBM i	Ja		Ja		Ja	
Linux Linux	Ja	Ja	Ja			
Windows Windows	Ja	Ja	Ja			Ja

Multi *C-Anwendungen mit IBM MQ MQI client-Code verknüpfen*

Nachdem Sie die IBM MQ-Anwendung, die Sie auf dem IBM MQ MQI client ausführen möchten, geschrieben haben, müssen Sie diese noch mit dem IBM MQ MQI client-Code verknüpfen.

Sie können Ihre Anwendung auf zwei Weisen mit dem IBM MQ MQI client-Code verknüpfen:

1. Direkt, indem Sie Ihre Anwendung mit einem Warteschlangenmanager verbinden. In diesem Fall muss sich der Warteschlangenmanager auf dem gleichen System befinden wie die Anwendung.
2. Über eine Clientbibliotheksdatei, die Ihnen Zugriff auf Warteschlangenmanager auf dem gleichen oder auf anderen Systemen gibt.

IBM MQ bietet für jede Umgebung eine Clientbibliotheksdatei:

AIX **AIX**
Bibliothek libmqic.a für Nicht-Thread-Anwendungen bzw. Bibliothek libmqic_r.a für Thread-Anwendungen

Linux **Linux**
Bibliothek libmqic.so für Nicht-Thread-Anwendungen bzw. Bibliothek libmqic_r.so für Thread-Anwendungen

IBM i **IBM i**
Binden Sie die Clientanwendung bei einer Nicht-Thread-Anwendung mit dem Clientserviceprogramm LIBMQIC bzw. bei einer Thread-Anwendung mit dem Clientserviceprogramm LIBMQIC_R.

Windows **Windows**
MQIC32.LIB.

ALW *C++-Anwendungen mit IBM MQ MQI client-Code verknüpfen*

Die Anwendungen, die auf dem Client ausgeführt werden sollen, können Sie in der Programmiersprache C++ schreiben. Die Erstellungsmethoden variieren je nach Umgebung.

Informationen zum Verknüpfen der C++-Anwendungen finden Sie im Abschnitt IBM MQ-C++-Programme erstellen.

Ausführliche Informationen zur Verwendung der Programmiersprache C++ finden Sie im Abschnitt C++ verwenden.

Multi *COBOL-Anwendungen mit IBM MQ MQI client-Code verknüpfen*

Nachdem Sie die COBOL-Anwendung, die Sie auf dem IBM MQ MQI client ausführen möchten, geschrieben haben, müssen Sie diese noch mit der richtigen Bibliothek verknüpfen.

IBM MQ bietet für jede Umgebung eine Clientbibliotheksdatei:

AIX **AIX**
: Verknüpfen Sie eine Nicht-Thread-COBOL-Anwendung mit der Bibliothek libmqicb.a und eine Thread-COBOL-Anwendung mit der Bibliothek libmqicb_r.a.

IBM i **IBM i**
: Binden Sie die COBOL-Clientanwendung bei einer Nicht-Thread-Anwendung mit dem Serviceprogramm AMQCSTUB bzw. bei einer Thread-Anwendung mit dem Serviceprogramm AMQCSTUB_R.

Windows **Windows**
: Verknüpfen Sie Ihren Anwendungscode mit der MQICCBB-Bibliothek für die 32-Bit-Version von COBOL. Der IBM MQ MQI client for Windows unterstützt die 16-Bit-Version von COBOL nicht.

Windows *Visual Basic-Anwendungen mit IBM MQ MQI client-Code verknüpfen*

Unter Windows können Sie Microsoft Visual Basic-Anwendungen mit dem IBM MQ MQI client-Code verknüpfen.

Deprecated

Ab IBM MQ 9.0 wird die Unterstützung für Microsoft Visual Basic 6.0 nicht mehr verwendet. IBM MQ-Klassen für .NET sind die empfohlene Ersatztechnologie. Weitere Informationen finden Sie unter .NET-Anwendungen entwickeln.

Verknüpfen Sie Ihre Visual Basic-Anwendung mit folgenden Include-Dateien:

CMQB.bas
: MQI

CMQBB.bas
: MQAI

CMQCFB.bas
: PCF-Befehle

CMQXB.bas
: Kanäle

Legen Sie `mqtype=2` für den Client im Visual Basic -Compiler fest, um die korrekte automatische Auswahl der Client-DLL sicherzustellen:

MQIC32.dll
: Windows 7, Windows 8, Windows 2008 und Windows 2012

Zugehörige Konzepte

„Codierung in Visual Basic” auf Seite 1106
Informationen, die beim Codieren von IBM MQ-Programmen in Microsoft Visual Basic zu berücksichtigen sind. Visual Basic wird nur unter Windows unterstützt.

„Visual Basic-Programme in Windows vorbereiten” auf Seite 1073

Informationen, die bei der Verwendung von Microsoft Visual Basic-Programmen unter Windows zu beachten sind.

Anwendungen in der IBM MQ MQI client-Umgebung ausführen

Sie können eine IBM MQ-Anwendung sowohl in einer vollständigen IBM MQ-Umgebung als auch in einer IBM MQ MQI client-Umgebung ausführen, ohne Ihren Code ändern zu müssen. Allerdings müssen hierfür bestimmte Voraussetzungen erfüllt sein.

Bei den Bedingungen handelt es sich um folgende:

- Die Anwendung muss zu jedem Zeitpunkt nicht mehr als eine Warteschlangenmanagerverbindung herstellen.
- Dem Warteschlangenmanagername wird in einem `MQCONN`- oder `MQCONNX`-Aufruf kein Stern (*) als Präfix vorangestellt.
- Für die Anwendung trifft keine der Ausnahmen zu, die im Abschnitt Welche Anwendungen werden auf einem IBM MQ MQI client ausgeführt? aufgeführt sind.

Anmerkung: Die Bibliotheken, die bei der Programmverbindung verwendet werden, geben die Umgebung vor, in der die Anwendung ausgeführt werden muss.

Berücksichtigen Sie in der IBM MQ MQI client-Umgebung auch Folgendes:

- Jede Anwendung in der IBM MQ MQI client-Umgebung hat ihre eigenen Serververbindungen. Eine Anwendung erstellt bei jeder Ausgabe eines `MQCONN`- oder `MQCONNX`-Aufrufs eine Verbindung zu einem Server.
- Eine Anwendung sendet und empfängt Nachrichten synchron. Daher kommt es zwischen der Ausgabe des Aufrufs auf dem Client und der Rückgabe eines Beendigungs- und Ursachencodes über das Netz zu einer zeitlichen Verzögerung.
- Eine eventuell erforderliche Datenkonvertierung wird vom Server ausgeführt, jedoch können Sie die konfigurierte CCSID des Systems auch überschreiben (siehe MQCCSID).

IBM MQ MQI client-Anwendungen mit Warteschlangenmanagern verbinden

Eine Anwendung in einer IBM MQ MQI client-Umgebung kann auf verschiedene Weisen Verbindungen zu einem Warteschlangenmanager herstellen: über Umgebungsvariablen, die MQCNO-Struktur oder eine Clientdefinitionstabelle.

Wenn eine in einer IBM MQ-Clientumgebung ausgeführte Anwendung einen MQCONN- oder MQCONNX-Aufruf ausgibt, ermittelt der Client, wie er die Verbindung herstellen soll. Führt eine Anwendung auf einem IBM MQ-Client einen MQCONNX-Aufruf aus, sucht die MQI-Clientbibliothek in der folgenden Reihenfolge nach den Clientkanalinformationen:

1. Anhand des Inhalts des Felds `ClientConnOffset` oder `ClientConnPtr` der MQCNO-Struktur (sofern vorhanden). Diese Felder geben die Kanaldefinitionsstruktur (MQCD) an, die als Definition für den Clientverbindungskanal verwendet werden soll. Verbindungsdetails können mithilfe eines Pre-Connect-Exits überschrieben werden. Weitere Informationen finden Sie unter „Verbindungsdefinitionen unter Verwendung eines Vorverbindungsexits aus einem Repository referenzieren” auf Seite 1045.
2. Wenn die Umgebungsvariable **MQSERVER** festgelegt ist, wird der von ihr definierte Kanal verwendet.
3. Wenn eine `mqclient.ini` -Datei definiert ist und die Zeilengruppe 'Channels' ein Attribut **ServerConnectionParms** enthält, wird der Kanal verwendet, den sie definiert. Weitere Informationen finden Sie unter IBM MQ MQI client -Konfigurationsdatei, `mqclient.ini` und Zeilengruppe 'Channels' der Clientkonfigurationsdatei.
4. Wenn die Umgebungsvariablen **MQCHLLIB** und **MQCHLTAB** festgelegt sind, wird die Definitionstabelle für Clientkanäle verwendet, auf die sie verweisen. Alternativ stellt die Umgebungsvariable **MQCCDTURL** die entsprechende Funktionalität zum Festlegen einer Kombination der Umgebungsvariablen **MQCHLLIB** und **MQCHLTAB** bereit. Wenn **MQCCDTURL** festgelegt ist, wird die Clientkanaldefinitionstabelle verwendet, auf die sie verweist. Weitere Informationen finden Sie unter URL-Zugriff auf die CCDT.

5. Wenn eine Datei `mqclient.ini` definiert ist und die Zeilengruppe 'Channels' die Attribute **ChannelDefinitionDirectory** und **ChannelDefinitionFile** enthält, werden diese Attribute zum Lokalisieren der Definitionstabelle für Clientkanäle verwendet. Weitere Informationen finden Sie unter IBM MQ MQI client -Konfigurationsdatei, `mqclient.ini` und Zeilengruppe 'Channels' der Clientkonfigurationsdatei.
6. Wenn die Umgebungsvariablen nicht gesetzt sind, sucht der Client schließlich nach einer Definitionstabelle für Clientkanäle mit einem Pfad und Namen, die über das Attribut **DefaultPrefix** der Zeilengruppe AllQueueManagers in der Datei `mqs.ini` festgelegt wurden. Weitere Informationen finden Sie unter ZeilengruppeAllQueueManagers in der Datei mqs.ini.

   Verläuft die Suche nach einer Clientkanaldefinitionstabelle ergebnislos, verwendet der Client folgende Pfade:

   - 

     Unter AIX and Linux: `/var/mqm/AMQCLCHL.TAB`
   - 

     Unter Windows: `C:\Programme\IBM\MQ\amqclchl.tab`
   - 

     Unter IBM i: `/QIBM/UserData/mqm/@ipcc`
   - 

     Unter IBM MQ Appliance: *QMname*`_AMQCLCHL.TAB`. Die Pfade sind unter `mqbackup://` `URI` zu finden.

Die erste der zuvor beschriebenen Optionen (anhand des Inhalts des Felds `ClientConnOffset` oder `ClientConnPtr` der MQCNO-Struktur) wird nur vom MQCONNX-Aufruf unterstützt. Wenn die Anwendung MQCONN statt MQCONNX verwendet, werden die Kanalinformationen in der angegebenen Reihenfolge mit den übrigen fünf Verfahren gesucht. Kann der Client die Kanalinformationen nicht finden, schlägt der MQCONN- oder MQCONNX-Aufruf fehl.

Der Kanalname (für die Clientverbindung) muss dem auf dem Server definierten Kanalnamen für die Serververbindung entsprechen, damit der MQCONN- oder MQCONNX-Aufruf erfolgreich ausgeführt werden kann.

Zugehörige Konzepte

Webadressierbarer Zugriff auf die Definitionstabelle für den Clientkanal

Zugehörige Tasks

Verbindungen zwischen dem Server und dem Client konfigurieren

Zugehörige Verweise

Definitionstabelle für den Clientkanal

MQCNO - Verbindungsoptionen

Clientanwendungen mithilfe von Umgebungsvariablen mit Warteschlangenmanagern verbinden

Clientkanalinformationen können einer in einer Clientumgebung ausgeführten Anwendung durch Umgebungsvariablen bereitgestellt werden.

Eine Anwendung, die in einer IBM MQ MQI client-Umgebung ausgeführt wird, kann mithilfe der folgenden Umgebungsvariablen eine Verbindung mit einem Warteschlangenmanager herstellen:

MQSERVER
: Mit der Umgebungsvariablen **MQSERVER** wird ein minimaler Kanal definiert. **MQSERVER** gibt die Position des IBM MQ -Servers und die zu verwendende Übertragungsmethode an.

MQCHLLIB
: Die Umgebungsvariable **MQCHLLIB** gibt den Verzeichnispfad zu der Datei an, die die Definitionstabelle für Clientkanäle (CCDT) enthält. Die Datei wird auf dem Server erstellt, kann aber auf die IBM MQ MQI client-Workstation kopiert werden.

MQCHLTAB
: Die Umgebungsvariable **MQCHLTAB** gibt den Namen der Datei an, die die Clientkanaldefinitionstabelle (CCDT) enthält.

Die Umgebungsvariable **MQCCDTURL** bietet die entsprechende Funktionalität zum Festlegen einer Kombination der Umgebungsvariablen **MQCHLLIB** und **MQCHLTAB** . Mit **MQCCDTURL** können Sie eine Datei-, FTP-

oder HTTP-URL als Einzelwert angeben, aus dem eine Clientkanaldefinitionstabelle abgerufen werden kann. Weitere Informationen finden Sie im Abschnitt Webadressierbarer Zugriff auf die Definitionstabelle für den Clientkanal.

Clientanwendungen mithilfe der MQCNO-Struktur mit Warteschlangenmanagern verbinden
Die Definition des Kanals können Sie in einer Kanaldefinitionsstruktur (MQCD) angeben, die in der MQCNO-Struktur des MQCONNX-Aufrufs bereitgestellt wird.

Weitere Informationen finden Sie unter Clientverbindungskanal auf dem IBM MQ MQI client mit MQCNO erstellen.

Clientanwendungen mithilfe von Clientkanal-Definitionstabellen mit Warteschlangenmanagern verbinden
Wenn Sie den MQSC-Befehl DEFINE CHANNEL verwenden, werden die von Ihnen bereitgestellten Angaben in der Clientkanal-Definitionstabelle (CCDT) gespeichert. Der Inhalt des Parameters **QMgrName** des MQCONN- oder MQCONNX-Aufrufs bestimmt, zu welchem Warteschlangenmanager der Client eine Verbindung herstellt.

Auf diese Datei greift der Client zu, um zu ermitteln, welchen Kanal eine Anwendung verwenden soll. Falls mehrere geeignete Kanaldefinitionen vorliegen, wird die Auswahl des Kanals durch die Kanalattribute CLNTWGHT (Gewichtung des Clientkanals) und AFFINITY (Verbindungsaffinität) beeinflusst.

Automatische Clientverbindungswiederholung verwenden
Sie können Ihre Clientanwendungen automatisch erneut verbinden, ohne zusätzlichen Code schreiben zu müssen, indem Sie eine Reihe von Komponenten konfigurieren.

Die automatische Verbindungswiederholung von Clients erfolgt *integriert*. Die Verbindung wird automatisch an einem beliebigen Punkt im Clientanwendungsprogramm wiederhergestellt und es werden alle Kennungen für geöffnete Objekte wiederhergestellt.

Bei einer manuellen Verbindungswiederholung dagegen muss die Clientanwendung die Verbindung mithilfe von `MQCONN` oder `MQCONNX` erneut herstellen und die Objekte erneut öffnen. Die automatische Clientverbindungswiederholung ist für viele, nicht jedoch für alle Clientanwendungen geeignet.

Weitere Informationen finden Sie im Abschnitt Automatische Clientverbindungswiederholung.

Aufgaben von Clientkanal-Definitionstabellen
Die Clientkanal-Definitionstabelle (CCDT) enthält Definitionen der Clientverbindungskanäle. Diese Tabelle ist besonders hilfreich, wenn Ihre Clientanwendungen Verbindungen zu verschiedenen Warteschlangenmanagern herstellen müssen.

Die Clientkanal-Definitionstabelle wird bei der Definition eines Warteschlangenmanagers erstellt. Die gleiche Datei kann auch von mehreren IBM MQ-Clients verwendet werden.

Eine Clientanwendung kann eine CCDT auf verschiedene Arten verwenden. Die CCDT kann auf den Client-Computer kopiert werden. Sie können die CCDT an eine Position kopieren, die von mehreren Clients gemeinsam genutzt wird. Sie können die CCDT als gemeinsam genutzte Datei für den Client zugänglich machen, während sie sich weiterhin auf dem Server befindet.

Die CCDT kann an einer zentralen Position gehostet werden, auf die über einen URI zugegriffen werden kann. Dadurch entfällt die Notwendigkeit, die CCDT für jeden implementierten Client einzeln zu aktualisieren.

Zugehörige Konzepte
Webadressierbarer Zugriff auf die Definitionstabelle für den Clientkanal
Zugehörige Tasks
Zugreifen auf Clientverbindungskanaldefinitionen
Zugehörige Verweise
Definitionstabelle für den Clientkanal

Warteschlangenmanagergruppen in der CCDT
Sie können eine Gruppe von Verbindungen in der Clientkanal-Definitionstabelle (CCDT) als eine *Warteschlangenmanagergruppe* definieren. Sie können eine Anwendung mit einem Warteschlangenmanager

verbinden, der Teil einer Warteschlangenmanagergruppe ist. Dazu kann dem Warteschlangenmanagernamen in einem MQCONN- oder MQCONNX-Aufruf ein Stern vorangestellt werden.

Sie könnten sich aus folgenden Gründen dafür entscheiden, Verbindungen zu mehreren Servermaschinen zu definieren:

- Sie möchten einen Client mit einem beliebigen aus einer Gruppe von aktiven Warteschlangenmanagern verbinden, um die Verfügbarkeit zu verbessern.
- Sie möchten die Verbindung eines Clients mit demselben Warteschlangenmanager wiederherstellen, mit dem er beim letzten Mal erfolgreich verbunden wurde, aber eine Verbindung mit einem anderen Warteschlangenmanager herstellen, wenn die Verbindung fehlschlägt.
- Sie möchten in der Lage sein, eine neue Clientverbindung zu einem anderen Warteschlangenmanager zu versuchen, wenn die Verbindung fehlschlägt, indem Sie noch einmal MQCONN im Clientprogramm ausgeben.
- Sie möchten automatisch, ohne Clientcode zu schreiben, eine neue Clientverbindung zu einem anderen Warteschlangenmanager herstellen, wenn die Verbindung fehlschlägt.
- Sie möchten automatisch, ohne Clientcode zu schreiben, eine neue Clientverbindung zu einer anderen Instanz eines Mehrinstanz-Warteschlangenmanagers herstellen, wenn eine Standby-Instanz übernimmt.
- Sie möchten Ihre Clientverbindungen zwischen einer Reihe von Warteschlangenmanagern ausgleichen, wobei einige Warteschlangenmanager mehr Clientverbindungen als andere besitzen.
- Sie möchten die Wiederholung zahlreicher Clientverbindungen auf mehrere Warteschlangenmanager und zeitlich verteilen, falls das hohe Verbindungsvolumen einen Ausfall verursacht.
- Sie möchten in der Lage sein, Ihre Warteschlangenmanager zu verschieben, ohne Code der Clientanwendung zu ändern.
- Sie möchten Clientanwendungsprogramme schreiben, denen keine Warteschlangenmanagernamen bekannt sein müssen.

Es ist nicht immer angebracht, Verbindungen zu anderen Warteschlangenmanagern herzustellen. So mag es beispielsweise für einen erweiterten transaktionsorientierten Client oder einen Java-Client in WebSphere Application Server erforderlich sein, eine Verbindung zu einer vorhersehbaren Warteschlangenmanagerinstanz herzustellen. Eine automatische Wiederherstellung einer Client-Verbindung wird von IBM MQ classes for Java nicht unterstützt.

Eine Warteschlangenmanagergruppe ist eine Gruppe von Verbindungen, die in der Definitionstabelle für Clientkanäle (CCDT) definiert wird. Die Gruppe wird dadurch definiert, dass ihre Mitglieder in ihren Kanaldefinitionen denselben Wert für das Attribut **QMNAME** besitzen.

Abbildung 97 auf Seite 974 zeigt eine Clientverbindungstabelle mit drei Warteschlangenmanagergruppen, davon zwei in der CCDT als **QMNAME** (QM1) und **QMNAME** (QMGrp1) definierten Gruppen und einer leeren bzw. Standardgruppe definiert als **QMNAME** (' ').

1. Warteschlangenmanagergruppe QM1 besitzt drei Clientverbindungskanäle, die sie mit den Warteschlangenmanagern QM1 und QM2 verbinden. QM1 kann ein Mehrinstanz-Warteschlangenmanager sein, der sich auf zwei verschiedenen Servern befindet.
2. Die Standard-Warteschlangenmanagergruppe besitzt sechs Clientverbindungskanäle, die sie mit allen Warteschlangenmanagern verbindet.
3. QMGrp1 besitzt Clientverbindungskanäle zu zwei Warteschlangenmanagern, QM4 und QM5.

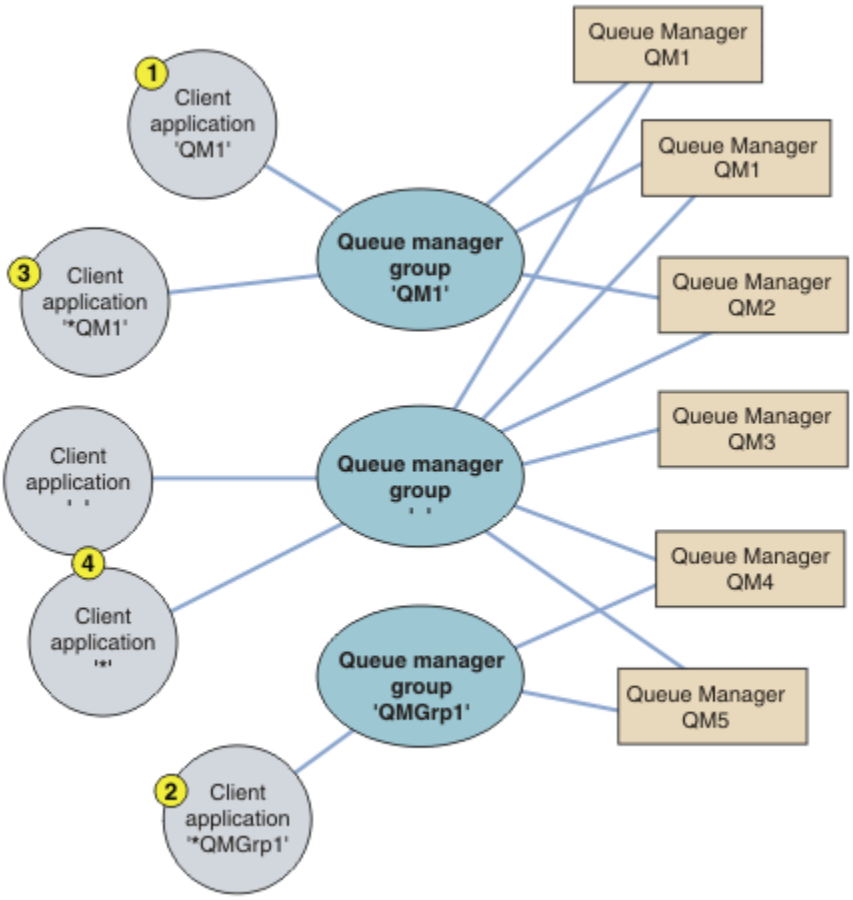


Abbildung 97. WS-Manager-Gruppen

Vier Beispiele für die Verwendung dieser Clientverbindungstabelle werden mithilfe der nummerierten Clientanwendungen in Abbildung 97 auf Seite 974 beschrieben.

1. Im ersten Beispiel übergibt die Clientanwendung einen Warteschlangenmanagernamen, `QM1`, als **`QmgrName`**-Parameter an ihren MQI-Aufruf `MQCONN` oder `MQCONNX`. Der IBM MQ-Clientcode wählt die passende Warteschlangenmanagergruppe QM1 aus. Die Gruppe enthält drei Verbindungskanäle, und der IBM MQ MQI client versucht der Reihe nach über diese Kanäle eine Verbindung zu `QM1` herzustellen, bis er ein IBM MQ-Empfangsprogramm für die Verbindung findet, das mit einem aktiven Warteschlangenmanager mit dem Namen `QM1` verbunden ist.

   Die Reihenfolge der Verbindungsversuche hängt vom Wert des Attributs AFFINITY der Clientverbindung und der Clientkanalgewichtung ab. Im Rahmen dieser Einschränkungen ist die Reihenfolge der Verbindungsversuche wahlfrei, sowohl über die drei möglichen Verbindungen als auch zeitlich, um die Arbeitslast bei der Verbindungsherstellung zu verteilen.

   Der von der Clientanwendung ausgegebene Aufruf `MQCONN` oder `MQCONNX` ist erfolgreich, wenn eine Verbindung zu einer aktiven Instanz von `QM1` aufgebaut wird.
2. Im zweiten Beispiel übergibt die Clientanwendung einen Warteschlangenmanagernamen mit vorangestelltem Stern, `*QMGrp1`, als **`QmgrName`**-Parameter an ihren MQI-Aufruf `MQCONN` oder `MQCONNX`. Der IBM MQ-Client wählt die passende Warteschlangenmanagergruppe QMGrp1 aus. Diese Gruppe enthält zwei Clientverbindungskanäle, und der IBM MQ MQI client versucht über diese Kanäle eine Verbindung zu einem *beliebigen* der Warteschlangenmanager herzustellen. In diesem Beispiel muss der IBM MQ MQI client eine erfolgreiche Verbindung herstellen. Dabei ist der Name des Warteschlangenmanagers, mit dem die Verbindung hergestellt wird, unerheblich.

   Die Regel für die Reihenfolge der Verbindungsversuche entspricht der vorhergehenden. Der einzige Unterschied besteht darin, dass der Client mit dem Stern vor dem Namen des Warteschlangenmanagers angibt, dass der Name des Warteschlangenmanagers keine Rolle spielt.

Der von der Clientanwendung ausgegebene Aufruf MQCONN oder MQCONNX ist erfolgreich, wenn eine Verbindung zu einer aktiven Instanz eines Warteschlangenmanagers aufgebaut wird, der über die Kanäle in der Warteschlangenmanagergruppe QMGrp1 verbunden ist.

3. Das dritte Beispiel entspricht im Wesentlichen dem zweiten, da dem Parameter **QmgrName** ein Stern vorangestellt wird: *QM1. Das Beispiel veranschaulicht, dass Sie nicht den Warteschlangenmanager bestimmen können, zu dem eine Clientkanalverbindung hergestellt wird, indem Sie allein das Attribut QMNAME in einer Kanaldefinition überprüfen. Das Attribut **QMNAME** der Kanaldefinition lautet QM1, aber das reicht nicht für die Anforderung aus, dass eine Verbindung zu einem Warteschlangenmanager namens QM1 hergestellt wird. Wenn die Clientanwendung dem Parameter **QmgrName** einen Stern voranstellt, ist jeder Warteschlangenmanager ein mögliches Verbindungsziel.

   In diesem Fall ist der von der Clientanwendung ausgegebene Aufruf MQCONN oder MQCONNX erfolgreich, wenn eine Verbindung zu einer aktiven Instanz von QM1 oder QM2 hergestellt wird.

4. Das vierte Beispiel stellt die Verwendung der Standardgruppe dar. In diesem Fall übergibt die Clientanwendung einen Stern, '*', oder ein Leerzeichen, ' ', als **QmgrName**-Parameter an ihren MQI-Aufruf MQCONN oder MQCONNX. Gemäß Konvention bezeichnet ein leeres **QMNAME**-Attribut in der Clientkanaldefinition die Standard-Warteschlangenmanagergruppe und ein **QmgrName**-Parameter, der leer ist oder einen Stern enthält, entspricht einem leeren **QMNAME**-Attribut.

   In diesem Beispiel besitzt die Standard-Warteschlangenmanagergruppe Clientverbindungskanäle zu allen Warteschlangenmanagern. Durch Auswahl der Standard-Warteschlangenmanagergruppe kann die Anwendung mit jedem Warteschlangenmanager in der Gruppe verbunden werden.

   Der von der Clientanwendung ausgegebene Aufruf MQCONN oder MQCONNX ist erfolgreich, wenn eine Verbindung zu einer aktiven Instanz eines Warteschlangenmanagers aufgebaut wird.

   Anmerkung: Die Standardgruppe unterscheidet sich von einem Standard-Warteschlangenmanager, auch wenn eine Anwendung einen leeren **QmgrName**-Parameter verwendet, um eine Verbindung zur Standard-Warteschlangenmanagergruppe oder zum Standard-Warteschlangenmanager herzustellen. Der Begriff einer Standard-Warteschlangenmanagergruppe ist nur für eine Clientanwendung relevant, der eines Standard-Warteschlangenmanagers für eine Serveranwendung.

Definieren Sie Ihre Clientverbindungskanäle auf nur einem Warteschlangenmanager, einschließlich der Kanäle, die Verbindungen zu einem zweiten oder dritten Warteschlangenmanager herstellen. Definieren Sie sie nicht auf zwei Warteschlangenmanagern, um dann zu versuchen, die beiden Definitionstabellen für Clientkanäle zusammenzuführen. Der Client kann nur auf eine Definitionstabelle für Clientkanäle zugreifen.

Beispiele

Schauen Sie sich noch einmal die am Anfang des Abschnitts aufgeführte Liste der Gründe für die Verwendung von Warteschlangenmanagergruppen an. Wie wird diese Funktionalität mit der Verwendung einer Warteschlangenmanagergruppe bereitgestellt?

Verbindung mit einem beliebigen aus einer Gruppe von Warteschlangenmanagern.
: Definieren Sie eine Warteschlangenmanagergruppe mit Verbindungen zu allen Warteschlangenmanagern in der Gruppe und stellen Sie mit dem Parameter **QmgrName** mit vorangestelltem Stern eine Verbindung zur Gruppe her.

Wiederherstellung der Verbindung zum selben Warteschlangenmanager, aber Verbindung zu einem anderen, wenn der Warteschlangenmanager, zu dem die letzte Verbindung hergestellt wurde, nicht verfügbar ist.
: Definieren Sie wie zuvor eine Warteschlangenmanagergruppe, setzen Sie das Attribut **AFFINITY** jedoch für jede Clientkanaldefinition auf (PREFERRED).

Versuch, eine neue Verbindung zu einem anderen Warteschlangenmanager herzustellen, wenn eine Verbindung fehlschlägt.
: Stellen Sie eine Verbindung zu einer Warteschlangenmanagergruppe her und geben Sie den MQI-Aufruf MQCONN oder MQCONNX erneut aus, wenn die Verbindung unterbrochen wird oder der Warteschlangenmanager ausfällt.

Automatische Herstellung einer neuen Verbindung zu einem anderen Warteschlangenmanager, wenn eine Verbindung fehlschlägt.

Stellen Sie mit der **MQCNO**-Option MQCNO_RECONNECT für MQCONNX eine Verbindung zu einer Warteschlangenmanagergruppe her.

Automatische Herstellung einer neuen Verbindung zu einer anderen Instanz eines Mehrinstanz-Warteschlangenmanagers.

Gehen Sie wie im vorherigen Beispiel vor. Wenn Sie in diesem Fall die Einschränkung angeben möchten, dass die Warteschlangenmanagergruppe Verbindungen zu den Instanzen eines bestimmten Mehrinstanz-Warteschlangenmanagers herstellt, definieren Sie die Gruppe nur mit Verbindungen zu den Instanzen des Mehrinstanz-Warteschlangenmanagers.

Sie können auch für die Clientanwendung vorgeben, dass ihr MQI-Aufruf MQCONN oder MQCONNX ohne Stern vor dem Parameter **QmgrName** ausgegeben wird. So kann die Clientanwendung nur eine Verbindung zum benannten Warteschlangenmanager herstellen. Schließlich können Sie die Option **MQCNO** auf MQCNO_RECONNECT_Q_MGR festlegen. Diese Option akzeptiert wiederholte Verbindungen zu dem Warteschlangenmanager, der vorher verbunden war. Sie können mit diesem Wert auch die Verbindungswiederholungen zur selben Instanz eines normalen Warteschlangenmanagers beschränken.

Ausgleich von Clientverbindungen zwischen Warteschlangenmanagern, wobei einige Warteschlangenmanager mehr Clientverbindungen als andere besitzen.

Definieren Sie eine Warteschlangenmanagergruppe und legen Sie das Attribut **CLNTWGHT** in jeder Clientkanaldefinition für eine ungleichmäßige Verteilung der Verbindungen fest.

Ungleichmäßige und zeitliche Verteilung der Arbeitslast bei der Wiederholung von Clientverbindungen nach Ausfall einer Verbindung oder eines Warteschlangenmanagers.

Gehen Sie wie im vorherigen Beispiel vor. Der IBM MQ MQI client verteilt erneute Verbindungen mit den Warteschlangenmanagern nach dem Zufallsprinzip, so dass sich Neuverbindungen mit der Zeit auf alle Warteschlangenmanager verteilen.

Verschieben von Warteschlangenmanagern ohne Ändern von Clientcode.

Die Definitionstabelle für Clientkanäle isoliert die Clientanwendung von der Position des Warteschlangenmanagers. Die CCDT (Client Channel Definition Table) ist eine Datendatei, die entweder auf dem Client definiert, aus einer gemeinsam genutzten Speicherposition ausgelesen oder von einem Web-Server abgerufen werden kann. Weitere Informationen hierzu finden Sie im Abschnitt Definitionstabelle für Clientkanal.

Schreiben einer Clientanwendung, der keine Namen von Warteschlangenmanagern bekannt sind.

Verwenden Sie Namen für Warteschlangenmanagergruppen und erstellen Sie eine Namenskonvention für Namen von Warteschlangenmanagergruppen, die für die Clientanwendungen in Ihrem Unternehmen relevant ist und die Architektur Ihrer Lösungen widerspiegelt, nicht die Benennung von Warteschlangenmanagern.

z/OS *Connecting to queue sharing groups*

You can connect your application to a queue manager that is part of a queue sharing group. This can be done by using the queue sharing group name instead of the queue manager name on the MQCONN or MQCONNX call.

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.

The client channel definition should use the queue sharing group generic interface to connect to an available queue manager in the group. For more information, see Connecting a client to a queue sharing group. A check is made to ensure that the queue manager the listener connects to is a member of the queue sharing group.

For more information on shared queues, see Shared queues and queue sharing groups.

Beispiele für Kanalgewichtung und Affinität

Diese Beispiele veranschaulichen die Auswahl von Clientverbindungskanälen bei Verwendung von Kanalgewichtungen (ClientChannelWeights) ungleich null.

Die Kanalattribute `ClientChannelWeight` und `ConnectionAffinity` steuern die Auswahl der Clientverbindungskanäle, wenn für die Verbindung mehrere geeignete Kanäle verfügbar sind. Zur Sicherstellung einer höheren Verfügbarkeit und eines Lastausgleichs sind diese Kanäle so konfiguriert, dass sie Verbindungen mit verschiedenen Warteschlangenmanagern herstellen können. Wenn ein MQCONN-Aufruf eine Verbindung mit einem von mehreren Warteschlangenmanagern herstellen könnte, muss dem Namen des Warteschlangenmanagers ein Stern vorangestellt werden. Dies ist im folgenden Abschnitt beschrieben: Beispiele für MQCONN-Aufrufe: Beispiel 1: Der Name des Warteschlangenmanagers enthält einen Stern (*).

Kanalkandidaten für eine Verbindung sind alle Kanäle, deren Attribut QMNAME mit dem im MQCONN-Aufruf angegebenen Warteschlangenmanagernamen übereinstimmt. Wenn alle Kanalkandidaten für eine Verbindung die Standard-Clientkanalgewichtung (`ClientChannelWeight`) null haben, werden sie in alphabetischer Reihenfolge ausgewählt. Dies ist im folgenden Beispiel gezeigt: Beispiele für MQCONN-Aufrufe: Beispiel 1: Der Name des Warteschlangenmanagers enthält einen Stern (*).

Die folgenden Beispiele veranschaulichen, was geschieht, wenn Clientkanalgewichtungen (`ClientChannelWeight`) ungleich null verwendet werden. Da die Auswahl der Kanäle bei dieser Funktion pseudo-zufällig erfolgt, handelt es sich bei der in den Beispielen gezeigten Aktionsfolge lediglich um ein Beispiel, das stattfinden kann, aber nicht muss.

Beispiel 1. Kanäle bei Festlegung von PREFERRED für 'ConnectionAffinity' auswählen
Dieses Beispiel zeigt, wie ein IBM MQ MQI client einen Kanal aus einer CCDT auswählt, wenn die Verbindungsaffinität (ConnectionAffinity) auf PREFERRED gesetzt ist.

In diesem Beispiel verwenden mehrere Clientsysteme eine von einem Warteschlangenmanager bereitgestellte Clientkanal-Definitionstabelle (CCDT). Die CCDT enthält Clientverbindungskanäle mit den folgenden Attributen (in der Syntax des Befehls DEFINE CHANNEL angegeben):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

Die Anwendung gibt MQCONN(*CORE) aus.

Kanal A ist kein Kandidat für diese Verbindung, da das Attribut QMNAME nicht übereinstimmt. Die Kanäle B, C und D werden als Kandidaten ermittelt und in der Reihenfolge ihrer Gewichtung ausprobiert. In diesem Beispiel kann die Reihenfolge C, B, D sein. Der Client versucht, eine Verbindung zum Warteschlangenmanager bei core2.ops.company.example herzustellen. Der Name des Warteschlangenmanagers an dieser Adresse wird nicht überprüft, da der MQCONN-Aufruf anstelle des Namens eines Warteschlangenmanagers einen Stern enthielt.

Beachten Sie, dass dieses Clientsystem bei der Einstellung `AFFINITY(PREFERRED)` die Kanäle bei jedem Verbindungsversuch in der gleichen Anfangsreihenfolge durchprobiert. Dies gilt auch, wenn die Verbindungen aus unterschiedlichen Prozessen oder zu verschiedenen Zeiten erfolgen.

In diesem Beispiel kann der Warteschlangenmanager unter core.2.ops.company.example nicht erreicht werden. Der Client versucht daraufhin, eine Verbindung mit core1.ops.company.example herzustellen, da Kanal B in der bevorzugten Reihenfolge als nächstes folgt. Außerdem rückt Kanal C nun in der Reihenfolge ganz nach hinten.

Von der gleichen Anwendung wird ein zweiter MQCONN(*CORE)-Aufruf ausgegeben. Da Kanal C durch die vorherige Verbindung herabgestuft wurde, ist nun also Kanal B an der Reihe; diese Verbindung erfolgt zu core1.ops.company.example.

Ein zweites System, das die gleiche Definitionstabelle für Clientkanäle verwendet, kann für die Kanäle eine andere Anfangsreihenfolge Beispiel: D, B, C. Wenn alle Kanäle funktionieren, was normalerweise ja der Fall ist, werden die Anwendungen auf diesem System mit core3.ops.company.example verbunden, während diejenigen des ersten Systems mit core2.ops.company.example verbunden werden. Auf diese Weise kann auch bei sehr vielen Clients ein Lastausgleich über mehrere Warteschlangenmanager

erfolgen, wobei die einzelnen Clients ihre Verbindungen jeweils zum gleichen Warteschlangenmanager herstellen, sofern dieser verfügbar ist.

Beispiel 2. Kanäle auswählen, wenn 'ConnectionAffinity' auf NONE gesetzt ist
Dieses Beispiel zeigt, wie ein IBM MQ MQI client einen Kanal aus einer CCDT auswählt, wenn die Verbindungsaffinität (ConnectionAffinity) auf NONE gesetzt ist.

In diesem Beispiel verwenden mehrere Clients eine von einem Warteschlangenmanager bereitgestellte Clientkanal-Definitionstabelle (CCDT). Die CCDT enthält Clientverbindungskanäle mit den folgenden Attributen (in der Syntax des Befehls DEFINE CHANNEL angegeben):

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

Die Anwendung gibt MQCONN(*CORE) aus. Wie im vorherigen Beispiel wird Kanal A nicht berücksichtigt, weil der QMNAME nicht übereinstimmt. Kanal B, C oder D werden aufgrund ihrer Gewichtung ausgewählt, und zwar mit einer Wahrscheinlichkeit von 50%, 30% bzw. 20%. In diesem Beispiel wird Kanal B gewählt. Eine feste Präferenzreihenfolge wird nicht festgelegt.

Ein zweiter MQCONN(*CORE)-Aufruf wird ausgegeben. Wiederum wird einer der drei übereinstimmenden Kanäle mit der gleichen Wahrscheinlichkeit ausgewählt. In diesem Beispiel wird Kanal C gewählt. Der Warteschlangenmanager an core2.ops.company.example antwortet jedoch nicht, weshalb aus den verbleibenden Kanaloptionen ein anderer Kanal gewählt wird. Nun wird Kanal B gewählt, und die Anwendung wird mit core1.ops.company.example verbunden.

Bei AFFINITY(NONE) ist jeder MQCONN-Aufruf von allen vorangegangenen Aufrufen unabhängig. Bei einem dritten MQCONN(*CORE)-Aufruf dieser Beispielanwendung kann es daher durchaus passieren, dass wiederum zunächst eine Verbindung über den nicht reagierenden Kanal C versucht wird, bevor Kanal B oder D ausprobiert wird.

Beispiele für MQCONN-Aufrufe
Beispiele für die Verwendung von MQCONN für die Verbindung mit einem bestimmten Warteschlangenmanager oder für die Verbindung mit einem Warteschlangenmanager aus einer Warteschlangenmanagergruppe.

In allen folgenden Beispielen ist das Netz gleich. Vom gleichen IBM MQ MQI client ist eine Verbindung zu zwei Servern definiert. (Anstelle des MQCONN-Aufrufs kann in allen diesen Beispielen auch der MQCONNX-Aufruf verwendet werden.)

Auf den Serversystemen laufen die beiden Warteschlangenmanager SALE und SALE_BACKUP.

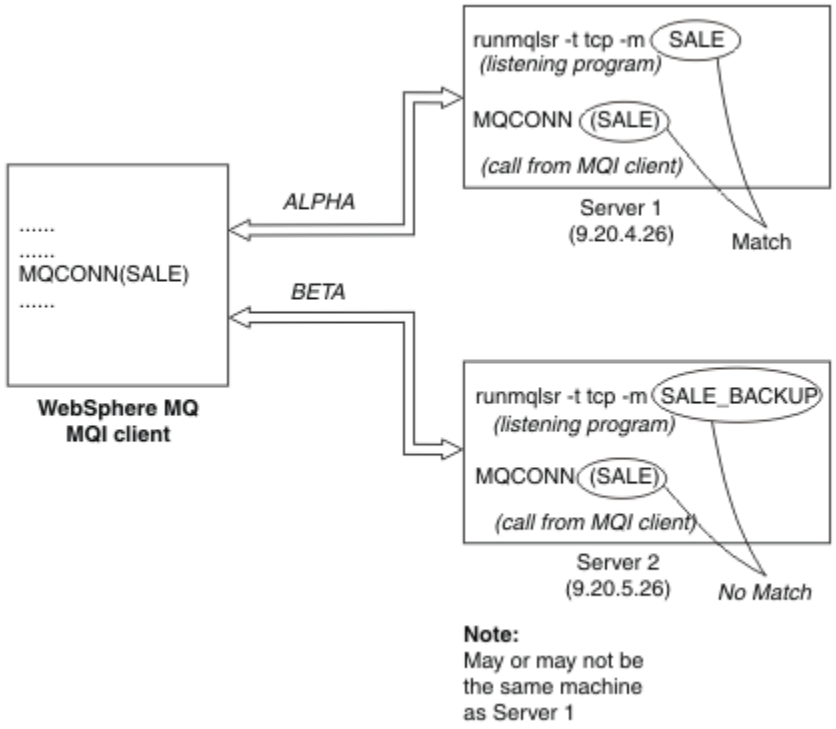


Abbildung 98. Beispiel für MQCONN-Aufruf

Die Kanäle in diesen Beispielen sind wie folgt definiert:

Definitionen für SALE:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Definition für SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

Hier eine Zusammenfassung der Clientkanaldefinitionen:

Name	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

Was die MQCONN-Beispiele veranschaulichen
Die Beispiele veranschaulichen die Verwendung mehrerer Warteschlangenmanager als Ausweichsystem.

Nehmen wir an, die Kommunikation mit Server 1 ist vorübergehend unterbrochen. In einem solchen Fall ist die Verwendung mehrerer Warteschlangenmanager als Ausweichsystem sehr nützlich. Dieses soll hier veranschaulicht werden.

Jedes Beispiel behandelt einen anderen MQCONN-Aufruf und erläutert die Vorgänge im jeweiligen Fall, wobei die folgenden Regeln gelten:

1. Die Clientkanal-Definitionstabelle (CCDT) wird in alphabetischer Reihenfolge der Kanalnamen nach dem im MQCONN-Aufruf angegebenen Warteschlangenmanagernamen (QMNAME) durchsucht.
2. Bei einer Übereinstimmung wird die Kanaldefinition verwendet.
3. Es wird versucht, den Kanal zu dem durch den Verbindungsnamen (CONNAME) angegebenen System zu starten. Bei Erfolg wird die Anwendung fortgesetzt. Für eine erfolgreiche Verbindung ist Folgendes erforderlich:
   - Ein aktives Empfangsprogramm auf dem Server.
   - Das Empfangsprogramm muss mit dem Warteschlangenmanager verbunden sein, zu dem der Client eine Verbindung herstellen will (sofern angegeben).
4. Falls der Versuch, den Kanal zu starten, fehlschlägt und die Clientkanal-Definitionstabelle noch weitere Einträge enthält (im Beispiel enthält sie zwei Einträge), wird die Tabelle nach einer weiteren Übereinstimmung durchsucht. Wird eine solche gefunden, so beginnt die Verarbeitung wieder bei Schritt 1.
5. Wird keine Übereinstimmung gefunden bzw. enthält die Clientkanal-Definitionstabelle keine weiteren Einträge mehr und der Kanal konnte nicht gestartet werden, so kann die Anwendung keine Verbindung herstellen. In diesem Fall gibt der MQCONN-Aufruf einen entsprechenden Beendigungs- und Ursachencode zurück. Auf Grundlage der zurückgegebenen Codes kann die Anwendung geeignete Schritte unternehmen.

Beispiel 1. Name des Warteschlangenmanagers enthält Stern (*)
In diesem Beispiel spielt es für die Anwendung keine Rolle, mit welchem Warteschlangenmanager sie verbunden wird. Die Anwendung gibt einen MQCONN-Aufruf mit einem Warteschlangenmanagernamen aus, der einen Stern (*) enthält. Daraufhin wird ein geeigneter Kanal ausgewählt.

Die Anwendung gibt folgenden Aufruf aus:

```
MQCONN (*SALE)
```

Den Regeln entsprechend geschieht nun Folgendes:

1. Die Clientkanal-Definitionstabelle (CCDT) wird nach dem im MQCONN-Aufruf angegebenen Warteschlangenmanagernamen SALE durchsucht.
2. Die Kanaldefinitionen für ALPHA und BETA werden gefunden.
3. Wenn ein Kanal den CLNTWGHT-Wert 0 hat, wird dieser Kanal ausgewählt. Wenn beide Kanäle den CLNTWGHT-Wert 0 haben, wird Kanal ALPHA ausgewählt, da er in alphabetischer Reihenfolge der erste Kanal ist. Wenn beide Kanäle einen CLNTWGHT-Wert ungleich null haben, wird der Kanal nach seiner Gewichtung ausgewählt.
4. Es wird versucht, den Kanal zu starten.
5. Wenn Kanal BETA ausgewählt wurde, ist der Start erfolgreich.
6. Wenn Kanal ALPHA ausgewählt wurde, ist der Start nicht erfolgreich, da die Kommunikationsverbindung unterbrochen ist. In diesem Fall werden folgende Schritte ausgeführt:
   a. Der einzige andere Kanal für den Warteschlangenmanager SALE ist BETA.
   b. Es wird versucht, diesen Kanal zu starten - mit Erfolg.
7. Die Überprüfung, ob ein Empfangsprogramm aktiv ist, ist erfolgreich. Dieses ist zwar nicht mit dem Warteschlangenmanager SALE verbunden, da der MQI-Aufrufparameter jedoch einen Stern (*) im Warteschlangenmanagernamen enthält, wird diesbezüglich keine Überprüfung vorgenommen. Die Anwendung wird mit dem Warteschlangenmanager SALE_BACKUP verbunden und die Verarbeitung kann fortgesetzt werden.

Beispiel 2. Name des WS-Managers angegeben
In diesem Beispiel muss die Anwendung eine Verbindung zu einem bestimmten Warteschlangenmanager herstellen. Die Anwendung gibt einen MQCONN-Aufruf mit dem Namen dieses Warteschlangenmanagers aus. Daraufhin wird ein geeigneter Kanal ausgewählt.

Die Anwendung fordert eine Verbindung zu einem bestimmten Warteschlangenmanager namens SALE an und gibt dazu folgenden MQI-Aufruf aus:

```
MQCONN (SALE)
```

Den Regeln entsprechend geschieht nun Folgendes:

1. Die Clientkanal-Definitionstabelle (CCDT) wird in alphabetischer Reihenfolge der Kanalnamen nach dem im MQCONN-Aufruf angegebenen Warteschlangenmanagernamen SALE durchsucht.
2. Die erste Kanaldefinition, die übereinstimmt, ist ALPHA.
3. Es wird versucht, diesen Kanal zu starten - erfolglos, weil die Kommunikationsverbindung unterbrochen ist.
4. Die Clientkanal-Definitionstabelle (CCDT) wird erneut nach dem Warteschlangenmanagernamen SALE durchsucht, wobei der Kanal BETA gefunden wird.
5. Es wird versucht, diesen Kanal zu starten - mit Erfolg.
6. Die Überprüfung, ob ein Empfangsprogramm aktiv ist, ist erfolgreich, jedoch ist dieses Empfangsprogramm nicht mit dem Warteschlangenmanager SALE verbunden.
7. Die Definitionstabelle für Clientkanäle enthält keine weiteren Einträge mehr. Die Anwendung kann die Verarbeitung nicht fortsetzen und erhält den Rückgabecode MQRC_Q_MGR_NOT_AVAILABLE.

Beispiel 3. Name des Warteschlangenmanagers ist leer oder ein Stern (*)
In diesem Beispiel spielt es für die Anwendung keine Rolle, mit welchem Warteschlangenmanager sie verbunden wird. Die Anwendung gibt einen MQCONN-Aufruf ohne Warteschlangenmanagername bzw. mit einem Stern als Warteschlangenmanagername aus. Daraufhin wird ein geeigneter Kanal ausgewählt.

Dieser Aufruf wird genauso behandelt wie der Aufruf in „Beispiel 1. Name des Warteschlangenmanagers enthält Stern (*)" auf Seite 980.

Anmerkung: Wird die Anwendung allerdings in einer anderen Umgebung als der IBM MQ MQI client-Umgebung ausgeführt und der Name ist leer, so würde der Aufruf versuchen, eine Verbindung zum Standard-Warteschlangenmanager herzustellen. Bei Ausführung aus einer Clientumgebung ist dies allerdings nicht der Fall. Hier erfolgt der Zugriff auf den Warteschlangenmanager, mit dem der Listener des ausgewählten Kanals verbunden ist.

Die Anwendung gibt folgenden Aufruf aus:

```
MQCONN ("")
```

oder

```
MQCONN (*)
```

Den Regeln entsprechend geschieht nun Folgendes:

1. Die Clientkanal-Definitionstabelle (CCDT) wird in alphabetischer Reihenfolge der Kanalnamen nach einem leeren Warteschlangenmanagernamen durchsucht, wie im MQCONN-Aufruf angegeben.
2. Der Eintrag für den Kanalnamen ALPHA enthält einen Warteschlangenmanagernamen in der Definition von SALE. Dies entspricht nicht dem MQCONN-Aufrufparameter, für den ein leerer Warteschlangenmanagername erforderlich ist.
3. Der nächste Eintrag gehört zum Kanalnamen BETA.
4. Die `queue manager name` in der Definition ist SALE. Auch hier entspricht dies nicht dem MQCONN-Aufrufparameter, für den ein leerer Warteschlangenmanagername erforderlich ist.

5. Die Definitionstabelle für Clientkanäle enthält keine weiteren Einträge mehr. Die Anwendung kann die Verarbeitung nicht fortsetzen und erhält den Rückgabecode MQRC_Q_MGR_NOT_AVAILABLE.

Auslösen in der Clientumgebung

Nachrichten, die von IBM MQ-Anwendungen auf IBM MQ MQI clients gesendet werden, tragen genauso zum Auslösen bei, wie jede andere Nachricht auch. Dabei können sie zum Auslösen von Programmen sowohl auf dem Server als auch auf dem Client verwendet werden.

Die Auslösefunktion wird in allen Einzelheiten im Abschnitt Kanalauslösung beschrieben.

Der Auslösemonitor und die zu startende Anwendung müssen sich auf dem gleichen System befinden.

Die Standardeigenschaften der ausgelösten Warteschlange sind die gleichen wie in der Serverumgebung. Wenn in einer Clientanwendung, die Nachrichten in eine ausgelöste Warteschlange einreiht, die für einen z/OS -Warteschlangenmanager lokal ist, keine MQPMO-Optionen für die Synchronisationspunktsteuerung angegeben sind, werden die Nachrichten innerhalb einer Arbeitseinheit eingereiht. Wird die Auslöserbedingung in diesem Fall erfüllt, so wird die Auslösenachricht innerhalb der gleichen Arbeitseinheit in die Initialisierungswarteschlange eingereiht und kann vom Auslösemonitor so lange nicht abgerufen werden, bis die Arbeitseinheit beendet ist. Der auszulösende Prozess wird also nicht vor Ende der Arbeitseinheit gestartet.

Prozessdefinition
Die Prozessdefinition müssen Sie auf dem Server definieren, da dieser der Warteschlange zugeordnet ist, für die die Auslösefunktion aktiviert ist.

Das Prozessobjekt definiert, was ausgelöst werden soll. Wenn Client und Server nicht auf derselben Plattform ausgeführt werden, muss in allen durch den Auslösemonitor gestarteten Prozessen der Anwendungstyp (*ApplType*) definiert werden. Andernfalls verwendet der Server seine Standarddefinitionen (d. h. den Anwendungstyp, der normalerweise der Servermaschine zugeordnet ist) und verursacht einen Fehler.

Läuft der Auslösemonitor beispielsweise auf einem IBM MQ MQI client und es soll eine Anforderung an einen Server gesendet werden, der unter einem anderen Betriebssystem ausgeführt wird, muss MQAT_WINDOWS_NT angegeben werden. Andernfalls verwendet das andere Betriebssystem die Standarddefinitionen und der Prozess schlägt fehl.

Multi *Auslösemonitor*
Der von IBM MQ for Multiplatforms bereitgestellte Auslösemonitor wird in den Clientumgebungen für Multiplatforms-Systeme ausgeführt.

Führen Sie zur Ausführung des Auslösemonitors einen der folgenden Befehle aus:

- IBM i Unter IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW Auf AIX, Linux, and Windows-Plattformen:

```
runmqtmc [-m QMgrName] [-q InitQ]
```

Die Standardinitialisierungswarteschlange ist die Warteschlange SYSTEM.DEFAULT.INITIATION.QUEUE auf dem Standardwarteschlangenmanager. Die Initialisierungswarteschlange ist die Warteschlange, die der Auslösemonitor nach Auslösenachrichten durchsucht. Danach ruft er die entsprechenden Programme für die Auslösenachrichten auf. Dieser Auslösemonitor unterstützt den Standardanwendungstyp und ist identisch mit `runmqtrm`, mit der Ausnahme, dass er die Clientbibliotheken verknüpft.

Die vom Auslösemonitor erstellte Befehlszeichenfolge besteht aus folgenden Elementen:

1. Die Anwendungs-ID (*ApplicId*) aus der relevanten Prozessdefinition. Bei *ApplicId* handelt es sich um den Namen des auszuführenden Programms in dem Format, in dem er in der Befehlszeile eingegeben wird.

2. Die MQTMC2-Struktur aus der Initialisierungswarteschlange, eingeschlossen in Anführungszeichen. Eine Befehlsfolge mit exakt der dargestellten Zeichenfolge wird in doppelten Anführungszeichen gestartet, damit der Systembefehl diese als einen Parameter akzeptiert.
3. Die Anwendungs-ID (*EnvrData*) aus der relevanten Prozessdefinition.

Erst nach Abschluss der gestarteten Anwendung durchsucht der Auslösemonitor die Initialisierungswarteschlange wieder nach einer weiteren Nachricht. Falls die Anwendungsverarbeitung lange dauert, kommt der Auslösemonitor daher unter Umständen den in der Warteschlange auflaufenden Auslösenachrichten nicht mehr nach. In einer solchen Situation haben Sie zwei Möglichkeiten:

1. Richten Sie weitere Auslösemonitore ein.

   Bei mehreren Auslösemonitoren können Sie die maximale Anzahl der gleichzeitig ausführbaren Anwendungen eingrenzen.
2. Führen Sie die gestarteten Anwendungen im Hintergrund aus.

   Bei einer Hintergrundausführung grenzt IBM MQ die Anzahl der ausführbaren Anwendungen nicht ein.

Um die gestartete Anwendung im Hintergrund auf AIX and Linux -Systemen auszuführen, müssen Sie am Ende der *EnvrData* der Prozessdefinition ein Et-Zeichen (&) eingeben.

CICS-Anwendungen (außer z/OS)
Ein Nicht-z/OS CICS-Anwendungsprogramm, das einen MQCONN- oder MQCONNX-Aufruf ausgibt, muss für CEDA als RESIDENT definiert sein. Wenn Sie eine CICS-Serveranwendung als Client neu verbinden, geht unter Umständen die Synchronisationspunktunterstützung verloren.

Ein Nicht-z/OS CICS-Anwendungsprogramm, das einen MQCONN- oder MQCONNX-Aufruf ausgibt, muss für CEDA als RESIDENT definiert sein. Um den residenten Code so klein wie möglich zu halten, können Sie für die Ausgabe eines MQCONN- oder MQCONNX-Aufrufs eine Verbindung zu einem separaten Programm herstellen.

Wenn die MQSERVER-Umgebungsvariable zum Definieren der Clientverbindung verwendet wird, muss sie in der Datei CICSENV.CMD angegeben werden.

IBM MQ-Anwendungen können ohne Codeänderung in einer IBM MQ-Serverumgebung oder auf einem IBM MQ-Client ausgeführt werden. In einer IBM MQ-Serverumgebung kann CICS jedoch als Synchronisationspunktkoordinator fungieren, und Sie verwenden in dieser Umgebung EXEC CICS SYNCPOINT und EXEC CICS SYNCPOINT ROLLBACK anstelle von **MQCMIT** und **MQBACK**. Wenn eine CICS-Anwendung aus einer solchen Umgebung nun einfach als Client neu verbunden wird, geht die Synchronisationspunktunterstützung verloren. **MQCMIT** und **MQBACK** müssen für die Anwendung verwendet werden, die auf einem IBM MQ MQI clientausgeführt wird.

ALW CICS- und Tuxedo-Anwendungen vorbereiten und ausführen

Wenn Sie CICS- und Tuxedo-Anwendungen als Clientanwendungen ausführen möchten, müssen Sie andere Bibliotheken als diejenigen für Ihre Serveranwendungen verwenden. Ebenso unterscheidet sich auch die Benutzer-ID, unter denen die Anwendungen aktiv sind.

Zur Vorbereitung von CICS- und Tuxedo-Anwendungen für die Ausführung als IBM MQ MQI client-Anwendungen folgen Sie den Anweisungen unter Erweiterten transaktionsorientierten Client konfigurieren.

Beachten Sie jedoch, dass in den Informationen, die speziell auf die Vorbereitung von CICS- und Tuxedo-Anwendungen eingehen (auch in den mit IBM MQ bereitgestellten Beispielprogrammen), davon ausgegangen wird, dass Sie Ihre Anwendungen für die Ausführung auf einem IBM MQ-Serversystem vorbereiten. Daher beziehen sich diese Informationen nur auf IBM MQ-Bibliotheken für Serversysteme. Wenn Sie dagegen Clientanwendungen vorbereiten, müssen Sie Folgendes beachten:

- Verwenden Sie für das von Ihrer Anwendung verwendete programmiersprachenbezogene Binden die entsprechende Clientsystembibliothek. For example:
  - Linux AIX Verwenden Sie für in C geschriebene Anwendungen unter AIX and Linux die Bibliothek 'libmqic' (statt 'libmqm').

– Windows Oder unter Windows die Bibliothek mqic.lib (statt mqm.lib).

- Verwenden Sie statt der in Tabelle 134 auf Seite 984 und Tabelle 135 auf Seite 984 auf geführten Serversystembibliotheken die entsprechenden Clientsystembibliotheken. Ist eine Serversystembibliothek nicht in diesen Tabellen aufgeführt, verwenden Sie die entsprechende Bibliothek für Clientsysteme.

Tabelle 134. Clientsystembibliotheken unter AIX and Linux

Bibliothek für ein IBM MQ-Serversystem	**Entsprechende Bibliothek für ein IBM MQ-Clientsystem**
libmqmxa	libmqcxa
V 9.4.0 V 9.4.0 libmqmxa64	V 9.4.0 V 9.4.0 libmqcxa64

Tabelle 135. Bibliotheken für Clientsysteme unter Windows

Bibliothek für ein IBM MQ-Serversystem	**Entsprechende Bibliothek für ein IBM MQ-Clientsystem**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

Von einer Clientanwendung verwendete Benutzer-ID

Wenn Sie eine IBM MQ-Serveranwendung unter CICS ausführen, wechselt die Anwendung normalerweise vom CICS-Benutzer zur Benutzer-ID der Transaktion. Bei einer IBM MQ MQI client-Anwendung, die Sie unter CICS ausführen, bleiben hingegen die privilegierten CICS-Berechtigungen erhalten.

ALW *Beispielprogramme für CICS und Tuxedo*

Beispielprogramme für CICS und Tuxedo für die Verwendung auf AIX, Linux, and Windows-Systemen.

In Tabelle 136 auf Seite 984 finden Sie eine Liste der CICS- und Tuxedo-Beispielprogramme für AIX and Linux-Clientsysteme. In Tabelle 137 auf Seite 985 finden Sie die entsprechenden Informationen für Windows-Clientsysteme. Außerdem aufgeführt in den Tabellen sind die für die Vorbereitung und Ausführung der Programme benötigten Dateien. Eine Beschreibung der Beispielprogramme finden Sie in den Abschnitten „Das CICS Transaction-Beispielprogramm” auf Seite 1137 und „TUXEDO-Beispiele unter AIX, Linux, and Windows verwenden” auf Seite 1184.

Tabelle 136. Beispielprogramme für AIX and Linux-Clientsysteme

Beschreibung	**Quelle**	**Ausführbares Modul**
CICS-Programm	amqscic0.ccs	amqscicc
Headerdatei für das CICS-Programm	amqscih0.h	-
Tuxedo-Clientprogramm für das Einreihen von Nachrichten	amqstxpx.c	-
Tuxedo-Clientprogramm für das Abrufen von Nachrichten	amqstxgx.c	-
Tuxedo-Serverprogramm für die beiden Clientprogramme	amqstxsx.c	-
UBBCONFIG-Datei für die Tuxedo-Programme	ubbstxcx.cfg	-
Datei mit den Feldtabellen für die Tuxedo-Programme	amqstxvx.flds	-
Datei mit den Ansichtsdefinitionen für die Tuxedo-Programme	amqstxvx.v	-

Tabelle 137. Beispielprogramme für Windows-Clientsysteme		
Beschreibung	**Quelle**	**Ausführbares Modul**
CICS-Transaktion	amqscic0.ccs	amqscicc
Headerdatei für die CICS-Transaktion	amqscih0.h	-
Tuxedo-Clientprogramm für das Einreihen von Nachrichten	amqstxpx.c	-
Tuxedo-Clientprogramm für das Abrufen von Nachrichten	amqstxgx.c	-
Tuxedo-Serverprogramm für die beiden Clientprogramme	amqstxsx.c	-
UBBCONFIG-Datei für die Tuxedo-Programme	ubbstxcx.cfg	-
Datei mit den Feldtabellen für die Tuxedo-Programme	amqstxvx.fld	-
Datei mit den Ansichtsdefinitionen für die Tuxedo-Programme	amqstxvx.v	-
Makefile für die Tuxedo-Programme	amqstxmc.mak	-
ENVFILE-Datei für die Tuxedo-Programme	amqstxen.env	-

ALW *Fehlernachricht AMQ5203, modifiziert für CICS- und Tuxedo-Anwendungen*

Wenn Sie CICS- oder Tuxedo-Anwendungen mit einem erweiterten transaktionsorientierten Client verwenden, werden, sofern nötig, Standarddiagnosenachrichten angezeigt. Eine dieser Nachrichten wurde für die Verwendung mit einem erweiterten transaktionsorientierten Client modifiziert.

Die Nachrichten, die in die Fehlerprotokolldateien von IBM MQ ausgegeben werden können, sind im Abschnitt Diagnosenachrichten: AMQ4000-9999 dokumentiert. Nachricht AMQ5203 wurde allerdings für die Verwendung mit einem erweiterten transaktionsorientierten Client modifiziert. Die modifizierte Nachricht lautet wie folgt:

AMQ5203: Fehler beim Aufrufen der XA-Schnittstelle

Erklärung

Die Fehlernummer lautet '&2', wobei der Wert 1 anzeigt, dass der bereitgestellte Flagwert von '&1' ungültig war, 2 anzeigt, dass versucht wurde, Threadbibliotheken und Nicht-Threadbibliotheken im gleichen Prozess zu verwenden, 3 anzeigt, dass ein Fehler mit dem bereitgestellten Warteschlangenmanagernamen '&3' aufgetreten ist, 4 anzeigt, dass die Ressourcenmanager-ID von '&1' ungültig war, 5 anzeigt, dass versucht wurde, einen zweiten Warteschlangenmanager mit der Bezeichnung '&3' zu verwenden, als der Warteschlangenmanager bereits verbunden war, 6 anzeigt, dass der Transaktionsmanager aufgerufen wurde, als die Anwendung bereits mit einem Warteschlangenmanager verbunden war, 7 anzeigt, dass der XA-Aufruf erfolgte, während ein anderer Aufruf in Bearbeitung war, 8 anzeigt, dass die xa_info-Zeichenfolge '&4' im xa_open-Aufruf einen ungültigen Parameterwert für den Parameternamen '&5' enthielt, und 9 anzeigt, dass der xa_info-Zeichenfolge '&4' im xa_open-Aufruf ein erforderlicher Parameter mit dem Parameternamen '&5' fehlt.

Benutzerantwort

Beheben Sie den Fehler und wiederholen Sie den Vorgang.

Windows Microsoft Transaction Server-Anwendungen vorbereiten und ausführen

Befolgen Sie die nachstehenden Anweisungen für Ihre Umgebung, um eine MTS-Anwendung für die Ausführung als IBM MQ MQI client-Anwendung vorzubereiten.

Allgemeine Informationen zur Entwicklung von Microsoft Transaction Server (MTS)-Anwendungen, die auf IBM MQ-Ressourcen zugreifen, finden Sie im Kapitel zu MTS im IBM MQ Help Center.

Befolgen Sie die eine der nachstehenden Anweisungen für jede Komponente der Anwendung, um eine MTS-Anwendung für die Ausführung als IBM MQ MQI client-Anwendung vorzubereiten

- Verwendet die Komponente Bindungen der Programmiersprache C für die MQI, müssen Sie entsprechend den Anweisungen im Abschnitt „C-Programme in Windows vorbereiten” auf Seite 1069 vorgehen, die Komponente jedoch mit der Bibliothek 'mqicxa.lib' verbinden, nicht mit 'mqic.lib'.
- Verwendet die Komponente IBM MQ C++-Klassen, befolgen Sie die Anweisungen im Abschnitt „C++-Programme unter Windows erstellen” auf Seite 578 und verbinden Sie die Komponente mit der Bibliothek 'imqx23vn.lib', nicht mit 'imqc23vn.lib'.
- Verwendet die Komponente die Bindungen von Visual Basic für das MQI, müssen Sie entsprechend den Anweisungen im Abschnitt „Visual Basic-Programme in Windows vorbereiten” auf Seite 1073 vorgehen, bei der Definition des Visual Basic-Projekts im Feld **Argumente für bedingte Kompilierung** jedoch `MqType=3` eingeben.

IBM MQ JMS-Anwendungen vorbereiten und ausführen

Sie können IBM MQ JMS-Anwendungen im Clientmodus mit WebSphere Application Server als Transaktionsmanager ausführen. Möglicherweise werden bestimmte Warnhinweise angezeigt.

Zur Vorbereitung und Ausführung von IBM MQ JMS-Anwendungen im Clientmodus mit WebSphere Application Server als Transaktionsmanager folgen Sie den Anweisungen unter „IBM MQ classes for JMS/Jakarta Messaging verwenden” auf Seite 85.

Bei der Ausführung einer IBM MQ JMS-Clientanwendung können folgende Warnhinweise angezeigt werden:

MQJE080
Insufficient license units - run setmqcap (Unzureichende Anzahl an Lizenzeinheiten - setmqcap ausführen)

MQJE081
File containing the license unit information is in the wrong format - run setmqcap (Die Datei mit den Informationen zu den Lizenzeinheiten hat das falsche Format - setmqcap ausführen)

MQJE082
File containing the license unit information could not be found - run setmqcap (Die Datei mit den Informationen zu den Lizenzeinheiten konnte nicht gefunden werden - setmqcap ausführen)

Benutzerexits, API-Exits und installierbare IBM MQ-Services

Dieser Abschnitt enthält Links zu Informationen zur Verwendung und Entwicklung dieser Programme.

Eine allgemeine Beschreibung, wie Warteschlangenmanagerfunktionen durch Benutzerexits, API-Exits und installierbare Services erweitert werden können, finden Sie im Abschnitt Funktionen des Warteschlangenmanagers erweitern.

Informationen zur Entwicklung und Kompilierung der Exits und installierbaren Services finden Sie in den entsprechenden Unterabschnitten.

Zugehörige Konzepte
Kanalexitprogramme für MQI-Kanäle
Zugehörige Verweise
API-Exitreferenz
Referenzinformationen zu installierbaren Services
IBM i Referenzinformationen zu installierbaren Services unter IBM i

ALW Exits und installierbare Services unter AIX, Linux, and Windows schreiben

Sie können Exits schreiben und kompilieren, ohne eine Verbindung zu IBM MQ -Bibliotheken unter AIX, Linux, and Windowsherzustellen.

Informationen zu diesem Vorgang

Dieser Abschnitt gilt nur für Systeme unter AIX, Linux, and Windows. Informationen zum Schreiben von Exits und installierbaren Services für andere Plattformen finden Sie in den relevanten plattformspezifischen Abschnitten.

Wenn IBM MQ nicht in einem Standardverzeichnis installiert ist, müssen Sie Ihre Exits ohne Verbindung zu irgendwelchen IBM MQ-Bibliotheken schreiben und kompilieren.

Auf Systemen unter AIX, Linux, and Windows können Sie Exits ohne Verknüpfung zu IBM MQ-Bibliotheken schreiben und kompilieren:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

Bestehende Exits, die mit diesen Bibliotheken verbunden sind, funktionieren weiterhin, sofern auf AIX and Linux-Systemen IBM MQ im Standardverzeichnis installiert ist.

Vorgehensweise

1. Schließen Sie die Headerdatei 'cmqec.h' ein.

   Durch Einschließen dieser Headerdatei werden die Headerdateien 'cmqc.h', 'cmqxc.h' und 'cmqzc.h' automatisch ebenfalls eingeschlossen.
2. Schreiben Sie den Exit so, dass MQI- und DCI-Aufrufe über die MQIEP-Struktur erfolgen. Weitere Informationen über die MQIEP-Struktur finden Sie unter MQIEP-Struktur.
   - Installierbare Services
     - Verwenden Sie den Parameter **Hconfig**, um auf den MQZEP-Aufruf zu verweisen.
     - Sie müssen überprüfen, ob die ersten vier Byte des Parameters **Hconfig** mit dem Parameter **StrucId** der MQIEP-Struktur übereinstimmen, bevor Sie **Hconfig** verwenden.
     - Weitere Informationen zum Schreiben installierbarer Servicekomponenten finden Sie im Abschnitt MQIEP.
   - API-Exits
     - Verwenden Sie den Parameter **Hconfig**, um auf den MQXEP-Aufruf zu verweisen.
     - Sie müssen überprüfen, ob die ersten vier Byte des Parameters **Hconfig** mit dem Parameter **StrucId** der MQIEP-Struktur übereinstimmen, bevor Sie **Hconfig** verwenden.
     - Weitere Informationen über das Schreiben von API-Exits finden Sie unter „API-Exits schreiben” auf Seite 1006.
   - Kanalexits
     - Verwenden Sie den Parameter **pEntryPoints** der MQCXP-Struktur, um auf MQI- und DCI-Aufrufe zu verweisen.
     - Sie müssen überprüfen, ob die MQCXP-Versionsnummer auf Version 8 oder eine höhere Version verweist, bevor Sie **pEntryPoints** verwenden.
     - Weitere Informationen über das Schreiben von Kanalexits finden Sie unter „Kanalexitprogramme schreiben” auf Seite 1016.
   - Datenkonvertierungsexits
     - Verwenden Sie den Parameter **pEntryPoints** der MQDXP-Struktur, um auf MQI- und DCI-Aufrufe zu verweisen.

- Sie müssen überprüfen, ob die MQDXP-Versionsnummer auf Version 2 oder eine höhere Version verweist, bevor Sie **pEntryPoints** verwenden.
- Sie können unter Verwendung des Befehls **crtmqcvx** und der Quellendatei 'amqsvfc0.c' einen Datenkonvertierungscode erstellen, der den Parameter **pEntryPoints** verwendet. Weitere Informationen finden Sie in den Abschnitten „Datenkonvertierungsexit für IBM MQ for Windows schreiben" auf Seite 1042 und „Datenkonvertierungsexit für IBM MQ for AIX or Linux-Systeme schreiben" auf Seite 1040.
- Wenn Sie über bestehende Datenkonvertierungsexits verfügen, die mit dem Befehl **crtmqcvx** generiert wurden, müssen Sie den Exit mit dem aktualisierten Befehl neu generieren.
- Weitere Informationen über das Schreiben von Datenkonvertierungsexits finden Sie unter „Datenkonvertierungsexits schreiben" auf Seite 1035.

- Pre-Connect-Exits
  - Verwenden Sie den Parameter **pEntryPoints** der MQNXP-Struktur, um auf MQI- und DCI-Aufrufe zu verweisen.
  - Sie müssen überprüfen, ob die MQNXP-Versionsnummer auf Version 2 oder eine höhere Version verweist, bevor Sie **pEntryPoints** verwenden.
  - Weitere Informationen über das Schreiben von Pre-Connect-Exits finden Sie unter „Verbindungsdefinitionen unter Verwendung eines Vorverbindungsexits aus einem Repository referenzieren" auf Seite 1045.
- Veröffentlichungsexits
  - Verwenden Sie den Parameter **pEntryPoints** der MQPSXP-Struktur, um auf MQI- und DCI-Aufrufe zu verweisen.
  - Sie müssen überprüfen, ob die MQPSXP-Versionsnummer auf Version 2 oder eine höhere Version verweist, bevor Sie **pEntryPoints** verwenden.
  - Weitere Informationen über das Schreiben von Veröffentlichungsexits finden Sie unter „Veröffentlichungsexits schreiben und kompilieren" auf Seite 1046.
- Exits für Clusterauslastung
  - Verwenden Sie den Parameter **pEntryPoints** der MQWXP-Struktur, um auf MQXCLWLN-Aufrufe zu verweisen.
  - Sie müssen überprüfen, ob die MQWXP-Versionsnummer auf Version 4 oder eine höhere Version verweist, bevor Sie **pEntryPoints** verwenden.
  - Weitere Informationen über das Schreiben von Exits für Clusterauslastung finden Sie unter „Exits für Clusterauslastung schreiben und kompilieren" auf Seite 1048.

Beispiel für einen Kanalexit, der MQPUT aufruft:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

Weitere Beispiele finden Sie in den „Prozedurale IBM MQ-Beispielprogramme verwenden" auf Seite 1111.

3. Kompilieren Sie den Exit:
   - Stellen Sie keine Verbindung zu den IBM MQ-Bibliotheken her.
   - Nehmen Sie keinen integrierten RPath in IBM MQ-Bibliotheken in Ihrem Exit auf.
   - Weitere Informationen über die Kompilierung des Exits finden Sie in einem der folgenden Abschnitte:

- API-Exits: „API-Exits kompilieren” auf Seite 1007.
- Kanalexits, Veröffentlichungsexits, Exits für Clusterauslastung: „Kanalexitprogramme auf AIX, Linux, and Windows-Systemen kompilieren” auf Seite 1034.
- Datenkonvertierungsexits: „Datenkonvertierungsexits schreiben” auf Seite 1035.

4. Speichern Sie den Exit an einem der folgenden Orte:
   - Pfad Ihrer Wahl, den Sie bei der Konfiguration des Exits vollständig qualifizieren
   - Exit-Standardpfad in einem bestimmten Installationsverzeichnis. Beispiel: *MQ_DATA_PATH*/exits/installation2.
   - Exit-Standardpfad

     Der Standardexitpfad ist *MQ_DATA_PATH*/exits für 32-Bit-Exits und *MQ_DATA_PATH*/exits64 für 64-Bit-Exits. Sie können diese Pfade in der Datei 'qm.ini' oder 'mqclient.ini' ändern. Weitere Informationen finden Sie im Abschnitt Exit-Pfad. UnterWindows und Linux können Sie den Pfad in IBM MQ Explorer ändern:

     a. Klicken Sie mit der rechten Maustaste auf den Namen des Warteschlangenmanagers.
     b. Klicken Sie auf **Eigenschaften...**
     c. Klicken Sie auf **Exits**.
     d. Geben Sie im Feld für den Exit-Standardpfad das Verzeichnis an, das das Exitprogramm enthält.

   Wenn sich ein Exit sowohl im speziellen Installationsverzeichnis als auch im Standardpfadverzeichnis befindet, wird für die im Pfad genannte IBM MQ-Installation der Exit aus dem speziellen Installationsverzeichnis verwendet. Der Exit wird beispielsweise in /exits/installation2 und /exits, jedoch nicht in /exits/installation1 gespeichert. Die IBM MQ-Installation installation2 verwendet in diesem Fall den Exit aus /exits/installation2. Die IBM MQ-Installation installation1 verwendet hingegen den Exit aus dem Verzeichnis /exits.
5. Konfigurieren Sie den Exit, falls erforderlich:
   - Installierbare Services: „Services und Komponenten konfigurieren” auf Seite 998
   - API-Exits: „API-Exits konfigurieren” auf Seite 1010.
   - Kanalexits: „Kanalexits konfigurieren” auf Seite 1035
   - Veröffentlichungsexits: „Veröffentlichungsexits konfigurieren” auf Seite 1048
   - Pre-Connect-Exits: Pre-Connect-Zeilengruppe der Clientkonfigurationsdatei.

ALW *Nicht mit einer MQI-Bibliothek verlinkte API-Exits*

Unter bestimmten Umständen sollten Sie Ihren vorhandenen API-Exit, dessen Programmcode nicht für die Verwendung der MQIEP-Funktionszeiger geändert werden kann, mit einer IBM MQ-API-Bibliothek verlinken.

Dies ist nötig, damit Ihr vorhandener API-Exit vom Laufzeit-Linker Ihres Systems erfolgreich in Programme geladen werden kann, die noch nicht über die geladenen Funktionszeiger verfügen.

Anmerkung: Diese Informationen sind auf solche vorhandenen API-Exits begrenzt, die MQI-Aufrufe direkt ausführen. Das sind die Exits, die MQIEP nicht verwenden. Wo dies möglich ist, sollten Sie eine Änderung des Programmcodes des Exits planen, damit stattdessen die MQIEP-Eingangspunkte verwendet werden.

runmqsc ist ein Beispiel für ein Programm, das nicht direkt mit einer MQI-Bibliothek verknüpft ist.

Deshalb kann ein API-Exit, der nicht mit der erforderlichen IBM MQ-API-Bibliothek verlinkt oder dessen Programmcode für die Verwendung von MQIEP geändert wurde, nicht in **runmqsc** geladen werden.

Im Fehlerprotokoll des Warteschlangenmanagers werden Fehler, z. B. AMQ6175: Das System konnte die gemeinsam genutzte Bibliothek nicht dynamisch laden (zusammen mit qualifizierendem Text wie undefined symbol: MQCONN)

und AMQ7214: `Das Modul für API-Exit 'myexitname' konnte nicht geladen werden` angezeigt.

Zugehörige Tasks

„Exits und installierbare Services unter AIX, Linux, and Windows schreiben” auf Seite 986
Sie können Exits schreiben und kompilieren, ohne eine Verbindung zu IBM MQ -Bibliotheken unter AIX, Linux, and Windowsherzustellen.

ALW Installierbare Services und Komponenten für AIX, Linux, and Windows

In diesem Abschnitt erhalten Sie eine Einführung in die installierbaren Services und die zugehörigen Funktionen und Komponenten. Die Schnittstelle zu diesen Funktionen ist dokumentiert, sodass Sie oder Softwareanbieter Komponenten bereitstellen können.

Installierbare IBM MQ-Services werden in erster Linie aus den folgenden Gründen bereitgestellt:

- Um Ihnen die Flexibilität zu bieten, selbst auszuwählen, ob Sie die von den IBM MQ-Produkten bereitgestellten Komponenten verwenden oder diese durch andere Komponenten ersetzen oder erweitern.
- Um Anbietern die Möglichkeit zu bieten, mit ihren eigenen Komponenten neue Technologien bereitzustellen, ohne die IBM MQ-Produkte intern ändern zu müssen.
- Um auch IBM MQ die Möglichkeit zu geben, neue Technologien schneller und kostengünstiger zu nutzen und so Produkte früher und günstiger bereitzustellen.

Installierbare Services und *Servicekomponenten* sind Bestandteil der Produktstruktur von IBM MQ. Im Zentrum dieser Struktur steht der Warteschlangenmanager, der die Funktionen und die Regeln implementiert, die mit der Message Queue Interface (MQI) in Zusammenhang stehen. Für diesen zentralen Bestandteil sind eine Reihe von Servicefunktionen, auch als *installierbare Services* bezeichnet, erforderlich. Es gibt folgende installierbare Services:

- Berechtigungsservice
- Namensservice

Jeder installierbare Service setzt sich aus einer Reihe zusammengehöriger Funktionen zusammen und wird unter Verwendung einer oder mehrerer *Servicekomponenten* installiert. Bei jeder dieser Komponenten handelt es sich um eine Schnittstelle mit einer festen Architektur, die öffentlich verfügbar ist. So können auch unabhängige Softwareanbieter und andere Drittanbieter als Ersatz oder Erweiterung für die von IBM MQ bereitgestellten Produkte installierbare Komponenten bereitstellen. In Tabelle 138 auf Seite 990 sehen Sie eine Zusammenfassung der installierbaren Services und ihrer Komponenten.

Tabelle 138. Zusammenfassung der installierbaren Services und ihrer Komponenten

Installierbarer Service	Bereitgestellte Komponente	Funktion	Voraussetzungen
Berechtigungsservice	Objektberechtigungsmanager (OAM)	Führt Berechtigungsprüfungen für Befehle und MQI-Aufrufe aus. Benutzer können eigene Komponenten erstellen, die den OAM ergänzen oder ersetzen. Beispiel: Überprüfung, ob eine Benutzer-ID zum Öffnen einer Warteschlange berechtigt ist.	(Geeignete Plattformberechtigungsfunktionen werden vorausgesetzt)

Tabelle 138. Zusammenfassung der installierbaren Services und ihrer Komponenten (Forts.)			
Installierbarer Service	**Bereitgestellte Komponente**	**Funktion**	**Voraussetzungen**
Namensservice	--	Der Namensservice ist ein installierbarer Service, mit dem ein Warteschlangenmanager den Namen eines anderen Warteschlangenmanagers ermitteln kann, dem eine bestimmte Warteschlange zugeordnet ist. • Benutzerdefiniert	• Ein eigener Namensmanager oder der Namensmanager eines Drittanbieters

Eine Beschreibung der Schnittstelle für installierbare Services finden Sie im Abschnitt Referenzinformationen zur Schnittstelle für installierbare Services.

Zugehörige Tasks
Installierbare Services konfigurieren

Servicekomponente schreiben
In diesem Abschnitt wird die Beziehung zwischen Services, Komponenten, Eingangspunkten und Rückkehrcodes beschrieben.

Funktionen und Komponenten

Jeder Service besteht aus einer Reihe zusammengehöriger Funktionen. Der Namensservice enthält beispielsweise Funktionen für folgende Vorgänge:

- Suchen nach dem Namen einer Warteschlange, wobei der Name des Warteschlangenmanagers zurückgegeben wird, in dem die Warteschlange definiert ist
- Einfügen eines Warteschlangennamens in das Serviceverzeichnis
- Löschen eines Warteschlangennamens aus dem Serviceverzeichnis

Darüber hinaus sind Initialisierungs- und Beendigungsfunktionen enthalten.

Ein installierbarer Service wird von einer oder mehreren Servicekomponenten bereitgestellt. Jede Komponente kann einige oder alle der für den betreffenden Service definierten Funktionen ausführen. In IBM MQ for AIX führt die bereitgestellte Berechtigungsservicekomponente, der OAM, beispielsweise alle verfügbaren Funktionen aus. Weitere Informationen hierzu finden Sie unter „Schnittstelle für Berechtigungsservice” auf Seite 995. Die Komponente ist auch für die Verwaltung der zugrunde liegenden, für die Implementierung des Service erforderlichen Ressourcen bzw. Software zuständig (z. B. ein LDAP-Verzeichnis). Konfigurationsdateien stellen eine bewährte Methode zum Laden der Komponente und Bestimmen der Adressen der bereitgestellten funktionalen Routinen dar.

Abbildung 99 auf Seite 992 veranschaulicht die Beziehung zwischen Services und Komponenten:

- Ein Service wird für einen Warteschlangenmanager über die Zeilengruppen einer Konfigurationsdatei definiert.
- Jeder einzelne Service wird durch bereitgestellten Code im Warteschlangenmanager unterstützt. Dieser Code kann von den Benutzern nicht geändert werden, d. h. sie können keine eigenen Services erstellen.
- Jeder Service wird durch mindestens eine Komponente implementiert, die entweder im Rahmen des Produkts bereitgestellt oder benutzerdefiniert sein kann. Für einen Service können mehrere Komponenten aufgerufen werden, von denen jede unterschiedliche Funktionen innerhalb des Service übernimmt.
- Eingangspunkte verbinden die Servicekomponenten mit dem entsprechenden Code im Warteschlangenmanager.

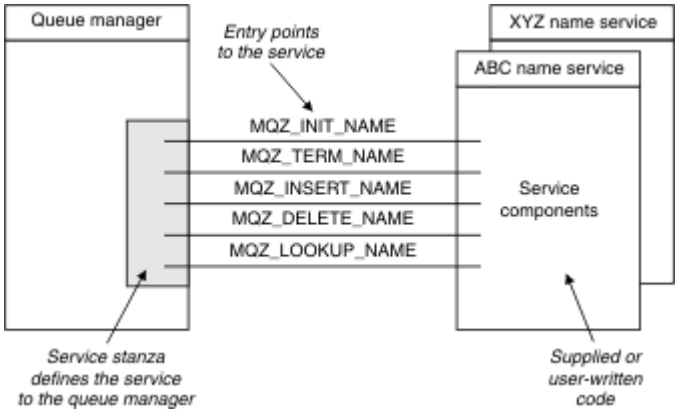


Abbildung 99. Zusammenhang zwischen Services, Komponenten und Eingangspunkten

Eingangspunkte

Jede Servicekomponente ist dargestellt durch eine Liste der Eingangspunktadressen der Routinen, die einen bestimmten installierbaren Service unterstützen. Der installierbare Service definiert die Funktionen, die von den einzelnen Routinen ausgeführt werden.

Die Reihenfolge der Servicekomponenten bei der Konfiguration bestimmt auch, in welcher Reihenfolge die Eingangspunkte aufgerufen werden, um eine Serviceanforderung zu erfüllen.

In der bereitgestellten Headerdatei `cmqzc.h` sind die Eingangspunkte für die einzelnen Services mit dem Präfix 'MQZID_' versehen.

Vorhandene Services werden in einer vordefinierten Reihenfolge geladen. In der folgenden Liste sind die Services aufgeführt, dabei ist auch angegeben, in welcher Reihenfolge sie initialisiert werden.

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

Der `AuthorizationService` (Berechtigungsservice) ist der einzige standardmäßig konfigurierte Service. Falls Sie den Namensservice (`NameService`) und den Benutzer-ID-Service (`UserIdentifierService`) verwenden möchten, müssen Sie sie manuell konfigurieren.

Für Services und Servicekomponenten gilt eine Eins-zu-eins- oder eine Eins-zu-viele-Zuordnung. Für jeden Service können mehrere Servicekomponenten definiert sein. Auf AIX and Linux-Systemen muss der Wert von 'Service' in der Zeilengruppe `ServiceComponent` mit dem Wert von 'Name' der Zeilengruppe 'Service' in der Datei 'qm.ini' übereinstimmen. Unter Windowsmuss der Wert des Service-Registry-Schlüssels von `ServiceComponent` mit dem Wert des Namens-Registry-Schlüssels übereinstimmen und wie folgt definiert sein: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`MQ_INSTALLATION_NAME`*`\Configuration\QueueManager\`*`qmname`*`\` . Dabei ist *qmname* der Name des Warteschlangenmanagers.

Bei AIX and Linux-Systemen werden Servicekomponenten in der Reihenfolge ihrer Definition in der Datei 'qm.ini' gestartet. Weil unter Windows die Windows-Registrierung verwendet wird, gibt IBM MQ hier den Aufruf **RegEnumKey** aus, der die Werte in alphabetischer Reihenfolge zurückgibt. Daher werden die Services unter Windows in alphabetischer Reihenfolge aufgerufen, so wie sie in der Registrierung definiert sind.

Die Anordnung der `ServiceComponent`-Definitionen ist wichtig. Diese Anordnung legt fest, in welcher Reihenfolge Komponenten für einen bestimmten Service ausgeführt werden. So ist beispielsweise der `AuthorizationService` unter Windows mit der OAM-Standardkomponente `MQSeries.WindowsNT.auth.service` konfiguriert. Um den Standardberechtigungsmanager zu überschreiben, können weitere Komponenten für diesen Service definiert werden. Sofern nicht `MQCACF_SERVICE_COMPONENT` angegeben ist, wird zur Verarbeitung der Anforderung die erste Komponente in der alphabetischen Reihenfolge und auch deren Name verwendet.

Rückkehrcodes

Servicekomponenten senden Rückgabecodes an den Warteschlangenmanager, um diesem verschiedene Bedingungen zu melden. Sie dokumentieren den Erfolg oder das Fehlschlagen der Operation und geben an, ob der Warteschlangenmanager mit der nächsten Servicekomponente fortfahren soll. Diese Angabe erfolgt über einen separaten Fortsetzungsparameter (*Continuation*).

Komponentendaten

Möglicherweise müssen die Daten von den verschiedenen Funktionen einer Servicekomponente gemeinsam genutzt werden. Installierbare Services bieten einen optionalen Datenbereich, der bei jedem Aufruf einer Servicekomponente übergeben werden kann. Dieser Datenbereich ist ausschließlich für die Nutzung durch die Servicekomponente vorgesehen. Er wird von allen Aufrufen einer angegebenen Funktion gemeinsam genutzt, selbst wenn diese aus unterschiedlichen Adressräumen oder Prozessen erfolgen. Dieser Datenbereich ist für die Servicekomponente jederzeit zugänglich, wann immer sie aufgerufen wird. Die Größe dieses Datenbereichs wird in der Zeilengruppe *ServiceComponent* festgelegt.

Komponenten initialisieren und beenden
Optionen für die Initialisierung und Beendigung von Komponenten.

Wenn die Routine der Komponenteninitialisierung aufgerufen wird, muss sie die Funktion **MQZEP** des Warteschlangenmanagers für jeden Eingangspunkt aufrufen, der von der Komponente unterstützt wird. **MQZEP** definiert einen Eingangspunkt für den Service. Für alle nicht definierten Exitpunkte wird der Wert NULL angenommen.

Eine Komponente wird auf jeden Fall einmal mit der primären Initialisierungsoption aufgerufen, bevor ihr Aufruf auf eine andere Weise erfolgen kann.

Auf bestimmten Plattformen kann eine Komponente mit der sekundären Initialisierungsoption aufgerufen werden. Sie kann beispielsweise einmal für alle Betriebssystemprozesse, Threads oder Tasks aufgerufen werden, über die auf den Service zugegriffen wird.

Bei Verwendung der sekundären Initialisierung:

- Die Komponente kann für eine sekundäre Initialisierung mehrmals aufgerufen werden. Für jeden derartigen Aufruf wird ein entsprechender sekundärer Beendigungsaufruf ausgegeben, wenn der Service nicht mehr benötigt wird.

  Für Namensservices ist dies der Aufruf MQZ_TERM_NAME.

  Für Berechtigungsservices ist dies der Aufruf MQZ_TERM_AUTHORITY.
- Bei jedem primären und sekundären Initialisierungsaufruf für eine Komponente müssen die Eingangspunkte durch Aufruf von MQZEP erneut angegeben werden.
- Für die Komponente wird nur eine Kopie der Komponentendaten verwendet; es gibt also nicht für jede sekundäre Initialisierung eine eigene Kopie.
- Erst nach Ausführung der sekundären Initialisierung kann die Komponente vom Betriebssystemprozess, Thread oder einer Task für andere Aufrufe des Service aufgerufen werden.
- Der Parameter **Version** der Komponente muss für die primäre und die sekundäre Initialisierung auf den gleichen Wert gesetzt sein.

Wenn die Komponente nicht mehr benötigt wird, wird sie auf jeden Fall einmal mit der primären Beendigungsoption aufgerufen. Danach erfolgen keine weiteren Aufrufe mehr für diese Komponente.

Die Komponente wird mit der sekundären Beendigungsoption aufgerufen, wenn sie zuvor für die sekundäre Initialisierung aufgerufen wurde.

Objektberechtigungsmanager (OAM)
Die mit IBM MQ-Produkten bereitgestellte Berechtigungsservicekomponente ist der Objektberechtigungsmanager (Object Authority Manager; OAM).

Der Objektberechtigungsmanager ist standardmäßig aktiv und arbeitet mit den Steuerbefehlen **dspmqaut** (Bildschirmberechtigung), **dmpmqaut** (Speicherauszugsberechtigung) und **setmqaut** (Einstellungs- und Zurücksetzungsberechtigung).

Die Syntax dieser Befehle und ihre Verwendung werden in IBM MQ for Multiplatforms mit Steuerbefehlen verwaltenbeschrieben.

Der Objektberechtigungsmanager arbeitet mit der *Entität* eines Principals oder einer Gruppe:

- Linux AIX Unter AIX and Linux ist ein Principal eine Benutzer-ID oder eine mit einem Anwendungsprogramm verknüpfte ID, das für einen Benutzer ausgeführt wird; eine Gruppe ist eine vom System definierte Principalgruppe.
- Windows Unter Windows ist ein Principal eine Windows-Benutzer-ID oder eine mit einem Anwendungsprogramm verknüpfte ID, das für einen Benutzer ausgeführt wird; eine Gruppe ist eine Windows-Gruppe.

Berechtigungen können auf Principal- oder Gruppenebene erteilt oder entzogen werden.

Bei Ausgabe einer MQI-Anforderung oder Aufruf eines Befehls prüft der Objektberechtigungsmanager, ob die mit der angeforderten Operation verbundene Entität über die Berechtigung zur Durchführung dieser Operation und zum Zugriff auf die angegebenen Warteschlangenmanagerressourcen verfügt.

Mit dem Berechtigungsservice können Sie die Berechtigungsprüfungen für Warteschlangenmanager erweitern oder ersetzen, indem Sie eine eigene Berechtigungsservicekomponente erstellen.

Namensservice
Der Namensservice ist ein installierbarer Service, der den Warteschlangenmanager beim Ermitteln des Namens eines Warteschlangenmanagers unterstützt, dem eine bestimmte Warteschlange zugeordnet ist. Von einem Namensservice können keine anderen Warteschlangenattribute abgerufen werden.

Mit Hilfe des Namensservices kann eine Anwendung ferne Warteschlangen genau so für die Ausgabe öffnen wie lokale Warteschlangen. Ein Namensservice kann für keine anderen Objekte als Warteschlangen aufgerufen werden.

Anmerkung: Für die fernen Warteschlangen muss das Attribut **Scope** auf CELL gesetzt sein.

Beim Versuch, eine Warteschlange zu öffnen, sucht die Anwendung den Namen der Warteschlange zunächst im Verzeichnis des Warteschlangenmanagers. Findet sie ihn dort nicht, fragt sie nacheinander bei den konfigurierten Namensservices nach dem Warteschlangennamen an, bis einer den Namen erkennt. Wird der Name nicht erkannt, schlägt das Öffnen fehl.

Der Namensservice gibt für die Warteschlange den Namen des Warteschlangenmanagers zurück, dem die Warteschlange zugeordnet ist. Anschließend setzt der Warteschlangenmanager die Verarbeitung der MQOPEN-Anforderung fort, als seien die Namen der Warteschlange und des Warteschlangenmanagers bereits in der ursprünglichen Anforderung angegeben worden.

Die Namensserviceschnittstelle (NSI, Name Service Interface) ist Teil des IBM MQ-Frameworks.

So funktioniert der Namensservice

Wenn in MQSC in einer Warteschlangendefinition für das Attribut **Scope** der Warteschlangenmanager (SCOPE(QMGR)) angegeben ist, wird die Warteschlangendefinition (mit allen Warteschlangenattributen) nur im Verzeichnis des Warteschlangenmanagers gespeichert. Dies kann nicht durch einen installierbaren Service ersetzt werden.

Wenn in MQSC in einer Warteschlangendefinition für das Attribut **Scope** die Zelle (SCOPE(CELL)) angegeben ist, wird die Warteschlangendefinition ebenso mit allen Warteschlangenattributen im Verzeichnis des Warteschlangenmanagers gespeichert. Die Namen der Warteschlange und des Warteschlangenmanagers werden darüber hinaus jedoch auch in einem Namensservice gespeichert. Wenn kein Service verfügbar ist, der diese Informationen speichern kann, ist es nicht möglich, eine Warteschlange mit dem Wert 'Cell' (Zelle) für das Attribut *Scope* zu definieren.

Das Verzeichnis mit diesen Informationen kann vom Service selbst verwaltet werden, oder der Service kann hierzu einen untergeordneten Service verwenden, beispielsweise ein LDAP-Verzeichnis, der diesen Zweck erfüllt. In beiden Fällen müssen die im Verzeichnis gespeicherte Definitionen persistent bleiben, selbst wenn die Komponente und der Warteschlangenmanager beendet wurden, und zwar so lange, bis diese explizit gelöscht werden.

Anmerkung:

1. Wenn Sie eine Nachricht an die lokale Warteschlangendefinition (mit 'Scope' gleich CELL) eines fernen Hosts auf einem anderen Warteschlangenmanager innerhalb einer Namensverzeichniszelle senden möchten, müssen Sie einen Kanal definieren.
2. Sie können keine Nachrichten direkt aus der fernen Warteschlange abrufen, selbst wenn sie den Wert CELL für das Attribut 'Scope' hat.
3. Beim Senden von Nachrichten an eine Warteschlange mit dem Wert CELL für das Attribut 'Scope' ist keine Definition der fernen Warteschlange erforderlich.
4. Der Namensservice definiert die Zielwarteschlange zentral. Nach wie vor benötigen Sie allerdings eine Übertragungswarteschlange zum Zielwarteschlangenmanager und ein Paar Kanaldefinitionen. Außerdem muss die Übertragungswarteschlange im lokalen System denselben Namen haben wie der Warteschlangenmanager des fernen Systems, dem die Zielwarteschlange (mit dem Wert CELL für das Attribut 'Scope') zugeordnet ist.

   Wenn beispielsweise der ferne Warteschlangenmanager den Namen 'QM01' hat, muss die Übertragungswarteschlange des lokalen Systems auch den Namen 'QM01' haben.

Schnittstelle für Berechtigungsservice
Der Berechtigungsservice stellt Eingangspunkte für die Verwendung durch den Warteschlangenmanager bereit.

Es handelt sich um folgende Eingangspunkte:

MQZ_AUTHENTICATE_USER
Authentifiziert eine Benutzer-ID und ein Kennwort und kann Identitätskontextfelder festlegen.

MQZ_CHECK_AUTHORITY
Prüft, ob eine Entität die Berechtigung besitzt, eine oder mehrere Operationen für ein angegebenes Objekt auszuführen.

MQZ_CHECK_PRIVILEGED
Prüft, ob ein angegebener Benutzer ein privilegierter Benutzer ist.

MQZ_COPY_ALL_AUTHORITY
Kopiert alle aktuellen Berechtigungen, die für ein Referenzobjekt vorhanden sind, in ein anderes Objekt.

MQZ_DELETE_AUTHORITY
Löscht alle Berechtigungen, die einem angegebenen Objekt zugeordnet sind.

MQZ_ENUMERATE_AUTHORITY_DATA
Ruft alle Berechtigungsdaten ab, die den angegebenen Auswahlkriterien entsprechen.

MQZ_FREE_USER
Gibt zugeordnete Ressourcen frei.

MQZ_GET_AUTHORITY
Ruft die Berechtigung ab, über die eine Entität zum Zugriff auf ein bestimmtes Objekt verfügt.

MQZ_GET_EXPLICIT_AUTHORITY
Ruft entweder die Berechtigung ab, die eine benannte Gruppe für den Zugriff auf ein angegebenes Objekt besitzt (jedoch ohne die Zusatzberechtigung der Gruppe **nobody**), oder die Berechtigung, die die Primärgruppe des benannten Principals für den Zugriff auf ein angegebenes Objekt besitzt.

MQZ_INIT_AUTHORITY
Initialisiert die Berechtigungsservicekomponente.

MQZ_INQUIRE
Fragt die unterstützte Funktionalität des Berechtigungsservice ab.

MQZ_REFRESH_CACHE
Aktualisiert alle Berechtigungen.

MQZ_SET_AUTHORITY
Legt die Berechtigung fest, die eine Entität für ein angegebenes Objekt besitzt.

MQZ_TERM_AUTHORITY
Beendet die Berechtigungsservicekomponente.

Unter IBM MQ for Windows stellt der Berechtigungsservice zusätzlich die folgenden Eingangspunkte für den Warteschlangenmanager bereit:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

Diese Eingangspunkte unterstützen die Verwendung der Windows Sicherheits-ID (NT SID).

Diese Namen sind in der Headerdatei `cmqzc.h`, die dazu verwendet werden kann, die Komponentenfunktionen mithilfe eines Prototyps zu testen, als **typedef**s definiert.

Die Initialisierungsfunktion (**MQZ_INIT_AUTHORITY**) muss der Haupteingangspunkt für die Komponente sein. Die übrigen Funktionen werden über die Eingangspunktadresse aufgerufen, die von der Initialisierungsfunktion in den Eingangspunktvektor der Komponente eingefügt wurde.

Schnittstelle für Namensservice
Ein Namensservice stellt Eingangspunkte für die Verwendung durch den Warteschlangenmanager bereit.

Die folgenden Eingangspunkte werden bereitgestellt:

MQZ_INIT_NAME
Initialisiert die Namensservicekomponente.

MQZ_TERM_NAME
Beendet die Namensservicekomponente.

MQZ_LOOKUP_NAME
Sucht nach dem Warteschlangenmanagernamen für die angegebenen Warteschlange.

MQZ_INSERT_NAME
Fügt einen Eintrag mit dem Namen des Warteschlangenmanagers, dem die angegebene Warteschlange zugeordnet ist, in das vom Service verwendete Verzeichnis ein.

MQZ_DELETE_NAME
Löscht den Eintrag für die angegebene Warteschlange aus dem vom Service verwendeten Verzeichnis.

Wenn mehr als ein Namensservice konfiguriert ist, wird wie folgt vorgegangen:

- Beim Ermitteln eines Namens wird die Funktion MQZ_LOOKUP_NAME für jeden Service in der Liste aufgerufen, bis der Warteschlangenname aufgelöst wurde (sofern keine der Komponenten anfordert, die Suche zu stoppen).
- Beim Einfügen wird die Funktion MQZ_INSERT_NAME für den ersten Service in der Liste aufgerufen, der diese Funktion unterstützt.
- Beim Löschen wird die Funktion MQZ_DELETE_NAME für den ersten Service in der Liste aufgerufen, der diese Funktion unterstützt.

Es ist sinnvoll, für die Funktionen zum Einfügen und Löschen zusammen nur eine Komponente zu verwenden. Aber auch eine Komponente, die nur eine Suchfunktion unterstützt, ist möglich, beispielsweise als letzte Komponente in der Liste, durch die alle Namen aufgelöst werden sollen, die keiner anderen Namensservicekomponente eines Warteschlangenmanagers bekannt sind, auf denen diese Namen definiert sein können.

In der Programmiersprache C werden die Namen als Funktionsdatentypen mit der Anweisung 'typedef' definiert. Diese können zum Erstellen von Prototypen für die Servicefunktionen verwendet werden, um sicherzustellen, dass die Parameter korrekt sind.

Das gesamte für installierbare Services spezifische Material für die Programmiersprache C befindet sich in der Headerdatei `cmqzc.h`.

Abgesehen von der Initialisierungsfunktion (MQZ_INIT_NAME), die der Haupteingangspunkt der Komponente sein muss, werden die Funktionen mit dem Aufruf MQZEP über die von der Initialisierungsfunktion hinzugefügte Eingangspunktadresse aufgerufen.

Mehrere Servicekomponenten verwenden
Für einen Service können Sie auch mehrere Komponenten installieren. In diesem Fall implementieren die einzelnen Komponenten nur jeweils einen Teil des Service und sind darauf angewiesen, dass die anderen Funktionen von anderen Komponenten bereitgestellt werden.

Beispiel für die Verwendung mehrerer Servicekomponenten

Sie erstellen die beiden Namensservicekomponenten `ABC_name_serv` und `XYZ_name_serv`.

`ABC_name_serv`
Diese Komponente unterstützt das Einfügen bzw. Löschen eines Namens im Serviceverzeichnis, jedoch nicht die Suche nach einem Warteschlangennamen.

`XYZ_name_serv`
Diese Komponente unterstützt die Suche nach einem Warteschlangennamen, jedoch nicht das Einfügen bzw. Löschen eines Namens im Serviceverzeichnis.

Komponente `ABC_name_serv` führt eine Datenbank mit Warteschlangennamen und verwendet zwei einfache Algorithmen zum Einfügen bzw. Löschen eines Namens im Serviceverzeichnis.

Komponente `XYZ_name_serv` verwendet einen einfachen Algorithmus, der für jeden Warteschlangennamen, mit dem die Komponente aufgerufen wurde, einen festen Warteschlangenmanagernamen zurückgibt. Sie führt keine Datenbank mit Warteschlangennamen und unterstützt daher auch kein Einfügen und Löschen von Namen.

Die Komponenten werden auf dem gleichen Warteschlangenmanager installiert. Die Zeilengruppen für *`ServiceComponent`* sind so angeordnet, dass die Komponente `ABC_name_serv` zuerst aufgerufen wird. Alle Aufrufe zum Einfügen oder Löschen einer Warteschlange in einem Komponentenverzeichnis werden von der Komponente `ABC_name_serv` verarbeitet; diese Funktionen werden nur von dieser Komponente implementiert. Ein Suchaufruf, den die Komponente `ABC_name_serv` nicht lösen kann, wird jedoch an die Suchkomponente `XYZ_name_serv` weitergeleitet. Diese Komponente löst den Namen eines Warteschlangenmanagers mithilfe ihres einfachen Algorithmus auf.

Ausschluss von Eingangspunkten bei Verwendung mehrerer Komponenten

Wenn Sie einen Service in Form mehrerer Komponenten implementieren, können Sie eine Servicekomponente erstellen, die bestimmte Funktionen nicht bereitstellt. Das Framework der installierbaren Services legt Ihnen hinsichtlich des Ausschlusses von Funktionen keine Einschränkungen auf. Für bestimmte installierbare Services allerdings kann sich das Weglassen einer oder mehrerer Funktionen als falsch erweisen, wenn dadurch der Service, der eigentlich benötigt wird, nicht erbracht werden kann.

Beispiel für Eingangspunkte bei Verwendung mehrerer Komponenten

In Tabelle 139 auf Seite 998 ist ein Beispiel für den installierbaren Namensservice aufgeführt, für den zwei Komponenten installiert wurden. Jede Komponente unterstützt eine bestimmte Reihe von Funktionen für diesen installierbaren Service. Für die Einfügefunktion wird zuerst der Eingangspunkt für die Komponente ABC aufgerufen. Für Eingangspunkte, die nicht mit **MQZEP** für den Service definiert wurden, wird standardmäßig NULL verwendet. In der Tabelle wird zwar ein Eingangspunkt für die Initialisierung aufgeführt, er ist jedoch nicht erforderlich, da die Initialisierung vom Haupteingangspunkt der Komponente durchgeführt wird.

Wenn der Warteschlangenmanager einen installierbaren Service benötigt, verwendet er die für diesen Service definierten Eingangspunkte (die Spalten in Tabelle 139 auf Seite 998). Der Warteschlangenmanager ermittelt abwechselnd für jede Komponente die Adresse der Routine, mit der die gewünschte Funktion implementiert wird. Anschließend ruft er die Routine auf, sofern sie vorhanden ist. War der

Vorgang erfolgreich, werden alle eventuell vorhandenen Ergebnisse und Statusinformationen vom Warteschlangenmanager verwendet.

Tabelle 139. Beispiel für Eingangspunkte für einen installierbaren Service

Funktionsnummer	**Namensservicekomponente ABC**	**Namensservicekomponente XYZ**
MQZID_INIT_NAME (Initialisieren)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (Beenden)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (Einfügen)	ABC_Insert()	NULL
MQZID_DELETE_NAME (Löschen)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME (Suchen)	NULL	XYZ_Lookup()

Ist eine Routine nicht vorhanden, wiederholt der Warteschlangenmanager diesen Prozess für die nächsten Komponente in der Liste. Wenn die Routine vorhanden ist, aber ein Code zurückgegeben wird, der darauf hindeutet, dass sie die Operation nicht ausführen konnte, wird der Vorgang mit der nächsten verfügbaren Komponente wiederholt. Routinen in Servicekomponenten können unter Umständen einen Code zurückgeben, der angibt, dass kein weiterer Versuch unternommen werden soll, die Operation auszuführen.

Services und Komponenten konfigurieren

Servicekomponenten werden über die Konfigurationsdateien des Warteschlangenmanagers konfiguriert; ausgenommen hiervon sind Windows-Systeme, auf denen jeder Warteschlangenmanager über eine eigene Zeilengruppe in der Registry verfügt.

Vorgehensweise

1. Fügen Sie der Konfigurationsdatei des Warteschlangenmanagers `qm.ini`Zeilengruppen hinzu, um den Service für den Warteschlangenmanager zu definieren, und geben Sie die Position des Moduls an:
   - Diese Datei muss für jeden verwendeten Service die Zeilengruppe `Service` enthalten, die den Service für den Warteschlangenmanager definiert. Weitere Informationen finden Sie unter Zeilengruppe 'Service' der Datei qm.ini.
   - Für jede Komponente innerhalb eines Service muss die Zeilengruppe `ServiceComponent` vorhanden sein. Diese Zeilengruppe gibt den Namen und den Pfad des Moduls an, das den Code für diese Komponente enthält. Weitere Informationen finden Sie unter ZeilengruppeServiceComponent in der Datei qm.ini.

   Die Berechtigungsservicekomponente, der Objektberechtigungsmanager oder OAM, wird im Rahmen des Produkts bereitgestellt. Bei der Erstellung eines Warteschlangenmanagers wird die Konfigurationsdatei für den Warteschlangenmanager (bzw. unter Windows die Registry) automatisch mit den entsprechenden Zeilengruppen für den Berechtigungsservice und dessen Standardkomponente (Objektberechtigungsmanager) aktualisiert. Für andere Komponenten muss die Konfigurationsdatei des Warteschlangenmanagers manuell konfiguriert werden.

   Der Code für die einzelnen Servicekomponenten wird beim Start des Warteschlangenmanagers in den Warteschlangenmanager geladen. Dabei wird dynamisches Binden eingesetzt, sofern dies auf der Plattform unterstützt wird.
2. Stoppen Sie den Warteschlangenmanager und starten Sie ihn anschließend erneut, um die Komponente zu aktivieren.

Zugehörige Verweise

Zeilengruppe 'Service' in der Datei 'qm.ini'

Zeilengruppe ServiceComponent in der Datei qm.ini

OAM nach Änderung einer Benutzerberechtigung aktualisieren
In IBM MQ können Sie die Berechtigungsgruppeninformationen des OAM sofort nach der Änderung der Zugehörigkeit eines Benutzers zu einer Berechtigungsgruppe aktualisieren und so Änderungen auf Betriebssystemebene übernehmen, ohne den Warteschlangenmanager stoppen und neu starten zu müssen. Dazu geben Sie den Befehl **REFRESH SECURITY** aus.

Anmerkung: Mit dem Befehl `setmqaut` geänderte Berechtigungen werden vom OAM sofort implementiert.

Warteschlangenmanager speichern Berechtigungsdaten in der lokalen Warteschlange SYSTEM.AUTH.DATA.QUEUE. Diese Daten werden von **amqzfuma.exe** verwaltet.

Zugehörige Verweise
REFRESH SECURITY

IBM i Installierbare Services und Komponenten unter IBM i

Dieser Abschnitt enthält Informationen zu den installierbaren Services und den zugehörigen Funktionen und Komponenten. Die Schnittstelle zu diesen Funktionen ist dokumentiert, sodass Sie oder Softwareanbieter Komponenten bereitstellen können.

Installierbare IBM MQ-Services werden in erster Linie aus den folgenden Gründen bereitgestellt:

- Um Ihnen die Flexibilität zu bieten, selbst auszuwählen, ob Sie die von IBM MQ for IBM i bereitgestellten Komponenten verwenden oder diese durch andere Komponenten ersetzen oder erweitern.
- Um Anbietern die Möglichkeit zu bieten, mit ihren eigenen Komponenten neue Technologien bereitzustellen, ohne interne Änderungen an IBM MQ for IBM i vornehmen zu müssen.
- Um auch IBM MQ die Möglichkeit zu geben, neue Technologien schneller und kostengünstiger zu nutzen und so Produkte früher und günstiger bereitzustellen.

Installierbare Services und *Servicekomponenten* sind Bestandteil der Produktstruktur von IBM MQ. Im Zentrum dieser Struktur steht der Warteschlangenmanager, der die Funktionen und die Regeln implementiert, die mit der Message Queue Interface (MQI) in Zusammenhang stehen. Für diesen zentralen Bestandteil sind eine Reihe von Servicefunktionen, auch als *installierbare Services* bezeichnet, erforderlich. In IBM MQ for IBM i steht als installierbarer Service der Berechtigungsservice zur Verfügung.

Jeder installierbare Service setzt sich aus einer Reihe zusammengehöriger Funktionen zusammen und wird unter Verwendung einer oder mehrerer *Servicekomponenten* installiert. Bei jeder dieser Komponenten handelt es sich um eine Schnittstelle mit einer festen Architektur, die öffentlich verfügbar ist. So können auch unabhängige Softwareanbieter und andere Drittanbieter als Ersatz oder Erweiterung für die von IBM MQ for IBM i bereitgestellten Produkte installierbare Komponenten bereitstellen. Tabelle 140 auf Seite 999 enthält eine Zusammenfassung der Unterstützung für den Berechtigungsservice.

Tabelle 140. Zusammenfassung der Komponenten des Berechtigungsservice

Bereitgestellte Komponente	**Funktion**	**Voraussetzungen**
Objektberechtigungsmanager (OAM)	Führt Berechtigungsprüfungen für Befehle und MQI-Aufrufe aus. Benutzer können eigene Komponenten erstellen, die den OAM ergänzen oder ersetzen.	(Geeignete Plattformberechtigungsfunktionen werden vorausgesetzt)
Namensservicekomponente DCE **Anmerkung:** DCE wird nur von IBM MQ vor V6.0 unterstützt.	• Ermöglicht Warteschlangenmanagern die gemeinsame Nutzung von Warteschlangen oder • Benutzerdefiniert **Anmerkung:** Für gemeinsam genutzte Warteschlangen muss das Attribut **Scope** auf CELL gesetzt sein.	• Für die bereitgestellte Komponente ist DCE erforderlich oder • Ein eigener Namensmanager oder der Namensmanager eines Drittanbieters

IBM i *Funktionen und Komponenten unter IBM i*

In diesem Abschnitt erhalten Sie Informationen zu Funktionen und Komponenten, Eingangspunkten, Rückgabecodes und Komponentendaten, die Sie in IBM MQ for IBM i verwenden können.

Jeder Service besteht aus einer Reihe zusammengehöriger Funktionen. Der Namensservice enthält beispielsweise Funktionen für folgende Vorgänge:

- Suchen nach dem Namen einer Warteschlange, wobei der Name des Warteschlangenmanagers zurückgegeben wird, in dem die Warteschlange definiert ist
- Einfügen eines Warteschlangennamens in das Serviceverzeichnis
- Löschen eines Warteschlangennamens aus dem Serviceverzeichnis

Darüber hinaus sind Initialisierungs- und Beendigungsfunktionen enthalten.

Ein installierbarer Service wird von einer oder mehreren Servicekomponenten bereitgestellt. Jede Komponente kann einige oder alle der für den betreffenden Service definierten Funktionen ausführen. Außerdem ist die Komponente für die Verwaltung aller zu Grunde liegenden Ressourcen oder Softwareprodukte verantwortlich, die zur Implementierung des Service erforderlich sind. Konfigurationsdateien stellen eine bewährte Methode zum Laden der Komponente und Bestimmen der Adressen der bereitgestellten funktionalen Routinen dar.

Services und Komponenten stehen wie folgt in Zusammenhang:

- Ein Service wird für einen Warteschlangenmanager über die Zeilengruppen einer Konfigurationsdatei definiert.
- Jeder einzelne Service wird durch bereitgestellten Code im Warteschlangenmanager unterstützt. Dieser Code kann von den Benutzern nicht geändert werden, d. h. sie können keine eigenen Services erstellen.
- Jeder Service wird durch mindestens eine Komponente implementiert, die entweder im Rahmen des Produkts bereitgestellt oder benutzerdefiniert sein kann. Für einen Service können mehrere Komponenten aufgerufen werden, von denen jede unterschiedliche Funktionen innerhalb des Service übernimmt.
- Eingangspunkte verbinden die Servicekomponenten mit dem entsprechenden Code im Warteschlangenmanager.

Eingangspunkte

Jede Servicekomponente ist dargestellt durch eine Liste der Eingangspunktadressen der Routinen, die einen bestimmten installierbaren Service unterstützen. Der installierbare Service definiert die Funktionen, die von den einzelnen Routinen ausgeführt werden. Die Reihenfolge der Servicekomponenten bei der Konfiguration bestimmt auch, in welcher Reihenfolge die Eingangspunkte aufgerufen werden, um eine Serviceanforderung zu erfüllen. In der bereitgestellten Headerdatei `cmqzc.h` sind die Eingangspunkte für die einzelnen Services mit dem Präfix 'MQZID_' versehen.

Rückkehrcodes

Servicekomponenten stellen dem Warteschlangenmanager Rückgabecodes zur Verfügung, anhand derer verschiedene Bedingungen gemeldet werden. Sie dokumentieren den Erfolg oder das Fehlschlagen der Operation und geben an, ob der Warteschlangenmanager mit der nächsten Servicekomponente fortfahren soll. Diese Angabe erfolgt über einen separaten Fortsetzungsparameter (*Continuation*).

Komponentendaten

Möglicherweise müssen die Daten von den verschiedenen Funktionen einer Servicekomponente gemeinsam genutzt werden. Installierbare Services bieten einen optionalen Datenbereich, der bei jedem Aufruf einer bestimmten Servicekomponente übergeben werden kann. Dieser Datenbereich ist ausschließlich für die Nutzung durch die Servicekomponente vorgesehen. Er wird von allen Aufrufen einer angegebenen Funktion gemeinsam genutzt, selbst wenn diese aus unterschiedlichen Adressräumen oder Prozessen erfolgen. Dieser Datenbereich ist für die Servicekomponente jederzeit zugänglich, wann immer sie aufgerufen wird. Die Größe dieses Datenbereichs wird in der Zeilengruppe *`ServiceComponent`* festgelegt.

IBM i *Initialisierung unter IBM i*

Wenn die Routine der Komponenteninitialisierung aufgerufen wird, muss sie die Funktion MQZEP des Warteschlangenmanagers für jeden Eingangspunkt aufrufen, der von der Komponente unterstützt wird. MQZEP definiert einen Eingangspunkt für den Service. Für alle nicht definierten Exitpunkte wird der Wert NULL angenommen.

Primäre Initialisierung
Eine Komponente wird auf jeden Fall einmal mit dieser Initialisierungsoption aufgerufen, bevor ihr Aufruf auf eine andere Weise erfolgen kann.

Sekundäre Initialisierung
Auf bestimmten Plattformen kann eine Komponente mit dieser Initialisierungsoption aufgerufen werden. Sie kann beispielsweise einmal für alle Betriebssystemprozesse, Threads oder Tasks aufgerufen werden, über die auf den Service zugegriffen wird.

Bei Verwendung der sekundären Initialisierung:

- Die Komponente kann für eine sekundäre Initialisierung mehrmals aufgerufen werden. Für jeden derartigen Aufruf wird ein entsprechender sekundärer Beendigungsaufruf ausgegeben, wenn der Service nicht mehr benötigt wird.

  Für Berechtigungsservices ist dies der Aufruf MQZ_TERM_AUTHORITY.
- Bei jedem primären und sekundären Initialisierungsaufruf für eine Komponente müssen die Eingangspunkte durch Aufruf von MQZEP erneut angegeben werden.
- Für die Komponente wird nur eine Kopie der Komponentendaten verwendet; es gibt also nicht für jede sekundäre Initialisierung eine eigene Kopie.
- Erst nach Ausführung der sekundären Initialisierung kann die Komponente vom Betriebssystemprozess, Thread oder einer Task für andere Aufrufe des Service aufgerufen werden.
- Der Parameter **`Version`** der Komponente muss für die primäre und die sekundäre Initialisierung auf den gleichen Wert gesetzt sein.

Primäre Beendigung
Wenn die Komponente nicht mehr benötigt wird, wird sie auf jeden Fall einmal mit dieser Beendigungsoption aufgerufen. Danach erfolgen keine weiteren Aufrufe mehr für diese Komponente.

Sekundäre Beendigung
Die Komponente wird mit dieser Beendigungsoption aufgerufen, wenn sie zuvor für die sekundäre Initialisierung aufgerufen wurde.

IBM i *Services und Komponenten unter IBM i konfigurieren*

Servicekomponenten werden mithilfe der Konfigurationsdateien des Warteschlangenmanagers konfiguriert.

Vorgehensweise

1. Fügen Sie der Konfigurationsdatei des Warteschlangenmanagers `qm.ini`Zeilengruppen hinzu, um den Service für den Warteschlangenmanager zu definieren, und geben Sie die Position des Moduls an:
   - Diese Datei muss für jeden verwendeten Service die Zeilengruppe `Service` enthalten, die den Service für den Warteschlangenmanager definiert. Weitere Informationen finden Sie unter Zeilengruppe 'Service' der Datei qm.ini.
   - Für jede Komponente innerhalb eines Service muss die Zeilengruppe `ServiceComponent` vorhanden sein. Diese Zeilengruppe gibt den Namen und den Pfad des Moduls an, das den Code für diese Komponente enthält. Weitere Informationen finden Sie unter ZeilengruppeServiceComponent in der Datei qm.ini.

   Die Berechtigungsservicekomponente, der Objektberechtigungsmanager oder OAM, wird im Rahmen des Produkts bereitgestellt. Bei der Erstellung eines Warteschlangenmanagers wird die Konfigurationsdatei für Warteschlangenmanager automatisch mit den entsprechenden Zeilengruppen für den Berechtigungsservice und dessen Standardkomponente (Objektberechtigungsmanager) aktualisiert.

Für andere Komponenten muss die Konfigurationsdatei des Warteschlangenmanagers manuell konfiguriert werden.

Der Code für die einzelnen Servicekomponenten wird beim Start des Warteschlangenmanagers in den Warteschlangenmanager geladen. Dabei wird dynamisches Binden eingesetzt, sofern dies auf der Plattform unterstützt wird.

2.

IBM i *Eigene Servicekomponente unter IBM i erstellen*

In diesem Abschnitt erfahren Sie, wie eine Servicekomponente unter IBM MQ for IBM i erstellt wird.

So erstellen Sie eine eigene Servicekomponente:

- Vergewissern Sie sich, dass die Headerdatei `cmqzc.h` in Ihrem Programm enthalten ist.
- Erstellen Sie die gemeinsam genutzte Bibliothek, indem Sie das Programm kompilieren und mit den gemeinsam genutzten Bibliotheken `libmqm*` und `libmqmzf*` verknüpfen.

  Anmerkung: Da der Agent in einer Threadumgebung ausgeführt werden kann, müssen Sie den Objektberechtigungsmanager für die Ausführung in einer Threadumgebung erstellen. Dies umfasst die Verwendung der Threadversionen von `libmqm` und `libmqmzf`.
- Fügen Sie der Konfigurationsdatei des Warteschlangenmanagers Zeilengruppen hinzu, um den Service beim Warteschlangenmanager zu definieren und die Position des Moduls anzugeben.
- Stoppen Sie den Warteschlangenmanager und starten Sie ihn anschließend erneut, um die Komponente zu aktivieren.

IBM i *Berechtigungsservice unter IBM i*

Der Berechtigungsservice ist ein installierbarer Service, mit dem Warteschlangenmanager Berechtigungsfunktionen aufrufen können, beispielsweise um zu prüfen, ob eine Benutzer-ID zum Öffnen einer Warteschlange berechtigt ist.

Dieser Service ist eine Komponente der IBM MQ-Schnittstelle für die Sicherheitsaktivierung (SEI), die Teil des IBM MQ-Frameworks ist. Die folgenden Themen werden behandelt:

- „Objektberechtigungsmanager (OAM)” auf Seite 1002
- „Service im Betriebssystem definieren” auf Seite 1003
- „Zeilengruppen für Berechtigungsservice konfigurieren” auf Seite 1003
- „Berechtigungsserviceschnittstelle unter IBM i” auf Seite 1003

Objektberechtigungsmanager (OAM)

Die mit IBM MQ-Produkten bereitgestellte Berechtigungsservicekomponente ist der Objektberechtigungsmanager (Object Authority Manager; OAM). Der OAM ist standardmäßig aktiv und verwendet die folgenden Steuerbefehle:

- **WRKMQMAUT** Arbeit mit Berechtigungen
- **WRKMQMAUTD** Arbeit mit Berechtigungsdaten
- **DSPMQMAUT** Anzeige von Objektberechtigungen
- **GRTMQMAUT** Erteilen von Objektberechtigungen
- **RVKMQMAUT** Widerrufen von Objektberechtigungen
- **RFRMQMAUT** Aktualisieren der Sicherheit

Die Syntax dieser Befehle und deren Verwendung werden im Hilfetext zu den Steuersprachenbefehlen beschrieben. Der Objektberechtigungsmanager arbeitet mit der *Entität* eines Principals oder einer Gruppe.

Bei Ausgabe einer MQI-Anforderung oder Aufruf eines Befehls prüft der OAM die Berechtigung der mit der Operation verknüpften Entität, um festzustellen, ob er die folgenden Aktionen ausführen kann:

- Angeforderte Operation ausführen
- Auf die angegebenen Warteschlangenmanagerressourcen zugreifen

Mit dem Berechtigungsservice können Sie die Berechtigungsprüfungen für Warteschlangenmanager erweitern oder ersetzen, indem Sie eine eigene Berechtigungsservicekomponente erstellen.

Service im Betriebssystem definieren

Die Zeilengruppen des Berechtigungsservice in der Konfigurationsdatei `qm.ini` des Warteschlangenmanagers definieren den Berechtigungsservice beim Warteschlangenmanager. Informationen zu den verschiedenen Zeilengruppen finden Sie im Abschnitt „Services und Komponenten unter IBM i konfigurieren" auf Seite 1001.

Zeilengruppen für Berechtigungsservice konfigurieren

Unter IBM MQ for IBM i:

Principal
Ein IBM i-Systembenutzerprofil.

Gruppe
Ein IBM i-Systemgruppenprofil.

Berechtigungen können nur auf Gruppenebene erteilt oder entzogen werden. Eine Anforderung hinsichtlich der Erteilung oder Entziehung der Berechtigung eines Benutzers aktualisiert die Primärgruppe für diesen Benutzer.

Jeder Warteschlangenmanager verfügt über eine eigene Konfigurationsdatei. Der Standardpfad und Dateiname der warteschlangenmanagerspezifischen Konfigurationsdatei für den Warteschlangenmanager QMNAME lauten beispielsweise `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`.

Die Zeilengruppen *`Service`* und *`ServiceComponent`* für die Standardberechtigungskomponente werden der Datei `qm.ini` automatisch hinzugefügt, können aber durch `WRKENVVAR` überschrieben werden. Alle anderen *`ServiceComponent`*-Zeilengruppen müssen manuell hinzugefügt werden.

Die folgenden Zeilengruppen in der Konfigurationsdatei für Warteschlangenmanager definieren beispielsweise zwei Berechtigungsservicekomponenten:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

Abbildung 100. Zeilengruppen für den Berechtigungsservice in der Datei 'qm.ini' unter IBM i

Die erste Servicekomponenten-Zeilengruppe mit dem Wert `MQ.UNIX.authorization.service` definiert die Standardkomponente des Berechtigungsservice, d. h. den Objektberechtigungsmanager (OAM). Wenn Sie diese Zeilengruppe entfernen und den Warteschlangenmanager erneut starten, wird der Objektberechtigungsmanager inaktiviert und es werden keine Berechtigungsprüfungen vorgenommen.

IBM i *Berechtigungsserviceschnittstelle unter IBM i*

Die Berechtigungsserviceschnittstelle stellt mehrere Eingangspunkte für die Verwendung durch den Warteschlangenmanager bereit.

MQZ_AUTHENTICATE_USER
Authentifiziert eine Benutzer-ID und ein Kennwort und kann Identitätskontextfelder festlegen.

MQZ_CHECK_AUTHORITY
Prüft, ob eine Entität die Berechtigung besitzt, eine oder mehrere Operationen für ein angegebenes Objekt auszuführen.

MQZ_COPY_ALL_AUTHORITY
Kopiert alle aktuellen Berechtigungen, die für ein Referenzobjekt vorhanden sind, in ein anderes Objekt.

MQZ_DELETE_AUTHORITY
Löscht alle Berechtigungen, die einem angegebenen Objekt zugeordnet sind.

MQZ_ENUMERATE_AUTHORITY_DATA
Ruft alle Berechtigungsdaten ab, die den angegebenen Auswahlkriterien entsprechen.

MQZ_FREE_USER
Gibt zugeordnete Ressourcen frei.

MQZ_GET_AUTHORITY
Ruft die Berechtigung ab, über die eine Entität zum Zugriff auf ein bestimmtes Objekt verfügt.

MQZ_GET_EXPLICIT_AUTHORITY
Ruft entweder die Berechtigung ab, die eine benannte Gruppe für den Zugriff auf ein angegebenes Objekt besitzt (jedoch ohne die Zusatzberechtigung der Gruppe **nobody**), oder die Berechtigung, die die Primärgruppe des benannten Principals für den Zugriff auf ein angegebenes Objekt besitzt.

MQZ_INIT_AUTHORITY
Initialisiert die Berechtigungsservicekomponente.

MQZ_INQUIRE
Fragt die unterstützte Funktionalität des Berechtigungsservice ab.

MQZ_REFRESH_CACHE
Aktualisiert alle Berechtigungen.

MQZ_SET_AUTHORITY
Legt die Berechtigung fest, die eine Entität für ein angegebenes Objekt besitzt.

MQZ_TERM_AUTHORITY
Beendet die Berechtigungsservicekomponente.

Diese Eingangspunkte unterstützen die Verwendung der Windows Sicherheits-ID (NT SID).

Diese Namen sind in der Headerdatei `cmqzc.h`, die dazu verwendet werden kann, die Komponentenfunktionen mithilfe eines Prototyps zu testen, als **typedef**s definiert.

Die Initialisierungsfunktion (**MQZ_INIT_AUTHORITY**) muss der Haupteingangspunkt für die Komponente sein. Die übrigen Funktionen werden über die Eingangspunktadresse aufgerufen, die von der Initialisierungsfunktion in den Eingangspunktvektor der Komponente eingefügt wurde.

Weitere Informationen finden Sie im Abschnitt „Eigene Servicekomponente unter IBM i erstellen” auf Seite 1002.

Multi API-Exits auf Multiplatforms schreiben und kompilieren

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.

Anmerkung: z/OS Wird bei IBM MQ for z/OS nicht unterstützt.

Gründe für die Verwendung von API-Exits

Jede Ihrer Anwendungen erfüllt eine bestimmte Aufgabe und der Anwendungscode sollte diese Aufgabe so effizient wie möglich ausführen. Auf einer höheren Ebene kann es sinnvoll sein, für alle Anwendungen, die einen bestimmten Warteschlangenmanager verwenden, Standards oder Geschäftsprozesse auf den

Warteschlangenmanager anzuwenden. Es ist effizienter, dies oberhalb der Ebene der einzelnen Anwendungen zu tun, da in diesem Fall nicht der Code jeder betroffenen Anwendung geändert werden muss.

Es folgen einige Vorschläge für Bereiche, in denen API-Exits sinnvoll sein können:

Sicherheit
Aus Sicherheitsgründen können Sie Authentifizierung bieten und prüfen, ob Anwendungen zum Zugriff auf eine Warteschlange oder einen Warteschlangenmanager berechtigt sind. Sie können auch die Verwendung der API durch die Anwendungen überwachen und die einzelnen API-Aufrufe oder sogar die dabei verwendeten Parameter authentifizieren.

Flexibilität
Zur Gewährleistung der Flexibilität können Sie auf plötzliche Änderungen in Ihrem Geschäftsumfeld reagieren, ohne die Anwendungen ändern zu müssen, die sich auf die Daten dieser Umgebung stützen. Sie können beispielsweise API-Exits einsetzen, die auf Änderungen von Zinssätzen oder Wechselkursen oder auf Preisänderungen von Komponenten in einer Produktionsumgebung reagieren.

Überwachung der Verwendung einer Warteschlange oder eines Warteschlangenmanagers
Zur Überwachung der Verwendung einer Warteschlange oder eines Warteschlangenmanagers können Sie den Datenfluss von Anwendungen und Nachrichten verfolgen, Fehler in den API-Aufrufen protokollieren, Prüfprotokolle zu Buchhaltungszwecken einrichten oder Nutzungsstatistiken für Planungszwecke erfassen.

Ablauf bei Ausführung eines API-Exits

Sobald Sie ein Exitprogramm geschrieben und bei IBM MQ angegeben haben, ruft der Warteschlangenmanager Ihren Exitcode an den registrierten Punkten automatisch auf.

Die auszuführenden API-Exit-Routinen sind in Zeilengruppen unter Multiplatforms angegeben. In diesem Abschnitt sind die Zeilengruppen in den Konfigurationsdateien `mqs.ini` und `qm.ini` beschrieben.

Diese Routinen können an drei Stellen definiert werden:

1. In 'ApiExitCommon' in der Datei 'mqs.ini' werden Routinen für den gesamten IBM MQ angegeben, die beim Start der Warteschlangenmanager ausgeführt werden. Diese können durch für einzelne Warteschlangenmanager definierte Routinen überschrieben werden (siehe Artikel „3” auf Seite 1005 in dieser Liste).
2. Die Vorlage ApiExitin der Datei mqs.ini gibt Routinen für die gesamte IBM MQan, die in die lokale Gruppe ApiExitkopiert werden (siehe Element „3” auf Seite 1005 in dieser Liste), wenn ein neuer Warteschlangenmanager erstellt wird.
3. In 'ApiExitLocal' in der Datei 'qm.ini' werden Routinen angegeben, die auf einen bestimmten Warteschlangenmanager angewendet werden.

Wenn ein neuer Warteschlangenmanager erstellt wird, werden für diesen die 'ApiExitTemplate'-Definitionen in 'mqs.ini' in die 'ApiExitLocal'-Definitionen in 'qm.ini' kopiert. Beim Start eines Warteschlangenmanagers werden sowohl die 'ApiExitCommon'- als auch die 'ApiExitLocal'-Definitionen verwendet. Die 'ApiExitLocal'-Definitionen ersetzen die 'ApiExitCommon'-Definitionen, wenn in beiden eine Routine mit demselben Namen angegeben ist. Das Attribut `Sequence` , das in „API-Exits konfigurieren” auf Seite 1010 beschrieben wird, bestimmt die Reihenfolge, in der die in den Zeilengruppen definierten Routinen ausgeführt werden.

API-Exits in mehreren Installationen von IBM MQ verwenden

Verwenden Sie bei der Nutzung mehrerer Installationen die API-Exits für die frühere Version von IBM MQ, da die in IBM WebSphere MQ 7.1 an den Exits vorgenommenen Änderungen mit früheren Versionen eventuell nicht funktionieren. Weitere Informationen zu den an den Exits vorgenommenen Änderungen finden Sie in „Exits und installierbare Services unter AIX, Linux, and Windows schreiben” auf Seite 986.

Die für die API-Exits amqsaem und amqsaxe bereitgestellten Beispiele spiegeln die Änderungen wider, die beim Schreiben von Exits erforderlich sind. Die Clientanwendung muss vor dem Anwendungsstart sicherstellen, dass die korrekten IBM MQ-Bibliotheken, die der Installation des der Anwendung zugeordneten Warteschlangenmanagers entsprechen, mit dieser Anwendung verknüpft sind.

Multi

API-Exits schreiben

Mithilfe der Programmiersprache C können Exits für jeden API-Aufruf geschrieben werden.

Verfügbare Exits

Für jeden API-Aufruf sind die folgenden Exits verfügbar:

- MQCB zur erneuten Registrierung eines Callbacks für die Objektkennung und zur Steuerung der Aktivierung und Änderungen des Callbacks
- MQCTL zur Ausführung von Steuerungsaktionen für die Objektkennungen, die für eine Verbindung geöffnet sind
- MQCONN/MQCONNX zur Bereitstellung einer Warteschlangenmanager-Verbindungskennung für die Verwendung in nachfolgenden API-Aufrufen
- MQDISC zur Trennung der Verbindung mit einem Warteschlangenmanager
- MQBEGIN zum Starten einer globalen Arbeitseinheit (UOW)
- MQBACK zum Zurücksetzen einer Arbeitseinheit (UOW)
- MQCMIT zum Festschreiben einer Arbeitseinheit (UOW)
- MQOPEN zum Öffnen einer IBM MQ-Ressource für nachfolgenden Zugriff
- MQCLOSE zum Schließen einer IBM MQ-Ressource, die zuvor für den Zugriff geöffnet wurde
- MQGET zum Abrufen einer Nachricht aus einer Warteschlange, die zuvor für den Zugriff geöffnet wurde
- MQPUT1 zum Einreihen einer Nachricht in eine Warteschlange
- MQPUT zum Einreihen einer Nachricht in eine Warteschlange, die zuvor für den Zugriff geöffnet wurde
- MQINQ zur Abfrage der Attribute einer IBM MQ-Ressource, die zuvor für den Zugriff geöffnet wurde
- MQSET zum Festlegen der Attribute einer Warteschlange, die zuvor für den Zugriff geöffnet wurde
- MQSTAT zum Abrufen von Statusinformationen
- MQSUB zum Registrieren der Anwendungssubskription für ein bestimmtes Thema
- MQSUBRQ zur Anforderung einer Subskription

MQ_CALLBACK_EXIT stellt eine Exitfunktion zur Verfügung, mit der eine Verarbeitung zur Vorbereitung und Nachbereitung eines Callbacks ausgeführt wird. Weitere Informationen finden Sie unter Callback - MQ_CALLBACK_EXIT.

API-Exits schreiben

In API-Exits verfügen die Aufrufe in der Regel über das Format:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

wobei es sich bei *call* um den MQI-Aufrufnamen ohne das MQ-Präfix handelt (beispielsweise PUT, GET). Die Angaben im Feld *parameters* steuern die Funktion des Exits und stellen in erster Linie die Datenübertragung zwischen dem Exit und den externen Steuerblocks MQAXP (der API-Exit-Parameterstruktur) und MQAXC (der API-Exit-Kontextstruktur) bereit. *context* beschreibt den Kontext, in dem der API-Exit aufgerufen wurde, und *ApiCallParameters* steht für die Parameter des MQI-Aufrufs.

Um Ihnen beim Schreiben eines eigenen API-Exits zu helfen, wird ein Musterexit (amqsaxe0.c) bereitgestellt; dieser Exit generiert Traceeinträge in einer von Ihnen angegebenen Datei. Das Muster soll Ihnen als Ausgangspunkt beim Schreiben von Exits dienen. Weitere Informationen zur Verwendung des Musterexits finden Sie im Abschnitt „API Exit-Beispielprogramm” auf Seite 1131.

Weitere Informationen zu API-Exit-Aufrufen, externen Steuerblöcken und zugehörigen Themen finden Sie im Abschnitt API exit reference.

Allgemeine Informationen zum Schreiben, Kompilieren und Konfigurieren eines Exits finden Sie im Abschnitt „Exits und installierbare Services unter AIX, Linux, and Windows schreiben” auf Seite 986.

Nachrichtenkennungen in API-Exits verwenden

Sie können steuern, auf welche Nachrichteneigenschaften ein API-Exit Zugriff hat. Eigenschaften sind einem 'ExitMsgHandle' zugeordnet. Eigenschaften, die in einem PUT-Exit festgelegt sind, werden in der Nachricht angegeben, die eingereiht wird, doch Eigenschaften, die in einem GET-Exit abgerufen werden, werden nicht an die Anwendung zurückgegeben.

Wenn Sie mit dem Aufruf MQXEP MQI, in dem **Function** auf MQXF_INIT und **ExitReason** auf MQXR_CONNECTION gesetzt sind, eine MQ_INIT_EXIT-Exitfunktion registrieren, übergeben Sie eine MQXEPO-Struktur als den Parameter **ExitOpts**. Die MQXEPO-Struktur enthält das Feld 'ExitProperties', in dem die Gruppe der Eigenschaften angegeben ist, die dem Exit zur Verfügung gestellt werden sollen. Die Angabe erfolgt in Form einer Zeichenfolge, die das Präfix der Eigenschaften darstellt und einem MQRFH2-Ordnernamen entspricht.

Jeder API-Exit empfängt eine MQAXP-Struktur, die ein 'ExitMsgHandle'-Feld enthält. Dieses Feld ist auf einen durch IBM MQ generierten Wert gesetzt und ist verbindungsspezifisch. Diese Kennung bleibt deshalb zwischen API-Exits desselben Typs oder verschiedener Typen auf derselben Verbindung unverändert.

In einem MQ_PUT_EXIT oder MQ_PUT1_EXIT, in dem **ExitReason** auf MQXR_BEFORE gesetzt ist (also ein API-Exit, der vor dem Einreihen einer Nachricht ausgeführt wird), werden alle Eigenschaften (außer Nachrichtendeskriptoreigenschaften), die 'ExitMsgHandle' bei Ausführung des Exits zugeordnet werden, in der Nachricht angegeben, die eingereiht wird. Um dies zu verhindern, muss 'ExitMsgHandle' auf MQHM_NONE gesetzt werden. Sie können auch eine andere Nachrichtenkennung angeben.

In einem MQ_GET_EXIT und MQ_CALLBACK_EXIT wird die Kennung ExitMsgvon Eigenschaften gelöscht und mit den Eigenschaften gefüllt, die im Feld ExitProperties angegeben wurden, als der MQ_INIT_EXIT registriert wurde, außer Nachrichtendeskriptoreigenschaften. Diese Eigenschaften werden der abrufenden Anwendung nicht zur Verfügung gestellt. Wenn die abrufende Anwendung eine Nachrichtenkennung im Feld 'MQGMO (Get message options)' angegeben hat, sind alle Eigenschaften, die dieser Kennung zugeordnet sind, einschließlich der Nachrichtendeskriptoreigenschaften, für den API-Exit verfügbar. Um zu verhindern, dass 'ExitMsgHandle' mit Eigenschaften gefüllt wird, können Sie MQHM_NONE angeben.

Ein Musterprogramm (amqsaem0.c) wird bereitgestellt, um die Verwendung von Nachrichtenkennungen in API-Exits zu veranschaulichen.

Zugehörige Verweise

Referenzinformationen zu Benutzerexits, API-Exits und installierbaren Services

Multi *API-Exits kompilieren*

Nachdem Sie einen Exit geschrieben haben, können Sie ihn wie hier beschrieben kompilieren und verbinden.

Die folgenden Beispiele zeigen die Befehle, die für das im Abschnitt „API Exit-Beispielprogramm” auf Seite 1131 beschriebene Musterprogramm verwendet werden. Für Plattformen mit Ausnahme von Windows-Systemen finden Sie den API-Exit-Mustercode im Verzeichnis *MQ_INSTALLATION_PATH*/`samp` und die kompilierte und verbundene gemeinsam genutzte Bibliothek im Verzeichnis *MQ_INSTALLATION_PATH*/`samp/bin`.

Windows Für Windows-Systeme finden Sie den API-Exit-Mustercode in *MQ_INSTALLATION_PATH* `\Tools\c\Samples`. *MQ_INSTALLATION_PATH* steht für das Verzeichnis, in dem IBM MQ installiert ist.

Anmerkung: Eine Anleitung zur Programmierung von 64-Bit-Anwendungen finden Sie im Abschnitt Codierungsstandards auf 64-Bit-Plattformen.

Für Multicasting-Clients müssen API-Exits und Datenkonvertierungsexits clientseitig ausgeführt werden können, da manche Nachrichten den Warteschlangenmanager unter Umständen nicht durchlaufen. Die folgenden Bibliotheken sind sowohl Teil der Client- als auch der Serverpakete:

Tabelle 141. In den Client- und Serverpaketen enthaltene Bibliotheken	
Betriebssystem	**Bibliotheken**
AIX	32 Bit & 64 Bit: libmqm.a & libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 Bit & 64 Bit: libmqm.so & libmqm_r.so
Windows	32 Bit & 64 Bit: mqm.dll & mqm.pdb

Linux AIX *API-Exits auf AIX and Linux-Systemen kompilieren*

Beispiele zur Kompilierung von API-Exits auf AIX and Linux-Systemen.

Auf allen Plattformen ist der Eingangspunkt in das Modul 'MQStart'.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

unter AIX

AIX

Kompilieren Sie den API-Exit-Quellcode, indem Sie einen der folgenden Befehle ausgeben:

32-Bit-Anwendungen

Nicht-thread-basiert

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

64-Bit-Anwendungen

Nicht-thread-basiert

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

unter Linux

Linux

Kompilieren Sie den API-Exit-Quellcode, indem Sie einen der folgenden Befehle ausgeben:

31-Bit-Anwendungen

Nicht-thread-basiert

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

32-Bit-Anwendungen

Nicht-thread-basiert

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

64-Bit-Anwendungen

Nicht-thread-basiert

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows *API-Exits auf Windows-Systemen kompilieren*

In diesem Abschnitt ist beschrieben, wie Sie das API-Exit-Musterprogramm `amqsaxe0.c` unter Windows kompilieren und verbinden.

Eine Manifestdatei ist ein optionales XML-Dokument, das die Versions- oder sonstigen -informationen enthält, die in eine kompilierte Anwendung oder DLL integriert werden können.

Wenn Sie über kein derartiges Dokument verfügen, übergehen Sie den Parameter -manifest *manifest.file* im Befehl **mt**.

Passen Sie die Befehle in den in Abbildung 101 auf Seite 1009 oder Abbildung 102 auf Seite 1010 gezeigten Beispielen an, um `amqsaxe0.c` unter Windows zu kompilieren und zu verbinden. Die Befehle funktionieren in Microsoft Visual Studio 2008, 2010 und 2012. Bei den Beispielen wird vorausgesetzt, dass das Verzeichnis `C:\Programme\IBM\MQ\tools\c\samples` das aktuelle Verzeichnis ist.

32 Bit

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def



amqsaxe0.obj \
   /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

Abbildung 101. amqsaxe0.c unter einer 32-Bit-Version von Windows kompilieren und verbinden

64-Bit

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \



amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

Abbildung 102. amqsaxe0.c unter einer 64-Bit-Version von Windows kompilieren und verbinden

Zugehörige Konzepte

„API Exit-Beispielprogramm” auf Seite 1131
Der API-Beispielexit generiert einen MQI-Trace in einer benutzerdefinierten Datei mit einem Präfix, das in der Umgebungsvariablen **MQAPI_TRACE_LOGFILE** definiert ist.

IBM i *API-Exits unter IBM i kompilieren*

API-Exits unter IBM i kompilieren.

Ein Exit wird wie folgt erstellt (in Programmiersprache C):

1. Erstellen Sie ein Modul mithilfe von CRTCMOD. Kompilieren Sie es für die Verwendung von Teraspace, indem Sie den Parameter TERASPACE(*YES *TSIFC) einschließen.
2. Erstellen Sie mithilfe von CRTSRVPGM ein Serviceprogramm aus dem Modul. Für Multithread-API-Exits ist die Bindung an das Serviceprogramm QMQM/LIBMQMZF_R erforderlich.

API-Exits konfigurieren

Durch eine Änderung der Konfigurationsdaten kann IBM MQ für die Verwendung von API-Exits konfiguriert werden.

Zum Ändern der Konfigurationsdaten müssen die Zeilengruppen zur Definition der Exitroutinen und ihrer Ausführungsreihenfolge geändert werden. Die Änderung kann wie folgt vorgenommen werden:

- Windows Linux Über den IBM MQ Explorer unter Windows und Linux (x86- und x86-64-Plattformen).
- Windows Mithilfe des Befehls **amqmdain** unter Windows.
- Multi Mithilfe der Dateien `mqs.ini` und `qm.ini` direkt unter Multiplatforms.

  Die Datei `mqs.ini` enthält relevante Informationen für alle Warteschlangenmanager auf einem bestimmten Knoten. Sie finden sie an den folgenden Positionen:

  – Linux AIX Im Verzeichnis `/var/mqm` unter AIX and Linux.

  – Windows In dem WorkPath , der im Schlüssel `HKLM\SOFTWARE\IBM\WebSphere MQ` auf Windows -Systemen angegeben ist

  – IBM i Im Verzeichnis `/QIBM/UserData/mqm` unter IBM i.

  Die Datei `qm.ini` enthält relevante Informationen für einen bestimmten Warteschlangenmanager. Für jeden Warteschlangenmanager gibt es eine Konfigurationsdatei, die sich im Stammverzeichnis der Baumstruktur des betreffenden Warteschlangenmanagers befindet. Der Pfad und der Name einer Konfigurationsdatei für einen WS-Manager mit dem Namen QMNAME sind beispielsweise:

IBM i Auf IBM i-Systemen:

```
/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
```

Linux AIX Auf AIX and Linux-Systemen:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

Windows Auf Windows-Systemen:

```
C:\ProgramData \IBM \MQ\qmgrs\QMNAME\qm.ini
```

Bevor Sie eine Konfigurationsdatei editieren, sichern Sie sie so, dass Sie eine Kopie haben, auf die Sie zurückgreifen können, wenn die Notwendigkeit besteht.

Konfigurationsdateien können auf eine der folgenden Arten bearbeitet werden:

– Automatisch mit Befehlen, die die Konfiguration von Warteschlangenmanagern auf dem Knoten ändern.
– Manuell mithilfe eines Standardtexteditors

Wenn Sie einen falschen Wert in einem Konfigurationsdateiattribut festlegen, wird der Wert ignoriert und eine Bedienernachricht ausgegeben, um das Problem anzugeben. Dies hat denselben Effekt wie die Nichtangabe des Attributs.

Zu konfigurierende Zeilengruppen

Folgende Zeilengruppen müssen geändert werden:

ApiExitCommon

Wird in der Datei `mqs.ini` und im IBM MQ Explorer auf der IBM MQ-Eigenschaftenseite unter 'Exits' definiert.

Beim Start eines Warteschlangenmanagers werden die Attribute in dieser Zeilengruppe gelesen und dann durch die API-Exits überschrieben, die in der Datei `qm.ini` definiert sind.

ApiExitTemplate

Wird in der Datei `mqs.ini` und im IBM MQ Explorer auf der IBM MQ-Eigenschaftenseite unter 'Exits' definiert.

Bei der Erstellung eines Warteschlangenmanagers werden die Attribute in dieser Zeilengruppe unter die Zeilengruppe 'ApiExitLocal' der neu erstellten Datei 'qm.ini' kopiert.

ApiExitLocal

Wird in der Datei `qm.ini` und im IBM MQ Explorer auf der Eigenschaftenseite für Warteschlangenmanager unter 'Exits' definiert.

Beim Start des Warteschlangenmanagers werden die in `mqs.ini` definierten Standardwerte durch die hier definierten API-Exits überschrieben.

Attribute für die Zeilengruppen

- Verwenden Sie zur Benennung des API-Exits das folgende Attribut:

Name=Name_des_API-Exits
Der beschreibende Name des API-Exits, der im Feld 'ExitInfoName' der MQAXP-Struktur übergeben wird.

Dieser Name muss eindeutig sein und darf maximal 48 Zeichen umfassen, wobei es sich um gültige Zeichen für die Namen von IBM MQ-Objekten (z. B. Warteschlangennamen) handeln muss.

- Mit den folgenden Attributen können Sie das Modul und den Eingangspunkt des auszuführenden API-Exitcodes angeben:

  Function=Funktionsname
  Der Name des Einstiegspunkts der Funktion in dem Modul, das den API-Exitcode enthält. Dieser Einstiegspunkt ist die Funktion MQ_INIT_EXIT.

  Die Länge dieses Felds ist auf den Wert von MQ_EXIT_NAME_LENGTH beschränkt.

  Module=Modulname
  Das Modul, das den API-Exitcode enthält.

  Wenn dieses Feld den vollständigen Pfadnamen enthält, wird es unverändert übernommen.

  Ist in diesem Feld nur der Modulname angegeben, wird das Modul über das Attribut `ExitsDefaultPath` im `ExitPath` in der Datei 'qm.ini' positioniert.

  Auf Plattformen, die separate Threadbibliotheken unterstützen, muss sowohl eine Threadversion als auch eine Version des API-Exitmoduls ohne Threads bereitgestellt werden. Die Threadversion muss das Suffix `_r` aufweisen. Von der Threadversion des IBM MQ-Anwendungsstubs wird dem angegebenen Modulnamen vor dem Laden implizit `_r` angehängt.

  Die Länge dieses Felds ist auf die maximale Pfadlänge begrenzt, die von der Plattform unterstützt wird.

- Optional können Sie mit dem Exit Daten übergeben. Verwenden Sie hierfür das folgende Attribut:

  Data=Datenname
  Die Daten, die an den API-Exit im Feld 'ExitData' der MQAXP-Struktur übergeben werden sollen.

  Wenn Sie dieses Attribut angeben, werden die führenden und abschließenden Leerzeichen entfernt. Die verbleibende Zeichenfolge wird auf 32 Zeichen abgeschnitten und das Ergebnis wird an den Exit übergeben. Wenn Sie dieses Attribut ausschließen, wird der Standardwert (32 Leerzeichen) an den Exit übergeben.

  Die maximale Länge dieses Felds beträgt 32 Zeichen.

- Geben Sie über das folgende Attribut die Position dieses Exits im Verhältnis zu anderen Exits an:

  Sequence=Folgenummer
  Die Reihenfolge, in der dieser API-Exit in Relation zu anderen API-Exits aufgerufen wird. Ein Exit mit einer niedrigen Folgenummer wird vor einem Exit mit einer höheren Folgenummer aufgerufen. Die Folgenummern für die Exits müssen nicht fortlaufend vergeben werden. Die Folge '1, 2, 3' führt zu demselben Ergebnis wie die Folge '7, 42, 1096'. Wenn zwei Exits dieselbe Folgenummer aufweisen, entscheidet der Warteschlangenmanager, welcher Exit zuerst aufgerufen wird. Nach dem Ereignis können Sie feststellen, welcher Exit aufgerufen wurde, indem Sie die Uhrzeit oder einen Datenpunkt in 'ExitChainArea' (durch 'ExitChainAreaPtr' in MQAXP angegeben) eingeben oder eine eigene Protokolldatei schreiben.

  Dieses Attribut ist ein numerischer Wert ohne Vorzeichen.

Musterzeilengruppen

Die Musterdatei 'mqs.ini' enthält folgende Zeilengruppen:

ApiExitTemplate
Diese Zeilengruppe definiert einen Exit mit dem beschreibenden Namen `OurPayrollQueueAuditor`, dem Modulnamen `auditor` und der Folgenummer 2. An den Exit wird der Datenwert '123' übergeben.

ApiExitCommon
Diese Zeilengruppe definiert einen Exit mit dem beschreibenden Namen `MQPoliceman`, dem Modulnamen `tmqp` und der Folgenummer 1. Bei den übergebenen Daten handelt es sich um eine Anweisung (`CheckEverything`).

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

In der folgenden Musterdatei 'qm.ini' ist eine ApiExitLocal-Definition eines Exits mit dem beschreibenden Namen `ClientApplicationAPIchecker`, dem Modulnamen `ClientAppChecker` und der Folgenummer 3 enthalten.

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

Kanalexitprogramme für Messaging-Kanäle

In dieser Themengruppe finden Sie Informationen zu IBM MQ-Kanalexitprogrammen für Nachrichtenkanäle.

Nachrichtenkanalagenten (Message Channel Agents, MCAs) können auch Datenkonvertierungsexits aufrufen. Weitere Informationen zum Schreiben von Datenkonvertierungsexits finden Sie im Abschnitt „Datenkonvertierungsexits schreiben” auf Seite 1035.

Einige dieser Informationen gelten auch für Exits auf MQI-Kanälen, die IBM MQ MQI clients mit Warteschlangenmanagern verbinden. Weitere Informationen finden Sie im Abschnitt Channel-exit programs for MQI channels.

Kanalexitprogramme werden an definierten Stellen in der Verarbeitung, die durch Nachrichtenkanalagentenprogramme vorgenommen wird, aufgerufen.

Einige dieser Benutzerexitprogramme agieren in komplementären Paaren. Wird ein Benutzerexitprogramm beispielsweise vom sendenden Nachrichtenkanalagenten zur Verschlüsselung der Nachrichten für die Übertragung aufgerufen, muss der komplementäre Prozess auf der Empfangsseite funktionieren, damit der Prozess umgekehrt werden kann.

Tabelle 142 auf Seite 1013 zeigt die Kanalexitarten, die für die einzelnen Kanaltypen verfügbar sind.

Tabelle 142. Für die einzelnen Kanaltypen verfügbare Kanalexits

Kanaltyp	Nachrichtenexit	Nachrichtenwiederholungsexit	Empfangsexit	Sicherheitsexit	Sendeexit	Autodefinitionsexit
Senderkanal	Ja		Ja	Ja	Ja	
Serverkanal	Ja		Ja	Ja	Ja	

Tabelle 142. Für die einzelnen Kanaltypen verfügbare Kanalexits (Forts.)						
Kanaltyp	**Nachrichtenexit**	**Nachrichtenwiederholungsexit**	**Empfangsexit**	**Sicherheitsexit**	**Sendeexit**	**Autodefinitionsexit**
Clustersenderkanal	Ja		Ja	Ja	Ja	Ja
Empfängerkanal	Ja	Ja	Ja	Ja	Ja	Ja
Requesterkanal	Ja	Ja	Ja	Ja	Ja	
Clusterempfängerkanal	Ja	Ja	Ja	Ja	Ja	Ja
Clientverbindungskanal			Ja	Ja	Ja	
Serververbindungskanal			Ja	Ja	Ja	Ja

Anmerkungen: z/OS

1. Unter z/OS gilt der Autodefinitionsexit nur für Clustersenderkanäle und Clusterempfängerkanäle.

Wenn Sie Kanalexits für einen Client ausführen möchten, können Sie die Umgebungsvariable MQSERVER nicht verwenden. Erstellen und referenzieren Sie stattdessen eine Definitionstabelle für Clientkanäle (CCDT - Client Channel Definition Table), wie im Abschnitt Client channel definition table beschrieben.

Verarbeitungsübersicht

Eine Übersicht über die Verwendung von Kanalexitprogrammen durch Nachrichtenkanalagenten.

Beim Start tauschen die Nachrichtenkanalagenten einen Startdialog aus, um die Verarbeitung zu synchronisieren. Dann wechseln sie zu einem Datenaustausch, der die Sicherheitsexits einschließt. Diese Exits müssen erfolgreich beendet werden, damit die Startphase abgeschlossen und Nachrichten übertragen werden können.

Bei der Sicherheitsprüfungsphase handelt es sich um eine Schleife wie in Abbildung 103 auf Seite 1014 dargestellt.

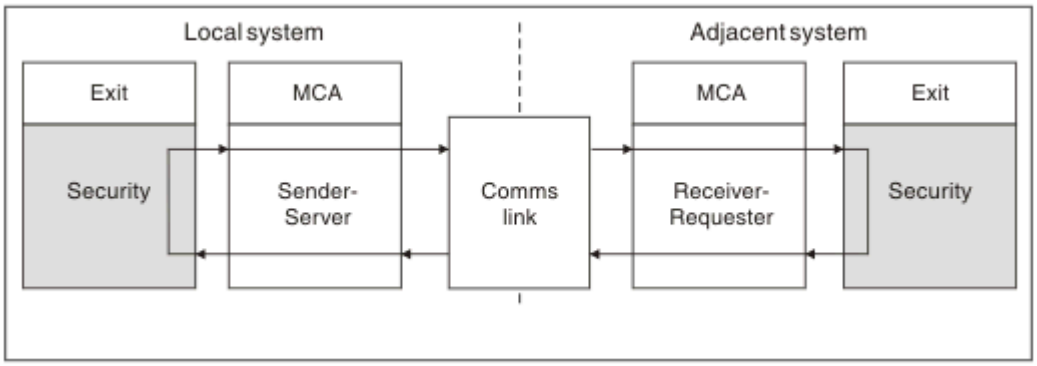


Abbildung 103. Sicherheitsexitschleife

Während der Nachrichtenübertragungsphase ruft der sendende Nachrichtenkanalagent Nachrichten aus einer Übertragungswarteschlange ab, ruft den Nachrichtenexit und dann den Sendeexit auf und sendet

die Nachricht dann wie in Abbildung 104 auf Seite 1015 dargestellt an den empfangenden Nachrichtenkanalagenten.

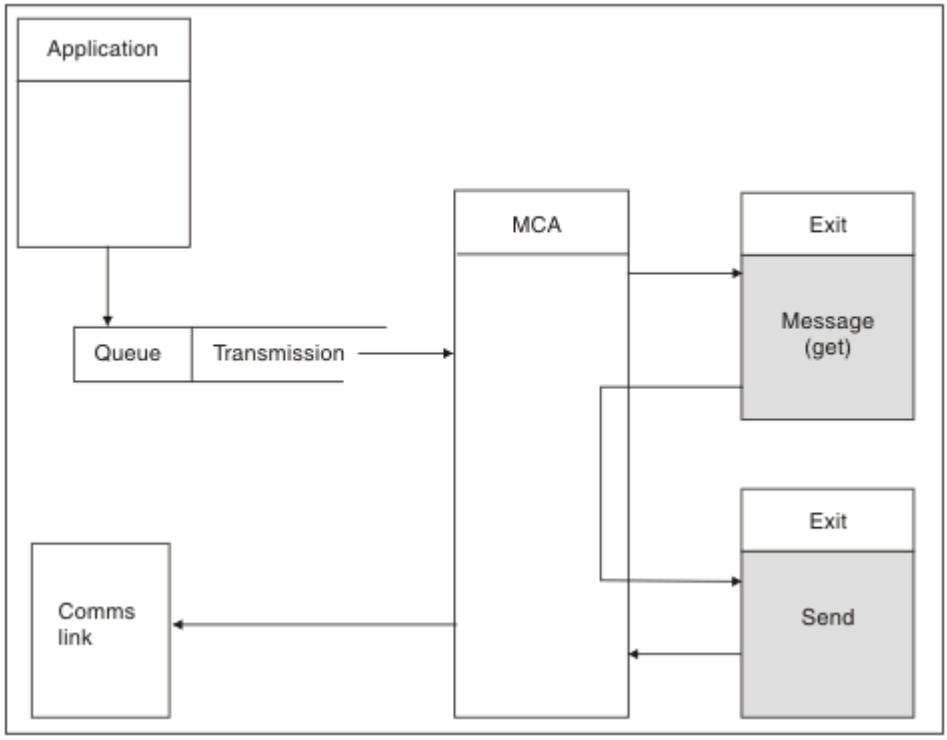


Abbildung 104. Beispiel eines Sendeexits auf der Sendeseite des Nachrichtenkanals

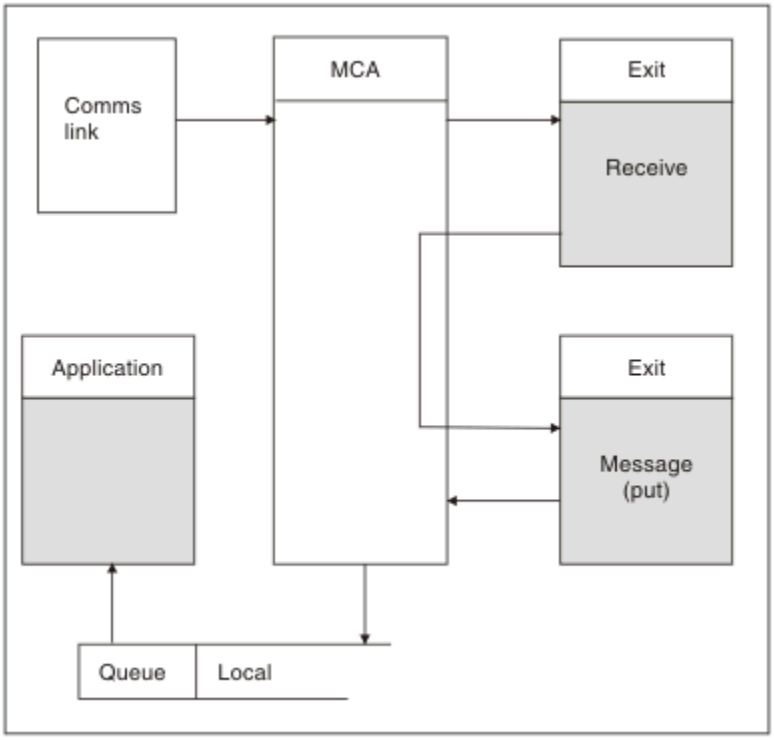


Abbildung 105. Beispiel eines Empfangsexits auf der Empfangsseite des Nachrichtenkanals

Der empfangende Nachrichtenkanalagent empfängt eine Nachricht vom der Datenübertragungsverbindung, ruft den Empfangsexit und dann den Nachrichtenexit auf und reiht die Nachricht dann wie in

Abbildung 105 auf Seite 1015 dargestellt in die lokale Warteschlange ein. (Der Empfangsexit kann vor dem Aufruf des Nachrichtenexits mehrmals aufgerufen werden.)

Kanalexitprogramme schreiben

Mithilfe der folgenden Informationen können Sie Kanalexitprogramme schreiben.

Benutzerexits und Kanalexitprogramme können mit Ausnahme der in den folgenden Abschnitten genannten Punkte alle MQI-Aufrufe verwenden. In MQ V7 und höher enthält die MQCXP-Struktur (Version 7 und höher) die Verbindungskennung 'hConn', die anstelle einer Ausgabe von MQCONN verwendet werden kann. In älteren Versionen muss zum Abruf der Verbindungskennung ein MQCONN-Aufruf ausgegeben werden; in diesem Fall wird allerdings die Warnung MQRC_ALREADY_CONNECTED gemeldet, da der Kanal selbst bereits mit dem Warteschlangenmanager verbunden ist.

Beachten Sie, dass der Kanalexit threadsicher sein muss.

Bei Exits in Clientverbindungskanälen hängt der Warteschlangenmanager, zu dem der Exit eine Verbindung herzustellen versucht, davon ab, wie der Exit verbunden war. Wenn der Exit mit MQM.LIB (bzw. QMQM/LIBMQM unter IBM i) verbunden war und Sie im MQCONN-Aufruf keinen Warteschlangenmanagernamen angeben, versucht der Exit, eine Verbindung zum Standardwarteschlangenmanager Ihres Systems herzustellen. Wenn der Exit mit MQM.LIB (bzw. QMQM/LIBMQM unter IBM i) verbunden war und Sie den Warteschlangenmanagernamen angeben, der über das Feld 'QMgrName' in der MQCD-Struktur an den Exit übergeben wurde, versucht der Exit, eine Verbindung zu diesem Warteschlangenmanager herzustellen. Wurde der Exit mit MQIC.LIB oder einer anderen Bibliothek verbunden, schlägt der MQCONN-Aufruf fehl, und zwar unabhängig davon, ob Sie einen Warteschlangenmanagernamen angeben oder nicht.

Sie sollten den Status der Transaktion, die dem übergebenen 'hConn'-Element in einem Kanalexit zugeordnet ist, nicht ändern; die MQCMIT-, MQBACK- oder MQDISC-Verben dürfen nicht mit dem 'hConn'-Element des Kanals verwendet werden und das MQBEGIN-Verb, dass das 'hConn'-Element des Kanals angibt, kann nicht genutzt werden.

Wird MQCONNX, das MQCNO_HANDLE_SHARE_BLOCK oder MQCNO_HANDLE_SHARE_NO_BLOCK angibt, zur Erstellung einer neuen IBM MQ-Verbindung verwendet, sind Sie dafür zuständig, für eine ordnungsgemäße Verwaltung des Kanals und die ordnungsgemäße Verbindungstrennung vom Warteschlangenmanager zu sorgen. Ein Kanalexit, der beispielsweise bei jedem Aufruf eine neue Verbindung zum Warteschlangenmanager erstellt, ohne die Verbindung zu trennen, führt zu einer Anhäufung von Verbindungskennungen und zu einer höheren Anzahl an Agententhreads.

Ein Exit wird in demselben Thread wie der Nachrichtenkanalagent selbst ausgeführt und verwendet dieselbe Verbindungskennung. Da er also in derselben UOW wie der Nachrichtenkanalagent ausgeführt wird, werden sämtliche Aufrufe, die unter einem Synchronisationspunkt getätigt werden, vom Kanal am Ende des Stapels festgeschrieben oder zurückgesetzt.

Daher könnte ein Kanalnachrichtenexit Benachrichtigungen senden, die bei der Festschreibung des Stapels, der die ursprüngliche Nachricht enthält, nur für diese Warteschlange festgeschrieben werden. Es ist also möglich, MQI-Aufrufe unter Synchronisationspunktsteuerung über einen Kanalnachrichtenexit auszugeben.

Die Felder im MQCD können vom Kanalexit geändert werden. Diese Änderungen werden jedoch nur unter den nachfolgend aufgeführten Umständen berücksichtigt. Wenn ein Kanalexitprogramm ein Feld in der MQCD-Datenstruktur ändert, wird der neue Wert vom IBM MQ-Kanalprozess ignoriert. Der neue Wert verbleibt jedoch im MQCD und wird an alle verbleibenden Exits in einer Exitkette übergeben sowie an jeden Datenaustausch mit einer gemeinsamen Nutzung der Kanalinstanz. Weitere Informationen finden Sie im Abschnitt Ändern von MQCD-Feldern in einem Kanalexit.

Außerdem darf die Funktion der nicht wiedereintrittsfähigen C-Bibliothek in einem Kanalexitprogramm nicht für Programme verwendet werden, die in der Programmiersprache C geschrieben sind.

Linux AIX Falls Sie mehrere Kanalexitbibliotheken gleichzeitig verwenden, können bei manchen UNIX and Linux-Plattformen Probleme auftreten, wenn der Code für zwei verschiedene Exits Funktionen mit identischem Namen enthält. Beim Laden eines Kanalexits löst das dynamische Ladeprogramm Funktionsnamen in der Exitbibliothek in die Adressen auf, an denen die Bibliothek geladen wird. Wenn zwei Exitbibliotheken separate Funktionen definieren, die zufällig identische Namen haben, löst

dieser Auflösungsprozess die Funktionsnamen der einen Bibliothek möglicherweise falsch auf, sodass die Funktionen einer anderen Bibliothek verwendet werden. Tritt dieses Problem auf, müssen Sie im Linker angeben, dass er nur die erforderlichen Exit- und 'MQStart'-Funktionen exportieren darf, da diese Funktionen davon nicht betroffen sind. Die übrigen Funktionen müssen lokal angezeigt werden, damit sie nicht von Funktionen außerhalb ihrer eigenen Exitbibliothek verwendet werden. In der Dokumentation des Linkers finden Sie weitere Informationen.

Alle Exits werden mit einer Parameterstruktur für Kanalexits (MQCXP), einer Kanaldefinitionsstruktur (MQCD), einem vorbereiteten Datenpuffer, einem Parameter für die Datenlänge und einem Parameter für die Puffergröße aufgerufen. Die Puffergröße darf folgende Grenzen nicht überschreiten:

- Bei Nachrichtenexits müssen Sie die längste Nachricht, die über den Kanal gesendet werden soll, plus die Länge der MQXQH-Struktur einkalkulieren.
- Bei Sende- und Empfangsexits muss der größte zulässige Puffer wie folgt lauten:

  LU 6.2
  32 KB

  TCP:
  IBM i IBM i 16 KB

  IBM i Sonstige 32 KB

  Anmerkung: Die maximal verwendbare Länge kann 2 Byte unter dieser Länge liegen. Überprüfen Sie den in **MaxSegmentLength** zurückgegebenen Wert auf Details. Weitere Informationen zu **MaxSegmentLength**finden Sie unter MaxSegmentLänge.

  NetBIOS:
  64 KB

  SPX:
  64 KB

  Anmerkung: Empfangsexits in Senderkanälen und Senderexits in Empfängerkanälen verwenden für TCP Puffer mit 2 KB.
- Für Sicherheitsexits wird von der Funktion der verteilten Steuerung von Warteschlangen ein Puffer mit 4000 Byte zugeordnet.

Es ist zulässig, dass der Exit einen alternativen Puffer gemeinsam mit den relevanten Parametern zurückgibt. Im Abschnitt „Kanalexitprogramme für Messaging-Kanäle” auf Seite 1013 finden Sie Details zum Aufruf.

z/OS *Writing channel exit programs on z/OS*

You can use the following information to help you write and compile channel-exit programs for z/OS.

The exits are started as if by a z/OS LINK, in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31 bit addressing mode

The link-edited modules must be placed in the data set specified by the CSQXLIB DD statement of the channel initiator address space procedure; the names of the load modules are specified as the exit names in the channel definition.

When writing channel exits for z/OS, the following rules apply:

- Exits must be written in assembler or C; if C is used, it must conform to the C systems programming environment for system exits, described in the z/OS C/C++ Programming Guide.

- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the channel initiator is running. The new version is used when the channel is restarted.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment, on return, to that at entry.
- Exits must free any storage obtained, or ensure that it is freed by a subsequent exit invocation.

  For storage that is to persist between invocations, use the z/OS STORAGE service, or the 4kmalc library function for System Programming C.

  For more information about this function, see 4kmalc() -- Allocate Page-Aligned Storage.
- All IBM MQ MQI calls except MQCMIT or CSQBCMT and MQBACK or CSQBBAK can be used. They must be contained after MQCONN (with a blank queue manager name). If these calls are used, the exit must be link-edited with the stub CSQXSTUB.

  The exception to this rule is that security channel exits can issue commit and backout MQI calls. To issue such calls, code the verbs CSQXCMT and CSQXBAK in place of MQCMIT or CSQBCMT and MQBACK or CSQBBAK.
- All exits that use stub CSQXSTUB from IBM WebSphere MQ 7.0 or later must be link-edited in a CSQXLIB load library with format PDS-E.
- Exits must not use any system services that cause a wait, because using system services would severely affect the handling of some or all the other channels. Many channels are run under a single TCB typically. If you do something in an exit that causes a wait and you do not use MQXWAIT, it causes all these channels to wait. Causing channels to wait does not give any functional problems, but might have an adverse effect on performance. Most SVCs involve waits, so you must avoid them, except for the following SVCs:
  - GETMAIN/FREEMAIN/STORAGE
  - LOAD/DELETE

  In general, therefore, avoid SVCs, PCs, and I/O. Instead, use the MQXWAIT call.
- Exits do not issue ESTAEs or SPIEs, apart from in any subtasks they attach, because their error handling might interfere with the error handling performed by IBM MQ. This means that IBM MQ might not be able to recover from an error, or that your exit program might not receive all the error information.
- The MQXWAIT call (see MQXWAIT) provides a wait service that waits for I/O and other events; if this service is used, exits must not use the linkage stack.

  For I/O and other facilities that do not provide non-blocking facilities or an ECB to wait on, a separate subtask must be ATTACHed, and its completion waited for by MQXWAIT; because of the processing that this technique incurs, this facility must be used only by the security exit.
- The MQDISC MQI call does not cause an implicit commit to occur within the exit program. A commit of the channel process is performed only when the channel protocol dictates.

The following exit samples are provided with IBM MQ for z/OS:

CSQ4BAX0
This sample is written in assembler, and illustrates the use of MQXWAIT.

CSQ4BCX1 and CSQ4BCX2
These samples are written in C and illustrate how to access the parameters.

CSQ4BCX3 and CSQ4BAX3
These samples are written in C and assembler respectively.

The CSQ4BCX3 sample (which is pre-compiled into the SCSQAUTH LOADLIB, should function with no changes necessary on the exit itself. You can create a LOADLIB (for example, called MY.TEST.LOAD-LIB) and copy the SCSQAUTH(CSQ4BCX3) member to it.

To set up a security exit on a client connection, carry out the following procedure:

1. Establish a valid OMVS segment for the user ID that the channel initiator uses.

This allows the IBM MQ for z/OS channel initiator to use TCP/IP with the z/OS UNIX System Services (z/OS UNIX) socket interface, in order to facilitate exit processing. Note that it is unnecessary to define an OMVS segment for the user ID of any connecting client.

2. Ensure that the exit code itself runs only in a program controlled environment.

   This means everything loaded into the CHINIT address space must be loaded from a program controlled library (meaning all libraries in the STEPLIB), and any libraries named on CSQXLIB and

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   To set a load library as program controlled, use a command similar to this example:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   Then you can activate or refresh the program controlled environment by issuing the command:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

3. Add the exit LOADLIB to the CSQXLIB DD (in the CHINIT started procedure), by issuing the following command:

   ```
   ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
   ```

   This activates the exit for the named channel.

4. Your external security manager (ESM) lists any other libraries to be program controlled, but note that none of the ESM or C libraries needs to be under program control.

See IBM MQ for z/OS server connection channel for further information on setting up a security exit using the sample CSQ4BCX3.

CSQ4BCX4
This sample is written in C and demonstrates using the **RemoteProduct** and **RemoteVersion** fields in MQCXP.

Related concepts

“Kanalexitprogramme unter IBM i schreiben” on page 1019
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für IBM i schreiben und kompilieren.

“Kanalexitprogramme unter AIX, Linux, and Windows schreiben” on page 1020
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für Systeme unter AIX, Linux, and Windows schreiben.

Related reference

IBM MQ for z/OS server connection channel

IBM i *Kanalexitprogramme unter IBM i schreiben*

Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für IBM i schreiben und kompilieren.

Ein Exit ist ein in ILE C, ILE RPG oder ILE COBOL geschriebenes Programmobjekt. Die Exitprogrammnamen und deren Bibliotheken sind in den Kanaldefinitionen benannt.

Bei der Erstellung und Kompilierung eines Exitprogramms sind die folgenden Bedingungen zu berücksichtigen:

- Das Programm muss threadsicher gemacht und mithilfe des ILE C-, ILE RPG- oder ILE COBOL-Compilers erstellt werden. Für ILE RPG muss die Steuerbestimmung THREAD(*SERIALIZE) angegeben werden. Für ILE COBOL muss für die Option THREAD der PROCESS-Anweisung SERIALIZE angegeben

werden. Zudem müssen die Programme an die IBM MQ-Threadbibliotheken gebunden sein: QMQM/LIBMQM_R im Fall von ILE C und ILE RPG bzw. AMQ0STUB_R im Fall von ILE COBOL. Weitere Informationen dazu, wie RPG- oder COBOL-Anwendungen threadsicher gemacht werden können, finden Sie im Programmierhandbuch der jeweiligen Sprache.

- Für IBM MQ for IBM i müssen die Exitprogramme für Teraspace-Support aktiviert sein. (Teraspace ist ein in OS/400 V4R4 eingeführter, gemeinsam genutzter Speicher.) Bei den ILE RPG- und COBOL-Compilern verfügen alle unter OS/400 V4R4 oder höher kompilierten Programme über diese Aktivierung. Für Programmiersprache C müssen die Programme mit den in den Befehlen CRTCMOD oder CRTBNDC angegebenen Optionen TERASPACE(*YES *TSIFC) kompiliert werden.
- Ein Exit, der einen Verweis an seinen eigenen Pufferspeicher zurückgibt, muss sicherstellen, dass das Objekt, auf das verwiesen wird, über den Zeitraum des Kanalexitprogramms hinaus vorhanden ist. Bei dem Verweis darf es sich nicht um die Adresse einer Variable im Programmstack oder Programmheapspeicher handeln. Stattdessen muss der Verweis vom System angefordert werden. Ein Beispiel hierfür ist ein im Benutzerexit erstellter Benutzeradressbereich. Um sicherzustellen, dass durch das Kanalexitprogramm zugeordnete Datenbereiche nach Beendigung des Programms weiterhin für den Nachrichtenkanalagenten verfügbar sind, muss der Kanalexit in der Aktivierungsgruppe des Anrufers oder einer benannten Aktivierungsgruppe ausgeführt werden. Legen Sie hierzu für den Parameter ACTGRP auf CRTPGM einen benutzerdefinierten Wert bzw. *CALLER fest. Wenn das Programm auf diese Weise erstellt wird, kann das Kanalexitprogramm dynamischen Speicher zuordnen und einem Verweis auf diesen Speicher an den Nachrichtenkanalagenten zurückgeben.

Zugehörige Konzepte

„Kanalexitprogramme unter AIX, Linux, and Windows schreiben” auf Seite 1020
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für Systeme unter AIX, Linux, and Windows schreiben.

„Writing channel exit programs on z/OS ” auf Seite 1017
You can use the following information to help you write and compile channel-exit programs for z/OS.

ALW *Kanalexitprogramme unter AIX, Linux, and Windows schreiben*

Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für Systeme unter AIX, Linux, and Windows schreiben.

Folgen Sie den unter „Exits und installierbare Services unter AIX, Linux, and Windows schreiben” auf Seite 986 beschriebenen Anweisungen. Berücksichtigen Sie ggf. die folgenden kanalexitspezifischen Informationen:

Der Exit ist eine DLL-Datei unter Windows und muss in der Programmiersprache C geschrieben werden..

Definieren Sie im Exit eine 'MQStart()'-Dummyroutine und geben Sie `MQStart` in der Bibliothek als Eingangspunkt an. In Abbildung 106 auf Seite 1020 wird die Vorgehensweise beim Einrichten eines Eingangs in Ihr Programm veranschaulicht:

```
#include <cmqec.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
... Insert code here
}
```

Abbildung 106. Beispielquellcode für einen Kanalexit

Beim Schreiben von Kanalexits für Windows unter Verwendung von Visual C++ müssen Sie Ihre eigene DEF-Datei schreiben. Wie Sie dies tun können, erfahren Sie in Abbildung 107 auf Seite 1021. Weitere

Informationen zum Schreiben von Kanalexitprogrammen finden Sie im Abschnitt „Kanalexitprogramme schreiben” auf Seite 1016.

```
EXPORTS
ChannelExit
```

Abbildung 107. DEF-Musterdatei für Windows

Zugehörige Konzepte

„Kanalexitprogramme unter IBM i schreiben” auf Seite 1019
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für IBM i schreiben und kompilieren.

„Writing channel exit programs on z/OS ” auf Seite 1017
You can use the following information to help you write and compile channel-exit programs for z/OS.

Sicherheitsexitprogramme für Kanäle
Sie können Sicherheitsexitprogramme verwenden, um sicherzustellen, dass der Partner am anderen Ende eines Kanals echt ist. Dieser Vorgang wird als Authentifizierung bezeichnet.

Um anzugeben, dass ein Kanal einen Sicherheitsexit verwenden muss, geben Sie den Exitnamen im Feld **SCYEXIT** der Kanaldefinition an.

Anmerkung: Eine Authentifizierung lässt sich auch mit den Datensätzen der Kanalauthentifizierung erreichen. Kanalauthentifizierungsdatensätze bieten eine große Flexibilität, wenn bestimmten Benutzern und Kanälen der Zugriff auf Warteschlangenmanager verweigert bzw. fernen Benutzern IBM MQ-Benutzer-IDs zugeordnet werden sollen. Die TLS-Unterstützung wird auch durch IBM MQ bereitgestellt, um die Benutzer zu authentifizieren und eine Verschlüsselung und Datenintegritätsprüfungen für Ihre Daten zu ermöglichen. Weitere Informationen zu SSL und TLS finden Sie im Abschnitt TLS-Sicherheitsprotokoll in IBM MQ. Wenn Sie jedoch eine noch ausgereiftere oder andere Form von Sicherheitsverarbeitung sowie andere Arten der Überprüfung und Einrichtung des Sicherheitskontexts benötigen, sollten Sie eigene Sicherheitsexits schreiben.

Die Attribute 'Registrierter Name des Zertifikatsinhabers' und 'Registrierter Name des Zertifikatsausstellers' sind in folgenden Kanalstatusattributen enthalten:

- SSLPEER (PCF selector MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (PCF selector MQCACH_SSL_CERT_ISSUER_NAME)

Diese Werte werden von Kanalstatusbefehlen zurückgegeben, ebenso wie die Daten, die an die nachfolgend aufgelisteten Kanalsicherheitsexits übergeben wurden:

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

Ein Sicherheitsexit kann in der Programmiersprache C oder Java geschrieben werden.

Kanalsicherheitsexits werden im Verarbeitungszyklus eines Nachrichtenkanalagenten an folgenden Stellen aufgerufen:

- Bei der Initialisierung und Beendigung eines Nachrichtenkanalagenten.
- Unmittelbar nach Abschluss der anfänglichen Datenvereinbarung beim Start des Kanals. Die Empfänger- oder Serverseite des Kanals kann einen Sicherheitsnachrichtenaustausch mit der fernen Seite einleiten, indem dem Sicherheitsexit auf der fernen Seite eine Nachricht zur Zustellung bereitgestellt wird. Sie kann dies jedoch auch ablehnen. Das Exitprogramm wird erneut gestartet, um die von der fernen Seite empfangene Sicherheitsnachricht zu verarbeiten.
- Unmittelbar nach Abschluss der anfänglichen Datenvereinbarung beim Start des Kanals. Die sendende oder anfordernde Seite des Kanals verarbeitet eine Sicherheitsnachricht, die von der fernen Seite empfangen wird, oder leitet einen Sicherheitsaustausch ein, wenn die ferne Seite dazu nicht in der Lage ist. Das Exitprogramm wird erneut gestartet, um alle eventuell empfangenen nachfolgenden Sicherheitsnachrichten zu verarbeiten.

Ein Requesterkanal wird niemals mit MQXR_INIT_SEC aufgerufen. Der Kanal benachrichtigt den Server darüber, dass er über ein Sicherheitsexitprogramm verfügt, und der Server kann dann einen Sicherheitsexit einleiten. Ist kein Programm vorhanden, wird der Requester entsprechend informiert und ein Datenfluss der Länge null an das Exitprogramm zurückgegeben.

Anmerkung: Vermeiden Sie das Senden von Sicherheitsnachrichten der Länge 'Null'.

Beispiele für durch Sicherheitsexitprogramme ausgetauschte Daten sind in den Abbildungen Abbildung 108 auf Seite 1022bis Abbildung 111 auf Seite 1024 aufgeführt. Diese Beispiele zeigen die Ereignisfolge im Zusammenhang mit dem Sicherheitsexit des Empfängers sowie mit dem Sicherheitsexit des Senders. Die aufeinanderfolgenden Zeilen in den Abbildungen repräsentieren den Zeitablauf. In einigen Fällen sind die Ereignisse beim Empfänger nicht mit denen beim Sender korreliert und können daher gleichzeitig oder auch zu unterschiedlichen Zeiten stattfinden. In anderen Fällen führt ein Ereignis bei dem einen Exitprogramm zu einem komplementären späteren Ereignis beim anderen Exitprogramm. Beispielsweise in Abbildung 108 auf Seite 1022:

1. Der Empfänger und der Sender werden mit MQXR_INIT aufgerufen, aber diese Aufrufe sind nicht korreliert und können daher gleichzeitig oder auch zu unterschiedlichen Zeiten stattfinden.
2. Der Empfänger wird dann mit MQXR_INIT_SEC aufgerufen, gibt jedoch MQXCC_OK zurück, für das beim Senderexit kein zusätzliches Ereignis erforderlich ist.
3. Der Sender wird dann mit MQXR_INIT_SEC aufgerufen. Dabei besteht keine Korrelation mit dem Aufruf des Empfängers mit MQXR_INIT_SEC. Der Sender gibt MQXCC_SEND_SEC_MSG zurück, was beim Empfängerexit zu einem zusätzlichen Ereignis führt.
4. Der Empfänger wird dann mit MQXR_SEC_MSG aufgerufen und gibt MQXCC_SEND_SEC_MSG zurück, was beim Senderexit zu einem zusätzlichen Ereignis führt.
5. Der Sender wird dann mit MQXR_SEC_MSG aufgerufen und gibt MQXCC_OK zurück, für das beim Empfängerexit kein komplementäres Ereignis erforderlich ist.

Receiver exit	**Sender exit**
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Message transfer begins	

Abbildung 108. Vom Sender eingeleiteter Austausch mit Vereinbarung

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Abbildung 109. Vom Sender eingeleiteter Austausch ohne Vereinbarung

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Abbildung 110. Vom Empfänger eingeleiteter Austausch mit Vereinbarung

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

Abbildung 111. Vom Empfänger eingeleiteter Austausch ohne Vereinbarung

Dem Kanalsicherheitsexitprogramm wird ein Agentenpuffer übergeben, der die vom Sicherheitsexit generierten Sicherheitsdaten enthält (ohne Übertragungsheader). Da es sich bei diesen Daten um beliebige geeignete Daten handeln kann, kann jede Kanalseite die Sicherheitsprüfung vornehmen.

Das Sicherheitsexitprogramm auf der sendenden und empfangenden Seite des Nachrichtenkanals kann für jeden Aufruf einen von zwei Antwortcodes zurückgeben:

- Der Sicherheitsaustausch wurde ohne Fehler beendet
- Der Kanal wird unterdrückt und geschlossen

Anmerkung:

1. Für gewöhnlich agieren die Kanalsicherheitsexits paarweise. Wenn Sie die entsprechenden Kanäle definieren, vergewissern Sie sich, dass für beide Kanalseiten kompatible Exitprogramme benannt werden.
2. IBM i In IBM i können Sicherheitsexitprogramme, die mit `Use adopted authority` (USEADPAUT=*YES) kompiliert wurden, über die Berechtigung QMQM oder QMQMADM verfügen. Achten Sie darauf, dass der Exit diese Funktion nicht verwendet, da dies ein Sicherheitsrisiko für Ihr System darstellen kann.
3. In einem TLS-Kanal, bei dem die andere Kanalseite ein Zertifikat bereitstellt, empfängt der Sicherheitsexit den definierten Namen des Gegenstands dieses Zertifikats im MQCD-Feld, auf das durch 'SSLPeerNamePtr' zugegriffen wird, und den definierten Namen des Ausstellers im MQCXP-Feld, auf das durch 'SSLRemCertIssNamePtr' zugegriffen wird. Dieser Name kann für folgende Zwecke eingesetzt werden:
   - Zur Beschränkung des Zugriffs über den TLS-Kanal.
   - Zur Änderung von MQCD.MCAUserIdentifier auf Basis des Namens.

Zugehörige Konzepte

Konzepte der Transport Layer Security (TLS)

Zugehörige Verweise

Kanalauthentifizierungsdatensätze

Sicherheitsexit schreiben

Sie können den Sicherheitsexit-Entwurfscode zum Schreiben von Sicherheitsexits verwenden.

Abbildung 112 auf Seite 1025 veranschaulicht, wie ein Sicherheitsexit geschrieben wird.

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

Abbildung 112. Sicherheitsexit-Entwurfscode

Der IBM MQ-Standardeingangspunkt 'MQStart' muss vorhanden sein, das Ausführen von Aufgaben ist jedoch nicht erforderlich. Die Änderung des Funktionsnamens ('EntryPoint' im vorliegenden Beispiel) ist möglich, die Funktion muss jedoch bei der Kompilierung und Verbindung der Bibliothek exportiert werden. Wie im vorherigen Beispiel muss 'pChannelExitParms' auf PMQCXP und 'pChannelDefinition' auf PMQCD verweisen. Allgemeine Informationen zum Aufruf von Kanalexits und der Verwendung von Parametern finden Sie im Abschnitt MQ_CHANNEL_EXIT. Diese Parameter werden wie folgt in einem Sicherheitsexit verwendet:

PMQVOID pChannelExitParms
Eingabe/Ausgabe

Zeiger zur MQCXP-Struktur - Verweis auf PMQCXP zum Zugriff auf Felder. Diese Struktur wird zur Kommunikation zwischen dem Exit und dem Nachrichtenkanalagenten verwendet. Die folgenden Felder in MQCXP sind in Bezug auf Sicherheitsexits von besonderem Interesse:

ExitReason
Informiert den Sicherheitsexit über den aktuellen Status des Sicherheitsaustauschs und wird verwendet, wenn über die zu ergreifende Maßnahme entschieden werden soll.

ExitResponse
Die Antwort auf den Nachrichtenkanalagenten, die den nächsten Schritt im Sicherheitsaustausch festlegt.

ExitResponse2
Zusätzliche Steuermarkierungen, die regeln, wie der Nachrichtenkanalagent die Antwort des Sicherheitsexits interpretiert.

ExitUserArea
16 Byte Speicher (maximal), der vom Sicherheitsexit zur Beibehaltung des Status zwischen Anrufen verwendet werden kann.

ExitData
Enthält die im Feld SCYDATA der Kanaldefinition angegebenen Daten (32 Bytes, die rechts durch Leerzeichen aufgefüllt sind).

PMQVOID pChannelDefinition
Eingabe/Ausgabe

Zeiger zur MQCD-Struktur - Verweis auf PMQCD zum Zugriff auf Felder. Dieser Parameter enthält die Kanaldefinition. Die folgenden Felder in der MQCD sind in Bezug auf Sicherheitsexits von besonderem Interesse:

ChannelName
Der Kanalname (20 Bytes, die rechts durch Leerzeichen aufgefüllt sind).

ChannelType
Ein Code zur Definition des Kanaltyps.

MCA User Identifier
Diese aus drei Feldern bestehende Gruppe wird mit dem Wert des in der Kanaldefinition angegebenen MCAUSER-Felds initialisiert. Durch den Sicherheitsexit in diesen Feldern angegebene Benutzer-IDs werden zur Zugriffssteuerung verwendet (nicht zutreffend für SDR-, SVR-, CLNTCONN- oder CLUSSDR-Kanäle).

MCAUserIdentifier
Die ersten 12 Bytes der Kennung, die rechts durch Leerzeichnen aufgefüllt sind.

LongMCAUserIdPtr
Zeiger zu einem Puffer, der die Kennung in gesamter Länge enthält (Nullabschluss nicht garantiert). Hat Vorrang vor 'MCAUserIdentifier'.

LongMCAUserIdLength
Zeichenfolge, auf die durch 'LongMCAUserIdPtr' verwiesen wird - muss festgelegt werden, wenn 'LongMCAUserIdPtr' festgelegt ist.

Remote User Identifier
Findet nur bei CLNTCONN/SVRCONN-Kanalpaaren Anwendung. Wenn kein CLNTCONN-Sicherheitsexit definiert ist, werden diese drei Felder durch den Client-Nachrichtenkanalagenten initialisiert und können eine Benutzer-ID aus der Clientumgebung enthalten, die durch einen SVRCONN-Sicherheitsexit zur Authentifizierung oder zur Angabe der Benutzer-ID des Nachrichtenkanalagenten verwendet werden kann. Wenn ein CLNTCONN-Sicherheitexit definiert ist, werden diese drei Felder nicht initialisiert und können durch den CLNTCONN-Sicherheitsexit festgelegt werden bzw. können Sicherheitsnachrichten dazu verwendet werden, eine Benutzer-ID vom Client zum Server zu senden.

Remote User Identifier
Die ersten 12 Bytes der Kennung, die rechts durch Leerzeichnen aufgefüllt sind.

LongRemoteUserIdPtr
Zeiger zu einem Puffer, der die Kennung in gesamter Länge enthält (Nullabschluss nicht garantiert). Hat Vorrang vor 'RemoteUserIdentifier'.

LongRemoteUserIdLength
Zeichenfolge, auf die durch 'LongRemoteUserIdPtr' verwiesen wird - muss festgelegt werden, wenn 'LongRemoteUserIdPtr' festgelegt ist.

PMQLONG pDataLength
Eingabe/Ausgabe

Zeiger zu MQLONG. Enthält die Länge der beim Aufruf des Sicherheitsexits in 'AgentBuffer' enthaltenen Sicherheitsexits. Muss durch einen Sicherheitsexit auf die Länge einer in 'AgentBuffer' oder 'ExitBuffer' gesendeten Nachricht festgelegt werden.

PMQLONG pAgentBufferLength
Eingabe

Zeiger zu MQLONG. Die Länge der beim Aufruf des Sicherheitsexits in 'AgentBuffer' enthaltenen Daten.

PMQVOID pAgentBuffer
Eingabe/Ausgabe

Beim Aufruf des Sicherheitsexits verweist dieser Parameter auf vom Partnerexit gesendete Nachrichten. Wenn bei 'ExitResponse2' in der MQCXP-Struktur das Flag MQXR2_USE_AGENT_BUFFER gesetzt ist (Standardeinstellung), muss dieser Parameter durch einen Sicherheitsexit so gesetzt werden, dass er auf die gesendeten Nachrichtendaten verweist.

PMQLONG pExitBufferLength
Eingabe/Ausgabe

Zeiger zu MQLONG. Dieser Parameter wird beim ersten Aufruf eines Sicherheitsexits auf 0 gesetzt und der zurückgegebene Wert wird zwischen Aufrufen des Sicherheitsexits während eines Sicherheitsaustauschs beibehalten.

PMQPTR pExitBufferAddr
Eingabe/Ausgabe

Dieser Parameter wird beim ersten Aufruf eines Sicherheitsexits auf einen Nullzeiger gesetzt und der zurückgegebene Wert wird zwischen Aufrufen des Sicherheitsexits während eines Sicherheitsaustauschs beibehalten. Wenn das Flag MQXR2_USE_EXIT_BUFFER in 'ExitResponse2' in der MQCXP-Structure gesetzt ist, muss dieser Parameter durch einen Sicherheitsexit so gesetzt werden, dass er auf die gesendeten Nachrichtendaten verweist.

Unterschiede im Verhalten von auf CLNTCONN/SVRCONN-Kanalpaaren und anderen Kanalpaaren definierten Sicherheitsexits
Sicherheitsexits können auf allen Kanaltypen definiert werden. Allerdings unterscheidet sich das Verhalten von auf CLNTCONN/SVRCONN-Kanalpaaren definierten Sicherheitsexits geringfügig von dem von Sicherheitsexits, die auf anderen Kanalpaaren definiert wurden.

Ein Sicherheitsexit auf einem CLNTCONN-Kanal kann die Remotebenutzer-ID in der Kanaldefinition so festlegen, dass die Verarbeitung durch einen SVRCONN-Partnerexit durchgeführt wird bzw. eine OAM-Autorisierung erfolgt, wenn kein SVRCONN-Sicherheitsexit definiert und das MCAUSER-Feld von SVRCONN nicht festgelegt ist.

Wenn kein CLNTCONN-Sicherheitsexit definiert ist, wird die Remotebenutzer-ID durch den Client-Nachrichtenkanalagenten in der Kanaldefinition auf eine Benutzer-ID aus der Clientumgebung gesetzt (die leer sein kann).

Ein Sicherheitsaustausch zwischen Sicherheitsexits auf einem CLNTCONN-und-SVRCONN-Kanalpaar wird erfolgreich abgeschlossen, wenn der SVRCONN-Sicherheitsexit als 'ExitResponse' MQXCC_OK zurückgibt.

Ein Sicherheitsaustausch zwischen anderen Kanalpaaren wird erfolgreich abgeschlossen, wenn der Sicherheitsexit, der den Austausch initiiert hat, als 'ExitResponse' MQXCC_OK zurückgibt.

Mit dem 'ExitResponse'-Code MQXCC_SEND_AND_REQUEST_SEC_MSG kann jedoch die Fortsetzung des Sicherheitsaustauschs erzwungen werden: Wenn im Feld 'ExitResponse' durch den Sicherheitsexit CLNTCONN oder SVRCONN MQXCC_SEND_AND_REQUEST_SEC_MSG zurückgegeben wird, muss der Partnerexit durch das Senden einer Sicherheitsnachricht (nicht MQXCC_OK oder eine Nullantwort) reagieren, um die Beendigung des Kanals zu verhindern. Bei auf anderen Kanälen definierten Sicherheitsexits resultiert die Rückgabe von MQXCC_OK im Feld 'ExitResponse' auf die Anfrage MQXCC_SEND_AND_REQUEST_SEC_MSG des Partnersicherheitsexits nicht in der Beendigung des Kanals, sondern in der Fortsetzung des Sicherheitsaustauschs, als wäre eine Nullantwort zurückgegeben worden.

SSPI-Sicherheitsexit
IBM MQ for Windows stellt einen Sicherheitsexit bereit, der die Programmierschnittstelle für Sicherheitsservices (Security Services Programming Interface, SSPI) dazu verwendet, Authentifizierung für IBM MQ zur Verfügung zu stellen. Die SSPI stellt die integrierten Sicherheitsfunktionen von Windows bereit.

Dieser Sicherheitsexit kann auf dem IBM MQ-Client und dem IBM MQ-Server eingesetzt werden.

Die Sicherheitspakete werden aus der Datei 'security.dll' oder 'secur32.dll' geladen. Diese DLLs werden mit Ihrem Betriebssystem geliefert.

Unidirektionale Authentifizierung wird über NTLM-Authentifizierungsservices unter Windows bereitgestellt. Zweiwegeauthentifizierung wird über Kerberos-Authentifizierungsservices unter Windows 2000 bereitgestellt.

Das Sicherheitsexitprogramm ist im Quellen- und Objektformat vorhanden. Sie können den Objektcode unverändert nutzen oder den Quellcode als Ausgangspunkt zur Erstellung Ihres eigenen Benutzerexitprogramms verwenden. Weitere Informationen zur Verwendung des Objekt- oder Quellcodes des SSPI-Sicherheitsexits finden Sie im Abschnitt „SSPI-Sicherheitsexit unter Windows verwenden” auf Seite 1194

Kanalsende- und -empfangsexitprogramme
Über die Sende- und Empfangsexits können Sie Aufgaben wie Datenkomprimierung und -dekomprimierung durchführen. Sie können eine Liste von Sende- und Empfangsexitsprogrammen angeben, die nacheinander ausgeführt werden sollen.

Kanalsende- und -empfangsexitprogramme werden im Verarbeitungszyklus eines Nachrichtenkanalagenten an folgenden Stellen aufgerufen:

- Die Sende- und Empfangsexitprogramme werden beim Start des Nachrichtenkanalagenten zur Initialisierung und bei der Beendigung des Nachrichtenkanalagenten zur Beendigung aufgerufen.
- Das Sendeexitprogramm wird auf einer der beiden Kanalseiten aufgerufen, abhängig davon, von welcher Seite unmittelbar vor dem Senden einer Übertragung über den Link ein Übermittlungsaufruf zur Übertragung einer Nachricht gesendet wird. In Anmerkung 4 wird erläutert, weshalb Exits in beide Richtungen verfügbar sind, obwohl das Versenden von Nachrichten über Nachrichtenkanäle nur in einer Richtung erfolgt.
- Das Empfangsexitprogramm wird auf einer der beiden Kanalseiten aufgerufen, abhängig davon, von welcher Seite unmittelbar nach dem Abrufen einer Übertragung aus dem Link ein Übermittlungsaufruf zur Übertragung einer Nachricht empfangen wird. In Anmerkung 4 wird erläutert, weshalb Exits in beide Richtungen verfügbar sind, obwohl das Versenden von Nachrichten über Nachrichtenkanäle nur in einer Richtung erfolgt.

Zur Übertragung einer Nachricht können mehrere Übermittlungsaufrufe erforderlich sein und es können viele Iterationen der Sende- und Empfangsexitprogramme stattfinden, bevor eine Nachricht den Nachrichtenexit auf der Empfangsseite erreicht.

Den Kanalsende- und -empfangsexitprogrammen wird ein Agentenpuffer übergeben, der die über die Kommunikationsverbindung gesendeten bzw. empfangenen Übertragungsdaten enthält. Bei Sendeexitprogrammen ist die Verwendung der ersten 8 Bytes des Puffers ausschließlich den Nachrichtenkanalagenten vorbehalten, die daher nicht verändert werden dürfen. Wenn das Programm einen anderen Puffer zurückgibt, müssen diese ersten 8 Bytes im neuen Puffer vorhanden sein. Das Format der an die Exitprogramme übergebenen Daten ist nicht definiert.

Von Sende- und Empfangsexitprogrammen muss ein intakter Antwortcode zurückgegeben werden. Andere Antworten führen zur abnormalen Beendigung (Abbruch) des Nachrichtenkanalagenten.

Anmerkung: Geben Sie keinen MQGET-, MQPUT- oder MQPUT1-Aufruf von einem Sende- oder Empfangsexit unter einem Synchronisationspunkt aus.

Anmerkung:

1. Sende-und Empfangsexits funktionieren in der Regel paarweise. Daten werden beispielsweise von einem Sendeexit komprimiert und von einem Empfangsexit dekomprimiert bzw. von einem Sendeexit verschlüsselt und von einem Empfangsexit wieder entschlüsselt. Wenn Sie die entsprechenden Kanäle definieren, vergewissern Sie sich, dass für beide Kanalseiten kompatible Exitprogramme benannt werden.
2. Wenn Komprimierung für den Kanal aktiviert ist, werden den Exits komprimierte Daten übergeben.
3. Kanalsende- und -empfangsexits können neben Anwendungsdaten auch für andere Nachrichtensegmente wie beispielsweise Statusnachrichten aufgerufen werden. Während des Startdialogs und der Sicherheitsprüfung werden sie nicht aufgerufen.
4. Obgleich Nachrichtenkanäle Nachrichten nur in einer Richtung versenden, fließen Kanalsteuerdaten wie Heartbeats und die Verarbeitung von Stapelenden in beide Richtungen, daher sind diese Exits auch in beiden Richtungen verfügbar. Einige der ursprünglichen Kanalstart-Datenflüsse sind jedoch von der Verarbeitung durch diese Exits ausgeschlossen.
5. Unter bestimmten Umständen können Sende- und Empfangsexits in falscher Reihenfolge aufgerufen werden, beispielsweise wenn Sie mehrere Exitprogramme bzw. auch Sicherheitsexits ausführen. Wenn der Empfangsexit dann zuerst zur Verarbeitung von Daten aufgerufen wird, kann er unter Umständen Daten empfangen, die nicht durch den entsprechenden Sendeexit übergeben wurden. Wenn der Empfangsexit einen Vorgang (z.B. eine Dekomprimierung) ausführt, ohne zuvor zu prüfen, ob er erforderlich ist, können unerwartete Ergebnisse auftreten.

   Sie müssen Ihren Sende- und Empfangsexit so codieren, dass der Empfangsexit prüfen kann, ob die empfangenen Daten durch den entsprechenden Sendeexit verarbeitet wurden. Es wird empfohlen, Exitprogramme so zu codieren, dass folgende Voraussetzungen erfüllt sind:

   - Der Sendeexit setzt den Wert des neunten Datenbyte auf 0 und verschiebt die gesamten Daten um 1 Byte, bevor der Vorgang ausgeführt wird. (Die Verwendung der ersten 8 Bytes ist ausschließlich den Nachrichtenkanalagenten vorbehalten.)
   - Wenn der Empfangsexit Daten empfängt, die in Byte 9 über eine 0 verfügen, weiß er, dass die Daten vom Sendeexit stammen. Er entfernt die 0, führt den komplementären Vorgang aus und verschiebt die resultierenden Daten um 1 Byte zurück.
   - Wenn der Empfangsexit Daten empfängt, die in Byte 9 nicht über eine 0 verfügen, setzt er voraus, dass der Sendeexit nicht ausgeführt wurde und sendet die Daten unverändert an das aufrufende Programm zurück.

   Wenn der Kanal bei Verwendung von Sicherheitsexits durch den Sicherheitsexit beendet wird, kann ein Sendeexit unter Umständen ohne den entsprechenden Empfangsexit aufgerufen werden. Dieses Problem kann u.a. dadurch vermieden werden, dass der Sicherheitsexit so codiert wird, dass er beispielsweise ein Attribut 'inMQCD.SecurityUserData' oder 'MQCD.SendUserData' angibt, wenn sich der Exit zur Beendigung des Kanals entscheidet. Der Sendeexit muss dieses Feld dann prüfen und verarbeitet die Daten nur, wenn kein Attribut angegeben ist. Diese Prüfung verhindert eine unnötige Änderung der Daten durch den Sendeexit und vermeidet somit Konvertierungsfehler, die beim Empfang veränderter Daten durch den Sicherheitsexit auftreten könnten.

Kanalsendeexitprogramme - Speicherplatz reservieren
Sie können Sende- und Empfangsexits dazu nutzen, Daten vor der Übertragung zu transformieren. Kanalsendeexitprogramme können eigene Daten über die Transformation hinzufügen, indem sie Speicherplatz im Übertragungspuffer reservieren.

Diese Daten werden durch das Empfangsexitprogramm verarbeitet und dann aus dem Puffer gelöscht. Sie können die Daten beispielsweise verschlüsseln und einen Sicherheitsschlüssel zur Entschlüsselung hinzufügen.

Vorgehensweise bei der Reservierung und Nutzung von Speicherplatz

Legen Sie beim Aufruf des Sendeexitprogramms zur Initialisierung im Feld *ExitSpace* von MQXCP die Anzahl der Bytes fest, die reserviert werden sollen. Details finden Sie im Abschnitt MQCXP. *ExitSpace* kann nur bei der Initialisierung festgelegt werden. Das heißt, wenn *ExitReason* den Wert MQXR_INIT hat. Wenn *ExitReason* auf MQXR_XMIT gesetzt ist, wird beim Aufruf des Sendeexits unmittelbar vor der Übertragung die im Feld *ExitSpace* angegebene Bytezahl im Übertragungspuffer reserviert. *ExitSpace* wird unter z/OS nicht unterstützt.

Der Sendeexit muss nicht den gesamten reservierten Speicherplatz verwenden. Er kann weniger als die im Feld *ExitSpace* angegebene Anzahl an Bytes oder bei einem nicht vollständig belegten Übertragungspuffer auch mehr als die reservierte Byteanzahl verwenden. Beim Festlegen des Werts für *ExitSpace* muss mindestens 1 KB im Übertragungspuffer für Nachrichtendaten freigelassen werden. Die Kanalleistung kann beeinträchtigt werden, wenn reservierter Speicherplatz für große Datenmengen verwendet wird.

Der Übertragungspuffer ist normalerweise 32KB lang. Wenn der Kanal jedoch TLS verwendet, wird die Übertragungspuffergröße auf 15.352 Byte reduziert, um der durch RFC 6101 und die zugehörige TLS-Standardfamilie definierten maximalen Satzlänge zu entsprechen. Weitere 1.024 Byte sind für die Verwendung durch IBM MQ reserviert, also beträgt die maximale, von Sendeexits verwendbare Übertragungspuffergröße 14.328 Byte.

Was geschieht auf der Empfangsseite des Kanals?

Kanalempfangsexitprogramme müssen so konfiguriert sein, dass sie mit den entsprechenden Sendeexits kompatibel sind. Empfangsexits müssen die Anzahl der Bytes des reservierten Speicherplatzes kennen und die Daten aus diesem Bereich entfernen.

Mehrere Sendeexits

Sie können eine Liste von Sende- und Empfangsexitsprogrammen angeben, die nacheinander ausgeführt werden sollen. IBM MQ hält einen Gesamtspeicherplatz für die durch alle Sendeexits reservierten Bereiche bereit. Mindestens 1 KB dieses Gesamtspeicherbereichs im Übertragungspuffer muss für Nachrichtendaten freigehalten werden .

Das folgende Beispiel veranschaulicht, wie drei nacheinander aufgerufenen Sendeexits Speicherplatz zugeordnet wird:

1. Beim Aufruf zur Initialisierung:
   - Für Sendeexit A wird 1 KB Speicherplatz reserviert.
   - Für Sendeexit B werden 2 KB Speicherplatz reserviert.
   - Für Sendeexit C werden 3 KB Speicherplatz reserviert.
2. Die maximale Übertragungsgröße beträgt 32 KB und die Benutzerdaten sind 5 KB lang.
3. Exit A wird mit 5 KB Daten aufgerufen; bis zu 27 KB sind verfügbar, da 5 KB für die Exits B und C reserviert sind. Exit A fügt 1 KB hinzu (den durch den Exit reservierten Speicherplatz).
4. Exit B wird mit 6 KB Daten aufgerufen; bis zu 29 KB sind verfügbar, da 3 KB für den Exit C reserviert sind. Exit B fügt 1 KB hinzu (weniger als den durch den Exit reservierten Speicherplatz von 2 KB).
5. Exit C wird mit 7 KB Daten aufgerufen; bis zu 32 KB sind verfügbar. Exit C fügt 10 KB hinzu (mehr als den durch den Exit reservierten Speicherplatz von 3 KB). Diese Datenmenge ist gültig, da die Gesamtdatenmenge von 17 KB unter dem maximal verfügbaren Speicherplatz von 32 KB liegt.

Die maximale Übertragungspuffergröße für einen Kanal, der TLS verwendet, beträgt 15.352 Byte, nicht 32 KB. Dies liegt daran, dass die zugrunde liegenden Übertragungssegmente für sichere Sockets auf 16 KB begrenzt sind und ein Teil des Speicherplatzes für TLS-Satzmehraufwand benötigt wird. Weitere 1.024 Byte sind für die Verwendung durch IBM MQ reserviert, also beträgt die maximale, von Sendeexits verwendbare Übertragungspuffergröße 14.328 Byte.

Kanalnachrichtenexitprogramme
Mithilfe des Kanalnachrichtenexits können Aufgaben wie die Verschlüsselung der Verbindung, die Validierung oder Ersetzung eingehender Benutzer-IDs, die Konvertierung von Nachrichtendaten, Journaling und die Behandlung von Referenznachrichten ausgeführt werden. Sie können eine Liste von Nachrichtenexitsprogrammen angeben, die nacheinander ausgeführt werden sollen.

Kanalnachrichtenexits werden im Verarbeitungszyklus des Nachrichtenkanalagenten an folgenden Stellen aufgerufen:

- Bei der Initialisierung und Beendigung des Nachrichtenkanalagenten
- Unmittelbar nach der Ausgabe eines MQGET-Aufrufs durch einen sendenden Nachrichtenkanalagenten
- Vor der Ausgabe eines MQPUT-Aufrufs durch den empfangenden Nachrichtenkanalagenten

Dem Nachrichtenexit wird ein Agentenpuffer übergeben, der den Header der Übertragungswarteschlange MQXQH und den Text der Anwendungsnachricht enthält, wie er von der Warteschlange abgerufen wurde. Informationen zum Format von MQXQH finden Sie im Abschnitt MQXQH - Übertragungswarteschlangenheader.

Multi Wenn Sie Referenznachrichten verwenden (d.h. Nachrichten, die nur einen Header enthalten, der auf ein zu sendendes anderes Objekt verweist), erkennt der Nachrichtenexit den Header MQRMH. Es identifiziert das Objekt, ruft es auf die geeignete Weise ab, hängt es an den Header an und übergibt es dann zur Übertragung an den empfangenden MCA. Beim empfangenden Nachrichtenkanalagenten erkennt ein anderer Nachrichtenexit, dass es sich bei der Nachricht um eine Referenznachricht handelt. Er extrahiert das Objekt und übergibt den Header an die Zielwarteschlange. Weitere Informationen zu Referenznachrichten und Beispiele zu deren Bearbeitung durch Nachrichtenexits finden Sie in den Abschnitten „Referenznachrichten und Übertragungen großer Objekte" auf Seite 836 und „Referenznachrichtenbeispiel ausführen" auf Seite 1164.

Nachrichtenexits können die folgenden Antworten zurückgeben:

- Nachricht senden (GET-Exit). Die Nachricht wurde möglicherweise durch den Exit geändert. (In diesem Fall wird MQXCC_OK zurückgegeben.)
- Nachricht in die Warteschlange einreihen (PUT-Exit). Die Nachricht wurde möglicherweise durch den Exit geändert. (In diesem Fall wird MQXCC_OK zurückgegeben.)
- Nachricht nicht verarbeiten. Die Nachricht wird durch den Nachrichtenkanalagenten in die Warteschlange für nicht zustellbare Nachrichten (nicht zugestellte Nachrichten) gestellt.
- Kanal schließen.
- Ungültiger Rückgabecode, der eine abnormale Beendigung des Nachrichtenkanalagenten zur Folge hat.

Anmerkung:

1. Nachrichtenexits werden einmal pro vollständiger Nachrichtenübertragung aufgerufen, auch wenn die Nachricht in mehrere Komponenten aufgeteilt ist.
2. Linux AIX Wenn Sie unter AIX oder Linux einen Nachrichtenexit bereitstellen, funktioniert die automatische Konvertierung von Benutzer-IDs in Kleinbuchstaben nicht (wie hier beschrieben).
3. Ein Exit wird in demselben Thread wie der Nachrichtenkanalagent selbst ausgeführt. Er wird zudem in derselben Arbeitseinheit (UOW) wie der Nachrichtenkanalagent ausgeführt, da er dieselbe Verbindungskennung verwendet. Daher werden sämtliche Aufrufe, die unter einem Synchronisationspunkt getätigt werden, vom Kanal am Ende des Stapels festgeschrieben oder zurückgesetzt. Beispielsweise kann ein Kanalnachrichtenexitprogramm Benachrichtigungsnachrichten an ein anderes senden und diese Nachrichten werden nur dann an die Warteschlange übergeben, wenn der Stapel, der die ursprüngliche Nachricht enthält, festgeschrieben wird.

   Die Ausgabe von MQI-Aufrufen unter Synchronisationspunktsteuerung über einen Kanalnachrichtenexit ist daher möglich.

Nachrichtenkonvertierung außerhalb des Nachrichtenexits
Vor dem Aufruf des Nachrichtenexits führt der empfangende Nachrichtenkanalagent einige Konvertierungen an der Nachricht durch. In diesem Abschnitt werden die zur Durchführung dieser Konvertierung verwendeten Algorithmen beschrieben.

Verarbeitete Header

Vor dem Aufruf des Nachrichtenexits wird im Nachrichtenkanalagenten des Empfängers eine Konvertierungsroutine ausgeführt. Die Konvertierungsroutine beginnt mit dem MQXQH-Header am Anfang der Nachricht. Die Konvertierungsroutine arbeitet sich dann durch die verketteten, auf MQXQH folgenden Header und führt bei Bedarf Konvertierungen durch. Die verketteten Header können die im Parameter 'HeaderLength' der MQCXP-Daten, die an den Nachrichtenexit des Empfängers übergeben werden, definierte Länge überschreiten. Die folgenden Header werden direkt konvertiert:

- MQXQH (Formatname " `MQXMIT` ")
- MQMD (dieser Header ist Teil von MQXQH und besitzt keinen Formatnamen)
- MQMDE (Formatname " `MQHMDE` ")
- MQDH (Formatname " `MQHDIST` ")
- MQWIH (Formatname " `MQHWIH` ")

Die folgenden Header werden nicht konvertiert, sondern bei der weiteren Verarbeitung der verketteten Header durch den Nachrichenkanalagenten übersprungen:

- MQDLH (Formatname " `MQDEAD` ")
- Alle Header, deren Formatnamen mit den drei Buchstaben 'MQH' beginnen (beispielsweise " `MQHRF` "), und die nicht anderweitig erwähnt wurden.

Vorgehensweise bei der Verarbeitung von Headern

Der Parameter für Formatangabe der einzelnen IBM MQ-Header wird durch den Nachrichtenkanalagenten gelesen. Der Parameter für Formatangabe im Header verfügt über eine Länge von 8 Bytes, bei denen es sich um einen Namen aus 8 Einzelbytezeichen handelt.

Der Nachrichtenkanalagent interpretiert die auf die Header folgenden Daten als Daten des benannten Typs. Wenn es sich bei dem Format um den Namen eines Headertyps handelt, der für die IBM MQ-Datenkonvertierung infrage kommt, wird es konvertiert. Wenn es sich um einen anderen Namen handelt, der auf Nicht-MQ-Daten verweist (beispielsweise MQFMT_NONE oder MQFMT_STRING), stoppt der Nachrichtenkanalagent die Verarbeitung der Header.

Was hat es mit MQCXP 'HeaderLength' auf sich?

Bei dem Parameter 'HeaderLength' in den an einen Nachrichtenexit übergebenen MQCXP-Daten handelt es sich um die Gesamtlänge der MQXQH- (einschließlich MQMD), MQMDE- und MQDH-Header am Nachrichtenanfang. Diese Header sind über die Formatnamen- und -längenwerte verkettet.

MQWIH

Verkettete Header können über die unter 'HeaderLength' angegebene Länge hinaus in den Benutzerdatenbereich hineinreichen. Der MQWIH-Header (falls vorhanden) ist einer der Header, die über 'HeaderLength' hinaus angezeigt werden.

Wenn sich ein MQWIH-Header unter den verketteten Headern befindet, wird dieser vor dem Aufruf des Nachrichtenexits des Empfängers direkt konvertiert.

Exitprogramm für Kanalnachrichtenwiederholung
Der Exit für Kanalnachrichtenwiederholung wird aufgerufen, wenn ein Versuch, die Zielwarteschlange zu öffnen, nicht erfolgreich war. Der Exit kann dazu verwendet werden, festzulegen, unter welchen Umständen, wie häufig und in welchen Abständen Wiederholungen durchgeführt werden sollen.

Dieser Exit wird bei der Initialisierung und Beendigung des Nachrichtenkanalagenten auch auf der Empfangsseite des Kanals aufgerufen.

Dem Exit für Kanalnachrichtenwiederholung wird ein Agentenpuffer übergeben, der den Header der Übertragungswarteschlange MQXQH und den Text der Anwendungsnachricht enthält, wie er von der Warteschlange abgerufen wurde. Informationen zum Format von MQXQH finden Sie im Abschnitt Overview for MQXQH.

Der Exit wird für alle Ursachencodes aufgerufen; er legt fest, bei welchen Ursachencodes, wie häufig und in welchen Intervallen der Nachrichtenkanalagent Wiederholungen durchführen soll. (Die Anzahl der Nachrichtenwiederholungen, die bei der Definition des Kanals festgelegt wurde, wird an den Exit in der MQCD übergeben, der Exit kann den Wert jedoch ignorieren.)

Der Wert im Feld 'MsgRetryCount' von MQCXP wird durch den Nachrichtenkanalagenten bei jedem Aufruf des Exits erhöht und der Exit gibt entweder MQXCC_OK mit der Wartezeit im Feld 'MsgRetryInterval' von MQCXP oder MQXCC_SUPPRESS_FUNCTION zurück. Die Wiederholungen werden so lange fortgesetzt, bis der Exit im Feld 'ExitResponse' von MQCXP den Wert MQXCC_SUPPRESS_FUNCTION zurückgibt. Informationen zu den durch den Nachrichtenkanalagenten ergriffenen Maßnahmen bei diesen Beendigungscodes finden Sie im Abschnitt MQCXP.

Wenn alle Wiederholungen nicht erfolgreich sind, wird die Nachricht in die Warteschlange für nicht zustellbare Nachrichten geschrieben. Wenn keine Warteschlange für nicht zustellbare Nachrichten verfügbar ist, wird der Kanal gestoppt.

Wenn Sie für einen Kanal keinen Exit für Nachrichtenwiederholung definieren und ein Fehler auftritt, der wahrscheinlich vorübergehend ist (z.B. MQRC_Q_FULL), verwendet der Nachrichtenkanalagent die bei der Definition des Kanals festgelegten Werte für die Anzahl und Intervalle der Wiederholungen. Wenn es sich um einen permanenten Fehler handelt und kein Exitprogramm zu dessen Behebung definiert ist, wird die Nachricht in die Warteschlange für nicht zustellbare Nachrichten geschrieben.

Exitprogramm für die automatische Kanaldefinition
Der Exit für die automatische Kanaldefinition kann verwendet werden, wenn eine Anforderung zum Starten eines Empfänger- oder Serververbindungskanals empfangen wird, jedoch keine Definition für diesen Kanal vorhanden ist (nicht für IBM MQ for z/OS). Er kann auch auf allen Plattformen für Clustersender- und Clusterempfängerkanäle aufgerufen werden, um die Definitionsänderung für eine Instanz des Kanals zu ermöglichen.

Der Exit für die automatische Kanaldefinition kann auf allen Plattformen mit Ausnahme von z/OS aufgerufen werden, wenn eine Anforderung zum Starten eines Empfänger- oder Serververbindungskanals empfangen wird, jedoch keine Kanaldefinition vorhanden ist. Mit dem Exit können Sie die bereitgestellte Standarddefinition für einen automatisch definierten Empfänger- oder Serververbindungskanal, SYSTEM.AUTO.RECEIVER bzw. SYSTEM.AUTO.SVRCON, ändern. Informationen zur automatischen Erstellung von Kanaldefinitionen finden Sie im Abschnitt Preparing channels.

Der Exit für die automatische Kanaldefinition kann auch aufgerufen werden, wenn eine Anforderung zum Starten eines Clustersenderkanals empfangen wird. Er kann für sämtliche Clustersender- und Clusterempfängerkanäle aufgerufen werden, um die Definitionsänderung für diese Instanz des Kanals zu ermöglichen. In diesem Fall findet der Exit auch unter IBM MQ for z/OS Anwendung. Im Allgemeinen wird der Exit für die automatische Kanaldefinition zum Ändern der Namen von Nachrichtenexits (MSGEXIT, RCVEXIT, SCYEXIT und SENDEXIT) verwendet, da Exitnamen auf unterschiedlichen Plattformen unterschiedlich formatiert sind. Wenn kein Exit für die automatische Kanaldefinition angegeben ist, besteht das Standardverhalten unter z/OS darin, einen verteilten Exitnamen im Format *[path]/libraryname(function)* zu untersuchen und bis zu acht Zeichen der Funktion (falls vorhanden) oder einen Bibliotheksnamen zu verwenden. Unter z/OS muss ein Exitprogramm für die automatische Kanaldefinition die Felder ändern, auf die 'MsgExitPtr', 'MsgUserDataPtr', 'SendExitPtr', 'SendUserDataPtr', 'ReceiveExitPtr' und 'ReceiveUserDataPtr' verweisen, nicht die Felder 'MsgExit', 'MsgUserData', 'SendExit', 'SendUserData', 'ReceiveExit' und 'ReceiveUserData' selbst.

Weitere Informationen finden Sie im Abschnitt Working with auto-defined channels.

Wie bei anderen Kanalexits lautet die Parameterliste:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

ChannelExitParms wird im Abschnitt MQCXP beschrieben. ChannelDefinition wird im Abschnitt MQCD beschrieben.

MQCD enthält die Werte, die in der Standardkanaldefinition verwendet werden, wenn sie nicht vom Exit geändert werden. Der Exit kann nur einen Teil der Felder ändern. Weitere Informationen hierzu finden Sie im Abschnitt MQ_CHANNEL_AUTO_DEF_EXIT. Der Versuch, andere Felder zu ändern, verursacht jedoch keinen Fehler.

Der Exit für die automatische Kanaldefinition gibt entweder die Antwort MQXCC_OK oder MQXCC_SUPPRESS_FUNCTION zurück. Wird keine dieser Antworten zurückgegeben, setzt der Nachrichtenkanalagent die Verarbeitung fort, als ob MQXCC_SUPPRESS_FUNCTION zurückgegeben wurde. Die automatische Definition wird also abgebrochen, es wird keine neue Kanaldefinition erstellt und der Kanal kann nicht gestartet werden.

ALW *Kanalexitprogramme auf AIX, Linux, and Windows-Systemen kompilieren*

Mithilfe der folgenden Beispiele können Sie Kanalexitprogramme für AIX, Linux, and Windows-Systeme kompilieren.

Windows

Windows

Die Compiler- und Linker-Befehle für Kanalexitprogramme unter Windows:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

Systeme mit AIX and Linux

Linux AIX

In diesen Beispielen ist exit der Bibliotheksname und ChannelExit der Funktionsname. Unter AIX heißt die Exportdatei exit.exp. Die Kanaldefinition verweist mit diesen Namen auf das Exitprogramm. Das entsprechende Format wird im Abschnitt MQCD- channel definition beschrieben. Lesen Sie auch die Informationen zum Parameter MSGEXIT des Befehls DEFINE CHANNEL.

AIX Compiler- und Linker-Beispielbefehle für Kanalexits unter AIX

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Compiler- und Linker-Beispielbefehle für Kanalexits unter Linux für 32-Bit-Warteschlangenmanager:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux Compiler- und Linker-Beispielbefehle für Kanalexits unter Linux für 64-Bit-Warteschlangenmanager:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Auf dem Client kann sowohl ein 32-Bit- als auch ein 64-Bit-Exit verwendet werden. Dieser Exit muss mit 'mqic_r' verknüpft sein.

AIX Unter AIX müssen alle von IBM MQ aufgerufenen Funktionen exportiert werden. Es folgt das Beispiel einer Exportdatei für diese Makefile:

```
#
!channelExit
MQStart
```

Kanalexits konfigurieren

Damit der Kanalexit aufgerufen werden kann, muss er in der Kanaldefinition benannt werden.

Kanalexits müssen in der Kanaldefinition benannt werden. Sie können diese Benennung vornehmen, wenn Sie die Kanäle zum ersten Mal definieren, oder Sie können die Informationen später beispielsweise unter Verwendung des MQSC-Befehls ALTER CHANNEL hinzufügen. Darüber hinaus können Sie den Kanalexit in der MQCD-Kanaldatenstruktur benennen. Das Format des Exitnamens hängt von Ihrer IBM MQ-Plattform ab; weitere Informationen finden Sie in unter MQCD oder MQSC-Befehle.

Wenn die Kanaldefinition keinen Benutzerexitprogrammnamen enthält, wird der Benutzerexit nicht aufgerufen.

Der Exit für die automatische Kanaldefinition ist die Eigenschaft des Warteschlangenmanagers, nicht der einzelne Kanal. Damit dieser Exit aufgerufen werden kann, muss er in der Definition des Warteschlangenmanagers benannt werden. Verwenden Sie den MQSC-Befehl ALTER QMGR, um die Definition eines Warteschlangenmanagers zu ändern.

Datenkonvertierungsexits schreiben

In dieser Themengruppe finden Sie Informationen zum Schreiben von Datenkonvertierungsexits.

Anmerkung: Wird unter MQSeries for VSE/ESA nicht unterstützt.

Wenn Sie einen Aufruf MQPUT ausgeben, erstellt die Anwendung den Nachrichtendeskriptor (MQMD) der Nachricht. Da IBM MQ den Inhalt des MQMD verstehen muss, unabhängig davon, auf welcher Plattform er erstellt wurde, wird er automatisch durch das System konvertiert.

Anwendungsdaten werden dagegen nicht automatisch konvertiert. Wenn Zeichendaten zwischen Plattformen ausgetauscht werden, auf denen die Felder `CodedCharSetId` und `Encoding` voneinander abweichen, beispielsweise bei ASCII und EBCDIC, muss die Anwendung die Konvertierung der Nachricht veranlassen. Anwendungsdaten können vom Warteschlangenmanager selbst oder von einem Benutzerexitprogramm, dem sogenannten *Datenkonvertierungsexit* konvertiert werden. Mithilfe einer der integrierten Konvertierungsroutinen kann der Warteschlangenmanager die Datenkonvertierung selbst vornehmen, wenn die Anwendungsdaten in einem der integrierten Formate (wie z. B. MQFMT_STRING) vorliegen. Dieser Abschnitt enthält Informationen zur Datenkonvertierungsexitfunktion, die IBM MQ für den Fall bereitstellt, dass die Anwendungsdaten nicht in einem integrierten Format vorliegen.

Während eines MQGET-Aufrufs kann die Steuerung an den Datenkonvertierungsexit übergeben werden. Auf diese Weise werden Konvertierungen über verschiedene Plattformen hinweg vor Erreichen des Zielorts vermieden. Handelt es sich jedoch bei dem Zielort um eine Plattform, die keine Datenkonvertierung im MQGET-Aufruf unterstützt, müssen Sie im Senderkanal, welcher die Daten an den Zielort sendet, CONVERT(YES) angeben. Dadurch wird die Konvertierung der Daten durch IBM MQ während der Übertragung sichergestellt. In diesem Fall muss sich der Datenkonvertierungsexit auf dem System befinden, in dem der Senderkanal definiert ist.

Der MQGET-Aufruf wird direkt von der Anwendung ausgegeben. Legen Sie für die Felder `CodedCharSetId` und `Encoding` im MQMD den erforderlichen Zeichensatz und die gewünschte Verschlüsselung fest. Wenn Ihre Anwendung denselben Zeichensatz und dieselbe Verschlüsselung wie der Warteschlangenmanager verwendet, setzen Sie `CodedCharSetId` auf MQCCSI_Q_MGR und `Encoding` auf MQENC_NATIVE. Nach Abschluss des MQGET-Aufrufs sind in diesen Feldern die geeigneten Werte für die zurückgegebenen Nachrichtendaten angegeben. Falls die Konvertierung nicht erfolgreich durchgeführt werden konnte, sind möglicherweise andere Werte erforderlich. Die Felder sollten von der Anwendung vor jedem MQGET-Aufruf auf die erforderlichen Werte zurückgesetzt werden.

Welche Bedingungen erfüllt sein müssen, damit der Datenkonvertierungsexit aufgerufen wird, ist für den MQGET-Aufruf im Abschnitt MQGET definiert.

Eine Beschreibung der Parameter, die an den Datenkonvertierungsexit übergeben werden, sowie ausführliche Hinweise zur Verwendung finden Sie im Abschnitt Data conversion für den Aufruf MQ_DATA_CONV_EXIT und die MQDXP-Struktur.

Programme, die Anwendungsdaten zwischen verschiedenen Maschinenverschlüsselungen und CCSIDs konvertieren, müssen mit der IBM MQ-Datenkonvertierungsschnittstelle (DCI) kompatibel sein.

Für Multicasting-Clients müssen API-Exits und Datenkonvertierungsexits clientseitig ausgeführt werden können, da manche Nachrichten den Warteschlangenmanager unter Umständen nicht durchlaufen. Die folgenden Bibliotheken sind sowohl Teil der Client- als auch der Serverpakete:

Tabelle 143. In den Client- und Serverpaketen enthaltene Bibliotheken

Betriebssystem	**Bibliotheken**
AIX	32 Bit & 64 Bit: libmqm.a & libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 Bit & 64 Bit: libmqm.so & libmqm_r.so
Windows	32 Bit & 64 Bit: mqm.dll & mqm.pdb

Datenkonvertierungsexit aufrufen

Ein Datenkonvertierungsexit ist ein benutzerdefinierter Exit, der während der Verarbeitung eines MQGET-Aufrufs die Kontrolle übernimmt.

Der Exit wird aufgerufen, wenn die folgenden Bedingungen erfüllt sind:

- Die Option MQGMO_CONVERT ist beim MQGET-Aufruf angegeben.
- Ein Teil bzw. die gesamten Nachrichtendaten verfügen nicht über den angeforderten Zeichensatz oder die angeforderte Verschlüsselung.
- Das Feld *Format* der MQMD-Struktur der Nachricht ist nicht auf MQFMT_NONE gesetzt.
- Die im MQGET-Aufruf angegebene *BufferLength* ist nicht null.
- Die Nachrichtendatenlänge ist nicht null.
- Die Nachricht enthält Daten in einem benutzerdefinierten Format. Das benutzerdefinierte Format kann die gesamte Nachricht ausfüllen oder ihm können ein oder mehrere integriertes Formate vorausgehen. Beispielsweise kann dem benutzerdefinierten Format ein MQFMT_DEAD_LETTER_HEADER-Format vorangestellt sein. Der Exit wird ausschließlich zur Konvertierung des benutzerdefinierten Formats aufgerufen. Der Warteschlangenmanager konvertiert alle integrierten Formate, die dem benutzerdefinierten Format vorausgehen.

  Ein benutzerdefinierter Exit kann auch zur Konvertierung integrierter Formate aufgerufen werden, dies geschieht jedoch nur, wenn die integrierten Konvertierungsroutinen das integrierte Format nicht erfolgreich konvertieren.

Einige weitere Bedingungen sind ausführlich in den Hinweisen zur Verwendung des MQ_DATA_CONV_EXIT-Aufrufs im Abschnitt MQ_DATA_CONV_EXIT beschrieben.

Detaillierte Informationen zum MQGET-Aufruf finden Sie im Abschnitt MQGET. Datenkonvertierungsexits können außer MQXCNVC keine MQI-Aufrufe verwenden.

Eine neue Kopie des Exits wird geladen, wenn eine Anwendung seit der Herstellung der Verbindung zum Warteschlangenmanager zum ersten Mal versucht, eine Nachricht mit diesem *Format* abzurufen. Zudem kann auch zu anderen Zeiten eine neue Kopie geladen werden, wenn der Warteschlangenmanager eine zuvor geladene Kopie gelöscht hat.

Der Datenkonvertierungsexit wird in einer ähnlichen Umgebung wie das Programm ausgeführt, das den MQGET-Aufruf ausgegeben hat. Neben Benutzeranwendungen kann es sich bei dem Programm auch um einen Nachrichtenkanalagenten handeln, der Nachrichten an einen Zielwarteschlangenmanager sendet, der keine Nachrichtenkonvertierung unterstützt. Die Umgebung enthält ggf. einen Adressraum und ein Benutzerprofil. Dieser Exit kann die Integrität des Warteschlangenmanagers nicht beeinträchtigen, da er nicht in der Umgebung des Warteschlangenmanagers ausgeführt wird.

Datenkonvertierung unter z/OS

z/OS

Unter z/OS sind folgende Punkte zu berücksichtigen:

- Exitprogramme können ausschließlich in Assemblersprache geschrieben werden.
- Exitprogramme müssen simultan verwendbar sein und überall im Speicher ausgeführt werden können.
- Exitprogramme müssen beim Verlassen der Umgebung den Startzustand wiederherstellen und angeforderten Speicherplatz wieder freigeben.
- Exitprogramme dürfen keine WAIT-Option oder ESTAEs oder SPIEs ausgeben.
- Exitprogramme werden in der Regel unter folgenden Bedingungen wie durch z/OS LINK aufgerufen:
  - Bei Auftreten eines Fehlers im Programm aufgrund einer fehlenden Zugriffsberechtigung
  - Im Steuermodus für den Primäradressraum
  - Im nicht speicherübergreifenden Modus
  - Im Nichtzugriffsregistermodus
  - Im 31-Bit-Adressierungsmodus
  - Im TCB-PRB-Modus
- Bei Verwendung durch eine CICS-Anwendung wird der Exit durch EXEC CICS LINK aufgerufen und muss den CICS-Programmierungskonventionen entsprechen. Die Parameter werden durch Verweise (Adressen) im CICS-Kommunikationsbereich (COMMAREA) übergeben.

  Obgleich dies nicht empfohlen wird, können Benutzerexitprogramme auch CICS-API-Aufrufe verwenden, wenn folgende Warnhinweise berücksichtigt werden:

  - Es dürfen keine Synchronisationspunkte ausgegeben werden, da die Ergebnisse durch den Nachrichtenkanalagenten deklarierte Arbeitseinheiten beeinträchtigen könnten.
  - Es dürfen keine durch einen anderen Ressourcenmanager als IBM MQ for z/OS gesteuerten Ressourcen aktualisiert werden, auch nicht solche, die durch den CICS-Transaktionsserver gesteuert werden.

  Bei Kanälen mit CONVERT=YES wird der Exit aus dem durch die CSQXLIB-DD-Anweisung angegebenen Datensatz geladen. Die von MQ für die IBM MQ CICS-Bridge bereitgestellten Exits CSQCBDCI und CSQCBDCO befinden sich in SCSQAUTH.

Datenkonvertierungsexitprogramm für IBM i schreiben

Schritte, die Sie beim Schreiben von MQ-Datenkonvertierungsprogrammen für IBM i beachten sollten.

Gehen Sie wie folgt vor:

1. Benennen Sie Ihr Nachrichtenformat. Der Name muss in das MQMD-Feld *Format* passen, wie z. B. MYFORMAT. Der Name für das Feld *Format* darf keine führenden eingebetteten Leerzeichen enthalten; abschließende Leerzeichen werden dagegen ignoriert. Da das Feld *Format* nur acht Zeichen lang ist, darf der Name des Objekts maximal acht belegte Zeichen haben. Denken Sie daran, diesen Name immer zu verwenden, wenn Sie eine Nachricht senden (unser Beispiel verwendet den Namen "Format").
2. Erstellen Sie eine Struktur zur Darstellung Ihrer Nachricht. Ein Beispiel hierfür finden Sie im Abschnitt Valid syntax.
3. Führen Sie diese Struktur über den Befehl CVTMQMDTA aus, um ein Codefragment für Ihren Datenkonvertierungsexit zu erstellen.

Die vom Befehl CVTMQMDTA erstellten Funktionen verwenden Makros, die in der Datei QMQM/H(AMQSVMHA) ausgeliefert werden. Diese Makros werden unter der Voraussetzung geschrieben, dass alle Strukturen gepackt sind; korrigieren Sie sie, wenn das nicht der Fall ist.

4. Erstellen Sie eine Kopie der bereitgestellten Entwurfsquellendatei QMQMSAMP/QCSRC(AMQSVFC4) und benennen Sie sie um. (In unserem Beispiel wird der Name EXIT_MOD verwendet.)
5. Suchen Sie in der Quellendatei nach den folgenden Kommentarboxes und fügen Sie wie hier beschrieben Code ein:

   a. Am Ende der Quellendatei beginnt eine Kommentarbox mit:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Fügen Sie hier das in Schritt „3” auf Seite 1037 generierte Codefragment ein.

   b. Ungefähr in der Mitte der Quellendatei befindet sich eine wie folgt beginnende Kommentarbox:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Hierauf folgt ein auf Kommentar gesetzter Aufruf der Funktion `ConverttagSTRUCT`.

   Ändern Sie den Namen der Funktion und geben Sie hierfür den Namen der in Schritt „5.a” auf Seite 1038 hinzugefügten Funktion an. Entfernen Sie die Kommentarzeichen, um die Funktion zu aktivieren. Erstellen Sie im Falle mehrerer Funktionen für jede einzelne einen Aufruf.

   c. Am Anfang der Quellendatei beginnt eine Kommentarbox mit:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   Fügen Sie hier die Funktionsprototypanweisungen für die in Schritt „5.a” auf Seite 1038 hinzugefügten Funktionen ein.

   Wenn die Nachricht Zeichendaten enthält, ruft der generierte Code MQXCNVC auf; die Auflösung kann durch Binden des Serviceprogramms QMQM/LIBMQM erfolgen.

6. Kompilieren Sie das Quellenmodul EXIT_MOD wie folgt:

   ```
   CRTCMOD MODULE(library/EXIT_MOD) +
   SRCFILE(QCSRC) +
   TERASPACE(*YES *TSIFC)
   ```

7. Erstellen/verlinken Sie das Programm.

   Verwenden Sie bei Nicht-Thread-Anwendungen Folgendes:

   ```
   CRTPGM PGM(library/Format)  +
      MODULE(library/EXIT_MOD)  +
      BNDSRVPGM(QMQM/LIBMQM)  +
      ACTGRP(QMQM)  +
      USRPRF(*USER)
   ```

   Neben der Erstellung des Datenkonvertierungsexits für die grundlegende Umgebung ist ein weiterer in der Threadumgebung erforderlich. Auf dieses ladbare Objekt muss '_R' folgen. Lösen Sie MQXCNVC-Aufrufe mithilfe der Bibliothek LIBMQM_R auf. Für eine Threadumgebung sind beide ladbaren Objekte erforderlich.

   ```
   CRTPGM PGM(library/Format_R)  +
      MODULE(library/EXIT_MOD)  +
      BNDSRVPGM(QMQM/LIBMQM_R)  +
      ACTGRP(QMQM)  +
      USRPRF(*USER)
   ```

8. Stellen Sie die Ausgabe in die Bibliotheksliste für den IBM MQ-Job. Es wird empfohlen, dass Datenkonvertierungsexitprogramme für die Produktion in QSYS gespeichert werden.

Anmerkung:

1. Bei Verwendung gepackter Strukturen durch CVTMQMDTA müssen alle IBM MQ-Anwendungen das Qualifikationsmerkmal'_Packed' verwenden.
2. Datenkonvertierungsexitprogramme müssen simultan verwendbar sein.
3. Von einem Datenkonvertierungsexit aus kann nur der MQI-Aufruf MQXCNVC ausgegeben werden.
4. Kompilieren Sie das Exitprogramm so, dass die Compileroption des Benutzerprofils auf *USER eingestellt ist, damit der Exit mit der Berechtigung des Benutzers ausgeführt wird.
5. Für alle Benutzerexits mit IBM MQ for IBM i ist eine Teraspace-Speicheraktivierung erforderlich ; Geben Sie TERASPACE(*YES *TSIFC) in den Befehlen CRTCMOD und CRTBNDC an.

Writing a data-conversion exit program for IBM MQ for z/OS

Information about steps to consider when writing data-conversion exit programs for IBM MQ for z/OS.

Follow these steps:

1. Take the supplied source skeleton CSQ4BAX9 (for non-CICS environments) or CSQ4CAX9 (for CICS) as your starting point.
2. Run the CSQUCVX utility.
3. Follow the instructions in the prolog of CSQ4BAX9 or CSQ4CAX9 to incorporate the routines generated by the CSQUCVX utility, in the order that the structures occur in the message that you want to convert.
4. The utility assumes that the data structures are not packed, that the implied alignment of the data is honored, and that the structures start on a fullword boundary, with bytes being skipped as required (as between ID and VERSION in the example in Valid syntax). If the structures are packed, omit the CMQXCALA macros that are generated. Therefore, consider declaring your structures in such a way that all fields are named and no bytes are skipped; in the example in Valid syntax, add a field "MQBYTE DUMMY;" between ID and VERSION.
5. The supplied exit returns an error if the input buffer is shorter than the message format to be converted. Although the exit converts as many complete fields as possible, the error causes an unconverted message to be returned to the application. If you want to allow short input buffers to be converted as far as possible, including partial fields, change the TRUNC= value on the CSQXCDFA macro to YES: no error is returned, so the application receives a converted message. The application must handle the truncation.
6. Add any other special processing code that you need.
7. Rename the program to your data format name.
8. Compile and link-edit your program like a batch application program (unless it is for use with CICS applications). The macros in the code generated by the utility are in the library, **thlqual**.SCSQMACS.

   If the message contains character data, the generated code calls MQXCNVC. If your exit uses this call, link-edit it with the exit stub program CSQASTUB. The stub is language-independent and environment-independent. Alternatively, you can load the stub dynamically using the dynamic call name CSQXCNVC. See “Dynamically calling the IBM MQ stub” on page 1082 for more information.

   Place the link-edited module in your application load library, and in a data set that is referenced by the CSQXLIB DD statement of your task procedure started by your channel initiator.
9. If the exit is for use by CICS applications, compile and link-edit it like a CICS application program, including CSQASTUB if required. Place it in your CICS application program library. Define the program to CICS in the typical way, specifying EXECKEY(CICS) in the definition.

Note: Although the LE/370 runtime libraries are needed for running the CSQUCVX utility (see step “2” on page 1039), they are not needed for link-editing or running the data-conversion exit itself (see steps “8” on page 1039 and “9” on page 1039).

See “Writing IMS bridge applications” on page 78 for information about data conversion within the IBM MQ - IMS bridge.

Linux AIX *Datenkonvertierungsexit für IBM MQ for AIX or Linux-Systeme schreiben*

Nachfolgend finden Sie die Schritte, die Sie beim Schreiben von Datenkonvertierungsprogrammen für IBM MQ for AIX or Linux-Systeme beachten sollten.

Gehen Sie wie folgt vor:

1. Benennen Sie Ihr Nachrichtenformat. Der Name muss ins Feld *Format* des MQMD passen und in Großbuchstaben angegeben werden, z. B. MYFORMAT. Der im Feld *Format* angegebene Name darf keine führenden Leerzeichen enthalten. Abschließende Leerzeichen werden ignoriert. Da das Feld *Format* nur acht Zeichen lang ist, darf der Name des Objekts maximal acht belegte Zeichen haben. Denken Sie daran, diesen Namen immer zu verwenden, wenn Sie eine Nachricht senden.

   Wird der Datenkonvertierungsexit in einer Threadumgebung verwendet, muss durch Angabe von '_r' hinter dem ladbaren Objekt angegeben werden, dass es sich um eine Threadversion handelt.
2. Erstellen Sie eine Struktur zur Darstellung Ihrer Nachricht. Ein Beispiel hierfür finden Sie im Abschnitt Valid syntax.
3. Führen Sie diese Struktur über den Befehl `crtmqcvx` aus und erstellen Sie auf diese Weise ein Codefragment für den Datenkonvertierungsexit.

   Die mit dem Befehl `crtmqcvx` generierten Funktionen verwenden Makros, die voraussetzen, dass alle Strukturen gepackt sind. Nehmen Sie gegebenenfalls die entsprechenden Korrekturen vor.
4. Kopieren Sie die bereitgestellte Entwurfsquellendatei, wobei Sie sie mit dem in Schritt „1” auf Seite 1040 festgelegten Nachrichtenformatnamen umbenennen. Die Entwurfsquellendatei und ihre Kopie sind schreibgeschützt.

   Die Entwurfsquellendatei trägt die Bezeichnung 'amqsvfc0.c'.
5. Unter IBM MQ for AIX wird zudem eine Entwurfsexportdatei namens 'amqsvfc.exp' bereitgestellt. Kopieren Sie diese Datei und benennen Sie sie dabei in MYFORMAT.EXP um.
6. Die Entwurfsdatei enthält eine Musterheaderdatei 'amqsvmha.h' im Verzeichnis *MQ_INSTALLATION_PATH*/inc, wobei *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis steht, in dem IBM MQ installiert ist. Vergewissern Sie sich, dass der include-Pfad auf dieses Verzeichnis zeigt, damit diese Datei berücksichtigt wird.

   Die Datei 'amqsvmha.h' enthält Makros, die von dem mit dem Befehl `crtmqcvx` generierten Code verwendet werden. Umfasst die zu konvertierende Struktur Zeichendaten, wird von diesen Makros MQXCNVC aufgerufen.
7. Suchen Sie in der Quellendatei nach den folgenden Kommentarboxes und fügen Sie wie hier beschrieben Code ein:

   a. Am Ende der Quellendatei beginnt eine Kommentarbox mit:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Fügen Sie hier das in Schritt „3” auf Seite 1040 generierte Codefragment ein.

   b. Ungefähr in der Mitte der Quellendatei befindet sich eine wie folgt beginnende Kommentarbox:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Hierauf folgt ein auf Kommentar gesetzter Aufruf der Funktion `ConverttagSTRUCT`.

   Ändern Sie den Namen der Funktion und geben Sie hierfür den Namen der in Schritt „7.a” auf Seite 1040 hinzugefügten Funktion an. Entfernen Sie die Kommentarzeichen, um die Funktion zu aktivieren. Erstellen Sie im Falle mehrerer Funktionen für jede einzelne einen Aufruf.

   c. Am Anfang der Quellendatei beginnt eine Kommentarbox mit:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

Fügen Sie hier die Funktionsprototypanweisungen für die in Schritt „3” auf Seite 1040 hinzugefügten Funktionen ein.

8. Kompilieren Sie den Exit als gemeinsam genutzte Bibliothek und verwenden Sie dabei als Eingangspunkt 'MQStart'. Informationen dazu finden Sie unter „Datenkonvertierungsexits auf AIX and Linux-Systemen kompilieren” auf Seite 1041.
9. Stellen Sie die Ausgabe in das Exitverzeichnis. Das Standardexitverzeichnis ist `/var/mqm/exits` (32-Bit-Systeme) bzw. `/var/mqm/exits64` (64-Bit-Systeme). Diese Verzeichnisse können in der Datei 'qm.ini' bzw. 'mqclient.ini' geändert werden. Dieser Pfad kann für jeden einzelnen Warteschlangenmanager definiert werden. Es wird nur in dem betreffenden Pfad bzw. den betreffenden Pfaden nach dem Exit gesucht.

Anmerkung:

1. Bei Verwendung gepackter Strukturen durch `crtmqcvx` müssen alle IBM MQ-Anwendungen auf diese Weise kompiliert sein.
2. Datenkonvertierungsexitprogramme müssen simultan verwendbar sein.
3. Von einem Datenkonvertierungsexit aus kann nur der MQI-Aufruf MQXCNVC ausgegeben werden.

Linux AIX *Datenkonvertierungsexits auf AIX and Linux-Systemen kompilieren*

Beispiele zur Kompilierung von Datenkonvertierungsexits auf AIX and Linux-Systemen.

Auf allen Plattformen ist der Eingangspunkt in das Modul 'MQStart'.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

AIX

AIX

Kompilieren Sie den Exit-Quellcode, indem Sie einen der folgenden Befehle ausgeben:

32-Bit-Anwendungen

Nicht-thread-basiert

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

64-Bit-Anwendungen

Nicht-thread-basiert

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

Kompilieren Sie den Exit-Quellcode, indem Sie einen der folgenden Befehle ausgeben:

31-Bit-Anwendungen

Nicht-thread-basiert

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
     -I  MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

32-Bit-Anwendungen

Nicht-thread-basiert

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

64-Bit-Anwendungen

Nicht-thread-basiert

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Windows *Datenkonvertierungsexit für IBM MQ for Windows schreiben*

Schritte, die Sie beim Schreiben von Datenkonvertierungsprogrammen für IBM MQ for Windows beachten sollten.

Gehen Sie wie folgt vor:

1. Benennen Sie Ihr Nachrichtenformat. Der Name muss in das MQMD-Feld *Format* passen, wie z. B. MYFORMAT. Der im Feld *Format* angegebene Name darf keine führenden Leerzeichen enthalten. Abschließende Leerzeichen werden ignoriert. Da das Feld *Format* nur acht Zeichen lang ist, darf der Name des Objekts maximal acht belegte Zeichen haben.

   Eine .DEF-Datei namens amqsvfcn.def wird ebenfalls im Beispielverzeichnis *MQ_INSTALLATION_PATH*\Tools\C\Samplesbereitgestellt. *MQ_INSTALLATION_PATH* ist das Verzeichnis, in dem IBM

MQ installiert ist. Erstellen Sie eine Kopie dieser Datei und benennen Sie sie um, beispielsweise in MYFORMAT.DEF. Der Name der erstellten DLL-Datei und der in MYFORMAT.DEF angegebene Name müssen identisch sein. Überschreiben Sie den Namen FORMAT1 in MYFORMAT.DEF mit dem neuen Formatnamen.

Denken Sie daran, diesen Namen immer zu verwenden, wenn Sie eine Nachricht senden.

2. Erstellen Sie eine Struktur zur Darstellung Ihrer Nachricht. Ein Beispiel hierfür finden Sie im Abschnitt Valid syntax.
3. Führen Sie diese Struktur über den Befehl `crtmqcvx` aus und erstellen Sie auf diese Weise ein Codefragment für den Datenkonvertierungsexit.

   Die mit dem Befehl CRTMQCVX generierten Funktionen verwenden Makros, die voraussetzen, dass alle Strukturen gepackt sind. Nehmen Sie gegebenenfalls die entsprechenden Korrekturen vor.
4. Kopieren Sie die bereitgestellte Entwurfsquellendatei 'amqsvfc0.c', wobei Sie sie in den in Schritt „1” auf Seite 1042 festgelegten Nachrichtenformatnamen umbenennen.

   amqsvfc0.c befindet sich in *MQ_INSTALLATION_PATH*`\Tools\C\Samples` , wobei *MQ_INSTALLATION_PATH* das Verzeichnis ist, in dem IBM MQ installiert ist. (Das Standardinstallationsverzeichnis ist `C:\Programme\IBM\MQ`.)

   Das Gerüst enthält eine Beispielheaderdatei amqsvmha.h im Verzeichnis *MQ_INSTALLATION_PATH*`\Tools\C\include` . Vergewissern Sie sich, dass der include-Pfad auf dieses Verzeichnis zeigt, damit diese Datei berücksichtigt wird.

   Die Datei 'amqsvmha.h' enthält Makros, die von dem mit dem Befehl CRTMQCVX generierten Code verwendet werden. Umfasst die zu konvertierende Struktur Zeichendaten, wird von diesen Makros MQXCNVC aufgerufen.
5. Suchen Sie in der Quellendatei nach den folgenden Kommentarboxes und fügen Sie wie hier beschrieben Code ein:

   a. Am Ende der Quellendatei beginnt eine Kommentarbox mit:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Fügen Sie hier das in Schritt „3” auf Seite 1043 generierte Codefragment ein.

   b. Ungefähr in der Mitte der Quellendatei befindet sich eine wie folgt beginnende Kommentarbox:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Hierauf folgt ein auf Kommentar gesetzter Aufruf der Funktion `ConverttagSTRUCT`.

   Ändern Sie den Namen der Funktion und geben Sie hierfür den Namen der in Schritt „5.a” auf Seite 1043 hinzugefügten Funktion an. Entfernen Sie die Kommentarzeichen, um die Funktion zu aktivieren. Erstellen Sie im Falle mehrerer Funktionen für jede einzelne einen Aufruf.

   c. Am Anfang der Quellendatei beginnt eine Kommentarbox mit:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   Fügen Sie hier die Funktionsprototypanweisungen für die in Schritt „3” auf Seite 1043 hinzugefügten Funktionen ein.
6. Erstellen Sie die folgende Befehlsdatei:

```
cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
MYFORMAT.C


MYFORMAT.DEF
```

wobei *MQ_INSTALLATION_PATH* für das Verzeichnis steht, in dem IBM MQ installiert ist.

7. Rufen Sie die Befehlsdatei auf, um den Exit als DLL-Datei zu kompilieren.
8. Stellen Sie die Ausgabe in das Exit-Unterverzeichnis unter dem IBM MQ-Datenverzeichnis. Das Standardverzeichnis zur Installation Ihrer Exits ist auf 32-Bit-Systemen *MQ_DATA_PATH*\Exits, auf 64-Bit-Systemen heißt es *MQ_DATA_PATH*\Exits64.

   Der Pfad, in dem nach den Datenkonvertierungsexits gesucht wird, ist in der Registrierungsdatenbank angegeben. Der Registrierungsdatenbankordner lautet:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\MQ_INSTALLATI
   ON_NAME\Configuration\ClientExitPath\
   ```

   und der Registrierungsschlüssel: ExitsDefaultPath. Dieser Pfad kann für jeden einzelnen Warteschlangenmanager definiert werden. Es wird nur in dem betreffenden Pfad bzw. den betreffenden Pfaden nach dem Exit gesucht.

Anmerkung:

1. Bei Verwendung gepackter Strukturen durch CRTMQCVX müssen alle IBM MQ-Anwendungen auf diese Weise kompiliert sein.
2. Datenkonvertierungsexitprogramme müssen simultan verwendbar sein.
3. Von einem Datenkonvertierungsexit aus kann nur der MQI-Aufruf MQXCNVC ausgegeben werden.

Windows *Exit- und Switchloaddateien auf Windows-Betriebssystemen*

Die Prozesse des Warteschlangenmanagers für IBM WebSphere MQ for Windows 7.5 werden im 32-Bit-Modus ausgeführt. Dadurch muss bei der Verwendung von 64-Bit-Anwendungen für einige Typen der Exit-Dateien und der XA-Switch-Ladedateien ebenfalls eine 32-Bit-Version für die Verwendung durch den Warteschlangenmanager verfügbar sein. Wenn die 32-Bit-Version der Exit-Datei oder der XA-Switch-Ladedatei erforderlich, jedoch nicht verfügbar ist, schlägt der relevante API-Aufruf oder API-Befehl fehl.

In der Datei qm.ini file für *ExitPath* werden zwei Attribute unterstützt. Dies sind ExitsDefaultPath= *MQ_INSTALLATION_PATH*\exits und ExitsDefaultPath64= *MQ_INSTALLATION_PATH*\exits64. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist. Durch die Verwendung wird sichergestellt, dass die entsprechende Bibliothek gefunden werden kann. Wenn eine Exit-Datei in einem IBM MQ-Cluster verwendet wird, wird zudem sichergestellt, dass die entsprechende Bibliothek auf einem fernen System gefunden werden kann.

In der folgenden Tabelle werden die unterschiedlichen Typen von Exit- und Switchloaddateien aufgeführt und sie enthält Hinweise dazu, ob die 32-Bit- und/oder die 64-Bit-Versionen erforderlich sind, abhängig davon, ob 32-Bit oder 64-Bit-Anwendungen verwendet werden:

Dateitypen	**32-Bit-Anwendungen**	**64-Bit-Anwendungen**
API-Exit	32 Bit und 64 Bit	64 Bit
Datenkonvertierungsexit	32 Bit	64 Bit
Serverkanalexits (alle Typen)	64 Bit	64 Bit
Clientkanalexits (alle Typen)	32 Bit	64 Bit
Exit für installierbaren Service	64 Bit	64 Bit
WLM-Exit des Clusters	64 Bit	64 Bit
Publish/Subscribe-Routing-Exit	64 Bit	64 Bit
Switchloaddateien der Datenbank	32 Bit und 64 Bit	64 Bit
AX-Bibliotheken für den externen Transaktionsmanager	32 Bit	64 Bit

Dateitypen	32-Bit-Anwendungen	64-Bit-Anwendungen
Exit vor Verbindungsaufbau	32 Bit	64 Bit

Verbindungsdefinitionen unter Verwendung eines Vorverbindungsexits aus einem Repository referenzieren

IBM MQ MQI clients kann für die Suche nach einem Repository konfiguriert werden, um Verbindungsdefinitionen mithilfe einer Bibliothek für den Pre-Connect-Exit abzurufen.

Einführung

Eine Clientanwendung kann mithilfe von Definitionstabellen für den Clientkanal (Client Channel Definition Tables, CCDT) eine Verbindung zu einem Warteschlangenmanager herstellen. Die CCDT-Datei befindet sich im Allgemeinen auf einem zentralen Netzdateiserver und verfügt über Clients, die auf die Datei verweisen. Da es schwierig ist, verschiedene Clientanwendungen zu steuern und zu verwalten, die auf die CCDT-Datei verweisen, ist auch ein flexibles Vorgehen möglich, bei dem die Clientdefinitionen in einem globalen Repository wie z. B. LDAP-Verzeichnis, WebSphere Registry and Repository oder einem anderen Repository gespeichert werden. Das Speichern der Definitionen für die Clientverbindung in einem Repository vereinfacht die Verwaltung der Clientverbindungsdefinitionen, und Anwendungen können auf die korrekten und aktuellsten Clientverbindungsdefinitionen zugreifen.

Während der Ausführung des MQCONN/X-Aufrufs lädt der IBM MQ MQI client eine Anwendung, die in einer Bibliothek des Pre-Connect-Exits angegeben wurde, und ruft eine Exit-Funktion zum Abrufen von Verbindungsdefinitionen auf. Die abgerufenen Verbindungsdefinitionen werden anschließend dazu verwendet, eine Verbindung mit einem Warteschlangenmanager herzustellen. Die Einzelheiten der aufzurufenden Exit-Bibliothek und Exit-Funktion werden in der Konfigurationsdatei 'mqclient.ini' angegeben.

Syntax

`void MQ_PRECONNECT_EXIT` (*pExitParms*, *pQMgrName*, *ppConnectOpts*, *pCompCode*, *pReason*);

Parameter

pExitParms
Typ: PMQNXP - Ein-/Ausgabe

Die Struktur des Exit-Parameters **PreConnection**.

Die Struktur wird von dem Programm zugeordnet und verwaltet, das den Exit aufruft.

pQMgrName
Typ: PMQCHAR Ein-/Ausgabe

Name des Warteschlangenmanagers.

Bei Eingabe ist dieser Parameter die Filterzeichenfolge, die über den Parameter **QMgrName** an den API-Aufruf MQCONN übergeben wird. Dieses Feld kann leer bleiben oder es kann einen definierten Namen oder bestimmte Platzhalterzeichen enthalten. Das Feld wird durch den Exit geändert. Der Parameter ist NULL, wenn der Exit mit MQXR_TERM aufgerufen wird.

ppConnectOpts
Typ: ppConnectOpts Ein-/Ausgabe

Optionen, mit denen die Aktion des MQCONNX-Aufrufs gesteuert wird.

Hierbei handelt es sich um einen Verweis auf eine MQCNO-Struktur für Verbindungsoptionen, mit denen die Aktion des API-Aufrufs MQCONN gesteuert wird. Der Parameter ist NULL, wenn der Exit mit MQXR_TERM aufgerufen wird. Der MQI-Client stellt immer eine MQCNO-Struktur für den Exit bereit, auch wenn sie von der Anwendung ursprünglich nicht bereitgestellt wurde. Stellt eine Anwendung eine MQCNO-Struktur bereit, erstellt der Client ein Duplikat davon, das an den Exit übergeben und dort geändert wird. Der Client bleibt Eigner der MQCNO-Struktur.

Eine MQCD-Struktur, auf die in der MQCNO-Struktur verwiesen wird, hat Vorrang vor allen Verbindungsdefinitionen, die über das Array bereitgestellt werden. Der Client stellt mithilfe der MQCNO-Struktur eine Verbindung zum Warteschlangenmanager her, während die anderen Strukturen ignoriert werden.

pCompCode
Typ: PMQLONG Ein-/Ausgabe

Beendigungscode.

Verweis auf eine MQLONG-Struktur, die den Beendigungscode des Exits empfängt. Folgende Werte sind zulässig:

- MQCC_OK - Erfolgreiche Ausführung
- MQCC_WARNING - Warnung (teilweise Ausführung)
- MQCC_FAILED - Aufruf fehlgeschlagen

pReason
Typ: PMQLONG Ein-/Ausgabe

Ursachencode zur näheren Bestimmung von 'pCompCode'.

Verweis auf eine MQLONG-Struktur, die den Ursachencode des Exits empfängt. Lautet der Beendigungscode MQCC_OK, ist nur der folgende Wert gültig:

- MQRC_NONE - (0, x'000') Keine Ursache zurückzumelden.

Wenn der Beendigungscode MQCC_FAILED oder MQCC_WARNING lautet, kann das Feld für den Ursachencode von der Exitfunktion auf einen beliebigen MQRC_*-Wert gesetzt werden.

C Invocation

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

Veröffentlichungsexits schreiben und kompilieren

Sie können einen Veröffentlichungsexit im Warteschlangenmanager konfigurieren, um die Inhalte einer veröffentlichten Nachricht zu ändern, bevor diese von Subskribenten empfangen wird. Sie können außerdem den Nachrichtenheader ändern oder die Nachricht nicht an eine Subskription übergeben.

Anmerkung: Veröffentlichungsexits werden unter z/OS nicht unterstützt.

Mit dem Veröffentlichungsexit können Sie Nachrichten überprüfen und ändern, die an Subskribenten übergeben wurden:

- Inhalte einer Nachricht überprüfen, die für jeden Subskribenten veröffentlicht wird
- Inhalte einer Nachricht ändern, die für jeden Subskribenten veröffentlicht wird
- Warteschlange ändern, in die eine Nachricht eingereiht wird
- Übermittlung einer Nachricht an einen Subskribenten stoppen

Veröffentlichungsexit schreiben

Die Schritte im Abschnitt „Exits und installierbare Services unter AIX, Linux, and Windows schreiben” auf Seite 986 helfen Ihnen beim Schreiben und Kompilieren Ihres Exits.

Der Bereitsteller des Veröffentlichungsexits definiert die Funktionen des Exits. Der Exit muss allerdings den in MQPSXP definierten Regeln entsprechen.

IBM MQ stellt keine Implementierung des Eingangspunkts MQ_PUBLISH_EXIT bereit. Es wird eine typedef-Deklaration in der Programmiersprache C bereitgestellt. Verwenden Sie die typedef-Deklaration, um die Parameter ordnungsgemäß für einen benutzerdefinierten Exit zu deklarieren. Im folgenden Beispiel wird die Verwendung der typedef-Deklaration dargestellt:

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
/* C language statements to perform the function of the exit */
}
```

Der Veröffentlichungsexit wird im Warteschlangenmanagerprozess als Folge der folgenden Operationen ausgeführt:

- Veröffentlichungsoperation, bei der eine Nachricht einem oder mehreren Subskribenten zugestellt wird
- Subskriptionsoperation, bei der eine oder mehrere Nachrichten einer ständigen Veröffentlichung zugestellt werden
- Subskriptionsanforderungsoperation, bei der eine oder mehrere Nachrichten einer ständigen Veröffentlichung zugestellt werden

Wenn der Veröffentlichungsexit das erste Mal aufgerufen wird, um eine Verbindung herzustellen, wird der *ExitReason*-Code MQXR_INIT festgelegt. Vor dem Trennen der Verbindung, nachdem Veröffentlichungsexit verwendet wurde, wird der Exit mit dem *ExitReason*-Code MQXR_TERM aufgerufen.

Wenn der Veröffentlichungsexit konfiguriert ist, aber beim Start des Warteschlangenmanagers nicht geladen werden kann, werden Operationen für Publish/Subscribe-Nachrichten für den Warteschlangenmanager übernommen. Sie müssen das Problem beheben oder den Warteschlangenmanager erneut starten, bevor die Publish/Subscribe-Nachrichtenübertragung wieder aktiviert wird.

Für jede IBM MQ-Verbindung, für die der Veröffentlichungsexit erforderlich ist, kann das Laden oder Initialisieren des Exits fehlschlagen. Wenn das Laden oder Initialisieren des Exits fehlschlägt, werden Publish/Subscribe-Operationen, für die der Veröffentlichungsexit erforderlich ist, für diese Verbindung inaktiviert. Die Operationen schlagen mit dem IBM MQ-Ursachencode MQRC_PUBLISH_EXIT_ERROR fehl.

Der Kontext, in dem der Veröffentlichungsexit aufgerufen wird, ist die Herstellung einer Verbindung zum Warteschlangenmanager durch eine Anwendung. Der Warteschlangenmanager verwaltet einen Benutzerdatenbereich für jede Verbindung, über die Veröffentlichungsoperationen ausgeführt werden. Der Exit kann Informationen im Benutzerdatenbereich für jede Verbindung speichern.

Ein Veröffentlichungsexit kann einige MQI-Aufrufe verwenden. Es können nur die MQI-Aufrufe verwendet werden, durch die Nachrichteneigenschaften bearbeitet werden. Dies sind die folgenden Aufrufe:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

Wenn der Veröffentlichungsexit den Zielwarteschlangenmanager oder den Warteschlangennamen ändert, wird keine neue Berechtigungsprüfung ausgeführt.

Veröffentlichungsexit kompilieren

Der Veröffentlichungsexit ist eine dynamisch geladene Bibliothek, die man sich als Kanalexit vorstellen kann. Weitere Informationen zum Kompilieren von Exits finden Sie unter „Exits und installierbare Services unter AIX, Linux, and Windows schreiben” auf Seite 986.

Beispiel für einen Veröffentlichungsexit

Das Beispielexitprogramm heißt `amqspse0.c`. Darin werden unterschiedliche Nachrichten in eine Protokolldatei geschrieben, abhängig davon, ob der Exit zum Initialisieren, Veröffentlichen oder Beenden von Operationen aufgerufen wurde. Es wird außerdem die Verwendung des Benutzerbereichfelds für den Exit und das entsprechende Freigeben von Speicherplatz gezeigt.

Veröffentlichungsexits konfigurieren

Zum Konfigurieren eines Veröffentlichungsexits müssen Sie bestimmte Attribute definieren.

Unter Windows und Linux können Sie zum Definieren der Attribute den IBM MQ-Explorer verwenden. Die Attribute werden auf der Eigenschaftenseite des Warteschlangenmanagers unter 'Publish/Subscribe' definiert.

Zum Konfigurieren des Veröffentlichungsexits in der Datei `qm.ini` erstellen Sie auf AIX and Linux-Systemen eine Zeilengruppe mit der Bezeichnung `PublishSubscribe`. Die Zeilengruppe `PublishSubscribe` enthält die folgenden Attribute:

PublishExitPath=[path]|*Modulname*
Modulname und Pfad, die den Veröffentlichungsexit-Code enthalten. Die maximale Länge dieses Felds beträgt `MQ_EXIT_NAME_LENGTH`. Standardmäßig wird kein Veröffentlichungsexit angegeben.

PublishExitFunction=*Funktionsname*
Name des Funktionseingangspunktes in dem Modul, das den Veröffentlichungsexit-Code enthält. Die maximale Länge dieses Felds beträgt `MQ_EXIT_NAME_LENGTH`.

IBM i Wenn unter IBM iein Programm verwendet wird, lassen Sie `PublishExitFunction` weg.

PublishExitData=*Zeichenfolge*
Wenn der Warteschlangenmanager einen Veröffentlichungsexit aufruft, wird eine `MQPSXP`-Struktur als Eingabe übergeben. Die Daten, die bei Verwendung des Attributs **PublishExitData** angegeben werden, werden im Feld *ExitData* der Struktur bereitgestellt. Die Zeichenfolge kann eine Länge von bis zu `MQ_EXIT_DATA_LENGTH` Zeichen haben. Standardeinstellung: 32 Leerzeichen.

Exits für Clusterauslastung schreiben und kompilieren

Sie können ein Programm für Exits für Clusterauslastung schreiben, um das Auslastungsmanagement von Clustern anzupassen. Beim Weiterleiten von Nachrichten können Sie das Verwenden eines Kanals zu unterschiedlichen Tageszeiten oder das Verwenden von Nachrichteninhalten in Betracht ziehen. Diese Faktoren werden vom Standardalgorithmus zum Workload-Management nicht berücksichtigt.

In den meisten Fällen reicht der Algorithmus für das Workload-Management für Ihre Anforderungen aus. IBM MQ enthält jedoch einen Benutzerexit, den Exit für Clusterauslastung, damit Sie Ihr eigenes Benutzerexitprogramm verwenden und auf Ihr Workload-Management anpassen können.

Es stehen Ihnen möglicherweise einige spezielle Informationen zu Ihren Netz oder Ihren Nachrichten zur Verfügung, mit denen Sie den Lastausgleich beeinflussen können. Sie wissen vielleicht, welche Kanäle eine hohe Speicherkapazität haben oder welche Netzrouten billiger sind, oder Sie möchten Nachrichten in Abhängigkeit ihres Inhalts weiterleiten. Sie könnten ein Programm für den Exit für Clusterauslastung schreiben oder ein von einem anderen Anbieter bereitgestelltes Programm verwenden.

Der Exit für Clusterauslastung wird beim Zugriff auf eine Clusterwarteschlange aufgerufen. Er wird von `MQOPEN`, `MQPUT1` und `MQPUT` aufgerufen.

Der zum Zeitpunkt `MQOPEN` ausgewählte Zielwarteschlangenmanager wird festgelegt, wenn `MQOO_BIND_ON_OPEN` angegeben ist. In diesem Fall wird der Exit nur einmal ausgeführt.

Wenn der Zielwarteschlangenmanager zum Zeitpunkt MQOPEN nicht festgelegt ist, wird er beim Aufruf von MQPUT ausgewählt. Wenn der Zielwarteschlangenmanager nicht verfügbar ist oder fehlschlägt, während sich die Nachricht noch in der Übertragungswarteschlange befindet, wird der Exit erneut aufgerufen. Es wird ein neuer Zielwarteschlangenmanager ausgewählt. Wenn der Nachrichtenkanal beim Übertragen der Nachricht fehlschlägt und die Nachricht zurückgesetzt wird, wird ein neuer Zielwarteschlangenmanager ausgewählt.

Multi Auf Multiplatforms-Systemen lädt der Warteschlangenmanager den neuen Exit für Clusterauslastung beim nächsten Start des Warteschlangenmanagers.

Wenn die Warteschlangenmanagerdefinition keinen Programmnamen für den Exit für Clusterauslastung enthält, wird der Exit für Clusterauslastung nicht aufgerufen.

In der Parameterstruktur MQWXP des Exits werden verschiedene Daten an einen Exit für Clusterauslastung übertragen:

- Die Struktur der Nachrichtendefinition (MQMD).
- Der Längenparameter für die Nachricht.
- Eine Kopie der Nachricht oder ein Teil der Nachricht.

Wenn Sie auf anderen Plattformen als z/OS das Attribut CLWLMode=FAST verwenden, lädt jeder Betriebssystemprozess seine eigene Kopie des Exits. Unterschiedliche Verbindungen zum Warteschlangenmanager können dazu führen, dass unterschiedliche Kopien des Exits aufgerufen werden. Wenn der Exit im abgesicherten Standardmodus (CLWLMode=SAFE) ausgeführt wird, wird eine einzelne Kopie des Exits in einem eigenen separaten Prozess ausgeführt.

Exit für Clusterauslastung schreiben

z/OS Informationen zum Schreiben von Exits für Clusterauslastung für z/OS finden Sie unter „Cluster workload exit programming for IBM MQ for z/OS” auf Seite 1051.

Ab IBM MQ 9.1.0 werden Exits für Clusterauslastung im Adressraum des Kanalinitiators ausgeführt, nicht im Adressraum des Warteschlangenmanagers. Wenn ein Exit für Clusterauslastung vorhanden ist, sollten Sie die CSQXLIB DD-Anweisung aus der Prozedur der gestarteten Task Ihres Warteschlangenmanagers entfernen und das Dataset, das den Exit für Clusterauslastung enthält, zur CSQXLIB-Verkettung in der Prozedur der gestarteten Task Ihres Kanalinitiators hinzufügen.

Multi Auf Multiplatforms-Systemen dürfen Exits für Clusterauslastung keine MQI-Aufrufe verwenden. Ansonsten entsprechen die Regeln zum Schreiben und Kompilieren von Programmen für den Exit für Clusterauslastung den Regeln, die für Kanalexitprogramme angewendet werden. Folgen Sie den Schritten im Abschnitt „Exits und installierbare Services unter AIX, Linux, and Windows schreiben” auf Seite 986 und verwenden Sie das Beispielprogramm „Beispielprogramm für Exit für Clusterauslastung” auf Seite 1050, das Sie beim Schreiben und Kompilieren Ihres Exits unterstützt.

Weitere Informationen zu Kanalexits finden Sie im Abschnitt „Kanalexitprogramme schreiben” auf Seite 1016.

Exits für Clusterauslastung konfigurieren

Sie benennen Exits für Clusterauslastung in der Warteschlangenmanagerdefinition, indem Sie das Attribut für den Exit für Clusterauslastung im Befehl ALTER QMGR angeben. For example:

```
ALTER QMGR CLWLEXIT(myexit)
```

Zugehörige Verweise

Aufruf des Exits für Clusterauslastung und Datenstrukturen

Beispielprogramm für Exit für Clusterauslastung

IBM MQ enthält ein Beispielprogramm für den Exit für Clusterauslastung. Sie können das Beispiel kopieren und es als Basis für Ihre eigenen Programme verwenden.

z/OS **IBM MQ for z/OS**

Das Beispielexitprogramm für Clusterauslastung wird in Assembler und in Cbereitgestellt. Die Assemblerversion heißt CSQ4BAF1 und befindet sich in der Bibliothek `thlqual.SCSQASMS`. Die Version für die Programmiersprache C heißt CSQ4BCF1 und befindet sich in der Bibliothek `thlqual.SCSQC37S`. `thlqual` ist das übergeordnete Qualifikationsmerkmal für die Zielbibliothek für IBM MQ-Datensätze in Ihrer Installation.

Multi **IBM MQ for Multiplatforms**

Das Beispielprogramm für den Exit für Clusterauslastung wird in der Programmiersprache C bereitgestellt und hat die Bezeichnung `amqswlm0.c`. Es befindet sich in folgender Position:

Tabelle 144. Speicherposition des Beispielprogramms für den Exit für Clusterauslastung auf Multiplatforms-Systemen

Plattform	**Dateipfad**
AIX AIX	*MQ_INSTALLATION_PATH*`/samp`
Windows Windows	*MQ_INSTALLATION_PATH* `\Tools\c\Samples`
IBM i IBM i	Bibliothek `qmqm`

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Mit diesem Beispielexit werden alle Nachrichten an einen bestimmten Warteschlangenmanager weitergeleitet, es sei denn, der Warteschlangenmanager ist nicht mehr verfügbar. Wenn der Warteschlangenmanager fehlschlägt, werden Nachrichten an einen anderen Warteschlangenmanager weitergeleitet.

Zeigen Sie an, an welchen Warteschlangenmanager Nachrichten gesendet werden sollen. Geben Sie den Namen des Clusterempfängerkanals in der Warteschlangenmanagerdefinition mit dem Attribut `CLWLDATA` an. For example:

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

Zum Aktivieren des Exits geben Sie den zugehörigen vollständigen Pfad und den Namen im Attribut `CLWLEXIT` an:

Linux AIX Unter AIX and Linux:

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows Unter Windows:

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS Unter z/OS:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

dabei ist x entweder 'A' oder 'C', abhängig von der Programmiersprache der von Ihnen verwendeten Version.

IBM i Verwenden Sie unter IBM i einen der folgenden Befehle:

- Syntax des MQSC-Befehls:

```
ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
```

  Der Programmname und der Bibliotheksname nutzen jeweils 10 Zeichen und werden bei Bedarf auf der rechten Seite mit Leerzeichen aufgefüllt.

- Syntax des CL-Befehls:

```
CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
```

IBM MQ verwendet jetzt nicht mehr den bereitgestellten Algorithmus zum Auslastungsmanagement, sondern ruft diesen Exit auf, um alle Nachrichten an den von Ihnen ausgewählten Warteschlangenmanager weiterzuleiten.

z/OS *Cluster workload exit programming for IBM MQ for z/OS*

Cluster workload exits are invoked as if by a z/OS **LINK** command. Exits are subject to a number of stringent programming rules. Avoid using most SVC commands that involve waits, or using a STAE or ESTAE in a workload exit.

Cluster workload exits are invoked as if by a z/OS **LINK** in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31 bit addressing mode
- Storage key 8
- Program Key Mask 8
- TCB key 8

Put the link-edited modules in the data set specified by the CSQXLIB DD statement of the started task procedure of the channel initiator. The names of the load modules are specified as the workload exit names in the queue manager definition.

When writing workload exits for IBM MQ for z/OS, the following rules apply:

- You must write exits in assembler or C. If you use C, it must conform to the C systems programming environment for system exits, described in the *z/OS C/C++ Programming Guide*, SC09-4765.
- If using the MQXCLWLN call, link edit with CSQMFCLW, supplied in *thlqual*.SCSQLOAD.
- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the queue manager is running, with the new version used in the next MQCONN thread the queue manager starts.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment on return to that at entry.
- Exits must free any storage obtained, or ensure that storage is freed by a subsequent exit invocation.
- No MQI calls are allowed.
- Exits must not use any system services that could cause a wait, because a wait severely degrades the performance of the queue manager. In general, therefore, avoid an SVC, PC, or I/O.
- Exits must not issue an ESTAE or SPIE, apart from within any subtasks they attach.

Note: There are no absolute restrictions on what you can do in an exit. However, most SVCs involve waits, so avoid them, except for the following commands:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

Do not use ESTAEs and ESPIEs because their error handling might interfere with the error handling performed by IBM MQ. IBM MQ might not be able to recover from an error, or your exit program might not receive all the error information.

The system parameter EXITLIM limits the amount of time an exit might run for. The default value for EXITLIM is 30 seconds. If you see the return code MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA' your exit might be looping. If you think the exit needs more than 30 seconds to complete, increase the value of EXITLIM.

Prozedurale Anwendung erstellen

Sie können eine IBM MQ-Anwendung in einer von mehreren prozeduralen Sprachen schreiben und die Anwendung auf mehreren unterschiedlichen Plattformen ausführen.

AIX Prozedurale Anwendung auf AIX erstellen

In den AIX-Veröffentlichungen ist beschrieben, wie Sie aus den von Ihnen geschriebenen Programmen ausführbare Anwendungen erstellen können.

In diesem Abschnitt werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie beim Erstellen von IBM MQ for AIX-Anwendungen für die Ausführung unter AIX durchführen müssen. Unterstützt werden C, C++ und COBOL. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter C++ verwenden.

Die Tasks, die Sie zum Erstellen einer ausführbaren Anwendung mithilfe von IBM MQ for AIX ausführen müssen, sind von der Programmiersprache abhängig, in der Ihr Quellcode geschrieben ist. Zusätzlich zur Codierung der MQI-Aufrufe in Ihrem Quellcode müssen Sie die entsprechenden Sprachanweisungen hinzufügen, um die IBM MQ for AIX-Include-Datei für die von Ihnen verwendete Sprache einzuschließen. Machen Sie sich mit den Inhalten dieser Dateien vertraut. Eine vollständige Beschreibung finden Sie unter „IBM MQ-Datendefinitionsdateien” auf Seite 755.

Legen Sie bei Ausführung eines Thread-Servers oder einer Thread-Clientanwendung die Umgebungsvariable THREAD_SCOPE=S fest.

AIX *C-Programme in AIX vorbereiten*

In diesem Abschnitt finden Sie Informationen zum Verknüpfen von Bibliotheken, die für das Vorbereiten von C-Programmen unter AIX erforderlich sind.

Vorkompilierte C-Programme werden im Verzeichnis *MQ_INSTALLATION_PATH*/samp/bin bereitgestellt. Verwenden Sie den ANSI-Compiler und führen Sie die folgenden Befehle aus. Weitere Informationen zur Programmierung von 64-Bit-Anwendungen finden Sie im Abschnitt Codierungsstandards auf 64-Bit-Plattformen.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Für 32-Bit-Anwendungen:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

Dabei ist amqsput0 ein Beispielprogramm.

Für 64-Bit-Anwendungen:

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

Dabei ist `amqsput0` ein Beispielprogramm.

V 9.4.0 Für 32-Bit-Anwendungen mit XLC 17 -Compiler:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

Dabei ist `amqsput0` ein Beispielprogramm.

V 9.4.0 Für 64-Bit-Anwendungen mit XLC 17 -Compiler:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

Dabei ist `amqsput0` ein Beispielprogramm.

Wenn Sie den VisualAge C/C++-Compiler für C++-Programme verwenden, müssen Sie die Option `-q na-memangling=v5` einschließen, damit beim Verknüpfen der Bibliotheken alle IBM MQ-Symbole aufgelöst werden.

Wenn Sie die Programme auf einem System verwenden möchten, auf dem nur der IBM MQ MQI client for AIX installiert ist, kompilieren Sie stattdessen nur die Programme erneut, um diese mit der Clientbibliothek (`-lmqic`) zu verknüpfen.

Bibliotheken verknüpfen

Sie benötigen die folgenden Bibliotheken:

- Verknüpfen Sie Ihre Programme mit der entsprechenden Bibliothek, die von IBM MQ bereitgestellt wird.

  Stellen Sie in einer Umgebung, bei der es sich nicht um eine Threadumgebung handelt, eine Verknüpfung zu einer der folgenden Bibliotheken her:

Bibliotheksdatei	**Programm/Exit-Typ**
libmqm.a	Server für C
libmqic.a & libmqm.a	Client für C

  Stellen Sie in einer Threadumgebung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Bibliotheksdatei	**Programm/Exit-Typ**
libmqm_r.a	Server für C
libmqic_r.a & libmqm_r.a	Client für C

  Zum Erstellen einer einfachen IBM MQ-Thread-Anwendung aus einer einzelnen Kompiliereinheit führen Sie beispielsweise die folgenden Befehle aus.

  Für 32-Bit-Anwendungen:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

  Dabei ist `amqsput0` ein Beispielprogramm.

  Für 64-Bit-Anwendungen:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

  Dabei ist `amqsput0` ein Beispielprogramm.

  V 9.4.0 Für 32-Bit-Anwendungen mit XLC 17 -Compiler:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib -lmqm_r
```

Dabei ist `amqsput0` ein Beispielprogramm.

V 9.4.0 Für 64-Bit-Anwendungen mit XLC 17 -Compiler:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib64 -lmqm_r
```

Dabei ist `amqsput0` ein Beispielprogramm.

Wenn Sie die Programme auf einem System verwenden möchten, auf dem nur der IBM MQ MQI client for AIX installiert ist, kompilieren Sie stattdessen nur die Programme erneut, um diese mit der Clientbibliothek (`-lmqic`) zu verknüpfen.

Anmerkung:

1. Verknüpfungen mit mehreren Bibliotheken sind nicht möglich. Sie können also beispielsweise nicht gleichzeitig eine Verknüpfung mit einer Bibliothek mit Threads und einer Bibliothek ohne Threads herstellen.
2. Wenn Sie einen installierbaren Service schreiben (weitere Informationen dazu finden Sie unter IBM MQ verwalten), müssen Sie eine Verknüpfung zu der Bibliothek `libmqmzf.a` in einer anderen Anwendung als einer Thread-Anwendung und zu der Bibliothek `libmqmzf_r.a` in einer Thread-Anwendung herstellen.
3. Wenn Sie eine Anwendung für die externe Koordination durch einen XA-konformen Transaktionsmanager (z. B. IBM TXSeries, Encina oder BEA Tuxedo) erstellen, müssen Sie eine Verknüpfung zu den Bibliotheken `libmqmxa.a` (oder `libmqmxa64.a`, falls Ihr Transaktionsmanager den Typ 'long' als 64-Bit behandelt) und `libmqz.a` in einer anderen Anwendung als einer Thread-Anwendung bzw. zu den Bibliotheken `libmqmxa_r.a` (oder `libmqmxa64_r.a`) und `libmqz_r.a` in einer Thread-Anwendung herstellen.
4. Sie müssen vertrauenswürdige Anwendungen mit den IBM MQ-Threadbibliotheken verknüpfen. Auf IBM MQ for AIX or Linux-Systemen kann allerdings jeweils nur eine Verbindung zu einem Thread in einer vertrauenswürdigen Anwendung hergestellt werden.
5. Sie müssen IBM MQ-Bibliotheken vor allen anderen Produktbibliotheken verknüpfen.

AIX *COBOL-Programme in AIX vorbereiten*

Verwenden Sie diese Informationen, wenn Sie COBOL-Programme in AIX mithilfe von IBM COBOL Set und Micro Focus COBOL vorbereiten.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

- COBOL-Copybooks im 32-Bit-Format sind im folgenden Verzeichnis installiert:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

  Symbolische Verbindungen werden im folgenden Verzeichnis erstellt:

```
MQ_INSTALLATION_PATH/inc
```

- COBOL-Copybooks im 64-Bit-Format sind im folgenden Verzeichnis installiert:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

In den folgenden Beispielen wird die Umgebungsvariable **COBCPY** wie folgt gesetzt:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

für 32-Bit-Anwendungen und auf

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

für 64-Bit-Anwendungen.

Sie müssen Ihr Programm mit einer der folgenden Bibliotheksdateien verknüpfen:

Bibliotheksdatei	Programm/Exit-Typ
libmqmcb.a	Server für COBOL (Anwendung ohne Threads)
libmqmcb_r.a	Server für COBOL (Thread-Anwendung)
libmqicb.a	Client für COBOL (Anwendung ohne Threads)
libmqicb_r.a	Client für COBOL (Thread-Anwendung)

Abhängig vom Programm können Sie den IBM COBOL Set-Compiler oder den Micro Focus COBOL-Compiler verwenden:

- Programme, die mit `amqm` beginnen, sind für den Micro Focus COBOL-Compiler geeignet, und
- Programme, die mit `amq0` beginnen, sind für beide Compiler geeignet.

COBOL-Programme mithilfe von IBM COBOL Set für AIX vorbereiten

COBOL-Beispielprogramme werden mit IBM MQ geliefert. Zum Kompilieren eines solchen Programms geben Sie den entsprechenden Befehl aus der folgenden Liste ein:

32-Bit-Serveranwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

32-Bit-Clientanwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

32-Bit-Serveranwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

32-Bit-Clientanwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

64-Bit-Serveranwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

64-Bit-Clientanwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

64-Bit-Serveranwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

64-Bit-Clientanwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

COBOL-Programme mithilfe von Micro Focus COBOL vorbereiten

Vor dem Kompilieren Ihres Programms legen Sie Umgebungsvariablen folgendermaßen fest:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Zum Kompilieren eines 32-Bit-COBOL-Programms mithilfe von Micro Focus COBOL geben Sie Folgendes ein:

- Server für COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- Client für COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- Thread-Server für COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- Thread-Client für COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Zum Kompilieren eines 64-Bit-COBOL-Programms mithilfe von Micro Focus COBOL geben Sie Folgendes ein:

- Server für COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- Client für COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- Thread-Server für COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- Thread-Client für COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

Dabei steht `amqminqx` für ein Beispielprogramm.

In der Dokumentation zu Micro Focus COBOL finden Sie eine Beschreibung der Umgebungsvariablen, die Sie einrichten müssen.

AIX *CICS-Anwendungsprogramme in AIX vorbereiten*

Verwenden Sie diese Informationen, wenn Sie CICS-Programme in AIX vorbereiten.

Verwenden Sie die Module für den *XA-Switch*, um CICS mit IBM MQ zu verknüpfen. Weitere Informationen zur Struktur des XA-Switch finden Sie unter Strukturen des XA -Switch.

Es wird die Beispieldatei für den Quellcode bereitgestellt, damit Sie die XA-Switches für andere Transaktionsnachrichten entwickeln können. Der Name des bereitgestellten Switchloadmoduls wird unter Tabelle 145 auf Seite 1057 aufgeführt.

Tabelle 145. Essenzieller Code für CICS-Anwendungsprogramme unter AIX: XA-Initialisierungsroutine

Beschreibung	**C (Quelle)**	**C (exec)-zu XAD.Stanza hinzufügen**
XA-Initialisierungsroutine	amqzscix.c	*amqzsc* - CICS für AIX

Verwenden Sie die vordefinierte Version der IBM MQ-Switchloaddatei *amqzsc*, die mit dem Produkt bereitgestellt wird.

Verbinden Sie Ihre C-Transaktionen immer mit der threadsicheren IBM MQ -Bibliothek *libmqm_r.a.*, und Ihre COBOL-Transaktionen mit der COBOL-Bibliothek *libmqmcb_r.a.*.

Weitere Informationen zu unterstützenden CICS-Transaktionen finden Sie im IBM MQ verwalten IBM MQ-Handbuch zur Systemverwaltung.

AIX *Unterstützung für TXSeries CICS*

IBM MQ on AIX unterstützt TXSeries CICS mithilfe der XA-Schnittstelle. Stellen Sie sicher, dass CICS-Anwendungen mit der Thread-Version der IBM MQ-Bibliotheken verknüpft sind.

Sie können CICS-Programme mithilfe von IBM COBOL Set for AIX oder Micro Focus COBOL ausführen. In den folgenden Abschnitten wird der Unterschied zwischen der Ausführung von CICS-Programmen unter IBM COBOL Set for AIX und Micro Focus COBOL beschrieben.

Schreiben Sie IBM MQ-Programme, die in die gleiche CICS-Region geladen werden, in C oder COBOL. Sie können C- und COBOL MQI-Aufrufe in der gleichen CICS-Region nicht kombinieren. Die meisten MQI-Aufrufe, die in der zweiten Sprache verwendet wurden, schlagen mit dem Ursachencode `MQRC_HOBJ_ERROR` fehl.

CICS-COBOL-Programme mithilfe von IBM COBOL Set for AIX vorbereiten

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Gehen Sie zur Verwendung von IBM COBOL folgendermaßen vor:

1. Exportieren Sie die folgende Umgebungsvariable:

   ```
   export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                   -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                   -e _iwz_cobol_main  \
   ```

   Dabei ist `LIB` eine Compilersteueranweisung.
2. Übersetzen, kompilieren und verknüpfen Sie das Programm, indem Sie Folgendes eingeben:

   ```
   cicstcl -l IBMCOB yourprog.ccp
   ```

CICS COBOL-Programme mithilfe von Micro Focus COBOL vorbereiten

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Gehen Sie zur Verwendung von Micro Focus COBOL folgendermaßen vor:

1. Fügen Sie das Modul für die IBM MQ COBOL-Laufzeitbibliothek mit folgendem Befehl zur Laufzeitbibliothek hinzu:

   ```
   cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Anmerkung: Bei Verwendung von `cicsmkcobol` ist es in IBM MQ nicht möglich, MQI-Aufrufe in der Programmiersprache C aus Ihrer COBOL-Anwendung vorzunehmen.

   Wenn Ihre vorhandene Anwendung solche Aufrufe enthält, wird empfohlen, diese Funktionen aus den COBOL-Anwendungen in Ihre eigene Bibliothek (z. B. `myMQ.so`) zu verschieben. Nach dem Verschieben der Funktionen schließen Sie die IBM MQ-Bibliothek `libmqmcbrt.o` beim Erstellen der COBOL-Anwendung für CICS nicht ein.

   Wenn Ihre COBOL-Anwendung keine COBOL MQI-Aufrufe vornimmt, stellen Sie außerdem keine Verknüpfung zwischen `libmqmz_r` und `cicsmkcobol` her.

   Dadurch wird die Methodendatei für die Micro Focus COBOL-Sprache erstellt und ermöglicht der CICS COBOL-Laufzeitbibliothek das Aufrufen von IBM MQ for AIX or Linux-Systemen.

   Anmerkung: Führen Sie `cicsmkcobol` nur aus, wenn Sie eines der folgenden Produkte installieren:

   - Neue Version oder neues Release von Micro Focus COBOL
   - Neue Version oder neues Release von CICS for AIX
   - Neue Version oder neues Release eines beliebigen unterstützten Datenbankprodukts (nur für COBOL-Transaktionen)
   - Neue Version oder neues Release von IBM MQ
2. Exportieren Sie die folgende Umgebungsvariable:

   ```
   COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
   ```
3. Übersetzen, kompilieren und verknüpfen Sie das Programm, indem Sie Folgendes eingeben:

   ```
   cicstcl -l COBOL -e yourprog.ccp
   ```

CICS C-Programme vorbereiten

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Erstellen Sie CICS C-Programme mithilfe der CICS-Standardfunktionen:

1. Exportieren Sie ***eine*** der folgenden Umgebungsvariablen:
   - LDFLAGS = "-L/ *MQ_INSTALLATION_PATH* Bibliothek -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* Bibliothek -lmqm_r" export USERLIB
2. Übersetzen, kompilieren und verknüpfen Sie das Programm, indem Sie Folgendes eingeben:

   ```
   cicstcl -l C amqscic0.ccs
   ```

CICS C-Beispieltransaktion

Ein Beispiel für einen C-Quellcode für eine AIX IBM MQ-Transaktion wird in AMQSCIC0.CCS bereitgestellt. Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE im Standardwarteschlangenmanager und stellt sie in die lokale Warteschlan-

ge mit einem Warteschlangennamen, der im Übertragungsheader der Nachricht enthalten ist. Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICSgesendet.DLQ Erstellen Sie mithilfe des MQSC-Beispielscripts AMQSCIC0.TST diese Warteschlangen und die Beispieleingabewarteschlangen.

IBM i Prozedurale Anwendung auf IBM i erstellen

In den IBM i-Veröffentlichungen wird beschrieben, wie Sie aus den von Ihnen geschriebenen Programmen ausführbare Anwendungen erstellen, die mit IBM i auf iSeries- oder System i-Systemen ausgeführt werden können.

In diesem Abschnitt werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie beim Erstellen von prozeduralen Anwendungen für IBM MQ for IBM i zur Ausführung auf IBM i-Systemen durchführen müssen. Die Programmiersprachen COBOL, C, C++, Java und RPG werden unterstützt. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter C++ verwenden. Informationen zur Vorbereitung Ihrer Java-Programme finden Sie unter IBM MQ classes for Java verwenden.

Die Tasks, die Sie zum Erstellen einer ausführbaren IBM MQ for IBM i-Anwendung ausführen müssen, sind von der Programmiersprache abhängig, in der der Quellcode geschrieben ist. Zusätzlich zur Codierung der MQI-Aufrufe in Ihrem Quellcode müssen Sie die entsprechenden Sprachanweisungen hinzufügen, um die IBM MQ for IBM i-Datendefinitionsdateien für die von Ihnen verwendete Sprache einzuschließen. Machen Sie sich mit den Inhalten dieser Dateien vertraut. Eine vollständige Beschreibung finden Sie unter „IBM MQ-Datendefinitionsdateien” auf Seite 755.

IBM i *C-Programme in IBM i vorbereiten*

In IBM MQ for IBM i werden Nachricht mit einer Größe von bis zu 100 MB unterstützt. Anwendungsprogramme, die in ILE C geschrieben sind und IBM MQ-Nachrichten mit einer Größe von mehr als 16 MB unterstützten, müssen die Compileroption `Teraspace` verwenden, um genügend Speicherplatz für diese Nachrichten zuzuordnen.

Weitere Informationen zu den Optionen des C-Compilers finden Sie im Handbuch *WebSphere Development Studio ILE C/C++ Programmer's Guide*.

Zum Kompilieren eines Moduls für die Programmiersprache C können Sie den IBM i-Befehl **CRTCMOD** verwenden. Stellen Sie sicher, dass sich die Bibliothek mit den Include-Dateien (QMQM) beim Kompilieren in der Bibliotheksliste befindet.

Anschließend müssen Sie die Ausgabe des Compilers mithilfe des Befehls **CRTPGM** mit dem Serviceprogramm verbinden.

Tabelle 146. Beispiel für CRTPGM in Umgebungen mit und ohne Threads

Umgebungstyp	Befehl	Programm/Exit-Typ
Umgebung ohne Threads	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM)`	Server oder Client für C
Umgebung mit Threads	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM_R)`	Server oder Client für C

Dabei ist *pgmname* der Name Ihres Programms.

Tabelle 147 auf Seite 1060 listet die Bibliotheken auf, die beim Vorbereiten von C-Programmen unter IBM i in einer Umgebung ohne Threads und einer Umgebung mit Threads erforderlich sind.

Tabelle 147. Benötigte Bibliotheken für Umgebungen mit und ohne Threads		
Umgebungstyp	**Bibliotheksdatei**	**Programm/Exit-Typ**
Umgebung ohne Threads	LIBMQM	Server für C
	LIBMQIC und LIBMQM	Client für C
Umgebung mit Threads	LIBMQM_R	Server für C
	LIBMQIC_R und LIBMQM_R	Client für C

IBM i *COBOL-Programme in IBM i vorbereiten*

Hier finden Sie Informationen zur Vorbereitung von COBOL-Programmen in IBM i und erfahren, wie Sie aus dem COBOL-Programm auf die MQI zugreifen können.

Informationen zu diesem Vorgang

Für den Zugriff auf die MQI von COBOL-Programmen aus bietet IBM MQ for IBM i eine gebundene prozedurale Aufrufschnittstelle, die von Serviceprogrammmen bereitgestellt wird. Diese ermöglicht Zugriff auf alle MQI-Funktionen in IBM MQ for IBM i und bietet Unterstützung für Thread-Anwendungen. Die Schnittstelle kann nur mit dem ILE COBOL-Compiler verwendet werden.

Der Zugriff auf die MQI-Funktionen erfolgt mit der standardmäßigen COBOL-CALL-Syntax.

Die COBOL-Kopierdateien mit den benannten Konstanten und Strukturdefinitionen für die Verwendung mit der MQI befinden sich in der physische Quellendatei QMQM/QCBLLESRC.

Die COBOL-Kopierdateien verwenden ein einfaches Anführungszeichen (') als Zeichenfolgebegrenzer. Die COBOL-Compiler für IBM i setzen ein Anführungszeichen (") als Begrenzer voraus. Um zu verhindern, dass die Compiler Warnhinweise generieren, geben Sie `OPTION(*APOST)` in den Befehlen **CRTCBLPGM**, **CRTBNDCBL** oder **CRTCBLMOD** an.

Verwenden Sie die Compileroption \APOST, damit der Compiler das einfache Anführungszeichen (') als Zeichenfolgebegrenzer in den COBOL-Kopierdateien akzeptiert.

Anmerkung: In IBM MQ 9.0 oder höher wird die dynamische Aufrufschnittstelle nicht bereitgestellt.

Gehen Sie wie folgt vor, um die gebundene prozedurale Aufrufschnittstelle zu verwenden:

Vorgehensweise

1. Erstellen Sie mit dem Compiler **CRTCBLMOD** ein Modul und geben Sie dabei den folgenden Parameter an:

   ```
   LINKLIT(*PRC)
   ```

2. Erstellen Sie mit dem Befehl **CRTPGM** das Programmobjekt und geben Sie dabei den entsprechenden Parameter an:

   Für Nicht-Thread-Anwendungen:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
   ```

   For Thread-Anwendungen:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
   ```

   Anmerkung: Mit Ausnahme von Programmen, die mithilfe des V4R4 ILE COBOL-Compilers erstellt wurden und die Option THREAD(SERIALIZE) in der Anweisung PROCESS enthalten, dürfen die IBM

MQ-Threadbibliotheken in COBOL-Programmen nicht verwendet werden. Selbst wenn ein COBOL-Programm auf diese Weise threadsicher erstellt wurde, müssen Sie beim Entwickeln der Anwendung sorgfältig vorgehen, da mit THREAD(SERIALIZE) die Serialisierung von COBOL-Prozeduren auf Modulebene erzwungen wird und dies die Gesamtleistung beeinträchtigen kann.

Weitere Informationen finden Sie in den Handbüchern *WebSphere Development Studio: ILE COBOL Programmer's Guide* und *WebSphere Development Studio: ILE COBOL Reference*.

Weitere Informationen zum Kompilieren einer CICS-Anwendung finden Sie im Handbuch *CICS for IBM i Application Programming Guide*, SC41-5454.

IBM i *CICS-Programme in IBM i vorbereiten*

Hier finden Sie Informationen zu den Schritten, die bei der Vorbereitung von CICS-Programmen in IBM i erforderlich sind.

Führen Sie die folgenden Schritte aus, um ein Programm zu erstellen, das EXEC CICS-Anweisungen und MQI-Aufrufe enthält:

1. Erstellen Sie bei Bedarf Zuordnungen mithilfe des Befehls CRTCICSMAP.
2. Übersetzen Sie die EXEC CICS-Befehle in Anweisungen der nativen Programmiersprache. Verwenden Sie den Befehl CRTCICSC für ein C-Programm. Verwenden Sie den Befehl CRTCICSCBL für ein COBOL-Programm.

   Schließen Sie `CICSOPT(*NOGEN)` in den Befehl CRTCICSC bzw. CRTCICSCBL ein. Dadurch wird die Verarbeitung gestoppt und Sie können die entsprechenden CICS- und IBM MQ-Serviceprogramme einschließen. Dieser Befehl stellt den Code standardmäßig in QTEMP/QACYCICS.
3. Kompilieren Sie den Quellcode mithilfe des Befehls CRTCMOD (für ein C-Programm) oder mithilfe des Befehls CRTCBLMOD (für ein COBOL-Programm).
4. Verwenden Sie den Befehl CRTPGM, um den kompilierten Code mit den entsprechenden CICS- und IBM MQ-Serviceprogrammen zu verbinden. Dadurch wird das ausführbare Programm erstellt.

Im Anschluss finden Sie ein Beispiel eines solchen Codes (dabei wird das gelieferte CICS-Beispielprogramm kompiliert):

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i *RPG-Programme in IBM i vorbereiten*

Wenn Sie IBM MQ for IBM i verwenden, können Sie Ihre Anwendungen in RPG schreiben.

Weitere Informationen finden Sie in den Abschnitten „IBM MQ-Programme in RPG codieren (nur IBM i)” auf Seite 1110 und Referenzinformationen zur Anwendungsprogrammierung für IBM i (ILE/RPG).

IBM i *Überlegungen zur SQL-Programmierung für IBM i*

Hier finden Sie Informationen zu den Schritten, die bei der Erstellung einer Anwendung in IBM i mithilfe von SQL erforderlich sind.

Wenn Ihr Programm EXEC SQL-Anweisungen und MQI-Aufrufe enthält, führen Sie die folgenden Schritte aus:

1. Übersetzen Sie die EXEC SQL-Befehle in Anweisungen der nativen Programmiersprache. Verwenden Sie den Befehl CRTSQLCI für ein Programm in der Programmiersprache C. Verwenden Sie den Befehl CRTSQLCBLI für ein COBOL-Programm.

Schließen Sie `OPTION(*NOGEN)` in den Befehl CRTSQLCI oder CRTSQLCBLI ein. Dadurch wird die Verarbeitung gestoppt und Sie können die entsprechenden Programme des IBM MQ-Service einschließen. Mit diesem Befehl wird der Code standardmäßig in QTEMP/QSQLTEMP integriert.

2. Kompilieren Sie den Quellcode mithilfe des Befehls CRTCMOD (für ein C-Programm) oder mithilfe des Befehls CRTCBLMOD (für ein COBOL-Programm).
3. Verwenden Sie den Befehl CRTPGM, um den kompilierten Code mit den entsprechenden Programmen des IBM MQ-Service zu verbinden. Dadurch wird das ausführbare Programm erstellt.

Im Anschluss finden Sie ein Beispiel eines solchen Codes (Programm SQLTEST wird in Bibliothek SQLUSER kompiliert):

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux Prozedurale Anwendung auf Linux erstellen

In diesen Informationen werden die zusätzlichen Tasks sowie die Änderungen an den Standardtasks beschrieben, die Sie bei der Erstellung von IBM MQ for Linux-Anwendung für die Ausführung vornehmen müssen.

C und C++ werden unterstützt. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter C++ verwenden.

Linux *C-Programme in Linux vorbereiten*

Vorkompilierte C-Programme werden im Verzeichnis *MQ_INSTALLATION_PATH*`/samp/bin` bereitgestellt. Verwenden Sie den Compiler `gcc`, um ein Beispiel aus einem Quellcode zu erstellen.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Arbeiten Sie in Ihrer normalen Umgebung. Weitere Informationen zur Programmierung von 64-Bit-Anwendungen finden Sie unter Codierungsstandards auf 64-Bit-Plattformen.

Bibliotheken verknüpfen

In der folgenden Tabelle werden die Bibliotheken aufgeführt, die beim Vorbereiten von C-Programmen unter Linux erforderlich sind.

- Sie müssen Ihre Programme mit der entsprechenden Bibliothek verknüpfen, die von IBM MQ bereitgestellt wird.

  Stellen Sie in einer Umgebung ohne Threads eine Verknüpfung zu einer einzigen der folgenden Bibliotheken her:

Bibliotheksdatei	Programm/Exit-Typ
libmqm.so	Server für C
libmqic.so & libmqm.so	Client für C

  Stellen Sie in einer Threadumgebung eine Verknüpfung zu einer einzigen der folgenden Bibliotheken her:

Bibliotheksdatei	Programm/Exit-Typ
libmqm_r.so	Server für C
libmqic_r.so & libmqm_r.so	Client für C

Anmerkung:

1. Verknüpfungen mit mehreren Bibliotheken sind nicht möglich. Sie können also beispielsweise nicht gleichzeitig eine Verknüpfung mit einer Bibliothek mit Threads und einer Bibliothek ohne Threads herstellen.
2. Wenn Sie einen installierbaren Service schreiben (weitere Informationen dazu finden Sie unter IBM MQ verwalten), müssen Sie eine Verknüpfung zu der Bibliothek `libmqmzf.so` herstellen.
3. Wenn Sie eine Anwendung für die externe Koordination durch einen XA-konformen Transaktionsmanager (z. B. IBM TXSeries, Encina oder BEA Tuxedo) erstellen, müssen Sie eine Verknüpfung zu den Bibliotheken `libmqmxa.so` (oder `libmqmxa64.so`, falls Ihr Transaktionsmanager den Typ 'long' als 64-Bit behandelt) und `libmqz.so` in einer anderen Anwendung als einer Thread-Anwendung bzw. zu den Bibliotheken `libmqmxa_r.so` (oder `libmqmxa64_r.so`) und `libmqz_r.so` in einer Thread-Anwendung herstellen.
4. Sie müssen IBM MQ-Bibliotheken vor allen anderen Produktbibliotheken verknüpfen.

Linux *31-Bit-Anwendungen erstellen*

In diesem Abschnitt finden Sie Beispiele für die Befehle, die zum Erstellen von 31-Bit-Programmen in verschiedenen Umgebungen verwendet werden.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

C-Clientanwendung, 31-Bit, kein Thread

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C-Clientanwendung, 31-Bit, mit Thread

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C-Serveranwendung, 31-Bit, kein Thread

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C-Serveranwendung, 31-Bit, mit Thread

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++-Clientanwendung, 31-Bit, kein Thread

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

C++-Clientanwendung, 31-Bit, mit Thread

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

C++-Serveranwendung, 31-Bit, kein Thread

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
```

```
-limqs23gl
-limqb23gl -lmqm
```

C++-Serveranwendung, 31-Bit, mit Thread

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

C-Client-Exit, 31-Bit, kein Thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

C-Client-Exit, 31-Bit, mit Thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C-Server-Exit, 31-Bit, kein Thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

C-Server-Exit, 31-Bit, mit Thread

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *32-Bit-Anwendungen erstellen*

In diesem Abschnitt finden Sie Beispiele für die Befehle, die zum Erstellen von 32-Bit-Programmen in verschiedenen Umgebungen verwendet werden.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

C-Clientanwendung, 32-Bit, kein Thread

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C-Clientanwendung, 32-Bit, mit Thread

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C-Serveranwendung, 32-Bit, kein Thread

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C-Serveranwendung, 32-Bit, mit Thread

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++-Clientanwendung, 32-Bit, kein Thread

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

C++-Clientanwendung, 32-Bit, mit Thread

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++-Serveranwendung, 32-Bit, kein Thread

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

C++-Serveranwendung, 32-Bit, mit Thread

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C-Client-Exit, 32-Bit, kein Thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

C-Client-Exit, 32-Bit, mit Thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C-Server-Exit, 32-Bit, kein Thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

C-Server-Exit, 32-Bit, mit Thread

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *64-Bit-Anwendungen erstellen*

In diesem Abschnitt finden Sie Beispiele für die Befehle, die zum Erstellen von 64-Bit-Programmen in verschiedenen Umgebungen verwendet werden.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

C-Clientanwendung, 64-Bit, kein Thread

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

C-Clientanwendung, 64-Bit, mit Thread

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

C-Serveranwendung, 64-Bit, kein Thread

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

C-Serveranwendung, 64-Bit, mit Thread

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

C++-Clientanwendung, 64-Bit, kein Thread

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

C++-Clientanwendung, 64-Bit, mit Thread

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++-Serveranwendung, 64-Bit, kein Thread

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

C++-Serveranwendung, 64-Bit, mit Thread

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C-Client-Exit, 64-Bit, kein Thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

C-Client-Exit, 64-Bit, mit Thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
```

```
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

C-Server-Exit, 64-Bit, kein Thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

C-Server-Exit, 64-Bit, mit Thread

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux *COBOL-Programme in Linux vorbereiten*

Hier erfahren Sie, wie Sie COBOL-Programme in Linux vorbereiten und COBOL-Programme mit IBM COBOL for Linux auf x86 und Micro Focus COBOL vorbereiten.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

1. 32-Bit-COBOL-Copybooks werden im folgenden Verzeichnis installiert:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   Symbolische Verbindungen werden im folgenden Verzeichnis erstellt:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. Auf 64-Bit-Plattformen werden 64-Bit-COBOL-Copybooks im folgenden Verzeichnis installiert:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. Setzen Sie COBCPY in den folgenden Beispielen auf

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   für 32-Bit-Anwendungen und:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   für 64-Bit-Anwendungen.

Sie müssen Ihr Programm mit einer der folgenden Bibliotheksdateien verknüpfen:

Bibliotheksdatei	Programm/Exit-Typ
libmqmcb.so	Server für COBOL
libmqicb.so	Client für COBOL
libmqmcb_r.so	Server für COBOL (Thread-Anwendung)
libmqicb_r.so	Client für COBOL (Thread-Anwendung)

COBOL-Programme mit IBM COBOL for Linux auf x86 vorbereiten

COBOL-Beispielprogramme werden mit IBM MQbereitgestellt. Zum Kompilieren eines solchen Programms geben Sie den entsprechenden Befehl aus der folgenden Liste ein:

32-Bit-Serveranwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

32-Bit-Clientanwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

32-Bit-Thread-Serveranwendung

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

32-Bit-Thread-Clientanwendung

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

COBOL-Programme mithilfe von Micro Focus COBOL vorbereiten

Vor dem Kompilieren Ihres Programms legen Sie Umgebungsvariablen folgendermaßen fest:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Geben Sie Folgendes ein, um ein 32-Bit-COBOL-Programm, sofern unterstützt, mit Micro Focus COBOL zu kompilieren:

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Geben Sie Folgendes ein, um ein 64-Bit-COBOL-Programm mit Micro Focus COBOL zu kompilieren:

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

Dabei steht `amqsput` für ein Beispielprogramm.

In der Dokumentation zu Micro Focus COBOL finden Sie eine Beschreibung der erforderlichen Umgebungsvariablen.

Windows Prozedurale Anwendung auf Windows erstellen

In den Veröffentlichungen zu Windows-Systemen ist beschrieben, wie Sie aus den von Ihnen geschriebenen Programmen ausführbare Anwendungen erstellen können.

In diesem Abschnitt werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie beim Erstellen von IBM MQ for Windows-Anwendungen für die Ausführung auf Windows-Systemen durchführen müssen. Die Programmiersprachen C, C++, COBOL und Visual Basic werden unterstützt. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter C++ verwenden.

Die Tasks, die Sie zum Erstellen einer ausführbaren Anwendung mithilfe von IBM MQ for Windows ausführen müssen, sind von der Programmiersprache abhängig, in der Ihr Quellcode geschrieben ist. Zusätzlich zur Codierung der MQI-Aufrufe in Ihrem Quellcode müssen Sie die entsprechenden Sprachanweisungen hinzufügen, um die IBM MQ for Windows-Include-Datei für die von Ihnen verwendete Sprache

einzuschließen. Machen Sie sich mit den Inhalten dieser Dateien vertraut. Eine vollständige Beschreibung finden Sie unter „IBM MQ-Datendefinitionsdateien” auf Seite 755.

Windows 64-Bit-Anwendungen unter Windows erstellen

Unter IBM MQ for Windows werden 32-Bit- und 64-Bit-Anwendung unterstützt. Die ausführbaren Dateien und Bibliotheksdateien für IBM MQ werden im 32-Bit- und 64-Bit-Format bereitgestellt. Verwenden Sie die Version, die für die von Ihnen verwendete Anwendung geeignet ist.

Ausführbare Dateien und Bibliotheken

Die 32-Bit- und 64-Bit-Versionen der IBM MQ-Bibliotheken werden in den folgenden Positionen bereitgestellt:

Tabelle 148. Position der IBM MQ-Bibliotheken

Bibliotheksversion	**Verzeichnis mit den Bibliotheksdateien**
32 Bit	*MQ_INSTALLATION_PATH* `\Tools\Lib`
64 Bit	*MQ_INSTALLATION_PATH* `\Tools\Lib64`

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

32-Bit-Anwendungen können nach der Migration weiterhin normal ausgeführt werden. Die 32-Bit-Dateien sind im gleichen Verzeichnis wie in der Vorgängerversion des Produkt vorhanden.

Wenn Sie eine 64-Bit-Version erstellen möchten, müssen Sie sicherstellen, dass Ihre Umgebung für die Verwendung der Bibliotheksdateien in *MQ_INSTALLATION_PATH*`\Tools\Lib64` konfiguriert ist. Stellen Sie sicher, dass die LIB-Umgebungsvariable so festgelegt ist, dass keine Ordner mit den 32-Bit-Bibliotheken durchsucht werden.

Windows C-Programme in Windows vorbereiten

Arbeiten Sie in Ihrer normalen Windows-Umgebung; für IBM MQ for Windows sind keine speziellen Funktionen erforderlich.

Weitere Informationen zur Programmierung von 64-Bit-Anwendungen finden Sie unter Codierungsstandards auf 64-Bit-Plattformen.

- Verknüpfen Sie Ihre Programme mit den entsprechenden Bibliotheken, die von IBM MQ bereitgestellt werden:

Bibliotheksdatei	**Programm/Exit-Typ**
MQ_INSTALLATION_PATH `\Tools\Lib\mqm.lib`	Server für 32-Bit-C
MQ_INSTALLATION_PATH `\Tools\Lib\mqic.lib`	Client für 32-Bit-C
MQ_INSTALLATION_PATH `\Tools\Lib\mqicxa.lib`	Client für 32-Bit-C mit Transaktionskoordination
MQ_INSTALLATION_PATH `\Tools\Lib64\mqm.lib`	Server für 64-Bit-C

Bibliotheksdatei	Programm/Exit-Typ
MQ_INSTALLATION_PATH \Tools\Lib64\mqic.lib	Client für 64-Bit-C
MQ_INSTALLATION_PATH \Tools\Lib64\mqicxa.lib	Client für 64-Bit-C mit Transaktionskoordination

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Im folgenden Befehl finden Sie ein Beispiel zur Kompilierung des Beispielprogramms 'amqsget0' (mithilfe des Microsoft Visual C++-Compilers).

Für 32-Bit-Anwendungen:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

Für 64-Bit-Anwendungen:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

Anmerkung:

- Wenn Sie einen installierbaren Service schreiben (weitere Informationen dazu finden Sie unter IBM MQ verwalten, müssen Sie eine Verknüpfung zu der Bibliothek 'mqmzf.lib' herstellen.
- Wenn Sie eine Anwendung für die externe Koordination durch einen XA-konformen Transaktionsmanager (z. B. IBM TXSeries Encina oder BEA Tuxedo) erstellen, müssen Sie eine Verknüpfung zu der Bibliothek 'mqmxa.lib' oder 'mqmxa.lib' herstellen.
- Wenn Sie einen CICS-Exit schreiben, stellen Sie eine Verknüpfung zu der Bibliothek 'mqmcics4.lib' her.
- Sie müssen IBM MQ-Bibliotheken vor allen anderen Produktbibliotheken verknüpfen.

- Die DLLs müssen sich in dem von Ihnen angegebenen Pfad (PATH) befinden.
- Wenn Sie Kleinbuchstaben verwenden, wann immer dies möglich ist, können Sie von IBM MQ for Windows zu IBM MQ for AIX or Linux wechseln, wo die Verwendung von Kleinbuchstaben erforderlich ist.

CICS- und Transaktionsserverprogramme vorbereiten

Ein Beispiel für einen C-Quellcode für eine CICS IBM MQ-Transaktion wird in AMQSCIC0.CCS bereitgestellt. Er wird mithilfe der CICS-Standardfunktionen erstellt. Gehen Sie für TXSeries for Windows 2000 beispielsweise folgendermaßen vor:

1. Legen Sie die Umgebungsvariable fest (geben Sie den folgenden Code in einer Zeile ein):

   ```
   set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
   %CICS_IBMC_FLAGS%
   ```

2. Legen Sie die Umgebungsvariable USERLIB fest:

   ```
   set USERLIB=MQM.LIB;%USERLIB%
   ```

3. Übersetzen, kompilieren und verknüpfen Sie das Beispielprogramm:

```
cicstcl -l IBMC amqscic0.ccs
```

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Dies wird im Handbuch *Transaction Server for Windows NT Application Programming Guide (CICS) V4* beschrieben.

Weitere Informationen zu unterstützenden CICS-Transaktionen finden Sie unter IBM MQ verwalten.

Windows *COBOL-Programme in Windows vorbereiten*

Verwenden Sie diese Informationen, um mehr zur Vorbereitung von COBOL-Programmen in Windows und zur Vorbereitung von CICS- und Transaktionsserverprogrammen zu erfahren.

1. Die 32-Bit-COBOL-Copybooks werden im folgenden Verzeichnis installiert: *MQ_INSTALLATION_PATH* `\Tools\cobol\CopyBook`.
2. Die 64-Bit-COBOL-Copybooks sind im folgenden Verzeichnis installiert: *MQ_INSTALLATION_PATH* `\Tools\cobol\CopyBook64`
3. Legen Sie für `CopyBook` in diesen Beispielen den folgenden Wert fest:

   ```
   CopyBook
   ```

   für 32-Bit-Anwendungen und:

   ```
   CopyBook64
   ```

   für 64-Bit-Anwendungen.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Zur Vorbereitung von COBOL-Programmen auf Windows-Systemen verknüpfen Sie Ihr Programm mit einer der folgenden Bibliotheken, die von IBM MQ bereitgestellt werden:

Bibliotheksdatei	**Programm oder Exittyp**
MQ_INSTALLATION_PATH `\Tools\Lib\mqmcb`	32-Bit-Server für Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib\mqiccb`	32-Bit-Client für Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib64\mqmcb`	64-Bit-Server für Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib64\mqiccb`	64-Bit-Client für Micro Focus COBOL

Bei der Ausführung eines Programms in der MQI-Clientumgebung stellen Sie sicher, dass die Bibliothek DOSCALLS vor einer COBOL- oder IBM MQ-Bibliothek angezeigt wird.

COBOL-Programme mithilfe von Micro Focus COBOL vorbereiten

Verknüpfen Sie alle vorhandenen 32 -Bit- IBM MQ Micro Focus COBOL-Programme mithilfe von `mqmcb.lib` oder `mqiccb.lib`anstelle der Bibliotheken `mqmcbb` und `mqiccbb` erneut.

So kompilieren Sie beispielsweise das Beispielprogramm `amq0put0` mithilfe von Micro Focus COBOL:

1. Legen Sie die Umgebungsvariable COBCPY so fest, dass sie auf die IBM MQ COBOL-Copybooks verweist (geben Sie den folgenden Code in einer Zeile ein):

   ```
   set COBCPY= MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook
   ```

2. Kompilieren Sie das Programm zur Ausgabe einer Objektdatei:

```
cobol amq0put0 LITLINK
```

3. Verknüpfen Sie die Objektdatei mit dem Laufzeitsystem.
   - Legen Sie die Umgebungsvariable LIB so fest, dass sie auf die COBOL-Bibliotheken des Compilers verweist.
   - Verknüpfen Sie die Objektdatei für die Verwendung auf dem IBM MQ-Server:

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - Alternativ verknüpfen Sie die Objektdatei für die Verwendung auf dem IBM MQ-Client:

     ```
     cbllink amq0put0.obj mqiccb.lib
     ```

CICS- und Transaktionsserverprogramme vorbereiten

So kompilieren und verknüpfen Sie ein Programm für TXSeries for Windows NT V5.1 mithilfe von IBM VisualAge COBOL:

1. Legen Sie die Umgebungsvariable fest (geben Sie den folgenden Code in einer Zeile ein):

   ```
   set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
   Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
   ```

2. Legen Sie die Umgebungsvariable USERLIB fest:

   ```
   set USERLIB=MQMCBB.LIB
   ```

3. Übersetzen, kompilieren und verknüpfen Sie Ihr Programm:

   ```
   cicstcl -l IBMCOB myprog.ccp
   ```

Dies wird im Handbuch *Transaction Server for Windows NT, V4 Application Programming* beschrieben.

So kompilieren und verknüpfen Sie ein Programm für CICS for Windows V5 mithilfe von Micro Focus COBOL:

- Legen Sie die Variable INCLUDE fest:

  ```
  set
  INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
          drive:\opt\cics\include;%INCLUDE%
  ```

- Legen Sie die Umgebungsvariable COBCPY fest:

  ```
  setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
            drive:\opt\cics\include
  ```

- Legen Sie die COBOL-Optionen fest:
  - `set`
  - `COBOPTS=/LITLINK /NOTRUNC`

Führen Sie folgenden Code aus:

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows *Visual Basic-Programme in Windows vorbereiten*

Informationen, die bei der Verwendung von Microsoft Visual Basic-Programmen unter Windows zu beachten sind.

Deprecated Ab IBM MQ 9.0 wird die Unterstützung für Microsoft Visual Basic 6.0 nicht mehr verwendet. IBM MQ-Klassen für .NET sind die empfohlene Ersatztechnologie. Weitere Informationen finden Sie unter .NET-Anwendungen entwickeln.

Anmerkung: 64-Bit-Versionen der Visual Basic-Moduldateien werden nicht bereitgestellt.

So bereiten Sie Visual Basic-Programme unter Windows vor:

1. Erstellen Sie ein neues Projekt.
2. Fügen Sie dem Projekt die bereitgestellte Moduldatei CMQB.BAS hinzu.
3. Fügen Sie bei Bedarf weitere bereitgestellte Moduldateien hinzu:
   - CMQBB.BAS: MQAI-Unterstützung
   - CMQCFB.BAS: PCF-Unterstützung
   - CMQXB.BAS: Unterstützung von Kanalexits
   - CMQPSB.BAS: Publish/Subscribe

Abschnitt „Codierung in Visual Basic” auf Seite 1106 enthält Informationen zur Verwendung des MQCONNXAny-Aufrufs in Visual Basic.

Rufen Sie die Prozedur MQ_SETDEFAULTS auf, bevor Sie MQI-Aufrufe im Projektcode vornehmen. Mit dieser Prozedur werden Standardstrukturen eingerichtet, die für die MQI-Aufrufe erforderlich sind.

Geben Sie vor dem Kompilieren oder Ausführen des Projekts an, ob Sie einen IBM MQ-Server oder -Client erstellen, indem Sie die Variable *MqType* für die bedingte Kompilierung festlegen. Setzen Sie *MqType* in einem Visual Basic-Projekt auf 1 für einen Server oder 2 für einen Client. Gehen Sie dazu wie folgt vor:

1. Wählen Sie das Menü 'Project' (Projekt) aus.
2. Wählen Sie Eigenschaften für *Name* aus (wobei *Name* für den Namen des aktuellen Projekts steht).
3. Wählen Sie im Dialogfeld die Registerkarte 'Make' (Erstellen) aus.
4. Geben Sie im Feld 'Conditional Compilation Arguments' (Argumente für bedingte Kompilierung) für einen Server Folgendes ein:

   ```
   MqType=1
   ```

   Geben Sie in diesem Feld für einen Client Folgendes ein:

   ```
   MqType=2
   ```

Zugehörige Konzepte

„Codierung in Visual Basic” auf Seite 1106
Informationen, die beim Codieren von IBM MQ-Programmen in Microsoft Visual Basic zu berücksichtigen sind. Visual Basic wird nur unter Windows unterstützt.

Zugehörige Verweise

„Visual Basic-Anwendungen mit IBM MQ MQI client-Code verknüpfen” auf Seite 969
Unter Windows können Sie Microsoft Visual Basic-Anwendungen mit dem IBM MQ MQI client-Code verknüpfen.

Windows *SSPI-Sicherheitsexit*

In IBM MQ for Windows wird ein Sicherheitsexit für den IBM MQ MQI client- und den IBM MQ-Server bereitgestellt. Dabei handelt es sich um ein Kanalexitprogramm zur Authentifizierung von IBM MQ-Kanälen mithilfe der SSPI-Schnittstelle. In der SSPI werden die integrierten Sicherheitsfunktionen von Windows-Systemen bereitgestellt.

Die Sicherheitspakete werden aus der Datei 'security.dll' oder 'secur32.dll' geladen. Diese DLLs werden mit Ihrem Betriebssystem geliefert.

Mithilfe von NTLM-Authentifizierungsservices wird die unidirektionale Authentifizierung bereitgestellt. Eine Zweiwegeauthentifizierung ist über die Services der Kerberos-Authentifizierung verfügbar.

Das Sicherheitsexitprogramm ist im Quellen- und Objektformat vorhanden. Sie können den Objektcode unverändert nutzen oder den Quellcode als Ausgangspunkt zur Erstellung Ihres eigenen Benutzerexitprogramms verwenden.

Weitere Informationen hierzu finden Sie im Abschnitt „SSPI-Sicherheitsexit unter Windows verwenden" auf Seite 1194.

Einführung in Sicherheitsexits

Ein Sicherheitsexit bildet eine sichere Verbindung zwischen zwei Sicherheitsexitprogrammen, von denen eines für den sendenden Nachrichtenkanalagenten und das andere für den empfangenden Nachrichtenkanalagenten verwendet wird.

Das Programm, das die sichere Verbindung einleitet (also das erste Programm, das die Kontrolle nach dem Herstellen der MCA-Sitzung übernimmt), wird als *Kontextinitiator* bezeichnet. Das Partnerprogramm wird *Kontextakzeptor* genannt.

In der folgenden Tabelle werden einige der Kanaltypen gezeigt, bei denen es sich um Kontextinitiatoren und die zugehörigen Kontextakzeptoren handelt.

Tabelle 149. Kontextinitiatoren und die zugehörigen Kontextakzeptoren

Kontextinitiator	**Kontextakzeptor**
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

Das Programm für den Sicherheitsexit verfügt über zwei Eingangspunkte:

- **SCY_NTLM**

  Bei diesem Einstiegspunkt werden NTLM-Authentifizierungsservices verwendet, in denen die unidirektionale Authentifizierung bereitgestellt wird. Mit NTLM können Server die Identitäten ihrer zugehörigen Clients bestätigen. Allerdings können Clients nicht die Identität eines Servers bestätigen und ein Server kann nicht die Identität eines anderen Servers bestätigen. Die NTLM-Authentifizierung wurde für eine Netzumgebung entwickelt, in der die Echtheit von Servern vorausgesetzt wird.

- **SCY_KERBEROS**

  Bei diesem Einstiegspunkt werden Kerberos-Services für die gegenseitige Authentifizierung verwendet. Das Kerberos-Protokoll nimmt nicht an, dass die Server in einer Netzumgebung echt sind. Die Parteien an beiden Enden einer Netzverbindung können die Identität der anderen Partei bestätigen. Also können Server die Identität von Clients und von anderen Servern bestätigen und Clients können die Identität eines Servers bestätigen.

Funktionen des Sicherheitsexits

In diesem Abschnitt werden die Funktionen von SSPI-Kanalexitprogrammen beschrieben.

In den bereitgestellten Kanalexitprogrammen wird die unidirektionale oder die bidirektional (gegenseitige) Authentifizierung eines Partnersystems beim Herstellen einer Sitzung bereitgestellt. Jedes Exitprogramm verfügt für einen bestimmten Kanal über einen zugehörigen *Prinzipal* (ähnlich der Benutzer-ID,

siehe „IBM MQ-Zugriffssteuerung und Windows-Principals” auf Seite 1075). Bei einer Verbindung zwischen zwei Exitprogrammen handelt es sich um eine Zuordnung zwischen den beiden Prinzipalen.

Nach den Herstellen der zugrundeliegenden Sitzung wird eine sichere Verbindung zwischen zwei Sicherheitsexitprogrammen hergestellt (ein Programm für den sendenden MCA und eins für den empfangenden MCA). Die Operationsfolge lautet folgendermaßen:

1. Jedes Programm wird einem bestimmten Prinzipal zugeordnet, z. B. als Ergebnis einer expliziten Anmeldeoperation.
2. Der Kontextinitiator fordert eine sichere Verbindung zum Partner aus dem Sicherheitspaket her (für Kerberos ist dies der genannte Partner) und empfängt ein Token (mit der Bezeichnung Token1). Das Token wird mithilfe der bereits hergestellten zugrundeliegenden Sitzung an das Partnerprogramm gesendet.
3. Das Partnerprogramm (der Kontextakzeptor) übergibt Token1 an das Sicherheitspaket, in dem die Authentifizierung des Kontextinitiators vorgenommen wird. Die Verbindung wurde für NTLM jetzt hergestellt.
4. Für den mit Kerberos gelieferten Sicherheitsexit (zur gegenseitigen Authentifizierung) generiert das Sicherheitspaket auch ein zweites Token (mit der Bezeichnung Token2), das der Kontextakzeptor mithilfe der zugrundeliegenden Sitzung an den Kontextinitiator zurückgibt.
5. Der Kontextinitiator authentifiziert den Kontextakzeptor mithilfe von Token2.
6. Wenn beide Anwendungen die Authentizität des Tokens vom Partner akzeptieren, wird in dieser Phase die sichere (d. h. authentifizierte) Verbindung hergestellt.

IBM MQ-Zugriffssteuerung und Windows-Principals

Die von IBM MQ bereitgestellte Zugriffssteuerung basiert auf dem Benutzer und der Gruppe. Die von Windows bereitgestellte Authentifizierung basiert auf Prinzipalen, z. B. Benutzer und Dienstprinzipalname (Service Principal Name, SPN). Bei SPN können viele Prinzipale einem einzelnen Benutzer zugeordnet sein.

Der SSPI-Sicherheitsexit verwendet die relevanten Windows-Prinzipale für die Authentifizierung. Wenn die Windows-Authentifizierung erfolgreich ist, übergibt der Exit die dem Windows-Prinzipal zugeordnete Benutzer-ID an IBM MQ zur Zugriffskontrolle.

Die für die Authentifizierung relevanten Windows-Prinzipale sind je nach dem zur Authentifizierung verwendeten Typ unterschiedlich.

- Bei der NTLM-Authentifizierung handelt es sich bei dem Windows-Prinzipal für den Kontextinitiator um die Benutzer-ID, die dem aktiven Prozess zugeordnet ist. Da es sich um eine unidirektionale Authentifizierung handelt, ist der dem Kontextakzeptor zugeordnete Prinzipal nicht relevant.
- Bei der Kerberos-Authentifizierung auf CLNTCONN-Kanälen handelt es sich beim Windows-Prinzipal um die dem aktiven Prozess zugeordnete Benutzer-ID. Andernfalls ist der Windows-Prinzipal der SPN, der durch Hinzufügen des folgenden Präfixes an den Warteschlangenmanagernamen gebildet wird.

  ```
  ibmMQSeries/
  ```

z/OS Building your procedural application on z/OS

The CICS, IMS, and z/OS publications describe how to build applications that run in these environments.

This collection of topics describes the additional tasks, and the changes to the standard tasks, that you must perform when building IBM MQ for z/OS applications for these environments. COBOL, C, C++, Assembler, and PL/I programming languages are supported. (For information about building C++ applications see C++ verwenden.)

The tasks that you must perform to create an executable IBM MQ for z/OS application depend on both the programming language that the program is written in, and the environment in which the application will run.

In addition to coding the MQI calls in your program, add the appropriate language statements to include the IBM MQ for z/OS data definition file for the language that you are using. Make yourself familiar with the contents of these files. See “IBM MQ-Datendefinitionsdateien” on page 755 for a full description.

Note

The name **thlqual** is the high-level qualifier of the installation library on z/OS.

z/OS *Preparing your program to run*

After you have written the program for your IBM MQ application to create an executable application, you have to compile or assemble it, then link-edit the resulting object code with the stub program that IBM MQ for z/OS supplies for each environment that it supports.

How you prepare your program depends on both the environment (batch, CICS, IMS(BMP or MPP), Linux or z/OS UNIX System Services) in which the application runs, and the structure of the data sets on your z/OS installation.

“Dynamically calling the IBM MQ stub” on page 1082 describes an alternative method of making MQI calls in your programs so that you do not need to link-edit an IBM MQ stub. This method is not available for all languages and environments.

Do not link-edit a higher level of stub program than that of the version of IBM MQ for z/OS on which your program is running. For example, a program running on MQSeries for OS/390®, V5.2 must not be link-edited with a stub program supplied with IBM MQ for z/OS V7.

z/OS *Building 64 bit C applications*

In z/OS, 64 bit C applications are built using the LP64 compiler and binder options. The IBM MQ for z/OS *cmqc.h* header file recognizes when this option is provided to the compiler, and generates IBM MQ data types and structures appropriate for 64 bit operation.

C code built with this option must be built to use dynamic-link libraries (DLLs) appropriate for the coordination semantic required. To achieve this, you bind the compiled code with the appropriate side-deck defined in the following table:

Table 150. Side-deck name required for each coordination semantic

Coordination	**Side-deck name**
Single phase commit MQI	CSQBMQ2X
Two phase commit with RRS coordination, using RRS verbs	CSQBRR2X
Two phase commit with RRS coordination, using MQI verbs	CSQBRI2X

Note: For 31-bit C applications you also set compiler options for the calling interface (either Language Environment or XPLINK), as described in “Building z/OS batch applications using 31-bit Language Environment or XPLINK” on page 1078. For 64-bit C applications you do not specify the calling interface, because the only supported linkage is XPLINK.

Use the EDCQCB JCL procedure, supplied with *z/OS XL C/C++*, to build a single phase commit IBM MQ program as a batch job, as follows:

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',            < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',            < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//        DD
//        DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
```

```
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

To build an RRS coordinated program in z/OS UNIX System Services, compile and link as follows:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *Building z/OS batch applications*

Learn how to build z/OS batch applications and the steps to consider when doing so.

To build an application for IBM MQ for z/OS that runs under z/OS batch, create job control language (JCL) that performs these tasks:

1. Compile (or assemble) the program to produce object code. The JCL for your compilation must include SYSLIB statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
   - For COBOL, **thlqual**.SCSQCOBC
   - For assembler language, **thlqual**.SCSQMACS
   - For C, **thlqual**.SCSQC370
   - For PL/I, **thlqual**.SCSQPLIC
2. For a C application, prelink the object code created in step “1” on page 1077.
3. For PL/I applications, use the compiler option EXTRN(SHORT).
4. Link-edit the object code created in step “1” on page 1077 (or step “2” on page 1077 for a C application) to produce a load module. When you link-edit the code, you must include one of the IBM MQ for z/OS batch stub programs (CSQBSTUB or one of the RRS stub programs: CSQBRRSI or CSQBRSTB).

   CSQBSTUB
   single-phase commit provided by IBM MQ for z/OS

   CSQBRRSI
   two-phase commit provided by RRS using the MQI

   CSQBRSTB
   two-phase commit provided by RRS directly

   Notes:

   a. If you use CSQBRSTB, you must also link-edit your application with ATRSCSS from SYS1.CSSLIB. Figure 113 on page 1077 and Figure 114 on page 1078 show fragments of JCL to do this. The stubs are language-independent and are supplied in library **thlqual**.SCSQLOAD.

   b. If your application runs under Language Environment, you should ensure you link-edit with the Language Environment DLL instead as described in “Building z/OS batch applications using 31-bit Language Environment or XPLINK” on page 1078.
5. Store the load module in an application load library.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

Figure 113. Fragments of JCL to link-edit the object module in the batch environment, using single-phase commit

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
⋮
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
⋮
/*
```

Figure 114. Fragments of JCL to link-edit the object module in the batch environment, using two-phase commit

To run a batch or RRS program, you must include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB or JOBLIB data set concatenation.

To run a TSO program, you must include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB used by the TSO session.

To run a batch program from the z/OS UNIX System Services shell, add the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD to the STEPLIB specification in your $HOME?.profile like this:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *Building z/OS batch applications using 31-bit Language Environment or XPLINK*

IBM MQ for z/OS provides a set of dynamic link libraries (DLLs) that must be used when you link-edit your applications.

There are two variants of the libraries that allow the application to use one of the following calling interfaces:

- The 31-bit Language Environment calling interface.
- The 31-bit XPLINK calling interface. z/OS XPLINK is a high performance calling convention available for C applications. See XPLINK | NOXPLINK in the z/OS 2.2 documentation.

To use the DLLs, the application is bound or linked against so called *sidedecks*, instead of the stubs provided with earlier versions. The sidedecks are found in the SCSQDEFS library (instead of the SCSQLOAD library).

Table 151. Variants of dynamic link libraries

Commit	31-bit Language Environment DLL	31-bit XPLINK DLL	Equivalent stub name
1 phase commit MQI libraries	CSQBMQ1	CSQBMQ1X	CSQBSTUB
2 phase commit with RRS co-ordination using RRS transaction-control verbs	CSQBRR1	CSQBRR1X	CSQBRSTB
2 phase commit with RRS co-ordination using MQI transaction-control verbs	CSQBRI1	CSQBRI1X	CSQBRRSI

Note: All sidedecks contain a definition of the data conversion entry point, MQXCNVC, previously resolved by including CSQASTUB.

Common issues:

- The following message appears on the job log if your application uses asynchronous message consume (MQCB, MQCTL or MQSUB calls) and the previous DLL interface is not used:

  ```
  CSQB001E Language environment programs running in z/OS batch or z/OS UNIX System Services must
  use the DLL interface to IBM MQ
  ```

  Solution: Rebuild your application using sidedecks instead of stubs as detailed previously.

- At program build time, the following message appears

  ```
  IEW2469E The Attributes of a reference to MQAPI-NAME from section your-code do not match the
  attributes of
  the target symbol
  ```

  Reason: This means that you have compiled your XPLINK program with V701 (or later) version of cmqc.h, but are not binding with sidedecks.

  Solution: Change your program's build file to bind against the appropriate sidedeck from SCSQDEFS instead of a stub from SCSQLOAD

The following sample JCL demonstrates how you can compile and link-edit a C program to use the 31 bit Language Environment DLL calling interface:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

Note: The compilation uses the **DLL** option. The link-edit uses **DYNAM=DLL** option and the references the **CSQBMQ1** library.

The following sample JCL demonstrates how you can compile and link-edit a C program to use the 31 bit XPLINK DLL calling interface:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
```

```
 NAME MYPROGAM(R)
//
```

Note: The compilation uses the **XPLINK** and **DLL** options. The link-edit uses **DYNAM=DLL** option and references the **CSQBMQ1X** library.

Ensure that you add the compilation option DLL to each program in the module. Messages such as IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED are an indication that you need to check that all of the programs have been compiled with the DLL option.

z/OS *Building CICS applications in z/OS*

Use this information when building CICS applications in z/OS.

To build an application for IBM MQ for z/OS that runs under CICS, you must:

- Translate the CICS commands in your program into the language in which the rest of your program is written.
- Compile or assemble the output from the translator to produce object code.
  - For PL/I programs, use the compiler option EXTRN(SHORT).
  - For C applications, if the application is not using XPLINK, use the compiler option DEFINE(MQ_OS_LINKAGE=1).
- Link-edit the object code to create a load module.

CICS provides a procedure to execute these steps in sequence for each of the programming languages it supports.

- For CICS Transaction Server for z/OS, the *CICS Transaction Server for z/OS System Definition Guide* describes how to use these procedures and the *CICS/ESA Application Programming Guide* gives more information on the translation process.

You must include:

- In the SYSLIB statement of the compilation (or assembly) stage, statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
  - For COBOL, **thlqual**.SCSQCOBC
  - For assembler language, **thlqual**.SCSQMACS
  - For C, **thlqual**.SCSQC370
  - For PL/I, **thlqual**.SCSQPLIC
- In your link-edit JCL, the IBM MQ for z/OS CICS stub program (CSQCSTUB). Figure 115 on page 1080 shows fragments of JCL code to do this. The stub is language-independent and is supplied in library **thlqual**.SCSQLOAD.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

Figure 115. Fragments of JCL to link-edit the object module in the CICS environment

- For CICS versions later than CICS TS 3.2, or, if you want to use IBM MQ message property APIs, or IBM MQ APIs MQCB, MQCTL, MQSTAT, MQSUB or MQSUBR, you must linkedit your object code with the CICS supplied stub, DFHMQSTB and not the IBM MQ supplied CSQCSTUB. For more information about

building IBM MQ programs for CICS, see API stub program to access IBM MQ MQI calls in the CICS product documentation.

When you have completed these steps, store the load module in an application load library and define the program to CICS in the usual way.

Before you run a CICS program, your system administrator must define it to CICS as an IBM MQ program and transaction, You can then run it in the typical way.

z/OS *Building IMS (BMP or MPP) applications*

Use this information when building IMS (BMP or MPP) applications.

If you are building batch DL/I programs, see "Building z/OS batch applications" on page 1077. To build other applications that run under IMS (either as a BMP or an MPP), create JCL that performs these tasks:

1. Compile (or assemble) the program to produce object code. The JCL for your compilation must include SYSLIB statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
   - For COBOL, **thlqual**.SCSQCOBC
   - For assembler language, **thlqual**.SCSQMACS
   - For C, **thlqual**.SCSQC370
   - For PL/I, **thlqual**.SCSQPLIC
2. For a C application, prelink the object module created in step "1" on page 1081.
3. For PL/I programs, use the compiler option EXTRN(SHORT).
4. For a C application, if the application is not using XPLINK, use the compiler option DEFINE(MQ_OS_LINKAGE=1).
5. Link-edit the object code created in step "1" on page 1081 (or step "2" on page 1081 for a C/370 application) to produce a load module:
   a. Include the IMS language interface module (DFSLI000).
   b. Include the IBM MQ for z/OS IMS stub program (CSQQSTUB). Figure 116 on page 1081 shows fragments of JCL to do this. The stub is language independent and is supplied in library **thlqual**.SCSQLOAD.

      Note: If you are using COBOL, select the NODYNAM compiler option to enable the linkage editor to resolve references to CSQQSTUB unless you intend to use dynamic linking as described in "Dynamically calling the IBM MQ stub" on page 1082.
6. Store the load module in an application load library.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Figure 116. Fragments of JCL to link-edit the object module in the IMS environment

Before you run an IMS program, your system administrator must define it to IMS as an IBM MQ program and transaction: you can then run it in the typical way.

z/OS *Building z/OS UNIX System Services applications*

Use this information when building z/OS UNIX System Services applications.

To build a C application for IBM MQ for z/OS that runs under z/OS UNIX System Services, compile and link your application as follows:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

where **thlqual** is the high-level qualifier used by your installation.

To run the C program, you need to add the following to your `.profile` file; this should be in your root directory:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Note that you need to exit from z/OS UNIX System Services, and enter z/OS UNIX System Services again, for the change to be recognized.

If you want to run multiple shells, add the word export at the beginning of the line, that is:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Once this completes successfully you can link the CSQBSTUB and issue IBM MQ calls.

"Dynamically calling the IBM MQ stub" on page 1082 describes an alternative method of making MQI calls in your programs so that you do not need to link-edit an IBM MQ stub. This method is not available for all languages and environments.

Do not link-edit a higher level of stub program than that of the version of IBM MQ for z/OS on which your program is running. For example, a program running on IBM WebSphere MQ for z/OS 7.1 must not be link-edited with a stub program supplied with IBM MQ for z/OS 8.0.

z/OS *Dynamically calling the IBM MQ stub*

Instead of link-editing the IBM MQ stub program with your object code, you can dynamically call the stub from within your program.

You can do this in the batch, IMS, and CICS environments. This facility is not supported in the RRS environment. If your application program uses RRS to coordinate updates, see "RRS Considerations" on page 1086.

However, this method:

- Increases the complexity of your programs
- Increases the storage required by your programs at execution time
- Reduces the performance of your programs
- Means that you cannot use the same programs in other environments

If you call the stub dynamically, the appropriate stub program and its aliases must be available at execution time. To ensure this, include the IBM MQ for z/OS data set SCSQLOAD:

- For batch and IMS, in the STEPLIB concatenation of the JCL.
- For CICS, in the CICS DFHRPL concatenation.

For IMS, ensure that the library containing the dynamic stub (built as described in the information about installing the IMS adapter in Setting up the IMS adapter) is ahead of the data set SCSQLOAD in the STEPLIB concatenation of the region JCL.

Use the names shown in Table 152 on page 1083 when you call the stub dynamically. In PL/I, only declare the call names used in your program.

Table 152. Call names for dynamic linking			
MQI call	**Batch (non-RRS) dynamic call names**	**CICS dynamic call names**	**IMS dynamic call names**
MQBACK	CSQBBACK	not supported	Not supported
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	Not supported
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	not supported	Not supported
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	Not supported
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGET	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Note: 1. These API calls are available only when using CICS TS 3.2 or later and the CSQCSTUB shipped with CICS must be used. For CICS TS 3.2, APAR PK66866 must be applied. For CICS TS 4.1, APAR PK89844 must be applied.

For examples of how to use this technique, see the following figures:

- Batch and COBOL: see Figure 117 on page 1084
- CICS and COBOL: see Figure 118 on page 1084
- IMS and COBOL: see Figure 119 on page 1084
- Batch and assembler: see Figure 120 on page 1085
- CICS and assembler: see Figure 121 on page 1085
- IMS and assembler: see Figure 122 on page 1085
- Batch and C: Figure 123 on page 1085

- CICS and C: see Figure 124 on page 1085
- IMS and C: see Figure 125 on page 1086
- Batch and PL/I: see Figure 126 on page 1086
- IMS and PL/I: see Figure 127 on page 1086

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                    PIC X(8) VALUE 'CSQBOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 117. Dynamic linking using COBOL in the batch environment

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                    PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 118. Dynamic linking using COBOL in the CICS environment

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                    PIC X(8) VALUE 'MQOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
      * ------------------------------------------------------------ *
      *
      *    If the compilation option 'DYNAM' is specified
      *    then you may code the MQ calls as follows
      *
      * ------------------------------------------------------------ *
...
           CALL 'MQOPEN'  WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 119. Dynamic linking using COBOL in the IMS environment

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

Figure 120. Dynamic linking using assembly language in the batch environment

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

Figure 121. Dynamic linking using assembly language in the CICS environment

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

Figure 122. Dynamic linking using assembly language in the IMS environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 123. Dynamic linking using C language in the batch environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 124. Dynamic linking using C language in the CICS environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 125. Dynamic linking using C language in the IMS environment

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

Figure 126. Dynamic linking using PL/I in the batch environment

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

Figure 127. Dynamic linking using PL/I in the IMS environment

z/OS *RRS Considerations*

Consider using this information if your application program uses RRS to coordinate updates.

IBM MQ provides two different stubs for batch programs which need RRS coordination - see “The RRS batch adapter” on page 941. The difference in behavior of later API calls is determined at MQCONN time by the batch adapter from information passed by the stub routine on the MQCONN or MQCONNX API. This means that dynamic API calls are available for batch programs which need RRS coordination, provided that the initial connection to IBM MQ was done by using the appropriate stub. The following example illustrates this:

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
```

```
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
                               W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS *Debugging your programs*

Use this information to learn about debugging TSO and CICS programs, and an insight into CICS trace.

The main aids to debugging IBM MQ for z/OS application programs are the reason codes returned by each API call. For a list of these, including ideas for corrective action, see:

- IBM MQ for z/OS -Nachrichten, -Beendigungscodes und -Ursachencodes for IBM MQ for z/OS
- Nachrichten und Ursachencodes for all other IBM MQ platforms

This topic also suggests other debugging tools to use in particular environments.

Debugging TSO programs

The following interactive debugging tools are available for TSO programs:

- TEST tool
- VS COBOL II interactive debugging tool
- INSPECT interactive debugging tool for C and PL/I programs

Debugging CICS programs

You can use the CICS Execution Diagnostic Facility (CEDF) to test your CICS programs interactively without having to modify the program or program-preparation procedure.

For more information about EDF, see the *CICS Transaction Server for z/OS CICS Application Programming Guide*.

CICS trace

You will probably also find it helpful to use the CICS Trace Control transaction (CETR) to control CICS trace activity.

For more information about CETR, see *CICS Transaction Server for z/OS CICS-Supplied Transactions* manual.

To determine whether CICS trace is active, display connection status using the CKQC panel. This panel also shows the trace number.

To interpret CICS trace entries, see Table 153 on page 1088.

The CICS trace entry for these values is AP0 *xxx* (where *xxx* is the trace number specified when the CICS adapter was enabled). All trace entries except CSQCTEST are issued by CSQCTRUE. CSQCTEST is issued by CSQCRST and CSQCDSP.

Table 153. CICS adapter trace entries

Name	Description	Trace sequence	Trace data
CSQCABNT	Abnormal termination	Before issuing END_THREAD ABNORMAL to IBM MQ. This is because of the end of the task and an implicit backout could be performed by the application. A ROLLBACK request is included in the END_THREAD call in this case.	Unit of work information. You can use this information when finding out about the status of work. (For example, it can be verified against the output produced by the DISPLAY THREAD command, or the IBM MQ for z/OS log print utility.)
CSQCBACK	Syncpoint backout	Before issuing BACKOUT to IBM MQ for z/OS. This is due to an explicit backout request from the application.	Unit of work information.
CSQCCCRC	Completion code and reason code	After unsuccessful return from API call.	Completion code and reason code.
CSQCCOMM	Syncpoint commit	Before issuing COMMIT to IBM MQ for z/OS. This can be due to a single-phase commit request or the second phase of a two-phase commit request. The request is due to an explicit syncpoint request from the application.	Unit of work information.
CSQCEXER	Execute resolve	Before issuing EXECUTE_RESOLVE to IBM MQ for z/OS.	The unit of work information of the unit of work issuing the EXECUTE_RESOLVE. This is the last indoubt unit of work in the resynchronization process.
CSQCGETW	GET wait	Before issuing CICS wait.	Address of the ECB to be waited on.
CSQCGMGD	GET message data	After successful return from MQGET.	Up to 40 bytes of the message data.
CSQCGMGH	GET message handle	Before issuing MQGET to IBM MQ for z/OS.	Object handle.
CSQCGMGI	Get message ID	After successful return from MQGET.	Message ID and correlation ID of the message.
CSQCINDL	Indoubt list	After successful return from the second INQUIRE_INDOUBT.	The indoubt units of work list.
CSQCINDO	IBM use only		
CSQCINDS	Indoubt list size	After successful return from the first INQUIRE_INDOUBT and the indoubt list is not empty.	Length of the list. Divided by 64 gives the number of indoubt units of work.
CSQCINQH	INQ handle	Before issuing MQINQ to IBM MQ for z/OS.	Object handle.
CSQCLOSH	CLOSE handle	Before issuing MQCLOSE to IBM MQ for z/OS.	Object handle.

Table 153. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCLOST	Disposition lost	During the resynchronization process, CICS informs the adapter that it has been restarted so no disposition information regarding the unit of work being resynchronized is available.	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCNIND	Disposition not indoubt	During the resynchronization process, CICS informs the adapter that the unit of work being resynchronized should not have been indoubt (that is, perhaps it is still running).	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCNORT	Normal termination	Before issuing END_THREAD NORMAL to IBM MQ for z/OS. This is due to the end of the task and therefore the application might perform an implicit syncpoint commit. A COMMIT request is included in the END_THREAD call in this case.	Unit of work information.
CSQCOPNH	OPEN handle	After successful return from MQOPEN.	Object handle.
CSQCOPNO	OPEN object	Before issuing MQOPEN to IBM MQ for z/OS.	Object name.
CSQCPMGD	PUT message data	Before issuing MQPUT to IBM MQ for z/OS.	Up to 40 bytes of the message data.
CSQCPMGH	PUT message handle	Before issuing MQPUT to IBM MQ for z/OS.	Object handle.
CSQCPMGI	PUT message ID	After successful MQPUT from IBM MQ for z/OS.	Message ID and correlation ID of the message.
CSQCPREP	Syncpoint prepare	Before issuing PREPARE to IBM MQ for z/OS in the first phase of two-phase commit processing. This call can also be issued from the distributed queuing component as an API call.	Unit of work information.
CSQCP1MD	PUTONE message data	Before issuing MQPUT1 to IBM MQ for z/OS.	Up to 40 bytes of data of the message.
CSQCP1MI	PUTONE message ID	After successful return from MQPUT1.	Message ID and correlation ID of the message.
CSQCP1ON	PUTONE object name	Before issuing MQPUT1 to IBM MQ for z/OS.	Object name.
CSQCRBAK	Resolved backout	Before issuing RESOLVE_ROLLBACK to IBM MQ for z/OS.	Unit of work information.
CSQCRCMT	Resolved commit	Before issuing RESOLVE_COMMIT to IBM MQ for z/OS.	Unit of work information.

Table 153. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCRMIR	RMI response	Before returning to the CICS RMI (resource manager interface) from a specific invocation.	Architected RMI response value. Its meaning depends of the type of the invocation. These values are documented in the *CICS Transaction Server for z/OS Customization Guide*. To determine the type of invocation, look at previous trace entries produced by the CICS RMI component.
CSQCRSYN	Resynchronization	Before the resynchronization process starts for the task.	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCSETH	SET handle	Before issuing MQSET to IBM MQ for z/OS.	Object handle.
CSQCTASE	IBM use only		
CSQCTEST	Trace test	Used in EXEC CICS ENTER TRACE call to verify the trace number supplied by the user or the trace status of the connection.	No data.
CSQCDCFF	IBM use only		

Prozedurale Programmfehler handhaben

In diesen Informationen werden Fehler erläutert, die den MQI-Aufrufen Ihrer Anwendungen beim Ausgeben eines Aufrufs oder bei der Übergabe einer Nachricht an den Zielort zugeordnet werden.

Wann immer dies möglich, gibt der Warteschlangenmanager eventuelle Fehler zurück, sobald ein MQI-Aufruf ausgegeben wird. Hierbei handelt es sich um *lokal ermittelte Fehler*.

Beim Senden von Nachrichten an eine ferne Warteschlange sind Fehler möglicherweise nicht offensichtlich, wenn der MQI-Aufruf vorgenommen wird. In diesem Fall meldet der Warteschlangenmanager, der die Fehler ermittelte, diese beim Senden einer weiteren Nachricht an das Programm, von dem die Nachricht ausging. Hierbei handelt es sich um *fern ermittelte Fehler*.

Lokal ermittelte Fehler

In diesem Abschnitt finden Sie Informationen zu lokal ermittelten Fehlern wie beispielsweise einem Fehler in einem MQI-Aufruf, Systemunterbrechungen und Nachrichten mit falschen Daten.

Dies sind die drei häufigsten Fehlerursachen, die der Warteschlangenmanager sofort melden kann:

- Fehler in einem MQI-Aufruf, weil beispielsweise eine Warteschlange voll ist
- Unterbrechung der Ausführung eines Teil des Systems, der für Ihre Anwendung erforderlich ist (z. B Warteschlangenmanager)
- Nachrichten mit Daten, die nicht erfolgreich verarbeitet werden können

Wenn Sie die asynchrone Put-Funktion verwenden, werden Fehler nicht sofort gemeldet. Rufen Sie Statusinformationen zu vorherigen asynchronen Put-Funktionen mit dem MQSTAT-Aufruf ab.

Fehler eines MQI-Aufrufs

Der Warteschlangenmanager kann Fehler in der Codierung eines MQI-Aufrufs sofort melden. Dazu wird eine Gruppe vordefinierter Rückgabecodes verwendet. Diese werden in Beendigungscodes und Ursachencodes eingeteilt.

Um anzuzeigen, ob ein Aufruf erfolgreich ist, gibt der Warteschlangenmanager einen *Beendigungscode* zurück, wenn der Aufruf abgeschlossen ist. Es gibt drei Beendigungscodes, mit denen der Erfolg, ein partieller Abschluss und das Fehlschlagen des Aufrufs angezeigt werden. Der Warteschlangenmanager gibt außerdem einen *Ursachencode* zurück, in dem die Ursache für den partiellen Abschluss oder das Fehlschlagen des Aufrufs angezeigt wird.

Die Beendigungs- und Ursachencodes für jeden Aufruf werden mit der Beschreibung dieses Aufrufs unter Rückgabecodes aufgeführt. Ausführlichere Informationen, einschließlich Vorschlägen für Korrekturmaßnahmen, finden Sie in folgenden Abschnitten:

- z/OS IBM MQ for z/OS -Nachrichten, -Beendigungscodes und -Ursachencodes fürIBM MQ for z/OS
- Nachrichten und Ursachencodes für alle anderen IBM MQ-Plattformen

Gestalten Sie Ihre Programme so, dass alle Rückgabecodes bearbeitet werden, die ein Aufruf ausgeben kann.

System i-Unterbrechungen

Ihre Anwendung erkennt möglicherweise keine Unterbrechungen, wenn der damit verbundene Warteschlangenmanager nach einem Systemausfall wiederhergestellt wird. Allerdings müssen Sie Ihre Anwendung so entwickeln, dass sichergestellt wird, dass Ihre Daten beim Auftreten einer Unterbrechung nicht verloren gehen.

Die Methoden, die Sie zum Sicherstellen der Datenkonsistenz verwenden können, sind von der Plattform abhängig, auf denen Ihr Warteschlangenmanager ausgeführt wird:

z/OS
z/OS

In den CICS- und IMS-Umgebungen können Sie MQPUT- und MQGET-Aufrufe innerhalb von Arbeitseinheiten vornehmen, die von CICS oder IMS verwaltet werden. In der Stapelumgebung können Sie MQPUT- und MQGET-Aufrufe auf die gleiche Weise ausgeben, müssen aber Synchronisationspunkte mithilfe folgender Funktionen vereinbaren:

- IBM MQ for z/OS MQCMIT- und MQBACK-Aufrufe (siehe „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901) oder
- Das z/OS-Transaktionsmanagement und Recoverable Resource Manager Services (RRS) zur Bereitstellung der Unterstützung für zweiphasige Synchronisationspunkte. Mit RRS können Sie IBM MQ und andere Produktressourcen, die RRS unterstützen (z. B. in Db2 gespeicherte Produktressourcen) in einer einzelnen logischen Arbeitseinheit aktualisieren. Informationen zur Unterstützung des RRS-Synchronisationspunkts finden Sie unter „Transaction management and recoverable resource manager services” auf Seite 905.

IBM i
IBM i

Sie können Ihre MQPUT- und MQGET-Aufrufe in globalen Arbeitseinheiten ausgeben, die von der IBM i-Commitsteuerung verwaltet werden. Mithilfe der nativen IBM i-Befehle COMMIT und ROLLBACK oder der sprachspezifischen Befehle können Sie Synchronisationspunkte deklarieren. Lokale Arbeitseinheiten werden von IBM MQ mithilfe der MQCMIT- und MQBACK-Aufrufe verwaltet.

AIX, Linux, and Windows-Systeme

In diesen Umgebungen können Sie Ihre MQPUT- und MQGET-Aufrufe wie gewohnt ausgeben, Sie müssen allerdings Synchronisationspunkte mithilfe der MQCMIT- und MQBACK-Aufrufe ausgeben (siehe „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 901). In der CICS-Umgebung sind MQCMIT- und MQBACK-Befehle inaktiviert, da Sie Ihre MQPUT- und MQGET-Aufrufe innerhalb von Arbeitseinheiten ausgeben können, die von CICS verwaltet werden.

Verwenden Sie persistente Nachrichten zur Übergabe aller Daten, die nicht verloren gehen dürfen. Persistente Nachrichten werden in Warteschlangen wiederhergestellt, falls der Warteschlangenmanager nach einem Fehler wiederhergestellt werden muss. ALW Bei IBM MQ on AIX, Linux, and Windowsschlägt ein MQGET-oder MQPUT-Aufruf in Ihrer Anwendung fehl, wenn alle Protokolldateien mit der Nachricht MQRC_RESOURCE_PROBLEM gefüllt werden. Weitere Informationen zu Protokolldateien unter AIX, Linux, and Windows finden Sie im Abschnitt IBM MQ verwalten. z/OS Weitere Informationen zu z/OS finden Sie im Abschnitt Planung für z/OS.

Wenn der Warteschlangenmanager während der Ausführung einer Anwendung von einem Bediener gestoppt wird, wird normalerweise die Option zum Versetzen in den Wartemodus verwendet. Der Warteschlangenmanager wird in den Wartemodus versetzt, in dem Anwendungen weiterhin ausgeführt werden können, aber zum nächsten günstigen Zeitpunkt beendet werden müssen. Kompakte, schnelle Anwendungen können den Wartemodus wahrscheinlich ignorieren und ihre Arbeit fortsetzen, bis sie wie gewohnt beendet werden. Für länger aktive Anwendungen oder für Anwendungen, die auf den Empfang von Nachrichten warten, sollte bei Verwendung der MQOPEN-, MQPUT-, MQPUT1- und MQGET-Aufrufe die Option *fail if quiescing* verwendet werden. Durch das Festlegen dieser Option schlagen die Aufrufe fehl, wenn der Warteschlangenmanager in den Wartemodus gesetzt wird, aber die Anwendung hat möglicherweise noch genügend Zeit für einen fehlerfreien Abschluss, indem Aufrufe ausgegeben werden, die den Wartestatus ignorieren. Diese Anwendungen können auch durchgeführte Änderungen festschreiben oder zurücksetzen, bevor Sie beendet werden.

Wenn das Stoppen des Warteschlangenmanagers erzwungen wird (also das Stoppen ohne Wartemodus), empfangen Anwendungen beim Ausgeben von MQI-Aufrufen den Ursachencode MQRC_CONNECTION_BROKEN. Beenden Sie die Anwendung oder geben Sie alternativ auf Systemen mit IBM i IBM MQ for IBM i, AIX, Linux, and Windows einen MQDISC-Aufruf aus.

Nachrichten mit falschen Daten

Wenn Sie in Ihrer Anwendung Arbeitseinheiten verwenden und ein Programm eine aus einer Warteschlange empfangene Nachricht nicht erfolgreich verarbeiten kann, wird der MQGET-Aufruf zurückgesetzt.

Im Feld *BackoutCount* (Rücksetzungszähler) des Nachrichtendeskriptor zählt der Warteschlangenmanager, wie oft dies geschieht. Dieser Zähler wird im Deskriptor jeder beteiligten Nachricht verwaltet. Durch diesen Zähler können wertvolle Informationen zur Effizienz einer Anwendung bereitgestellt werden. Nachrichten mit Rücksetzungszählern, die sich im Laufe der Zeit erhöhen, werden wiederholt abgelehnt. Stellen Sie beim Entwickeln Ihrer Anwendung sicher, dass die Ursachen für diese Zunahme analysiert und die Nachrichten entsprechend bearbeitet werden.

z/OS Setzen Sie in IBM MQ for z/OS das Attribut **HardenGetBackout** auf MQQA_BACKOUT_HARDENED, damit der Rücksetzungszähler bei Neustarts des Warteschlangenzählers erhalten bleibt; andernfalls wird der Rücksetzungszähler für die einzelnen Nachrichten bei einem Neustart des Warteschlangenmanagers nicht fortgeführt. Das Festlegen des Attributs auf diese Weise hat den Nachteil einer zusätzlichen Verarbeitung.

Auf IBM MQ for IBM i IBM i, AIX, Linux, and Windows-Systemen wird der Rücksetzungszähler auch bei einem Neustart des Warteschlangenmanagers fortgeführt.

z/OS Wenn Sie innerhalb einer Arbeitseinheit Nachrichten aus einer Warteschlange entfernen, können Sie in IBM MQ for z/OS auch eine Nachricht markieren, sodass sie nicht wieder verfügbar gemacht wird, wenn die Arbeitseinheit von der Anwendung zurückgesetzt wird. Die markierte Nachricht wird behandelt, wie wenn Sie in einer neuen Arbeitseinheit abgerufen worden wäre. Sie markieren die Nachricht, deren Zurücksetzen übersprungen werden soll, mit der Option MQGMO_MARK_SKIP_BACKOUT(in der MQGMO-Struktur), sobald Sie den MQGET-Aufruf verwenden. Weitere Informationen zu diesem Verfahren finden Sie unter „Backout überspringen” auf Seite 840.

Berichtsnachrichten zur Problembestimmung verwenden

Der ferne Warteschlangenmanager kann Fehler (z. B. das Fehlschlagen beim Einreihen einer Nachricht in einen Warteschlange während Ihres MQI-Aufrufs) nicht melden, aber er kann eine Berichtsnachricht mit den Informationen senden, wie Ihre Nachricht verarbeitet wurde.

In Ihrer Anwendung können Sie Berichtsnachrichten (MQPUT) erstellen und die Option zum Empfangen von Berichtsnachrichten auswählen (in diesem Fall werden sie von einer anderen Anwendung oder von einem Warteschlangenmanager gesendet).

Berichtsnachrichten erstellen

Durch Berichtsnachrichten kann eine Anwendung einer anderen Anwendung melden, dass sie die gesendete Nachricht nicht verarbeiten kann.

Allerdings muss das Feld *Report* zunächst analysiert werden, um zu ermitteln, ob die Anwendung, die die Nachricht gesendet hat, über Probleme informiert werden soll. Wenn ermittelt wurde, dass eine Berichtsnachricht erforderlich ist, müssen Sie Folgendes entscheiden:

- Sie müssen angeben, ob die gesamte ursprüngliche Nachricht, nur die ersten 100 Byte der Daten oder keine Teile der ursprünglichen Nachricht eingeschlossen werden sollen.
- Sie müssen angeben, was mit der ursprünglichen Nachricht geschehen soll. Sie können diese löschen oder sie kann in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden.
- Sie müssen angeben, ob die Inhalte der Felder *MsgId* und *CorrelId* ebenfalls erforderlich sind.

Geben Sie im Feld *Feedback* die Ursache für die Erstellung der Berichtsnachricht an. Reihen Sie Ihre Berichtsnachrichten in die Empfangswarteschlange für Antworten einer Anwendung ein. Weitere Informationen finden Sie unter Feedback.

Berichtsnachrichten (MQGET) anfordern und empfangen

Wenn Sie eine Nachricht an eine andere Anwendung senden, werden Sie nicht über Probleme informiert, es sei denn, Sie geben im Feld *Report* das gewünschte Feedback an. Informationen zu den verfügbaren Optionen finden Sie unter Struktur des Berichtsfelds.

Warteschlangenmanager reihen Berichtsnachrichten immer in die Empfangswarteschlange für Antworten einer Anwendung ein und es wird empfohlen, dass Ihre eigene Anwendung genauso vorgeht. Wenn Sie die Funktion für die Berichtsnachricht verwenden, geben Sie den Namen Ihrer Empfangswarteschlange für Antworten im Nachrichtendeskriptor Ihrer Nachricht an. Andernfalls schlägt der MQPUT-Aufruf fehl.

Ihre Anwendung muss Prozeduren enthalten, mit denen Ihre Empfangswarteschlange für Antworten überwacht wird und alle Nachrichten verarbeitet werden, die darin empfangen werden. Berücksichtigen Sie, dass eine Berichtsnachricht die gesamte ursprüngliche Nachricht, die ersten 100 Byte der ursprünglichen Nachricht oder keine Teile der ursprünglichen Nachrichten enthalten kann.

Der Warteschlangenmanager gibt im Feld *Feedback* der Berichtsnachricht die Ursache für den Fehler an, beispielsweise, dass die Zielwarteschlange nicht vorhanden ist. Ihre Programme sollten genauso vorgehen.

Weitere Informationen zu Berichtsnachrichten finden Sie unter „Berichtsnachrichten” auf Seite 21.

Über Fernzugriff ermittelte Fehler

Selbst wenn der lokale Warteschlangenmanager in Ihrem MQI-Aufruf beim Senden von Nachrichten an eine ferne Warteschlange keinen Fehler gefunden hat, können andere Faktoren die Verarbeitung durch einen fernen Warteschlangenmanager beeinflussen.

Beispielsweise ist die Zielwarteschlange voll oder eventuell überhaupt nicht vorhanden. Wenn Ihre Nachricht bei der Weiterleitung an die Zielwarteschlange von anderen temporären Warteschlangenmanagern verarbeitet werden muss, kann in jedem dieser Warteschlangenmanagern ein Fehler ermittelt werden.

Probleme bei der Übergabe einer Nachricht

Wenn ein MQPUT-Aufruf fehlschlägt, können Sie versuchen, die Nachricht erneut in die Warteschlange einzureihen, sie an den Absender zurückzugeben oder sie in eine Warteschlange für nicht zustellbare Nachrichten einzureihen.

Jede Option hat ihre Vorzüge, aber Sie sollten nicht versuchen, eine Nachricht erneut einzureihen, wenn der MQPUT-Aufruf fehlschlug, weil die Zielwarteschlange voll war. In diesem Fall kann die Nachricht durch das Einreihen in die Warteschlange für nicht zustellbare Nachrichten später an die ordnungsgemäße Zielwarteschlange übergeben werden.

Nachrichtenübermittlung wiederholen

Bevor die Nachricht in eine Warteschlange für nicht zustellbare Nachrichten eingereiht wird, versucht ein ferner Warteschlangenmanager, die Nachricht erneut in die Warteschlange einzureihen, wenn die Attribute *MsgRetryCount* und *MsgRetryInterval* für den Kanal festgelegt sind oder wenn ein Wiederholungsexitprogramm vorhanden ist, das der Warteschlangenmanager verwenden kann (der Name des Programms ist im Feld *MsgRetryExitId* des Kanalattributs enthalten).

Wenn das Feld *MsgRetryExitId* leer ist, werden die Werte der Attribute *MsgRetryCount* und *MsgRetryInterval* verwendet.

Wenn das Feld *MsgRetryExitId* nicht leer ist, wird das in diesem Feld angegebene Exitprogramm ausgeführt. Weitere Informationen zur Verwendung Ihrer eigenen Exitprogramme finden Sie im Abschnitt „Kanalexitprogramme für Messaging-Kanäle” auf Seite 1013.

Nachricht an Absender zurückgeben

Sie können eine Nachricht an den Absender zurückgeben, indem das Generieren einer Berichtsnachricht angefordert wird, die die vollständige ursprüngliche Nachricht enthält.

Einzelheiten zu den Optionen der Berichtsnachricht finden Sie unter „Berichtsnachrichten” auf Seite 21.

Warteschlange für nicht zustellbare Nachrichten verwenden

Wenn ein Warteschlangenmanager eine Nachricht nicht übergeben kann, versucht er, die Nachricht in die zugehörige Warteschlange für nicht zustellbare Nachrichten einzureihen. Diese Warteschlange sollte bei der Installation des Warteschlangenmanagers definiert werden.

Ihre Programme können die Warteschlange für nicht zustellbare Nachrichten auf die gleiche Weise wie der Warteschlangenmanager verwenden. Sie können den Namen der Warteschlange für nicht zustellbare Nachrichten ermitteln, indem Sie das Warteschlangenmanagerobjekt (mit dem MQOPEN-Aufruf) öffnen und das Attribut **DeadLetterQName** abfragen (mit dem MQINQ-Aufruf).

Wenn der Warteschlangenmanager eine Nachricht in diese Warteschlange einreiht, wird der Nachricht ein Header hinzugefügt. Das Format des Headers wird in der Struktur des Headers einer nicht zustellbaren Nachricht (MQDLH) beschrieben; siehe MQDLH - Header einer nicht zustellbaren Nachricht. Dieser Header enthält den Namen der Zielwarteschlange und den Grund für das Einreihen der Nachricht in die Warteschlange für nicht zustellbare Nachrichten. Er muss entfernt und das Problem behoben werden, damit die Nachricht in die vorgesehene Warteschlange eingereiht werden kann. Der Warteschlangenmanager ändert außerdem das Feld *Format* des Nachrichtendeskriptors (MQMD), um anzuzeigen, dass die Nachricht eine MQDLH-Struktur enthält.

MQDLH, Struktur

Es wird empfohlen, allen Nachrichten, die in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden, eine MQDLH-Struktur hinzuzufügen; wenn Sie jedoch die von bestimmten IBM MQ-Produkten bereitgestellte Steuerroutine der Warteschlange für nicht zustellbare Nachrichten verwenden möchten, müssen Sie Ihren Nachrichten eine MQDLH-Struktur hinzufügen.

Durch das Hinzufügen des Headers zu einer Nachricht wird die Nachricht möglicherweise zu lang für die Warteschlange für nicht zustellbare Nachrichten. Stellen Sie deshalb immer sicher, dass Ihre Nachrichten um mindestens den Wert der Konstante MQ_MSG_HEADER_LENGTH kürzer sind als die maximal

zulässige Größe für die Warteschlange für nicht zustellbare Nachrichten. Wie groß die Nachrichten in einer Warteschlange maximal sein dürfen, wird durch den Wert des Attributs **MaxMsgLength** der Warteschlange bestimmt. Stellen Sie für die Warteschlange für nicht zustellbare Nachrichten sicher, dass dieses Attribut auf den im Warteschlangenmanager maximal zulässigen Wert gesetzt ist. Wenn Ihre Anwendung eine Nachricht nicht übergeben kann und die Nachricht zum Einreihen in die Warteschlange für nicht zustellbare Nachrichten zu lang ist, befolgen Sie den Ratschlag in der Beschreibung der MQDLH-Struktur.

Stellen Sie sicher, dass die Warteschlange für nicht zustellbare Nachrichten überwacht wird und alle darin empfangenen Nachrichten verarbeitet werden. Die Steuerroutine der Warteschlange für nicht zustellbare Nachrichten wird als Stapeldienstprogramm ausgeführt und kann verschiedene Aktionen in ausgewählten Nachrichten in der Warteschlange für nicht zustellbare Nachrichten ausführen. Weitere Einzelheiten finden Sie unter „Verarbeitung der Warteschlange für nicht zustellbare Nachrichten” auf Seite 1095.

Wenn eine Konvertierung der Daten erforderlich ist, konvertiert der Warteschlangenmanager die Headerinformationen beim Verwenden der Option MQGMO_CONVERT im MQGET-Aufruf. Wenn es sich bei dem Prozess zum Einreihen der Nachricht um einen Nachrichtenkanalagenten handelt, folgt auf den Header der gesamte Text der ursprünglichen Nachricht.

Die in die Warteschlange für nicht zustellbare Nachrichten eingereihten Nachrichten werden möglicherweise abgeschnitten, wenn sie für diese Warteschlangen zu lang sind. Ein möglicher Hinweis darauf ist, dass die Nachrichten in der Warteschlange für nicht zustellbare Nachrichten die gleiche Länge haben wie der Wert des Attributs **MaxMsgLength** der Warteschlange.

Verarbeitung der Warteschlange für nicht zustellbare Nachrichten
In diesem Abschnitt finden Sie Informationen zur allgemeinen Programmierschnittstelle bei der Verarbeitung der Warteschlange für nicht zustellbare Nachrichten.

Die Verarbeitung der Warteschlange für nicht zustellbare Nachrichten ist von den Anforderungen des lokalen Systems abhängig, beachten Sie aber die folgenden Informationen, wenn Sie die Spezifikation zusammenstellen:

- Für die Nachricht kann ermittelt werden, dass ein Header für die Warteschlange für nicht zustellbare Nachrichten vorhanden ist, da der Wert MQFMT_DEAD_LETTER_HEADER im MQMD-Feld für das Format angegeben ist.
- Wenn ein Nachrichtenkanalagent diese Nachricht unter IBM MQ for z/OS mit CICS in die Warteschlange für nicht zustellbare Nachrichten einreiht, wird das Feld *PutApplType* auf MQAT_CICS gesetzt und das Feld *PutApplName* auf die *ApplId* (Anwendungs-ID) des CICS-Systems gefolgt vom Transaktionsnamen des Nachrichtenkanalagenten.
- Die Ursache für die Weiterleitung der Nachricht in die Warteschlange für nicht zustellbare Nachrichten wird im Feld *Reason* des Headers der Warteschlange für nicht zustellbare Nachrichten angegeben.
- Der Header der Warteschlange für nicht zustellbare Nachrichten enthält Einzelheiten zum Namen der Zielwarteschlange und des Warteschlangenmanagers.
- Der Header der Warteschlange für nicht zustellbare Nachrichten verfügt über Felder, die vor dem Einreihen der Nachricht in die Zielwarteschlange im Nachrichtendeskriptor wiederhergestellt werden müssen. Diese sind:
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- Der Nachrichtendeskriptor entspricht dem PUT-Aufruf durch die ursprüngliche Anwendung, mit Ausnahme der drei gezeigten Felder (Encoding, CodedCharSetId und Format).

Ihre Anwendung für die Warteschlange für nicht zustellbare Nachrichten muss mindestens eine der folgenden Tasks ausführen:

- Prüfen Sie das Feld *Reason*. Eine Nachricht kann von einem Nachrichtenkanalagenten aus den folgenden Gründen eingereiht worden sein:
  - Die Nachricht war länger als die maximale Nachrichtengröße für den Kanal

Die Ursache ist MQRC_MSG_TOO_BIG_FOR_CHANNEL

– Die Nachricht konnte nicht in die Zielwarteschlange eingereiht werden

  Die Ursache ist ein beliebiger MQRC_*-Ursachencode, der von einer MQPUT-Operation zurückgegeben werden kann.

– Diese Aktion wurde von einem Benutzerexit angefordert

  Der Ursachencode wurde vom Benutzerexit bereitgestellt oder ist der standardmäßige Ursachencode MQRC_SUPPRESSED_BY_EXIT

- Versuch der Weiterleitung der Nachricht an den vorgesehenen Empfängern, wann immer dies möglich ist.
- Aufbewahren der Nachricht für eine bestimmte Dauer, bevor diese gelöscht wird, wenn die Ursache für die Umleitung ermittelt wurde, das Problem aber nicht sofort behoben werden kann.
- Erteilen von Anweisungen an Administratoren zum Beheben von Problemen, wenn diese ermittelt wurden.
- Löschen von Nachrichten, die beschädigt sind oder aus anderen Gründen nicht verarbeitet werden können.

Nachrichten, die aus der Warteschlange für nicht zustellbare Nachrichten wiederhergestellt wurden, können auf zwei Arten bearbeitet werden:

1. Bei einer Nachricht für eine lokale Warteschlange:
   - Führen Sie alle erforderlichen Codeumsetzungen aus, um die Anwendungsdaten zu extrahieren
   - Führen Sie Codeumsetzugen in den Daten aus, falls es sich um eine lokale Funktion handelt
   - Reihen Sie die daraus resultierende Nachricht mit allen Einzelheiten, die der Nachrichtendeskriptor wiederhergestellt hat, in die lokale Warteschlange ein
2. Bei einer Nachricht für eine ferne Warteschlange reihen Sie die Nachricht in die Warteschlange ein.

Informationen zur Verarbeitung nicht zustellbarer Nachrichten in einer Umgebung für die verteilte Steuerung von Warteschlangen finden Sie im Abschnitt Was geschieht, wenn eine Nachricht nicht übergeben werden kann?.

Multicast-Programmierung

In diesem Abschnitt finden Sie Informationen zu den Tasks zum Programmieren von IBM MQ Multicast, beispielsweise das Herstellen einer Verbindung zu einem Warteschlangenmanager und die Erstellung von Ausnahmeberichten.

IBM MQ Multicast wurde so entwickelt, dass es möglichst transparent für den Benutzer und trotzdem weiterhin kompatibel mit vorhandenen Anwendungen ist. Durch das Definieren eines COMMINFO-Objekts und das Festlegen der Parameter **MCAST** und **COMMINFO** für das Objekt TOPIC ist für vorhandene IBM MQ-Anwendungen kein wesentliches Umarbeiten erforderlich, damit Multicast verwendet werden kann. Es müssen jedoch möglicherweise einige Einschränkungen (weitere Informationen dazu finden Sie im Abschnitt „Multicasting und die MQI” auf Seite 1096) und Sicherheitsprobleme (siehe Multicast-Sicherheit) beachtet werden.

Multicasting und die MQI

In diesem Abschnitt finden Sie Informationen, die Ihnen das Verständnis der wichtigsten MQI-Konzepte und deren Beziehung zu IBM MQ Multicast erleichtern.

Multicast-Subskriptionen sind nicht permanent. Da keine physischen Warteschlangen verwendet werden, können die von permanente Subskriptionen erstellten Offlinenachrichten nirgendwo gespeichert werden.

Nachdem eine Anwendung ein Multicastthema subskribiert hat, wird der Anwendung eine Objektkennung zurückgegeben, die verarbeitet oder aus der ein MQGET-Aufruf ausgegeben werden kann, wie wenn es sich um eine Kennung für eine Warteschlange handeln würde. Das bedeutet, dass nur verwaltete Multicast-Subskriptionen (mit MQSO_MANAGED erstellte Subskriptionen) unterstützt werden. Deshalb

ist es nicht möglich, eine Subskription vorzunehmen und die Nachrichten auf eine Warteschlange zu verweisen. Das bedeutet, dass Nachrichten, die beim Subskriptionsaufruf zurückgegeben wurden, von der Objektkennung gelesen werden müssen. Die Nachrichten werden im Client in einem Nachrichtenpuffer gespeichert, bis sie vom Client gelesen werden; weitere Informationen finden Sie unter Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei. Wenn der Client die Veröffentlichungsrate nicht aufrechterhalten kann, werden die Nachrichten bei Bedarf gelöscht, wobei die ältesten Nachrichten zuerst verworfen werden.

Normalerweise entscheidet der Administrator, ob Multicast für eine Anwendung verwendet wird. Dies wird durch das Festlegen des MCAST-Attributs eines TOPIC-Objekt angegeben. Wenn eine Veröffentlichungsanwendung sicherstellen muss, dass Multicast nicht verwendet wird, kann dazu die Option `MQOO_NO_MULTICAST` verwendet werden. Entsprechend kann eine Subskriptionsanwendung mit der Option `MQSO_NO_MULTICAST` sicherstellen, dass Multicast nicht verwendet wird.

IBM MQ Multicast unterstützt die Verwendung von Nachrichtenselektoren. Durch einen Selektor registriert sich eine Anwendung nur für die Nachrichten, deren Eigenschaften die als Auswahlzeichenfolge dargestellte SQL92-Abfrage erfüllen. Weitere Informationen zu Nachrichtenselektoren finden Sie unter „Selectors” auf Seite 32.

In der folgenden Tabelle werden alle wichtigen MQI-Konzepte und deren Beziehung zu Multicast aufgeführt:

Tabelle 154. MQI-Konzepte und deren Beziehung zu Multicast

MQI-Konzept	**Aktion beim Versuch, Multicast zu verwenden**	**Ursachencode**
Nachricht mit Nulllänge einreihen	Abgelehnt	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
Gruppe	Abgelehnt	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentation	Abgelehnt	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
Verteilerlisten	Abgelehnt	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR
MQINQ	Abgelehnt für Themenkennungen: MQINQ- und MQSET-Aufrufe von Themen werden nicht unterstützt.	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	Akzeptiert für verwaltetes Handle. Nur `Current Depth` (Aktuelle Tiefe) kann abgefragt werden.	• Wenn der Wert 'Current Depth' ist, gibt es keinen anwendbaren Ursachencode. • Wenn ein beliebiger anderer Wert als 'Current Depth' vorhanden ist, ist der Ursachencode 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR.
MQSET	Abgelehnt für alle Handles.	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
Transaktionen (XA oder nicht)	Abgelehnt	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
Nachricht durchsuchen	Abgelehnt	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE

Tabelle 154. MQI-Konzepte und deren Beziehung zu Multicast (Forts.)

MQI-Konzept	Aktion beim Versuch, Multicast zu verwenden	Ursachencode
Nachrichten sperren	Abgelehnt	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Mit Markierung durchsuchen	Abgelehnt	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Kontext übergeben	Abgelehnt	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	Abgelehnt. Zugriff und Aufruf von MQPUT1 in einem Thema, das nur für Multicast definiert ist, ist ungültig.	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
Permanente Subskription	Abgelehnt, wenn das Thema als reines Multicast-Thema markiert ist; andernfalls wird eine Nicht-Multicast-Subskription vorgenommen.	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	Abgelehnt. Wenn die Themenzeichenfolge größer als 255 Zeichen ist, wird sie im Client abgelehnt.	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
Nicht verwaltete Subskription vorgenommen	Abgelehnt, wenn das Thema als reines Multicast-Thema markiert ist; andernfalls wird eine Nicht-Multicast-Subskription vorgenommen.	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	Abgelehnt	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

Die folgenden Elemente gehen näher auf einige der MQI-Konzepte aus der vorherigen Tabelle ein und stellen Informationen zu einigen MQI-Konzepten bereit, die sich nicht in der Tabelle befinden:

Nachrichtenpersistenz
Für nicht permanente Multicast-Subskribenten werden persistente Nachrichten vom Publisher in nicht wiederherstellbarer Form übergeben.

Nachricht abschneiden
Das Abschneiden der Nachricht wird unterstützt, d. h. eine Anwendung hat folgende Möglichkeiten:

1. Ausgeben von MQGET.

2. Abrufen von MQRC_TRUNCATED_MSG_FAILED.
3. Zuordnen eines umfangreichen Puffers.
4. Erneutes Ausgeben von MQGET zum Abruf der Nachricht.

Ablauf der Subskription
Ablauf der Subskription wird nicht unterstützt. Jeder Versuch, einen Ablauf festzulegen, wird ignoriert.

Hochverfügbarkeit für Multicast

Verwenden Sie diese Informationen, um sich mit kontinuierlichen Peer-to-Peer-Operationen in IBM MQ Multicast vertraut zu machen;, obwohl IBM MQ eine Verbindung zu einem IBM MQ-Warteschlangenmanager herstellt, werden Nachrichten nicht durch diesen Warteschlangenmanager geleitet.

Obwohl eine Verbindung zu einem Warteschlangenmanager hergestellt werden muss, um den MQOPEN- oder MQSUB-Aufruf im Multicast-Themenobjekt ausgeben zu können, werden die Nachrichten selbst nicht durch den Warteschlangenmanager geleitet. Nachdem der MQOPEN- oder MQSUB-Aufruf im Multicast-Themenobjekt abgeschlossen wurde, können Multicastnachrichten daher weiterhin übertragen werden, selbst wenn die Verbindung zum Warteschlangenmanager getrennt wurde. Der Vorgang kann auf zwei Arten ausgeführt werden:

Es wird eine normale Verbindung zum Warteschlangenmanager hergestellt
Die Multicastkommunikation ist möglich, solange eine Verbindung zum Warteschlangenmanager vorhanden ist. Wenn die Verbindung fehlschlägt, werden die normalen MQI-Regeln angewendet; so gibt ein MQPUT-Aufruf an die Multicast-Objektkennung beispielsweise folgende Fehlermeldung zurück: 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN.

Die Verbindung von einem Client zum Warteschlangenmanager wird wiederhergestellt
Die Multicastkommunikation ist auch während des Zyklus zur Verbindungswiederholung möglich. Das bedeutet, dass selbst bei einer unterbrochenen Verbindung zum Warteschlangenmanager das Einreihen und Lesen von Multicastnachrichten nicht beeinträchtigt ist. Der Client versucht, die Verbindung zu einem Warteschlangenmanager wiederherzustellen, und wenn die Verbindungswiederholung fehlschlägt, ist die Verbindungskennung unterbrochen und alle MQI-Aufrufe, einschließlich der Multicastaufrufe, schlagen fehl. Weitere Informationen finden Sie im Abschnitt Automatische Clientverbindungswiederholung.

Wenn eine Anwendung explizit einen MQDISC-Aufruf ausgibt, werden alle Multicastsubskriptionen und Objektkennungen geschlossen.

Kontinuierliche Peer-to-Peer-Operation in Multicast

Ein Vorteil der Peer-to-Peer-Kommunikation zwischen den Clients ist, dass die Nachrichten nicht durch den Warteschlangenmanager geleitet werden müssen; wenn also die Verbindung zum Warteschlangenmanager unterbrochen wird, kann die Nachrichtenübertragung fortgesetzt werden. Die folgenden Einschränkungen gelten für die Anforderungen an die kontinuierliche Nachrichtenübertragung in diesem Modus:

- Die Verbindung muss mit einer der MQCNO_RECONNECT_*-Optionen für die kontinuierliche Operation hergestellt worden sein. Durch diesen Prozess wird bei einer möglichen Unterbrechung der Übertragungssitzung der tatsächliche Verbindungshandle nicht unterbrochen und ist stattdessen im Status der Verbindungswiederherstellung. Wenn die Verbindungswiederholung fehlschlägt, ist auch der Verbindungshandle unterbrochen, wodurch alle weiteren MQI-Aufrufe verhindert werden.
- In diesem Modus werden nur MQPUT, MQGET, MQINQ und Async Consume unterstützt. Für MQOPEN-, MQCLOSE- oder MQDISC-Verben muss die Verbindung zum Warteschlangenmanager wiederhergestellt werden, damit sie abgeschlossen werden können.
- Statusabläufe zum Warteschlangenmanager werden gestoppt; jeder Status im Warteschlangenmanager kann daher als abgelaufen markiert sein oder fehlen. Das bedeutet, dass der Client möglicherweise Nachrichten sendet und empfängt und im Warteschlangenmanager kein Status bekannt ist. Weitere Informationen finden Sie im Abschnitt Überwachung von Multicastanwendungen.

Datenkonvertierung in der MQI für die Multicast-Nachrichtenübertragung

In diesen Informationen erfahren Sie, wie die Datenkonvertierung für die IBM MQ-Multicast-Nachrichtenübertragung funktioniert.

IBM MQ Multicast ist ein gemeinsam genutzes, verbindungsunabhängiges Protokoll. Daher ist es nicht für jeden Client möglich, bestimmte Anforderungen an die Datenkonvertierung zu stellen. Jeder Client, der denselben Multicastdatenstrom subskribiert, empfängt dieselben Binärdaten; wenn eine Konvertierung von IBM MQ-Daten erforderlich ist, wird die Konvertierung deshalb lokal auf jedem Client ausgeführt.

Die Daten werden auf dem Client für den IBM MQ-Multicastverkehr konvertiert. Wenn die Option **MQGMO_CONVERT** angegeben ist, wird die Datenkonvertierung wie angefordert ausgeführt. Für benutzerdefinierte Formate muss der Datenkonvertierungsexit auf dem Client installiert sein; im Abschnitt „Datenkonvertierungsexits schreiben" auf Seite 1035 finden Sie Informationen darüber, welche Bibliotheken sich jetzt in den Client- und Serverpaketen befinden.

Informationen zur Verwaltung der Datenkonvertierung finden Sie unter Datenkonvertierung für die Multicast-Nachrichtenübertragung aktivieren.

Weitere Informationen zur Datenkonvertierung finden Sie unter Datenkonvertierung .

Weitere Informationen zu Datenkonvertierungsexits und `ClientExitPath`finden Sie im Abschnitt ZeilengruppeClientExitder Clientkonfigurationsdatei.

Erstellen von Ausnahmeberichten für Multicast

In diesem Abschnitt finden Sie Informationen zu Ereignishandlern für IBM MQ Multicast und zur Erstellung von Ausnahmeberichten für IBM MQ Multicast.

IBM MQ Multicast unterstützt Sie bei der Problembestimmung, indem der Ereignishandler zum Erstellen von Berichten für Multicast-Ereignisse aufgerufen wird, die mithilfe des Standardmechanismus für den IBM MQ-Ereignishandler dokumentiert werden.

Ein einzelnes Multicast-Ereignis kann dazu führen, dass mehr als ein IBM MQ-Ereignis aufgerufen wird, da möglicherweise mehrere MQHCONN-Verbindungshandles den gleichen Multicast-Sender oder -Empfänger verwenden. Durch jede Multicast-Ausnahme wird allerdings nur ein Ereignishandler pro IBM MQ-Verbindung aufgerufen.

Mit der Konstante IBM MQ MQCBDO_EVENT_CALL können Anwendungen einen Callback registrieren, um nur IBM MQ-Ereignisse zu empfangen, und mit der Konstante MQCBDO_MC_EVENT_CALL können Anwendungen einen Callback registrieren, um nur Multicast-Ereignisse zu empfangen. Wenn beide Konstanten verwendet werden, werden beide Ereignistypen empfangen.

Multicast-Ereignisse anfordern

IBM MQ Multicast-Ereignisse verwenden die Konstante MQCBDO_MC_EVENT_CALL im Feld `cbd.Options`. Im folgenden Beispiel wird gezeigt, wie Multicast-Ereignisse angefordert werden:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

Wenn die Option MQCBDO_MC_EVENT_CALL für das Feld `cbd.Options` angegeben ist, werden anstelle von Ereignissen auf Verbindungsebene nur IBM MQ Multicast-Ereignisse an den Ereignishandler gesendet. Um das Senden beider Ereignistypen an den Ereignishandler anzufordern, muss die Anwendung die Konstante MQCBDO_EVENT_CALL im Feld `cbd.Options` sowie die Konstante MQCBDO_MC_EVENT_CALL angeben, wie im folgenden Beispiel gezeigt wird:

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

Wenn keine dieser Konstanten verwendet wird, werden nur Ereignisse auf Verbindungsebene an den Ereignishandler gesendet.

Weitere Informationen zu den Werten für das Feld `Options` finden Sie im Abschnitt Optionen (MQLONG).

Format von Multicast-Ereignissen

IBM MQ Multicast-Ausnahmen enthalten einige unterstützende Informationen, die im Parameter **Buffer** derCallback-Funktion zurückgegeben werden. Der Zeiger **Buffer** verweist auf eine Feldgruppe von Zeigern und im Feld `MQCBC.DataLength` wird die Größe der Feldgruppe in Byte angegeben. Das erste Element der Feldgruppe verweist immer auf eine kurze Textbeschreibung des Ereignisses. Je nach Ereignistyp werden möglicherweise weitere Parameter angegeben. In der folgenden Tabelle werden die Ausnahmen aufgeführt:

Tabelle 155. Beschreibung des Multicast-Ereigniscodes

Ereigniscode	Beschreibung	Zusätzliche Daten
MQMCEV_PACKET_LOSS	Nicht behebbarer Paketverlust	Anzahl der verlorenen Pakete
MQMCEV_HEARTBEAT_TIMEOUT	Lange Abwesenheit des Steuerpakets für das Überwachungssignals	nicht zutreffend
MQMCEV_VERSION_CONFLICT	Empfang von neueren Protokollversionsdatenpaketen	nicht zutreffend
MQMCEV_RELIABILITY	Andere Zuverlässigkeitsmodi von Sender und Empfänger	nicht zutreffend
MQMCEV_CLOSED_TRANS	Themenübertragung wird von 1 Quelle geschlossen	nicht zutreffend
MQMCEV_STREAM_ERROR	Fehler im Datenstrom erkannt	nicht zutreffend
MQMCEV_NEW_SOURCE	Eine neue Quelle beginnt mit der Übertragung zu dem Thema	Quellenstruktur
MQMCEV_RECEIVE_QUEUE_TRIMMED	Aus PacketQ aufgrund von Zeit- oder Leerzeichenablauf entfernte Datenpakete	Anzahl der abgeschnittenen Pakete
MQMCEV_PACKET_LOSS_NACK_EXPIRE	Nicht behebbarer Paketverlust aufgrund von NACK-Ablauf	Anzahl der verlorenen Pakete
MQMCEV_ACK_RETRIES_EXCEEDED	Aus dem Protokoll entfernte Datenpakete, nachdem **max_ack_retries** überschritten wurde	Anzahl der entfernten Pakete
MQMCEV_STREAM_SUSPEND_NACK	NACKs sind auf einem von diesem Thema akzeptierten Datenstrom ausgesetzt worden	Aussetz-Datenstrom-ID Zeit in Millisekunden, für die der Datenstrom ausgesetzt wird
MQMCEV_STREAM_RESUME_NACK	NACKs sind fortgesetzt worden, nachdem sie auf einem Datenstrom ausgesetzt worden waren	Datenstrom-ID

Tabelle 155. Beschreibung des Multicast-Ereigniscodes (Forts.)

Ereigniscode	Beschreibung	Zusätzliche Daten
MQMCEV_STREAM_EXPELLED	Ein von diesem Thema akzeptierter Datenstrom ist aufgrund einer Ausschluss-Anforderung zurückgewiesen worden	Datenstrom-ID
MQMCEV_FIRST_MESSAGE	Erste Nachricht von einer Quelle	Nachrichtennummer
MQMCEV_LATE_JOIN_FAILURE	Fehler beim Starten einer Sitzung für eine späte Verknüpfung	nicht zutreffend
MQMCEV_MESSAGE_LOSS	Nicht behebbarer Nachrichtenverlust	Anzahl der verlorenen Nachrichten
MQMCEV_SEND_PACKET_FAILURE	Multicastsender konnte ein Multicastpaket nicht senden	nicht zutreffend
MQMCEV_REPAIR_DELAY	Multicastempfänger erhielt kein Reparaturdatenpaket für ein hervorragendes NAK	nicht zutreffend
MQMCEV_MEMORY_ALERT_ON	Empfangspuffer des Empfängers füllen sich	Prozentsatz der Pufferpool-Auslastung
MQMCEV_MEMORY_ALERT_OFF	Empfangspuffer des Empfängers sind wieder im Normalzustand	Prozentsatz der Pufferpool-Auslastung
MQMCEV_NACK_ALERT_ON	Anforderungsrate für Reparaturdatenpakete des Empfängers hat oberen Grenzwert erreicht	Aktuelle Anforderungsrate für Reparaturen in Datenpaketen pro Sekunde
MQMCEV_NACK_ALERT_OFF	Anforderungsrate für Reparaturdatenpakete des Empfängers ist wieder im Normalzustand	Aktuelle Anforderungsrate für Reparaturen in Datenpaketen pro Sekunde
MQMCEV_REPAIR_ALERT_ON	Senderate für Reparaturdatenpakete des Senders hat oberen Grenzwert erreicht	nicht zutreffend
MQMCEV_REPAIR_ALERT_OFF	Senderate für Reparaturdatenpakete des Senders ist wieder im Normalzustand	nicht zutreffend
MQMCEV_SHM_DEST_UNUSABLE	Der von einem Sender-Themenziel verwendete gemeinsam genutzte Speicherbereich wurde als unbrauchbar erkannt	nicht zutreffend
MQMCEV_SHM_PORT_UNUSABLE	Der von einer Empfängerinstanz verwendete gemeinsam genutzte Speicherport wurde als unbrauchbar erkannt	nicht zutreffend
MQMCEV_CCT_GETTIME_FAILED	Fehler bei der Abrufzeit von Coordinated Cluster Time	nicht zutreffend
MQMCEV_DEST_INTERFACE_FAILURE	Die von einem Sender-Themenziel verwendete Netzschnittstelle ist ausgefallen und eine Sicherungsnetzschnittstelle ist nicht verfügbar	

Tabelle 155. Beschreibung des Multicast-Ereigniscodes (Forts.)		
Ereigniscode	**Beschreibung**	**Zusätzliche Daten**
MQMCEV_DEST_INTERFACE_FAILOVER	Die von einem Sender-Themenziel verwendete Netzschnittstelle ist ausgefallen; die Überbrückung zu einer anderen Schnittstelle wurde erfolgreich abgeschlossen	
MQMCEV_PORT_INTERFACE-FAILURE	Die von einem Empfänger-rmmPort verwendete Netzschnittstelle ist ausgefallen und eine Sicherungsnetzschnittstelle ist nicht verfügbar (oder ist ebenfalls fehlgeschlagen)	RMM-Konfiguration
MQMCEV_PORT_INTERFACE_FAILOVER	Die von einem Empfänger-rmmPort verwendete Netzschnittstelle ist ausgefallen; die Überbrückung zu einer anderen Schnittstelle wurde erfolgreich abgeschlossen	RMM-Konfiguration

Codierung in C

Beachten Sie bei der Codierung von IBM MQ-Programmen in der Programmiersprache C die Informationen in den folgenden Abschnitten.

- „Parameter der MQI-Aufrufe” auf Seite 1103
- „Parameter mit einem nicht definierten Datentyp” auf Seite 1104
- „Datentypen” auf Seite 1104
- „Binärzeichenfolgen bearbeiten” auf Seite 1104
- „Zeichenfolgen bearbeiten” auf Seite 1104
- „Anfangswerte für Strukturen” auf Seite 1105
- „Anfangswerte für dynamische Strukturen” auf Seite 1105
- „Verwendung in der Programmiersprache C++” auf Seite 1106

Parameter der MQI-Aufrufe

Reine Eingabeparameter (*input-only*) vom Typ MQHCONN, MQHOBJ, MQHMSG oder MQLONG werden nach Wert weitergegeben. Bei allen anderen Parametern wird die *Adresse* des Parameters nach Wert weitergegeben.

Nicht alle Parameter, die nach Adresse übergeben werden, müssen bei jedem Aufruf einer Funktion angegeben werden. Ist ein bestimmter Parameter nicht erforderlich, kann beim Funktionsaufruf statt der Adresse der Parameterdaten ein Nullzeiger als Parameter angegeben werden. Parameter, bei denen dies möglich ist, sind in den Aufrufbeschreibungen angegeben.

Als Wert der Funktion wird kein Parameter zurückgegeben; dies bedeutet in der Terminologie der Programmiersprache C, dass alle Funktionen 'void' zurückgeben.

Die Attribute der Funktion werden durch die Makrovariable MQENTRY definiert. Der Wert dieser Makrovariablen hängt von der Umgebung ab.

Parameter mit einem nicht definierten Datentyp

Die Funktionen MQGET, MQPUT und MQPUT1 verfügen jeweils über den Parameter **Buffer**, der einen nicht definierten Datentyp hat. Mit diesem Parameter werden die Nachrichtendaten einer Anwendung gesendet und empfangen.

Parameter dieser Art werden in den C-Beispielen als Arrays von MQBYTE dargestllt. Parameter können auf diese Weise deklariert werden, es ist in der Regel jedoch praktischer, sie als Struktur zu deklarieren, die den Aufbau der Daten in der Nachricht beschreibt. Der Funktionsparameter wird als void-Zeiger deklariert; die Adresse von Daten kann jedoch als Parameter im Funktionsaufruf angegeben werden.

Datentypen

Alle Datentypen werden mit der Anweisung `typedef` definiert.

Für jeden Datentyp wird auch der entsprechende Zeigerdatentyp definiert. Als Name des Zeigerdatentyps wird der Name des Elementar- oder Strukturdatentyps mit dem Präfix P verwendet, das auf einen Zeiger hinweist. Die Attribute des Zeigers werden durch die Makrovariable MQPOINTER definiert. Der Wert dieser Makrovariablen hängt von der Umgebung ab. Der folgende Code veranschaulicht die Deklaration von Zeigerdatentypen:

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

Binärzeichenfolgen bearbeiten

Zeichenfolgen von Binärdaten werden als einer der MQBYTEn-Datentypen deklariert.

Verwenden Sie beim Kopieren, Vergleichen oder Festlegen von Feldern dieses Typs die C-Funktionen `memcpy`, `memcmp` oder `memset`:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant      */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

Sehen Sie von der Verwendung der Zeichenfolgefunktionen 'strcpy', 'strcmp', 'strncpy' oder 'strncmp' ab, da sie mit den als MQBYTE24 deklarierten Daten nicht ordnungsgemäß funktionieren.

Zeichenfolgen bearbeiten

Wenn der Warteschlangenmanager Zeichendaten an die Anwendung zurückgibt, füllt er die Zeichendaten immer bis zur definierten Feldlänge mit Leerzeichen auf. Der Warteschlangenmanager gibt keine auf NULL endenden Zeichenfolgen zurück, Sie können diese jedoch in Ihrer Eingabe verwenden. Verwenden Sie daher zum Kopieren, Vergleichen oder Verketten derartiger Zeichenfolgen die Zeichenfolgefunktionen 'strncpy', 'strncmp' bzw. 'strncat'.

Verwenden Sie keine Zeichenfolgefunktionen, bei denen die Zeichenfolge mit einer Null endet ('strcpy', 'strcmp' und 'strcat'). Darüber hinaus dürfen Sie zur Bestimmung der Zeichenfolgelänge nicht die Funktion 'strlen verwenden'; bestimmen Sie die Länge des Felds stattdessen mit der Funktion 'sizeof'.

Anfangswerte für Strukturen

Die Include-Datei <cmqc.h> definiert verschiedene Makrovariablen, mit denen Sie Anfangswerte für die Strukturen bereitstellen, wenn Sie die Instanzen dieser Strukturen deklarieren. Die Namen dieser Makrovariablen haben das Format MQxxx_DEFAULT. Dabei steht MQxxx für den Namen der Struktur. Verwenden Sie sie wie folgt:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

Bei manchen Zeichenfeldern definiert die MQI bestimmte gültige Werte (z. B. für die *StrucId*-Felder oder für das Feld *Format* im MQMD). Für jeden der gültigen Werte werden zwei Makrovariablen zur Verfügung gestellt:

- Eine Makrovariable definiert den Wert als Zeichenfolge mit einer Länge - ausschließlich der implizierten Null, die genau der definierten Länge des Felds entspricht. In den folgenden Beispielen stellt das Symbol ¬ ein einzelnes Leerzeichen dar:

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Verwenden Sie diese Form bei den Funktionen 'memcpy' und 'memcmp'.
- Die zweite Makrovariable definiert den Wert als Zeichen-Array; der Name dieser Makrovariablen ist der Name der Zeichenfolgeform mit dem Suffix _ARRAY. For example:

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  Verwenden Sie diese Form, um das Feld zu initialisieren, wenn eine Instanz der Struktur mit Werten deklariert wird, die nicht mit denen von der Makrovariablen MQMD_DEFAULT bereitgestellten Werten übereinstimmen.

Anfangswerte für dynamische Strukturen

Wenn eine variable Anzahl von Instanzen einer Struktur erforderlich ist, werden die Instanzen üblicherweise in einem mittels der Funktionen 'calloc' oder 'malloc' dynamisch abgerufenen Hauptspeicher erstellt.

Zur Initialisierung der Felder in solchen Strukturen empfiehlt sich folgendes Verfahren:

1. Deklarieren Sie eine Instanz der Struktur unter Verwendung der entsprechenden MQxxx_DEFAULT-Makrovariablen für die Initialisierung der Struktur. Diese Instanz dient als *Modell* für andere Instanzen:

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                    /* declare model instance */
   ```

   Codieren Sie die statischen oder automatischen Schlüsselwörter der Deklaration, um der Modellinstanz je nach Bedarf eine statische oder dynamische Lebensdauer zu geben.
2. Rufen Sie mit der Funktion 'calloc' oder 'malloc' einen Speicher für eine dynamische Instanz der Struktur ab:

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                    /* get storage for dynamic instance */
   ```
3. Kopieren Sie die Modellinstanz mit der Funktion 'memcpy' in die dynamische Instanz:

   ```
   memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                    /* initialize dynamic instance */
   ```

Verwendung in der Programmiersprache C++

In der Programmiersprache C++ enthalten die Headerdateien folgende zusätzliche Anweisungen, die nur dann eingeschlossen werden, wenn ein C++-Compiler verwendet wird:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Windows Codierung in Visual Basic

Informationen, die beim Codieren von IBM MQ-Programmen in Microsoft Visual Basic zu berücksichtigen sind. Visual Basic wird nur unter Windows unterstützt.

Anmerkung:

Stabilized Ab IBM WebSphere MQ 7.0, außerhalb der .NET-Umgebung, wurde die Unterstützung für Visual Basic (VB) auf der Stufe IBM WebSphere MQ 6.0 stabilisiert. Die meisten neuen Funktonen von IBM WebSphere MQ 7.0 oder höher stehen für VB-Anwendungen nicht zur Verfügung. Wenn Sie in VB.NET programmieren, verwenden Sie die IBM MQ-Klassen für .NET. Weitere Informationen finden Sie unter .NET-Anwendungen entwickeln.

Deprecated Ab IBM MQ 9.0 wird die Unterstützung für Microsoft Visual Basic 6.0 nicht mehr verwendet. IBM MQ-Klassen für .NET sind die empfohlene Ersatztechnologie.

Sie könne eine unbeabsichtigte Übersetzung von Binärdaten, die zwischen Visual Basic und IBM MQ übergeben werden, vermeiden, indem Sie anstelle von MQSTRING eine MQBYTE-Definition verwenden. CMQB.BAS definiert mehrere neue MQBYTE-Typen, die einer C-Byte-Definition entsprechen, und verwendet diese in IBM MQ-Strukturen. Für die MQMD-Struktur (Nachrichtendeskriptor) beispielsweise ist 'MsgId' (Nachrichten-ID) als MQBYTE24 definiert.

Visual Basic verwendet keinen Zeigerdatentyp, daher wird per Offset statt per Zeiger auf andere IBM MQ-Datenstrukturen verwiesen. Deklarieren Sie eine zusammengesetzte Struktur, die aus den beiden Komponentenstrukturen besteht, und geben Sie diese beim Aufruf an. Die IBM MQ-Unterstützung für Visual Basic stellt einen MQCONNXAny-Aufruf bereit, um dies zu ermöglichen und den Clientanwendungen zu gestatten, die Kanaleigenschaften bei einer Clientverbindung anzugeben. Sie akzeptiert eine Struktur ohne Typ (MQCNOCD) anstelle der typischen MQCNO-Struktur.

Die MQCNOCD-Struktur ist eine zusammengesetzte Struktur, die aus MQCNO-Parameter, gefolgt von einem MQCD-Parameter, besteht. Diese Struktur ist in der Exits-Headerdatei CMQXB deklariert. Initialisieren Sie eine MQCNOCD-Struktur mit der Routine MQCNOCD_DEFAULTS. Ein Beispiel für den Aufruf von MQCONNX steht zur Verfügung ('amqscnxb.vbp').

MQCONNXAny verwendet dieselben Parameter wie MQCONNX, außer dass der Parameter **ConnectOpts** als Datentyp 'Any' deklariert wird, nicht als MQCNO. So kann die Funktion die MQCNO- oder die MQCNOCD-Struktur akzeptieren. Diese Funktion ist in der Haupt-Headerdatei CMQB deklariert.

Zugehörige Konzepte

„Visual Basic-Programme in Windows vorbereiten” auf Seite 1073
Informationen, die bei der Verwendung von Microsoft Visual Basic-Programmen unter Windows zu beachten sind.

Zugehörige Verweise

„Visual Basic-Anwendungen mit IBM MQ MQI client-Code verknüpfen” auf Seite 969
Unter Windows können Sie Microsoft Visual Basic-Anwendungen mit dem IBM MQ MQI client-Code verknüpfen.

Codierung in COBOL

Beachten Sie bei der Codierung von IBM MQ-Programmen in COBOL die Informationen im folgenden Abschnitt.

Benannte Konstanten

Die angezeigten Namen von Konstanten enthalten den Unterstrich (_) als Teil des Namens. In COBOL müssen Sie den Bindestrich (-) anstelle des Unterstrichs verwenden. Konstanten mit Zeichenfolgewerte verwenden das einzelne Anführungszeichen (') als Zeichenfolgebegrenzer. Damit der Compiler dieses Zeichen akzeptiert, verwenden Sie die Compileroption APOST.

Die Kopierdatei CMQV enthält Deklarationen der benannten Konstanten als Elemente der Ebene 10. Um die Konstanten zu verwenden, deklarieren Sie das Element der Ebene 01 explizit und kopieren Sie dann die Deklarationen der Konstanten mithilfe der Anweisung COPY:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

Allerdings belegen die Konstanten durch diese Methode Programmspeicher, selbst wenn nicht auf sie verwiesen wird. Wenn die Konstanten in viele separate Programme innerhalb der gleichen Ausführungseinheit eingeschlossen werden, existieren mehrere Kopien der Konstanten; dies könnte dazu führen, dass eine erhebliche Menge an Hauptspeicher belegt wird. Sie können dies vermeiden, indem Sie der Ebene-01-Deklaration die Klausel GLOBAL hinzufügen:

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

Dadurch wird nur *einem* Konstantensatz Speicher in der Ausführungseinheit zugeordnet; ein Verweis auf die Konstanten ist jedoch durch *jedes* Programm in der Ausführungseinheit möglich und nicht nur durch das Programm, das die Ebene-01-Deklaration enthält.

Strukturausrichtung sicherstellen

Achten Sie darauf, dass die IBM MQ-Strukturen, die zum Start im MQ-Aufruf weitergegeben werden, an Wortgrenzen ausgerichtet sind. Eine Wortgrenze besteht aus 4 Bytes bei 32-Bit-Prozessen, 8 Bytes bei 64-Bit-Prozessen und 16 Bytes bei 128-Bit-Prozessen (IBM i).

Platzieren Sie wenn möglich alle IBM MQ-Strukturen gemeinsam, damit sie alle an Wortgrenzen ausgerichtet sind.

Coding in System/390 assembler language (Message queue interface)

Note the information in the following sections when coding IBM MQ for z/OS programs in assembler language.

Names

The names of parameters in the descriptions of calls, and the names of fields in the descriptions of structures are shown in mixed case. In the assembler-language macros supplied with IBM MQ, all names are in uppercase.

Using the MQI calls

The MQI is a call interface, so assembler-language programs must observe the OS linkage convention.

In particular, before they issue an MQI call, assembler-language programs must point register R13 at a save area of at least 18 full words. This save area provides storage for the called program. It stores the registers of the caller before their contents are destroyed, and restores the contents of the caller's registers on return.

Note: This is important for CICS assembler-language programs that use the DFHEIENT macro to set up their dynamic storage, but that choose to override the default DATAREG from R13 to other registers. When the CICS Resource Manager Interface receives control from the stub, it saves the current contents of the registers at the address to which R13 is pointing. Failing to reserve a save area for this purpose gives unpredictable results, and will probably cause an abend in CICS.

Declaring constants

Most constants are declared as equates in macro CMQA.

However, the following constants cannot be defined as equates, and these are not included when you call the macro using default options:

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

To include them, add the keyword EQUONLY=NO when you call the macro.

CMQA is protected against multiple declaration, so you can include it many times. However, the keyword EQUONLY takes effect only the first time that the macro is included.

Specifying the name of a structure

To allow more than one instance of a structure to be declared, the macro that generates the structure prefixes the name of each field with a user-specifiable string and an underscore character (_).

Specify the string when you invoke the macro. If you do not specify a string, the macro uses the name of the structure to construct the prefix:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

The structure declarations in Call descriptions show the default prefix.

Specifying the form of a structure

The macros can generate structure declarations in one of two forms, controlled by the DSECT parameter:

DSECT=YES
An assembler-language DSECT instruction is used to start a new data section; the structure definition immediately follows the DSECT statement. No storage is allocated, so no initialization is possible. The label on the macro invocation is used as the name of the data section; if no label is specified, the name of the structure is used.

DSECT=NO
Assembler-language DC instructions are used to define the structure at the current position in the routine. The fields are initialized with values, which you can specify by coding the relevant parameters on the macro invocation. Fields for which no values are specified on the macro invocation are initialized with default values.

DSECT=NO is assumed if the DSECT parameter is not specified.

Controlling the listing

You can control the appearance of the structure declaration in the assembler-language listing with the LIST parameter:

LIST=YES
The structure declaration appears in the assembler-language listing.

LIST=NO
The structure declaration does not appear in the assembler-language listing. This is assumed if the LIST parameter is not specified.

Specifying initial values for fields

You can specify the value to be used to initialize a field in a structure by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required.

For example, to declare a message descriptor structure with the *MsgType* field initialized with MQMT_REQUEST, and the *ReplyToQ* field initialized with the string MY_REPLY_TO_QUEUE, use the following code:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

If you specify a named constant (or equate) as a value on the macro invocation, use the CMQA macro to define the named constant. You must not enclose in single quotation marks (' ') values that are character strings.

Writing reenterable programs

IBM MQ uses its structures for both input and output. If you want your program to remain reenterable:

1. Define working storage versions of the structures as DSECTs, or define the structures inline within an already-defined DSECT. Then copy the DSECT to storage that is obtained using:
   - For batch and TSO programs, the STORAGE or GETMAIN z/OS assembler macros

- For CICS, the working storage DSECT (DFHEISTG) or the EXEC CICS GETMAIN command

  To correctly initialize these working storage structures, copy a constant version of the corresponding structure to the working storage version.

  Note: The MQMD and MQXQH structures are each more than 256 bytes long. To copy these structures to storage, use the MVCL assembler instruction.

2. Reserve space in storage by using the LIST form (MF=L) of the CALL macro. When you use the CALL macro to make an MQI call, use the EXECUTE form (MF=E) of the macro, using the storage reserved earlier, as shown in the example under “Using CEDF” on page 1110. For more examples of how to do this, see the assembler language sample programs as shipped with IBM MQ.

Use the assembler language RENT option to help you to determine if your program is reenterable.

For information on writing reenterable programs, see *z/OS MVS Application Development Guide: Assembler Language Programs*.

Using CEDF

If you want to use the CICS-supplied transaction, CEDF (CICS Execution Diagnostic Facility) to help you to debug your program, add the ,VL keyword to each CALL statement, for example:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

The previous example is reenterable assembler-language code where PARMAREA is an area in the working storage that you specified.

Using the MQI calls

The MQI is a call interface, so assembler-language programs must observe the OS linkage convention. In particular, before they issue an MQI call, assembler-language programs must point register R13 at a save area of at least 18 full words. This save area provides storage for the called program. It stores the registers of the caller before their contents are destroyed, and restores the contents of the caller's registers on return.

Note: This is important for CICS assembler-language programs that use the DFHEIENT macro to set up their dynamic storage, but that choose to override the default DATAREG from R13 to other registers. When the CICS Resource Manager Interface receives control from the stub, it saves the current contents of the registers at the address to which R13 is pointing. Failing to reserve a proper save area for this purpose gives unpredictable results, and will probably cause an abend in CICS.

IBM i IBM MQ-Programme in RPG codieren (nur IBM i)

In der IBM MQ-Dokumentation werden die Parameter von Aufrufen, die Namen der Datentypen, die Felder von Strukturen und die Namen von Konstanten jeweils mit ihren ausgeschriebenen Namen beschrieben. In RPG werden diese Namen auf maximal sechs Großbuchstaben abgekürzt.

So wird in RPG beispielsweise der Name des Felds *MsgType* zu *MDMT*. Weitere Informationen hierzu finden Sie im Referenzhandbuch IBM i Application Programming Reference (ILE/RPG).

Coding in PL/I (z/OS only)

Useful information when coding for IBM MQ in PL/I.

Structures

Structures are declared with the BASED attribute, and so do not occupy any storage unless the program declares one or more instances of a structure.

An instance of a structure can be declared using the `like` attribute, for example:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

The structure fields are declared with the INITIAL attribute; when the `like` attribute is used to declare an instance of a structure, that instance inherits the initial values defined for that structure. You need to set only those fields where the value required is different from the initial value.

PL/I is not sensitive to case, and so the names of calls, structure fields, and constants can be coded in lowercase, uppercase, or mixed case.

Named constants

The named constants are declared as macro variables; as a result, named constants that are not referred to by the program do not occupy any storage in the compiled procedure.

However, the compiler option that causes the source to be processed by the macro preprocessor must be specified when the program is compiled.

All the macro variables are character variables, even the ones that represent numeric values. Although this might seem counter intuitive, it does not result in any data-type conflict after the macro variables have been substituted by the macro processor, for example:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

Prozedurale IBM MQ-Beispielprogramme verwenden

Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

Informationen zu diesem Vorgang

Es gibt zwei Gruppen von Beispielprogrammen:

- Multi Beispielprogramme für Multiplatforms.
- z/OS Beispielprogramme für z/OS.

Prozedur

- Weitere Informationen zu den Beispielprogrammen finden Sie unter den folgenden Links:
  - Multi „Beispielprogramme plattformübergreifend verwenden” auf Seite 1112
  - z/OS „Using the sample programs for z/OS” auf Seite 1221

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 7
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Bevor Sie beginne, Ihre IBM MQ-Anwendungen zu entwerfen und zu schreiben, sollten Sie sich mit den grundlegenden IBM MQ-Konzepten vertraut machen.

„Anwendungen für IBM MQ entwickeln” auf Seite 5

Sie können Anwendungen zum Senden und Empfangen von Nachrichten sowie zum Verwalten Ihrer Warteschlangenmanager und der zugehörigen Ressourcen entwickeln. IBM MQ unterstützt Anwendungen, die in vielen verschiedenen Sprachen und Frameworks geschrieben sind.

„Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 52
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von IBM MQ bereitgestellten Funktionen verwendet werden sollen.

„Prozedurale Anwendungen für die Warteschlangensteuerung erstellen” auf Seite 758
Dieser Abschnitt enthält Informationen zum Erstellen von Anwendungen für die Warteschlangensteuerung, zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager und zum Öffnen und Schließen von Objekten.

„Prozedurale Clientanwendungen schreiben” auf Seite 961
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„Publish/Subscribe-Anwendungen schreiben” auf Seite 851
Beginnen Sie nun, Ihre eigenen IBM MQ-Publish/Subscribe-Anwendungen zu schreiben.

„Prozedurale Anwendung erstellen” auf Seite 1052
Sie können eine IBM MQ-Anwendung in einer von mehreren prozeduralen Sprachen schreiben und die Anwendung auf mehreren unterschiedlichen Plattformen ausführen.

„Prozedurale Programmfehler handhaben” auf Seite 1090
In diesen Informationen werden Fehler erläutert, die den MQI-Aufrufen Ihrer Anwendungen beim Ausgeben eines Aufrufs oder bei der Übergabe einer Nachricht an den Zielort zugeordnet werden.

Multi Beispielprogramme plattformübergreifend verwenden

Diese prozeduralen Beispielprogramme sind im Lieferumfang des Produkts enthalten. Sie sind in C und COBOL geschrieben und veranschaulichen die typische Verwendung der Message Queue Interface (MQI).

Informationen zu diesem Vorgang

Die Beispielprogramme sind nicht zur Darstellung allgemeiner Programmiertechniken gedacht. Daher sind keine Fehlerprüfungen für Produktionsprogramme enthalten.

Der Quellcode für die einzelnen Beispiele wird zusammen mit dem Produkt bereitgestellt; der Code enthält Kommentare zur Erläuterung der Message-Queuing-Techniken, die in den Programmen veranschaulicht werden.

IBM i Informationen zur RPG-Programmierung finden Sie in der IBM i Application Programming Reference (ILE/RPG).

Die Namen der Beispielprogramme beginnen mit dem Präfix `amq`. Das vierte Zeichen gibt die Programmiersprache und, wo erforderlich, den Compiler an:

- s: Programmiersprache C
- 0: COBOL-Sprache auf IBM und Micro Focus-Compilern
- i: COBOL-Sprache nur auf IBM-Compilern
- m: COBOL-Sprache nur auf Micro Focus-Compilern

Das achte Zeichen der ausführbaren Funktion gibt an, ob das Beispiel im lokalen Bindungsmodus oder Clientmodus ausgeführt wird. Wenn es kein achtes Zeichen gibt, wird das Beispiel im lokalen Bindungsmodus ausgeführt. Wenn das achte Zeichen "c" ist, wird das Beispiel im Clientmodus ausgeführt.

Bevor Sie die Beispielanwendungen ausführen können, müssen Sie zunächst einen Warteschlangenmanager erstellen und konfigurieren. Informationen zum Einrichten des Warteschlangenmanagers für die Annahme von Clientverbindungen finden Sie im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.

Prozedur

- Weitere Informationen zu den Beispielprogrammen finden Sie unter den folgenden Links:
  - „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114
  - „Beispielprogramme vorbereiten und ausführen” auf Seite 1123
  - „API Exit-Beispielprogramm” auf Seite 1131
  - „Das Asynchronous Consumption-Beispielprogramm” auf Seite 1132
  - „Das Asynchronous Put-Beispielprogramm” auf Seite 1133
  - „Die Browse-Beispielprogramme” auf Seite 1134
  - „Das Beispielprogramm 'Browser'” auf Seite 1135
  - „Das CICS Transaction-Beispielprogramm” auf Seite 1137
  - „Das Connect-Beispielprogramm” auf Seite 1137
  - „Das Data-Conversion-Beispielprogramm” auf Seite 1138
  - „Beispiele zur Datenbankkoordination” auf Seite 1139
  - „Beispielprogramm für eine Steuerroutine der Warteschlange für nicht zustellbare Nachrichten” auf Seite 1145
  - „Das Distribution List-Beispielprogramm” auf Seite 1146
  - „Die Echo-Beispielprogramme” auf Seite 1147
  - „Die Get-Beispielprogramme” auf Seite 1148
  - „Beispielprogramme zur Hochverfügbarkeit” auf Seite 1149
  - „Die Inquire-Beispielprogramme” auf Seite 1154
  - „Beispielprogramm zum Abfragen der Eigenschaften eines Nachrichtenhandles” auf Seite 1155
  - „Publish/Subscribe-Beispielprogramme” auf Seite 1155
  - „Das Publish Exit-Beispielprogramm” auf Seite 1160
  - „Die Put-Beispielprogramme” auf Seite 1161
  - „Die Reference Message-Beispielprogramme” auf Seite 1163
  - „Die Request-Beispielprogramme” auf Seite 1171
  - „Die Set--Beispielprogramme” auf Seite 1177
  - „TLS-Beispielprogramm” auf Seite 1178
  - „Auslösebeispielprogramme” auf Seite 1182
  - „TUXEDO-Beispiele unter AIX, Linux, and Windows verwenden” auf Seite 1184
  - „SSPI-Sicherheitsexit unter Windows verwenden” auf Seite 1194
  - „Beispiele unter Verwendung ferner Warteschlangen ausführen” auf Seite 1195
  - „Das Beispielprogramm 'Clusterwarteschlangenüberwachung' (AMQSCLM)” auf Seite 1195
  - „Das Beispielprogramm für die Suche nach Verbindungsendpunkten (CEPL)” auf Seite 1205

Zugehörige Konzepte

„Beispielprogramme in C++” auf Seite 557
In vier bereitgestellten Beispielprogrammen wird das Abrufen und Einreihen von Nachrichten veranschaulicht.

Zugehörige Tasks

„Using the sample programs for z/OS” auf Seite 1221
The sample procedural applications that are delivered with IBM MQ for z/OS demonstrate typical uses of the Message Queue Interface (MQI).

Multi *In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen*

Zusammenstellung von Tabellen, in denen die Verfahren aufgeführt sind, die von den IBM MQ-Beispielprogrammen veranschaulicht werden.

Alle Beispielprogramme öffnen und schließen Warteschlangen mittels MQOPEN- bzw. MQCLOSE-Aufrufen. Diese Methoden sind also nicht separat in den Tabellen aufgeführt. Weitere Informationen finden Sie unter der Überschrift der für Sie relevanten Plattform.

z/OS Weitere Informationen zur z/OS-Plattform finden Sie im Abschnitt „Using the sample programs for z/OS" auf Seite 1221.

Linux AIX *Beispiele für Systeme mit AIX and Linux*

In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ for AIX or Linux veranschaulicht werden.

Unter „Beispielprogramme unter AIX and Linux vorbereiten und ausführen" auf Seite 1127 erfahren Sie, wo die Beispielprogramme für IBM MQ for AIX or Linux gespeichert sind.

Tabelle 156 auf Seite 1114 Die Tabelle führt auf, welche C- und COBOL-Quellendateien bereitgestellt werden und ob eine ausführbare Server- oder Clientdatei enthalten ist.

Tabelle 156. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter AIX and Linux.

Eine Tabelle mit vier Spalten. In der ersten Spalte sind die von den Beispielen veranschaulichten Verfahren angegeben. In der zweiten Spalte sind die C-Beispiele und in der dritten Spalte die COBOL-Beispiele aufgeführt, die die in der ersten Spalte genannten Verfahren veranschaulichen. Die vierte Spalte zeigt, ob eine in C ausführbare Serverdatei enthalten ist oder nicht, und die fünfte Spalte, ob eine in C ausführbare Clientdatei enthalten ist oder nicht.

Verfahren	**C (Quelle) („1" auf Seite 1117)**	**COBOL (Quelle) („2" auf Seite 1117)**	**Server (ausführbar in C)**	**Client (ausführbar in C)**
Verwendung der Publish/Subscribe-Schnittstelle	amqspuba amqssuba amqssbxa	kein Beispiel	amqspub amqssub amqssbx	kein Beispiel
Einreihen von Nachrichten mithilfe des MQPUT-Aufrufs	amqsput0	amq0put0	amqsput	amqsputc
Einreihen einer einzelnen Nachricht mithilfe des MQPUT1-Aufrufs	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
Nachrichten in eine Verteilerliste stellen („3" auf Seite 1117)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Beantworten einer Anforderungsnachricht	amqsinqa	amqminqx amqiinqx	amqsinq	kein Beispiel
Abrufen von Nachrichten mittels Durchsuchen (ohne Wartezeit)	amqsgbr0	amq0gbr0	amqsgbr	kein Beispiel
Abrufen von Nachrichten (Wartezeit mit Zeitlimit)	amqsget0	amq0get0	amqsget	amqsgetc
Abruf von Nachrichten (unbegrenzte Wartezeit)	amqstrg0	kein Beispiel	amqstrg	amqstrgc

Tabelle 156. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter AIX and Linux.

Eine Tabelle mit vier Spalten. In der ersten Spalte sind die von den Beispielen veranschaulichten Verfahren angegeben. In der zweiten Spalte sind die C-Beispiele und in der dritten Spalte die COBOL-Beispiele aufgeführt, die die in der ersten Spalte genannten Verfahren veranschaulichen. Die vierte Spalte zeigt, ob eine in C ausführbare Serverdatei enthalten ist oder nicht, und die fünfte Spalte, ob eine in C ausführbare Clientdatei enthalten ist oder nicht.

(Forts.)

Verfahren	**C (Quelle) („1” auf Seite 1117)**	**COBOL (Quelle) („2” auf Seite 1117)**	**Server (ausführbar in C)**	**Client (ausführbar in C)**
Abrufen von Nachrichten (mit Datenkonvertierung)	amqsecha	kein Beispiel	amqsech	kein Beispiel
Referenznachrichten in eine Warteschlange einreihen („3” auf Seite 1117)	amqsprma	kein Beispiel	amqsprm	amqsprmc
Referenznachrichten aus einer Warteschlange abrufen („3” auf Seite 1117)	amqsgrma	kein Beispiel	amqsgrm	amqsgrmc
Referenznachricht-Kanalexit („3” auf Seite 1117)	amqsqrma amqsxrma	kein Beispiel	amqsxrm	kein Beispiel
Anzeige der ersten 20 Zeichen einer Nachricht	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Anzeige vollständiger Nachrichten	amqsbcg0	kein Beispiel	amqsbcg	amqsbcgc
Verwenden einer gemeinsam genutzten Eingabewarteschlange	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
Verwenden einer ausschließlichen Eingabewarteschlange	amqstrg0	amq0req0	amqstrg	amqstrgc
Verwenden des MQINQ-Aufrufs	amqsinqa	amqminqx amqiinqx	amqsinq	kein Beispiel
Verwenden des MQSET-Aufrufs	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
Verwenden einer Empfangswarteschlange für Antworten	amqsreq0	amq0req0	amqsreq	amqsreqc
Anforderung von Nachrichtenausnahmen	amqsreq0	amq0req0	amqsreq	kein Beispiel
Akzeptieren einer abgeschnittenen Nachricht	amqsgbr0	amq0gbr0	amqsgbr	kein Beispiel
Verwenden eines aufgelösten Warteschlangennamens	amqsgbr0	amq0gbr0	amqsgbr	kein Beispiel
Auslösen eines Prozesses	amqstrg0	kein Beispiel	amqstrg	amqstrgc
Verwendung der Datenkonvertierung	(„4” auf Seite 1117)	kein Beispiel	kein Beispiel	kein Beispiel
IBM MQ (koordinierende XA-konforme Datenbankmanager) ruft mittels SQL eine einzelne Datenbank auf	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	kein Beispiel	kein Beispiel

Tabelle 156. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter AIX and Linux.

Eine Tabelle mit vier Spalten. In der ersten Spalte sind die von den Beispielen veranschaulichten Verfahren angegeben. In der zweiten Spalte sind die C-Beispiele und in der dritten Spalte die COBOL-Beispiele aufgeführt, die die in der ersten Spalte genannten Verfahren veranschaulichen. Die vierte Spalte zeigt, ob eine in C ausführbare Serverdatei enthalten ist oder nicht, und die fünfte Spalte, ob eine in C ausführbare Clientdatei enthalten ist oder nicht.

(Forts.)

Verfahren	**C (Quelle) („1” auf Seite 1117)**	**COBOL (Quelle) („2” auf Seite 1117)**	**Server (ausführbar in C)**	**Client (ausführbar in C)**
IBM MQ (koordinierende XA-konforme Datenbankmanager) ruft mittels SQL zwei Datenbanken auf	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	kein Beispiel	kein Beispiel
CICS-Transaktion („5” auf Seite 1117)	amqscic0.ccs	kein Beispiel	amqscic0	kein Beispiel
Encina-Transaktion („3” auf Seite 1117)	amqsxae0	kein Beispiel	amqsxae0	kein Beispiel
TUXEDO-Transaktion zum Einreihen von Nachrichten („6” auf Seite 1117)	amqstxpx	kein Beispiel	kein Beispiel	kein Beispiel
TUXEDO-Transaktion zum Abrufen von Nachrichten („6” auf Seite 1117)	amqstxgx	kein Beispiel	kein Beispiel	kein Beispiel
Server für TUXEDO („6” auf Seite 1117)	amqstxsx	kein Beispiel	kein Beispiel	kein Beispiel
Steuerroutine der Warteschlange für nicht zustellbare Nachrichten	Verzeichnis `./tools/c/Samples/dlq` („7” auf Seite 1117)	kein Beispiel	amqsdlq	kein Beispiel
In einem MQI-Client, Einreihen einer Nachricht	kein Beispiel	kein Beispiel	kein Beispiel	amqsputc
In einem MQI-Client, Abrufen einer Nachricht	kein Beispiel	kein Beispiel	kein Beispiel	amqsgetc
Verbinden mit einem Warteschlangenmanager über MQCONNX	amqscnxc	kein Beispiel	kein Beispiel	amqscnxc
Verwendung von API-Exits	amqsaxe0	kein Beispiel	amqsaxe	kein Beispiel
Exit für Clusterlastausgleich	amqswlm0	kein Beispiel	amqswlm	kein Beispiel
Asynchrones Einreihen von Nachrichten und Abrufen des Status mit dem MQSTAT-Aufruf	amqsapt0	kein Beispiel	amqsapt	amqsaptc
Wiederverbindungsfähige Clients	amqsphac amqsghac amqsmhac	kein Beispiel	Nicht zutreffend	amqsphac amqsghac amqsmhac
Verwenden von Nachrichtenkonsumenten zum asynchronen Konsumieren von Nachrichten aus mehreren Warteschlangen	amqscbf0	kein Beispiel	amqscbf	amqscbfc

Tabelle 156. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter AIX and Linux.

Eine Tabelle mit vier Spalten. In der ersten Spalte sind die von den Beispielen veranschaulichten Verfahren angegeben. In der zweiten Spalte sind die C-Beispiele und in der dritten Spalte die COBOL-Beispiele aufgeführt, die die in der ersten Spalte genannten Verfahren veranschaulichen. Die vierte Spalte zeigt, ob eine in C ausführbare Serverdatei enthalten ist oder nicht, und die fünfte Spalte, ob eine in C ausführbare Clientdatei enthalten ist oder nicht.

(Forts.)

Verfahren	C (Quelle) („1” auf Seite 1117)	COBOL (Quelle) („2” auf Seite 1117)	Server (ausführbar in C)	Client (ausführbar in C)
Angeben von TLS-Verbindungsinformationen in MQCONNX	amqssslc	kein Beispiel	Nicht zutreffend	amqssslc

Anmerkungen:

1. Die ausführbare Version der IBM MQ MQI client-Beispielprogramme verwendet dieselbe Quelle wie die Beispielprogramme für eine Serverumgebung.
2. Kompilieren Sie Programme, die mit 'amqm' beginnen, mit dem Micro Focus COBOL-Compiler, Programme, die mit 'amqi' beginnen, mit dem IBM COBOL-Compiler und Programme, die mit 'amq0' beginnen, mit einem der beiden Compiler.
3. AIX Unterstützung nur unter IBM MQ for AIX.
4. AIX Unter IBM MQ for AIX heißt dieses Programm `amqsvfc0.c`.
5. AIX CICS wird nur von IBM MQ for AIX unterstützt.
6. Linux TUXEDO wird von IBM MQ for Linux on System p nicht unterstützt.
7. Die Quelle für die Steuerroutine der Warteschlange für nicht zustellbare Nachrichten besteht aus mehreren Dateien und wird in einem separaten Verzeichnis bereitgestellt.

Weitere Informationen zur Unterstützung von Systemen mit AIX and Linux finden Sie im Abschnitt Systemvoraussetzungen für IBM MQ.

Windows *Beispiele für IBM MQ for Windows*

In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ for Windows veranschaulicht werden.

Im Abschnitt Tabelle 157 auf Seite 1117 wird aufgelistet, welche C- bzw. COBOL-Quellendateien bereitgestellt werden und ob eine ausführbare Server- oder Clientdatei enthalten ist.

Tabelle 157. IBM MQ for Windows-Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL)

Verfahren	C (Quelle)	COBOL (Quelle)	Server (ausführbar in C)	Client (ausführbar in C)
Verwendung der Publish/Subscribe-Schnittstelle	amqspuba amqssuba amqssbxa	kein Beispiel	amqspub amqssub amqssbx	kein Beispiel
Einreihen von Nachrichten mithilfe des MQPUT-Aufrufs	amqsput0	amq0put0	amqsput	amqsputc

Tabelle 157. IBM MQ for Windows-Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) (Forts.)

Verfahren	C (Quelle)	COBOL (Quelle)	Server (ausführbar in C)	Client (ausführbar in C)
Einreihen einer einzelnen Nachricht mithilfe des MQPUT1-Aufrufs	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
Nachrichten in eine Verteilerliste einreihen	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Beantworten einer Anforderungsnachricht	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Abrufen von Nachrichten (ohne Wartezeit)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Abrufen von Nachrichten (Wartezeit mit Zeitlimit)	amqsget0	amq0get0	amqsget	amqsgetc
Abruf von Nachrichten (unbegrenzte Wartezeit)	amqstrg0	kein Beispiel	amqstrg	amqstrgc
Abrufen von Nachrichten (mit Datenkonvertierung)	amqsecha	kein Beispiel	amqsech	amqsechc
Einreihen von Referenznachrichten in eine Warteschlange	amqsprma	kein Beispiel	amqsprm	amqsprmc
Abruf von Referenznachrichten aus einer Warteschlange	amqsgrma	kein Beispiel	amqsgrm	amqsgrmc
Kanalexit für Referenznachrichten	amqsqrma amqsxrma	kein Beispiel	amqsxrm	kein Beispiel
Anzeige der ersten 20 Zeichen einer Nachricht	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Anzeige vollständiger Nachrichten	amqsbcg0	kein Beispiel	amqsbcg	amqsbcgc
Verwenden einer gemeinsam genutzten Eingabewarteschlange	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Verwenden einer ausschließlichen Eingabewarteschlange	amqstrg0	amq0req0	amqstrg	amqstrgc
Verwenden des MQINQ-Aufrufs	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Verwenden des MQSET-Aufrufs	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
MQINQMP-Aufruf verwenden	amqsiqma	kein Beispiel	kein Beispiel	kein Beispiel
Verwenden einer Empfangswarteschlange für Antworten	amqsreq0	amq0req0	amqsreq	amqsreqc
Anforderung von Nachrichtenausnahmen	amqsreq0	amq0req0	amqsreq	amqsreqc
Akzeptieren einer abgeschnittenen Nachricht	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Verwenden eines aufgelösten Warteschlangennamens	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Auslösen eines Prozesses	amqstrg0	kein Beispiel	amqstrg	amqstrgc
Verwendung der Datenkonvertierung	amqsvfc0	kein Beispiel	kein Beispiel	kein Beispiel

Tabelle 157. IBM MQ for Windows-Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) (Forts.)

Verfahren	C (Quelle)	COBOL (Quelle)	Server (ausführbar in C)	Client (ausführbar in C)
IBM MQ (koordinierende XA-konforme Datenbankmanager) ruft mittels SQL eine einzelne Datenbank auf	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	kein Beispiel	kein Beispiel
IBM MQ (koordinierende XA-konforme Datenbankmanager) ruft mittels SQL zwei Datenbanken auf	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	kein Beispiel	kein Beispiel
TUXEDO-Transaktion zum Einreihen von Nachrichten	amqstxpx	kein Beispiel	kein Beispiel	kein Beispiel
TUXEDO-Transaktion zum Abrufen von Nachrichten	amqstxgx	kein Beispiel	kein Beispiel	kein Beispiel
Server für TUXEDO	amqstxsx	kein Beispiel	kein Beispiel	kein Beispiel
Steuerroutine der Warteschlange für nicht zustellbare Nachrichten	Verzeichnis `./tools/c/Samples/dlq` („1” auf Seite 1120)	kein Beispiel	amqsdlq	kein Beispiel
In einem IBM MQ MQI client eine Nachricht einreihen	kein Beispiel	kein Beispiel	kein Beispiel	amqsputc
In einem IBM MQ MQI client eine Nachricht abrufen	kein Beispiel	kein Beispiel	kein Beispiel	amqsgetc
Verbinden mit einem Warteschlangenmanager über MQCONNX	amqscnxc	kein Beispiel	kein Beispiel	amqscnxc
Verwendung von API-Exits	amqsaxe0	kein Beispiel	amqsaxe	kein Beispiel
Clusterauslastungsausgleich	amqswlm0	kein Beispiel	amqswlm	kein Beispiel
SSPI-Sicherheitsroutinen	amqsspin	kein Beispiel	amqrspin.dll	amqrspin.dll
Asynchrones Einreihen von Nachrichten und Abrufen des Status mit dem MQSTAT-Aufruf	amqsapt0	kein Beispiel	amqsapt	amqsaptc
Wiederverbindungsfähige Clients	amqsphac amqsghac amqsmhac	kein Beispiel	Nicht zutreffend	amqsphac amqsghac amqsmhac
Verwenden von Nachrichtenkonsumenten zum asynchronen Konsumieren von Nachrichten aus mehreren Warteschlangen	amqscbf0	kein Beispiel	amqscbf	amqscbfc
Angeben von TLS-Verbindungsinformationen in MQCONNX	amqssslc	kein Beispiel	Nicht zutreffend	amqssslc

Anmerkungen:

1. Die Quelle für die Steuerroutine der Warteschlange für nicht zustellbare Nachrichten besteht aus mehreren Dateien und wird in einem separaten Verzeichnis bereitgestellt.

Windows *Visual Basic-Beispiele für IBM MQ for Windows*

In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit Windows veranschaulicht werden.

Tabelle 158 auf Seite 1120 zeigt die Verfahren, die von den IBM MQ for Windows-Beispielprogrammen veranschaulicht werden.

Ein Projekt kann mehrere Dateien enthalten. Wenn Sie in Visual Basic ein Projekt öffnen, werden die anderen Dateien automatisch geladen. Ausführbare Programme werden nicht bereitgestellt.

Alle Beispielprojekte - mit Ausnahme von 'mqtrivc.vbp' - wurden zur Verwendung mit dem IBM MQ-Server eingerichtet. Informationen darüber, wie Sie die Beispielprojekte zur Verwendung mit IBM MQ-Clients ändern, finden Sie unter „Visual Basic-Programme in Windows vorbereiten” auf Seite 1073.

Tabelle 158. IBM MQ for Windows-Beispielprogramme zur Veranschaulichung der Verwendung der MQI (Visual Basic)

Verfahren	**Projektdateiname**
Einreihen von Nachrichten mithilfe des MQPUT-Aufrufs	amqsputb.vbp
Nachrichten mit dem MQGET-Aufruf abrufen	amqsgetb.vbp
Warteschlange mit dem MQGET-Aufruf durchsuchen	amqsbcgb.vbp
Einfaches MQGET- und MQPUT-Beispiel (Client)	mqtrivc.vbp
Einfaches MQGET- und MQPUT-Beispiel (Server)	mqtrivs.vbp
Zeichenfolgen und benutzerdefinierte Strukturen mit MQPUT und MQGET einreihen und abrufen	strings.vbp
Kanal mittels PCF-Strukturen starten und stoppen	pcfsamp.vbp
Warteschlange mit der MQAI erstellen	amqsaicq.vbp
Warteschlangen eines Warteschlangenmanagers mit der MQAI auflisten	amqsailq.vbp
Ereignisse mit der MQAI überwachen	amqsaiem.vbp

IBM i *Beispiele für IBM i*

In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit IBM i veranschaulicht werden.

Tabelle 159 auf Seite 1120 zeigt die Verfahren, die von den IBM MQ for IBM i-Beispielprogrammen veranschaulicht werden. Einige der dargestellten Verfahren treten in mehr als einem Beispielprogramm auf, in der Tabelle wird aber nur ein Programm aufgelistet.

Tabelle 159. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter IBM i

Verfahren	**C (Quelle) („1” auf Seite 1122)**	**COBOL (Quelle) („2” auf Seite 1122)**	**RPG (Quelle) („3” auf Seite 1122)**	**Client (ausführbar in C) (4)**
Einreihen von Nachrichten mithilfe des MQPUT-Aufrufs	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	AMQSPUTC
Einreihen von Nachrichten aus einer Datendatei mit dem MQPUT-Aufruf	AMQSPUT4	kein Beispiel	kein Beispiel	kein Beispiel

Tabelle 159. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter IBM i (Forts.)

Verfahren	**C (Quelle) („1” auf Seite 1122)**	**COBOL (Quelle) („2” auf Seite 1122)**	**RPG (Quelle) („3” auf Seite 1122)**	**Client (ausführbar in C) (4)**
Einreihen einer einzelnen Nachricht mithilfe des MQPUT1-Aufrufs	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
Nachrichten in eine Verteilerliste einreihen	AMQSPTL4	kein Beispiel	kein Beispiel	AMQSPTLC
Beantworten einer Anforderungsnachricht	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Abrufen von Nachrichten (ohne Wartezeit)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Abrufen von Nachrichten (Wartezeit mit Zeitlimit)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
Abruf von Nachrichten (unbegrenzte Wartezeit)	AMQSTRG4	kein Beispiel	AMQ3TRG4	AMQSTRGC
Abrufen von Nachrichten (mit Datenkonvertierung)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
Einreihen von Referenznachrichten in eine Warteschlange	AMQSPRM4	kein Beispiel	kein Beispiel	AMQSPRMC
Abruf von Referenznachrichten aus einer Warteschlange	AMQSGRM4	kein Beispiel	kein Beispiel	AMQSGRMC
Kanalexit für Referenznachrichten	AMQSQRM4, AMQSXRM4	kein Beispiel	kein Beispiel	kein Beispiel
Nachrichtenexit	AMQSCMX4	kein Beispiel	kein Beispiel	kein Beispiel
Anzeigen der ersten 49 Zeichen einer Nachricht	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Anzeige vollständiger Nachrichten	AMQSBCG4	kein Beispiel	kein Beispiel	AMQSBCGC
Verwenden einer gemeinsam genutzten Eingabewarteschlange	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Verwenden einer ausschließlichen Eingabewarteschlange	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Verwenden des MQINQ-Aufrufs	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Verwenden des MQSET-Aufrufs	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
Verwenden einer Empfangswarteschlange für Antworten	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Anforderung von Nachrichtenausnahmen	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Akzeptieren einer abgeschnittenen Nachricht	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Verwenden eines aufgelösten Warteschlangennamens	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Auslösen eines Prozesses	AMQSTRG4	kein Beispiel	AMQ3TRG4	AMQSTRGC
Trigger-Server	AMQSERV4	kein Beispiel	AMQ3SRV4	kein Beispiel
Verwenden eines Trigger-Servers (einschließlich CICS-Transaktionen)	AMQSERV4	kein Beispiel	AMQ3SRV4	kein Beispiel

Tabelle 159. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter IBM i (Forts.)

Verfahren	**C (Quelle) („1” auf Seite 1122)**	**COBOL (Quelle) („2” auf Seite 1122)**	**RPG (Quelle) („3” auf Seite 1122)**	**Client (ausführbar in C) (4)**
Verwendung der Datenkonvertierung	AMQSVFC4	kein Beispiel	kein Beispiel	kein Beispiel
Verwendung von API-Exits	AMQSAXE0	kein Beispiel	kein Beispiel	kein Beispiel
Clusterauslastungsausgleich	AMQSWLM0	kein Beispiel	kein Beispiel	kein Beispiel
Asynchrones Einreihen von Nachrichten und Abrufen des Status mit dem MQSTAT-Aufruf	AMQSAPT0	kein Beispiel	kein Beispiel	AMQSAPTC
Verwendung der Publish/Subscribe-Schnittstelle	AMQSPUBA, AMQSSUBA, AMQSSBXA	kein Beispiel	kein Beispiel	AMQSPUBC, AMQSSUBC, AMQSSBXC
Wiederverbindbare Clients (5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	kein Beispiel	kein Beispiel	kein Beispiel
Verwenden der Nachrichtenkonsumenten zum asynchronen Konsumieren von Nachrichten aus mehreren Warteschlangen (5)	AMQSCBFO	kein Beispiel	kein Beispiel	kein Beispiel
Angeben von TLS-Verbindungsinformationen in MQCONNX	AMQSSSLC	kein Beispiel	kein Beispiel	AMQSSSLC
Verbinden mit einem Warteschlangenmanager über MQCONNX	AMQSCNXC	kein Beispiel	kein Beispiel	AMQSCNXC
Eigenschaften einer Nachrichtenkennung mit MQINQMP aus einer Nachrichtenwarteschlange abfragen	AMQISQMA	kein Beispiel	kein Beispiel	AMQISQMC
Eigenschaften einer Nachrichtenkennung mit MQSETMP festlegen und in eine Nachrichtenwarteschlange einreihen	AMQSSQMA	kein Beispiel	kein Beispiel	AMQSSWSMC

Anmerkungen:

1. Die Quelle für die Beispielprogramme in Programmiersprache C finden Sie in der Datei QMQMSAMP/QCSRC. Include-Dateien sind als Mitglieder in der Datei QMQM/H vorhanden.
2. Die Quelle für die Beispielprogramme in COBOL finden Sie in den Dateien QMQMSAMP/QCBLLESRC. Die Mitglieder heißen AMQ0 xxx 4, wobei xxx die Beispielfunktion angibt.
3. Die Quelle für die RPG-Beispielprogramme finden Sie in der Datei QMQMSAMP/QRPGLESRC. Die Mitglieder heißen AMQ3 xxx 4, wobei xxx die Beispielfunktion angibt. Kopiemitglieder (Member) sind in QMQM/QRPGLESRC vorhanden. Jeder Mitgliedsname hat das Suffix G.
4. Die ausführbare Version der IBM MQ MQI client-Beispielprogramme verwendet dieselbe Quelle wie die Beispielprogramme für eine Serverumgebung. Die Quelle für die Beispielprogramme in der Clientumgebung ist mit der Quelle für die Serverumgebung identisch. Beispielprogramme für den IBM MQ MQI client sind mit der Clientbibliothek LIBMQIC verknüpft, während Beispielprogramme für den IBM MQ-Server mit der Serverbibliothek LIBMQM verknüpft sind.
5. Wenn die Clientversion für die Beispielanwendung zum wiederverbindbaren Client und die asynchrone Konsumentenanwendung ausgeführt werden muss, muss sie kompiliert und mit der Threadbibliothek LIBMQIC_R verknüpft werden. Daher muss sie in einer Threadumgebung ausgeführt werden. Setzen Sie die Umgebungsvariable QIBM_MULTI_THREAD auf 'Y' und führen Sie die Anwendung über 'qsh' aus.

Weitere Informationen finden Sie unter IBM MQ mit Java und JMS einrichten .

Weitere Informationen finden Sie unter „Beispielprogramme unter IBM i vorbereiten und ausführen” auf Seite 1125.

Zusätzlich zu diesen Beispielprogrammen enthält das IBM MQ for IBM i-Beispielprogramm die Beispieldatendatei AMQSDATA, die Sie als Eingabe für die Beispielprogramme verwenden können, sowie CL-Programme zur Veranschaulichung von Verwaltungsaufgaben. Eine Beschreibung der CL-Beispielprogramme enthält der Abschnitt IBM i verwalten. Mit dem CL-Beispielprogramm `amqsamp4` könnten Sie Warteschlangen für die in diesem Abschnitt beschriebenen Beispielprogramme erstellen.

Multi *Beispielprogramme vorbereiten und ausführen*

Nach Abschluss einiger Vorbereitungen können Sie die Beispielprogramme ausführen.

Informationen zu diesem Vorgang

Bevor Sie die Beispielprogramme ausführen, müssen Sie zunächst einen Warteschlangenmanager und auch die benötigten Warteschlangen erstellen. Gegebenenfalls sind einige zusätzliche Vorbereitungen erforderlich, z. B. wenn Sie COBOL-Beispiele ausführen wollen. Nach Abschluss der notwendigen Vorbereitungen können Sie die Beispielprogramme ausführen.

Vorgehensweise

Informationen zur Vorbereitung und Ausführung der Beispielprogramme finden Sie in folgenden Abschnitten:

- „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123
- „Beispielprogramme unter IBM i vorbereiten und ausführen” auf Seite 1125
- „Beispielprogramme unter AIX and Linux vorbereiten und ausführen” auf Seite 1127
- „Beispielprogramme unter Windows vorbereiten und ausführen” auf Seite 1129

Multi *Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms*

Bevor Sie die Beispielanwendungen ausführen können, müssen Sie zunächst einen Warteschlangenmanager erstellen. Anschließend können Sie den Warteschlangenmanager so konfigurieren, dass er eingehende Verbindungsanforderungen von Anwendungen, die im Clientmodus ausgeführt werden, sicher akzeptiert.

Vorbereitende Schritte

Stellen Sie sicher, dass der Warteschlangenmanager vorhanden und gestartet ist. Stellen Sie fest, ob Kanalauthentifizierungsdatensätze bereits aktiviert sind, indem Sie folgenden MQSC-Befehl ausgeben:

```
DISPLAY QMGR CHLAUTH
```

Wichtig: Diese Task setzt voraus, dass Kanalauthentifizierungssätze aktiviert sind. Wenn dieser Warteschlangenmanager auch von anderen Benutzern und Anwendungen verwendet wird, wirkt sich eine Änderung dieser Einstellung auch auf diese Benutzer und Anwendungen aus. Wenn der Warteschlangenmanager keine Kanalauthentifizierungsdatensätze verwendet, kann Schritt 4 durch ein anderes Authentifizierungsverfahren ersetzt werden (z. B. ein Sicherheitsexit), das den MCAUSER auf die *Nicht_privilegierte_Benutzer-ID* setzt, die Sie in Schritt „1” auf Seite 1124 anfordern.

Sie müssen wissen, welchen Kanalnamen Ihre Anwendung verwenden möchte, damit der Anwendung die Berechtigung zur Verwendung des Kanals erteilt werden kann. Sie müssen auch wissen, welche zu verwendenden Objekte, zum Beispiel Warteschlangen oder Themen, Ihre Anwendung erwartet, damit sie sie verwenden darf.

Informationen zu diesem Vorgang

Durch diese Task wird eine nicht privilegierte Benutzer-ID erstellt, die von einer Clientanwendung zur Verbindung mit dem Warteschlangenmanager verwendet werden kann. Die Clientanwendung erhält mittels dieser Benutzer-ID allein die Berechtigung, den von ihr benötigten Kanal und die von ihr benötigte Warteschlange zu verwenden.

Vorgehensweise

1. Beschaffen Sie sich eine Benutzer-ID für das System, auf dem Ihr Warteschlangenmanager ausgeführt wird. Bei dieser Task darf diese Benutzer-ID keinen privilegierten Benutzer mit Verwaltungsaufgaben angeben. Mittels dieser Benutzer-ID erhalten Sie die Berechtigung zur Ausführung der Clientverbindung auf dem Warteschlangenmanager.
2. Starten Sie mit den folgenden Befehlen ein Listenerprogramm. Dabei gilt Folgendes:

   Warteschlangenmanagername ist der Name Ihres Warteschlangenmanagers.
   nnnn ist die von Ihnen ausgewählte Portnummer.

   a) ALW
   Für AIX, Linux, and Windows-Systeme:

   ```
   runmqlsr -t tcp -m qmgr-name -p nnnn
   ```

   b) IBM i
   Für IBM i:

   ```
   STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
   ```

3. Falls Ihre Anwendung den Kanal SYSTEM.DEF.SVRCONN verwendet, können Sie davon ausgehen, dass dieser bereits definiert ist. Verwendet Ihre Anwendung einen anderen Kanal, müssen Sie diesen mit folgendem MQSC-Befehl erstellen:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   Dabei ist *Kanalname* der Name Ihres Kanals.
4. Erstellen Sie eine Kanalauthentifizierungsregel, die eine Verwendung des Kanals nur durch die IP-Adresse Ihres Clientsystems zulässt. Geben Sie dazu folgenden MQSC-Befehl aus:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   Dabei gilt Folgendes:

   Kanalname ist der Name Ihres Kanals.
   IP-Adresse_Clientsystem ist die IP-Adresse Ihres Clientsystems. Wird Ihre Beispielclientanwendung auf demselben System wie der Warteschlangenmanager ausgeführt, verwenden Sie die IP-Adresse '127.0.0.1', falls Ihre Anwendung die Verbindung mittels 'localhost' herstellt. Sollen Verbindungen von mehreren Clientsystemen hergestellt werden, können Sie statt einer einzelnen IP-Adresse auch ein Adressmuster bzw. einen Adressbereich eingeben. Ausführliche Informationen hierzu finden Sie unter Generische IP-Adresse.
   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1” auf Seite 1124
5. Falls Ihre Anwendung die Warteschlange SYSTEM.DEFAULT.LOCAL.QUEUE verwendet, können Sie davon ausgehen, dass diese bereits definiert ist. Verwendet Ihre Anwendung eine andere Warteschlange, müssen Sie diese mit folgendem MQSC-Befehl erstellen:

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

Dabei ist *Warteschlangenname* der Name Ihrer Warteschlange.

6. Erteilen Sie den erforderlichen Zugriff für die Verbindung mit dem Warteschlangenmanager und dessen Abfrage, indem Sie den folgenden MQSC-Befehl absetzen:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
AUTHADD(CONNECT, INQ)
```

   Dabei ist *Nicht_privilegierte_Benutzer-ID* die Benutzer-ID aus Schritt „1” auf Seite 1124 .

7. Handelt es sich bei Ihrer Anwendung um eine Punkt-zu-Punkt-Anwendung (d. h., sie nutzt Warteschlangen), müssen Sie der zu verwendenden Benutzer-ID die Berechtigung zur Abfrage Ihrer Warteschlange sowie zum Einreihen und Abrufen von Nachrichten über diese Warteschlange einräumen. Geben Sie hierzu die folgenden MQSC-Befehle aus:

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

   Dabei gilt Folgendes:

   Warteschlangenname ist der Name Ihrer Warteschlange.
   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1” auf Seite 1124

8. Handelt es sich bei Ihrer Anwendung um eine Publish/Subscribe-Anwendung, d. h., sie nutzt Themen, so müssen Sie der zu verwendenden Benutzer-ID die Berechtigung zur Veröffentlichung und Subskription mittels Ihres Themas einräumen. Geben Sie hierzu die folgenden MQSC-Befehle aus:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

   Dabei gilt Folgendes:

   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1” auf Seite 1124
   Dadurch erhält *Nicht_privilegierte_Benutzer-ID* Zugriff auf jedes Thema in der Themenstruktur. Alternativ können Sie mit **DEFINE TOPIC** auch ein bestimmtes Themenobjekt definieren, um den Zugriff nur auf die entsprechenden Teile der Themenstruktur einzuräumen. Ausführliche Informationen hierzu finden Sie unter Benutzerzugriff auf Themen steuern.

Nächste Schritte

Ihre Clientanwendung kann nun eine Verbindung zum Warteschlangenmanager herstellen und Nachrichten über die Warteschlange einreihen oder abrufen.

Zugehörige Konzepte

ALW Zugriff auf ein IBM MQ-Objekt unter AIX, Linux, and Windows erteilen

Zugehörige Verweise

SET CHLAUTH
CHANNEL DEFINE CHANNEL
QLOCAL DEFINIER
SET AUTHREC

IBM i IBM MQ-Berechtigungen unter IBM i

IBM i *Beispielprogramme unter IBM i vorbereiten und ausführen*

Bevor Sie die Beispielprogramme unter IBM i ausführen, müssen Sie zunächst einen Warteschlangenmanager und auch die benötigten Warteschlangen erstellen. Wenn Sie COBOL-Beispiele ausführen möchten, sind gegebenenfalls zusätzliche Vorbereitungen erforderlich.

Informationen zu diesem Vorgang

Die Quellen für die IBM MQ for IBM i-Beispielprogramme werden in der Bibliothek QMQMSAMP als die Teildateien QCSRC, QCLSRC, QCBLLESRC und QRPGLESRC bereitgestellt.

Wenn Sie die Beispielprogramme ausführen, können Sie Ihre eigenen Warteschlangen verwenden oder mit dem Beispielprogramm AMQSAMP4 einige Beispielwarteschlangen erstellen. Die Quelle für das Programm AMQSAMP4 ist in der Datei QCLSRC in der Bibliothek QMQMSAMP enthalten. Sie können es mit dem Befehl CRTCLPGM kompilieren.

Wenn Sie die Beispielprogramme ausführen möchten, verwenden Sie entweder die ausführbaren C-Versionen in der Bibliothek QMQM oder kompilieren Sie sie ähnlich wie für andere IBM MQ-Anwendungen.

V 9.4.0 V 9.4.0 Die folgenden Beispielprogramme verfügen über Authentifizierungsfunktionen:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`
- `amqsmhac.c`
- `amqsphac.c`
- `amqspuba.c`
- `amqsput0.c`
- `amqsсslc.c`
- `amqssuba.c`

Für die ausführbaren Versionen dieser Beispiele ist die Authentifizierung aktiviert. Das Kompilieren der Quellenversionen mit aktivierter Authentifizierung erfordert jedoch die Definition des Kompilierungsflags **SAMPLE_AUTH_ENABLED** und die Kompilierung der `amqsauth.c` -Quellendatei mit dem gewünschten Beispiel. For example:

- Erstellen Sie das Programm `amqsсslc` ohne aktivierte Authentifizierung:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC) MODULE(MYLIB/AMQSSSLC) BNDSRVPGM(QMQM/LIBMQIC)
```

- `amqsсslc` mit aktivierter Authentifizierung erstellen:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) DEFINE('SAMPLE_AUTH_ENABLED') SRCFILE(QMQMSAMP/QCSRC)
CRTCMOD MODULE(MYLIB/AMQSAUTH) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC_AUTH) MODULE(MYLIB/AMQSSSLC MYLIB/AMQSAUTH) BNDSRVPGM(QMQM/LIBMQIC)
```

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager und konfigurieren Sie die Standarddefinitionen.

   Dies ist Voraussetzung für die Ausführung der Beispielprogramme. Weitere Informationen zum Erstellen eines Warteschlangenmanagers finden Sie unter IBM MQ verwalten. Informationen zum Konfigurieren eines Warteschlangenmanagers für die sichere Annahme eingehender Verbindungsanforderungen von Anwendungen, die im Clientmodus aktiv sind, finden Sie im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.
2. Rufen Sie beispielsweise mit folgendem Befehl eines der Beispielprogramme auf, das Daten aus der Teildatei PUT von Datei AMQSDATA der Bibliothek QMQMSAMP verwendet:

   ```
   CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
   ```

Anmerkung: Damit ein kompiliertes Modul das IFS-Dateisystem verwendet, geben Sie die Option SYSIFCOPT(*IFSIO) in CRTCMOD an. Dann muss der Dateiname, der als Parameter übermittelt wird, in folgendem Format angegeben werden:

```
home/me/myfile
```

3. Wenn Sie die COBOL-Versionen der Beispiele Inquire, Set und Echo verwenden möchten, ändern Sie die Prozessdefinitionen, bevor Sie diese Beispiele ausführen.

   Bei den Inquiry-, Set- und Echo-Beispielen lösen die Beispielprogrammdefinitionen die C-Versionen dieser Programme aus. Wenn Sie die COBOL-Versionen ausführen möchten, müssen Sie die Prozessdefinitionen ändern:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   Unter IBM i können Sie die Befehl **CHGMQMPRC** verwenden (ausführliche Informationen hierzu finden Sie unter Change MQ Process (CHGMQMPRC)) oder den Befehl **AMQSAMP4** bearbeiten und mit der alternativen Definition ausführen.
4. Führen Sie die Beispielprogramme aus.

   Weitere Informationen zu den Parametern, die von jedem der Beispiele erwartet werden, finden Sie in den Beschreibungen der einzelnen Beispiele.

   Anmerkung: Wenn Sie die COBOL-Beispielprogramme ausführen und die Warteschlangennamen als Parameter übergeben, müssen Sie 48 Zeichen angeben, die bei Bedarf mit Leerzeichen aufzufüllen sind. Sind nicht genau 48 Zeichen angegeben, schlägt das Programm mit Ursachencode 2085 fehl.

Zugehörige Verweise

„Beispiele für IBM i” auf Seite 1120
In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit IBM i veranschaulicht werden.

Linux AIX *Beispielprogramme unter AIX and Linux vorbereiten und ausführen*

Bevor Sie die Beispielprogramme unter AIX and Linux ausführen, müssen Sie zunächst einen Warteschlangenmanager und auch die benötigten Warteschlangen erstellen. Wenn Sie COBOL-Beispiele ausführen möchten, sind gegebenenfalls zusätzliche Vorbereitungen erforderlich.

Informationen zu diesem Vorgang

Die Beispieldateien für IBM MQ auf AIX and Linux-Systemen finden Sie in den unter Tabelle 160 auf Seite 1127 aufgeführten Verzeichnissen, wenn die Standardeinstellungen installiert wurden.

Tabelle 160. Verzeichnisse der Beispielprogramme für IBM MQ auf AIX and Linux-Systemen

Inhalt	**Directory**
Quellendateien	*MQ_INSTALLATION_PATH*/samp
Quellendateien der Steuerroutine der Warteschlange für nicht zustellbare Nachrichten	*MQ_INSTALLATION_PATH*/samp/dlq
Ausführbare Dateien	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Die Beispielprogramme benötigen mehrere Warteschlangen, mit denen sie arbeiten. Sie können entweder Ihre eigenen Warteschlangen verwenden oder die MQSC-Beispieldatei amqscos0.tst ausführen, um Warteschlangen zu erstellen. Verwenden Sie zum Ausführen der Beispielprogramme entweder die bereitgestellten ausführbaren Versionen oder kompilieren Sie die Quellenversionen wie jede andere Anwendung mithilfe eines ANSI-Compilers.

V 9.4.0 V 9.4.0 Die folgenden Beispielprogramme verfügen über Authentifizierungsfunktionen:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`
- `amqsmhac.c`
- `amqsphac.c`
- `amqspuba.c`
- `amqsput0.c`
- `amqsslc.c`
- `amqssuba.c`

Für die ausführbaren Versionen dieser Beispiele ist die Authentifizierung aktiviert. Das Kompilieren der Quellenversionen mit aktivierter Authentifizierung erfordert jedoch die Definition des Kompilierungsflags **SAMPLE_AUTH_ENABLED** und die Kompilierung der `amqsauth.c` -Quellendatei mit dem gewünschten Beispiel. For example:

- `amqsput0.c` ohne aktivierte Authentifizierung kompilieren:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -o /bin/amqsput amq
sput0.c
```

- Kompilieren von `amqsput0.c` mit aktivierter Authentifizierung:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -D SAMPLE_AUTH_ENABLED
-o /bin/amqsputc_auth amqsauth.c amqsput0.c
```

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager und konfigurieren Sie die Standarddefinitionen.

   Dies ist Voraussetzung für die Ausführung der Beispielprogramme. Weitere Informationen zum Erstellen eines Warteschlangenmanagers finden Sie unter IBM MQ verwalten. Informationen zum Konfigurieren eines Warteschlangenmanagers für die sichere Annahme eingehender Verbindungsanforderungen von Anwendungen, die im Clientmodus aktiv sind, finden Sie im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.
2. Wenn Sie nicht Ihre eigenen Warteschlangen verwenden, führen Sie die MQSC-Beispieldatei amqscos0.tst aus, um eine Gruppe von Warteschlangen zu erstellen.

   Geben Sie dazu unter AIX and Linux-Systemen Folgendes ein:

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   Prüfen Sie die Datei `sampobj.out`, um sicherzustellen, dass keine Fehler vorliegen.
3. Wenn Sie die COBOL-Versionen der Beispiele Inquire, Set und Echo verwenden möchten, ändern Sie die Prozessdefinitionen, bevor Sie diese Beispiele ausführen.

   Bei den Inquiry-, Set- und Echo-Beispielen lösen die Beispielprogrammdefinitionen die C-Versionen dieser Programme aus. Wenn Sie die COBOL-Versionen ausführen möchten, müssen Sie die Prozessdefinitionen ändern:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

Unter AIX and Linux müssen Sie dazu die Datei `amqscos0.tst` bearbeiten und die Namen der ausführbaren C-Dateien in die Namen der ausführbaren COBOL-Datei, bevor sie die Beispiele mit dem Befehl **runmqsc** ausführen.

4. Führen Sie die Beispielprogramme aus.

   Geben Sie zum Ausführen eines Beispiels seinen Namen gefolgt von Parametern ein, z. B.:

   ```
   amqsput myqueue qmanagername
   ```

   Dabei steht *myqueue* für den Namen der Warteschlange, in die die Nachrichten eingereiht werden, und *qmanagername* gibt den Warteschlangenmanager an, der Eigner von *myqueue* ist.

   Weitere Informationen zu den Parametern, die von jedem der Beispiele erwartet werden, finden Sie in den Beschreibungen der einzelnen Beispiele.

   Anmerkung: Wenn Sie die COBOL-Beispielprogramme ausführen und die Warteschlangennamen als Parameter übergeben, müssen Sie 48 Zeichen angeben, die bei Bedarf mit Leerzeichen aufzufüllen sind. Sind nicht genau 48 Zeichen angegeben, schlägt das Programm mit Ursachencode 2085 fehl.

Zugehörige Verweise

„Beispiele für Systeme mit AIX and Linux” auf Seite 1114
In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ for AIX or Linux veranschaulicht werden.

Windows *Beispielprogramme unter Windows vorbereiten und ausführen*

Bevor Sie die Beispielprogramme unter Windows ausführen, müssen Sie zunächst einen Warteschlangenmanager und auch die benötigten Warteschlangen erstellen. Wenn Sie COBOL-Beispiele ausführen möchten, sind gegebenenfalls zusätzliche Vorbereitungen erforderlich.

Informationen zu diesem Vorgang

Die Beispieldateien für IBM MQ for Windows befinden sich in den in Tabelle 161 auf Seite 1129 aufgeführten Verzeichnissen, sofern zur Installationszeit die Standardeinstellungen verwendet wurden. Als Installationslaufwerk ist standardmäßig <c:> eingestellt.

Tabelle 161. Verzeichnisse der Beispielprogramme für IBM MQ for Windows

Inhalt	**Directory**
C-Quellcode	*MQ_INSTALLATION_PATH*\Tools\C\Beispiele
Quellcode für das Beispielprogramm Steuerroutine der Warteschlange für nicht zustellbare Nachrichten	*MQ_INSTALLATION_PATH*\Tools\C\Samples\DLQ
COBOL-Quellcode	*MQ_INSTALLATION_PATH*\Tools\Cobol \ Beispiele
Ausführbare C-Dateien[1]	*MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin (32-Bit-Versionen) *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (64-Bit-Versionen)
MQSC-Beispieldateien	*MQ_INSTALLATION_PATH*\Tools\MQSC\Beispiele
Visual Basic-Quellcode	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET-Beispiele	*MQ_INSTALLATION_PATH*\Tools\dotnet \ Beispiele

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Anmerkung: Für einige ausführbare C-Beispieldateien stehen 64-Bit-Versionen zur Verfügung.

Die Beispielprogramme benötigen mehrere Warteschlangen, mit denen sie arbeiten. Sie können entweder Ihre eigenen Warteschlangen verwenden oder die MQSC-Beispieldatei amqscos0.tst ausführen, um

eine Gruppe von Warteschlangen zu erstellen. Verwenden Sie zum Ausführen der Beispielprogramme entweder die bereitgestellten ausführbaren Versionen oder kompilieren Sie die Quellenversionen wie jede andere IBM MQ for Windows-Anwendung.

V 9.4.0 V 9.4.0 Die folgenden Beispielprogramme verfügen über Authentifizierungsfunktionen:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`
- `amqsmhac.c`
- `amqsphac.c`
- `amqspuba.c`
- `amqsput0.c`
- `amqsсslc.c`
- `amqssuba.c`

Für die ausführbaren Versionen dieser Beispiele ist die Authentifizierung aktiviert. Das Kompilieren der Quellenversionen mit aktivierter Authentifizierung erfordert jedoch die Definition des Kompilierungsflags **SAMPLE_AUTH_ENABLED** und die Kompilierung der `amqsauth.c` -Quellendatei mit dem gewünschten Beispiel. For example:

- `amqsput0.c` ohne aktivierte Authentifizierung kompilieren:

```
CL amqsput0.c /link mqic.lib /OUT:Bin\amqsputc.exe
```

- Kompilieren von `amqsput0.c` mit aktivierter Authentifizierung:

```
CL /D SAMPLE_AUTH_ENABLED amqsauth.c amqsput0.c /link mqic.lib /OUT:Bin\amqsputc_auth.exe
```

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager und konfigurieren Sie die Standarddefinitionen.

   Dies ist Voraussetzung für die Ausführung der Beispielprogramme. Weitere Informationen zum Erstellen eines Warteschlangenmanagers finden Sie unter IBM MQ verwalten. Informationen zum Konfigurieren eines Warteschlangenmanagers für die sichere Annahme eingehender Verbindungsanforderungen von Anwendungen, die im Clientmodus aktiv sind, finden Sie im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.
2. Wenn Sie nicht Ihre eigenen Warteschlangen verwenden, führen Sie die MQSC-Beispieldatei amqscos0.tst aus, um eine Gruppe von Warteschlangen zu erstellen.

   Geben Sie dazu unter Windows-Systemen Folgendes ein:

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   Prüfen Sie die Datei `sampobj.out`, um sicherzustellen, dass keine Fehler vorliegen. Sie finden diese Datei in Ihrem aktuellen Verzeichnis.
3. Wenn Sie die COBOL-Versionen der Beispiele Inquire, Set und Echo verwenden möchten, ändern Sie die Prozessdefinitionen, bevor Sie diese Beispiele ausführen.

   Bei den Inquiry-, Set- und Echo-Beispielen lösen die Beispielprogrammdefinitionen die C-Versionen dieser Programme aus. Wenn Sie die COBOL-Versionen ausführen möchten, müssen Sie die Prozessdefinitionen ändern:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

Unter Windows müssen Sie dazu die Datei `amqscos0.tst` bearbeiten und die Namen der ausführbaren C-Dateien in die Namen der ausführbaren COBOL-Datei, bevor sie die Beispiele mit dem Befehl **runmqsc** ausführen.

4. Führen Sie die Beispielprogramme aus.

   Geben Sie zum Ausführen eines Beispiels seinen Namen gefolgt von Parametern ein, z. B.:

   ```
   amqsput myqueue qmanagername
   ```

   Dabei steht *myqueue* für den Namen der Warteschlange, in die die Nachrichten eingereiht werden, und *qmanagername* gibt den Warteschlangenmanager an, der Eigner von *myqueue* ist.

   Weitere Informationen zu den Parametern, die von jedem der Beispiele erwartet werden, finden Sie in den Beschreibungen der einzelnen Beispiele.

   Anmerkung: Wenn Sie die COBOL-Beispielprogramme ausführen und die Warteschlangennamen als Parameter übergeben, müssen Sie 48 Zeichen angeben, die bei Bedarf mit Leerzeichen aufzufüllen sind. Sind nicht genau 48 Zeichen angegeben, schlägt das Programm mit Ursachencode 2085 fehl.

Zugehörige Verweise

„Beispiele für IBM MQ for Windows" auf Seite 1117
In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ for Windows veranschaulicht werden.

„Visual Basic-Beispiele für IBM MQ for Windows" auf Seite 1120
In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit Windows veranschaulicht werden.

API Exit-Beispielprogramm

Der API-Beispielexit generiert einen MQI-Trace in einer benutzerdefinierten Datei mit einem Präfix, das in der Umgebungsvariablen **MQAPI_TRACE_LOGFILE** definiert ist.

Weitere Informationen zu API-Exits finden Sie im Abschnitt „API-Exits auf Multiplatforms schreiben und kompilieren" auf Seite 1004.

Quelle
amqsaxe0.c

Binär
amqsaxe

Konfiguration für den Beispielexit

1. Fügen Sie die folgenden Informationen zur ZeilengruppeApiExitLocal der Datei `qm.ini` hinzu.

   Andere Plattformen als Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem IBM MQ installiert ist.

   Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem IBM MQ installiert ist.

2. Legen Sie die Umgebungsvariable **MQAPI_TRACE_LOGFILE** fest

```
MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
```

3. Führen Sie Ihre Anwendung aus.

   Ausgabedateien werden im Verzeichnis `/tmp` mit Namen wie `MqiTrace.pid.tid.log`erstellt.

Das Asynchronous Consumption-Beispielprogramm

Das Beispielprogramm 'amqscbf' veranschaulicht die Verwendung von MQCB und MQCTL zur asynchronen Verarbeitung (Konsumierung) von Nachrichten aus mehreren Warteschlangen.

'amqscbf' wird auf AIX, Linux, and Windows-Plattformen als C-Quellcode, Binär-Client und ausführbare Serverdatei bereitgestellt.

Das Programm wird über die Befehlszeile gestartet und verwendet die folgenden optionalen Parameter:

```
Usage: [Options] Queue Name {queue_name}
  where Options are:
    -m Queue Manager Name
    -o Open options
    -r Reconnect Type
       d Reconnect Disabled
       r Reconnect
       m Reconnect Queue Manager
```

Geben Sie mehrere Warteschlangennamen an, um Nachrichten aus mehreren Warteschlangen zu lesen (das Beispielprogramm unterstützt maximal zehn Warteschlangen).

Anmerkung: Reconnect type ist nur für Clientprogramme gültig.

Beispiel

Das Beispielprogramm zeigt die Ausführung von 'amqscbf' als Serverprogramm. Es liest eine Nachricht aus QL1 und wird dann gestoppt.

Reihen Sie mit IBM MQ Explorer eine Testnachricht in QL1 ein. Stoppen Sie das Programm durch Drücken der Eingabetaste.

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

Was 'amqscbf' veranschaulicht

Das Beispielprogramm veranschaulicht, wie Nachrichten aus mehreren Warteschlangen in der Reihenfolge ihres Eintreffens gelesen werden. Dazu wäre sehr viel zusätzlicher Code unter Verwendung synchroner MQGET-Befehle erforderlich. Bei der asynchronen Verarbeitung ist keine Abfrage (Polling) erforderlich und IBM MQ verwaltet die Threads und Speicher. Ein konkretes Praxisbeispiel müsste auch Fehler behandeln. In dem Beispielprogramm werden Fehler in die Konsole ausgelagert.

Der Beispielcode besteht aus folgenden Schritten:

1. Definieren der Callback-Funktion der einzelnen Nachrichtenverarbeitung,

```
void MessageConsumer(MQHCONN      hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
```

```
                          MQCBC   * pContext)
{ ... }
```

2. Verbindungsherstellung zum Warteschlangenmanager,

```
MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
```

3. Öffnen der Eingabewarteschlangen, Zuordnen jeder Eingabewarteschlange zu Callback-Funktion `MessageConsumer`,

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   `cbd.CallbackFunction` muss nicht für jede Warteschlange festgelegt werden; es ist ein reines Eingabefeld. Sie könnten jedoch jeder Warteschlange eine andere Callback-Funktion zuordnen.

4. Starten der Nachrichtenverarbeitung,

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Warten, bis der Benutzer die Eingabetaste gedrückt hat, danach Stoppen der Nachrichtenverarbeitung,

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Abschließend Trennen der Verbindung zum Warteschlangenmanager,

```
MQDISC(&Hcon,&CompCode,&Reason);
```

Das Asynchronous Put-Beispielprogramm

Dieser Abschnitt enthält Informationen zum Ausführen des Beispielprogramms 'amqsapt' und zur Gestaltung des Asynchronous Put-Beispielprogramms.

Das Beispielprogramm für asynchrone Put-Operationen (Asynchronous Put) reiht Nachrichten mit dem asynchronen MQPUT-Aufruf in eine Warteschlange ein und ruft dann die Statusinformationen mit dem MQSTAT-Aufruf ab. Den Namen dieses Programms auf anderen Plattformen finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Beispielprogramm 'amqsapt' ausführen

Das Programm verwendet bis zu sechs Parameter:

1. Name der Zielwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (optional)
3. Optionen zum Öffnen (optional)
4. Optionen zum Schließen (optional)
5. Name des Zielwarteschlangenmanagers (optional)
6. Name der dynamischen Warteschlange (optional)

Wenn kein Warteschlangenmanager angegeben wird, stellt 'amqsapt' eine Verbindung zum Standardwarteschlangenmanager her.

Design des Beispielprogramms für asynchrone Put-Operationen

Das Programm verwendet den MQOPEN-Aufruf mit den bereitgestellten Ausgabeoptionen oder mit den Optionen MQOO_OUTPUT und MQOO_FAIL_IF_QUIESCING, um die Zielwarteschlange zum Einreihen von Nachrichten zu öffnen.

Wenn es die Warteschlange nicht öffnen kann, zeigt das Programm eine Fehlernachricht an, die den vom MQOPEN-Aufruf zurückgegebenen Ursachencode enthält. Um das Programm einfach zu halten, verwendet es für diesen und für nachfolgende MQI-Aufrufe die Standardwerte für die meisten Optionen.

Das Programm liest dann für jede Eingabezeile den Text in einen Puffer und verwendet den MQPUT-Aufruf mit MQPMO_ASYNC_RESPONSE, um eine Datagrammnachricht mit dem Text dieser Zeile zu erstellen und sie asynchron in die Zielwarteschlange einzureihen. Das Programm wird fortgesetzt, bis es entweder das Ende der Eingabe erreicht oder bis der MQPUT-Aufruf fehlschlägt. Wenn das Programm das Ende der Warteschlange erreicht, schließt es die Warteschlange unter Verwendung des MQCLOSE-Aufrufs.

Danach gibt das Programm den Aufruf MQSTAT zur Rückgabe einer MQSTS-Struktur aus und zeigt Nachrichten an, die die Anzahl der erfolgreich eingereihten Nachrichten, die Anzahl der mit einer Warnung eingereihten Nachrichten sowie die Anzahl der Nachrichten enthalten, deren Einreihung fehlgeschlagen ist.

Die Browse-Beispielprogramme

Die Browse-Beispielprogramme durchsuchen Nachrichten mit dem MQGET-Aufruf in einer Warteschlange.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Gestaltung des Browse-Beispielprogramms

Das Programm öffnet die Zielwarteschlange mit dem MQOPEN-Aufruf und der Option MQOO_BROWSE. Wenn es die Warteschlange nicht öffnen kann, zeigt das Programm eine Fehlernachricht an, die den vom MQOPEN-Aufruf zurückgegebenen Ursachencode enthält.

Das Programm verwendet für jede Nachricht, die es in der Warteschlange kopiert, den MQGET-Aufruf und zeigt dann die Daten an, die in der Nachricht enthalten sind. Der MQGET-Aufruf verwendet folgende Optionen:

MQGMO_BROWSE_NEXT
Nach Ausgabe des MQOPEN-Aufrufs wird der Anzeigecursor logisch vor der ersten Nachricht in der Warteschlange positioniert, sodass diese Option bei der ersten Ausführung des Aufrufs die Rückgabe der ***ersten*** Nachricht bewirkt.

MQGMO_NO_WAIT
Das Programm wartet nicht, wenn keine Nachrichten in der Warteschlange vorhanden sind.

MQGMO_ACCEPT_TRUNCATED_MSG
Der MQGET-Aufruf gibt einen Puffer mit einer festen Größe an. Wenn eine Nachricht länger als dieser Puffer ist, zeigt das Programm die abgeschnittene Nachricht zusammen mit der Warnung an, dass die Nachricht abgeschnitten wurde.

Das Programm veranschaulicht, wie die Felder *MsgId* und *CorrelId* der MQMD-Struktur nach jedem MQGET-Aufruf gelöscht werden müssen, da der Aufruf diese Felder auf die betreffenden Werte der abgerufenen Nachrichten setzt. Durch das Löschen dieser Felder rufen aufeinanderfolgende MQGET-Aufrufe die Nachrichten in der Reihenfolge ab, in der diese in der Warteschlange gehalten werden.

Das Programm wird bis zum Ende der Warteschlange ausgeführt; der MQGET-Aufruf gibt den Ursachencode MQRC_NO_MSG_AVAILABLE zurück und das Programm zeigt einen Warnhinweis an. Schlägt der MQGET-Aufruf fehl, zeigt das Programm eine Fehlernachricht mit dem Ursachencode an.

Das Programm schließt dann die Warteschlange mithilfe des MQCLOSE-Aufrufs.

Beispielprogramme für die Durchsuchen-Funktion unter AIX, Linux, and Windows
Dieser Abschnitt enthält Informationen zu den Beispielprogrammen für die die Durchsuchen-Funktion unter AIX, Linux, and Windows.

Die C-Version des Programms verwendet zwei Parameter:

1. Name der Quellenwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (optional)

Wird kein Warteschlangenmanager angegeben, wird der Standardwarteschlangenmanager verwendet. Geben Sie zum Beispiel einen der folgenden Befehle ein:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

Dabei steht `myqueue` für den Namen der Warteschlange, in der die Nachrichten angezeigt werden, und `qmanagername` gibt den Warteschlangenmanager an, der Eigner von `myqueue` ist.

Wenn Sie beim Ausführen des C-Beispielprogramms den Namen des Warteschlangenmanagers (`qmanagername`) übergehen, wird angenommen, dass der Standardwarteschlange der Eigner der Warteschlange ist.

Die COBOL-Version hat keine Parameter. Sie stellt eine Verbindung zum Standardwarteschlangenmanager her und gibt beim Ausführen folgende Eingabeaufforderungen aus:

```
Please enter the name of the target queue
```

Nur die ersten 50 Zeichen jeder Nachricht werden angezeigt, gefolgt von `- - - truncated`, wenn dies der Fall ist.

Die Browse-Beispielprogramme unter IBM i
Jedes Programm ruft Kopien aller Nachrichten in der Warteschlange ab, die Sie beim Programmaufruf angeben; die Nachrichten verbleiben in der Warteschlange.

Sie können die bereitgestellte Warteschlange SYSTEM.SAMPLE.LOCAL verwenden; führen Sie zunächst das Put-Beispielprogramm aus, um einige Nachrichten in die Warteschlange einzureihen. Sie können die Warteschlange SYSTEM.SAMPLE.ALIAS verwenden. Ihr Name ist ein Aliasname für die gleiche lokale Warteschlange. Das Programm wird fortgesetzt, bis es entweder das Ende der Warteschlange erreicht oder bis ein MQI-Aufruf fehlschlägt.

In den C-Beispielprogrammen können Sie den Namen des Warteschlangenmanagers (i.d.R. als zweiter Parameter) ähnlich wie in den Beispielprogrammen für Windows-Systeme angeben. For example:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

Wird kein Warteschlangenmanager angegeben, wird der Standardwarteschlangenmanager verwendet. Dies ist auch bei den RPG-Beispielprogrammen der Fall. Dort müssen Sie jedoch den Namen eines Warteschlangenmanagers angeben, statt die Standardeinstellung zu verwenden.

ALW *Das Beispielprogramm 'Browser'*

Das Browser-Beispielprogramm liest und schreibt sowohl die Felder für den Nachrichtendeskriptor als auch für den Nachrichtenkontext aller Nachrichten in einer Warteschlange.

Das Beispielprogramm ist als Dienstprogramm geschrieben und dient nicht allein zur Veranschaulichung eines Verfahrens. Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Das Programm verwendet folgende positionsgebundene Parameter:

1. Name der Quellenwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (erforderlich)
3. Optionaler Parameter für Eigenschaften (optional)

Verwenden Sie die folgenden Umgebungsvariablen, um Berechtigungsnachweise für die Authentifizierung beim Warteschlangenmanager anzugeben:

MQSAMP_USER_ID
Wird auf die für die Verbindungsauthentifizierung zu verwendende Benutzer-ID gesetzt, wenn Sie eine Benutzer-ID und ein Kennwort für die Authentifizierung beim Warteschlangenmanager verwenden wollen. Das Programm fordert zur Eingabe des Kennworts für die Benutzer-ID auf.

Linux V 9.4.0 AIX **MQSAMP_TOKEN**
Legen Sie einen Wert fest, der nicht leer ist, wenn Sie ein Authentifizierungstoken für die Authentifizierung beim Warteschlangenmanager bereitstellen wollen. Das Programm fordert zur Eingabe des Authentifizierungstoken auf. Authentifizierungstoken können nur vom **amqsbcgc** -Beispiel verwendet werden, das Clientbindungen verwendet.

Geben Sie zum Ausführen dieser Programme einen der folgenden Befehle ein:

- `amqsbcg` *`myqueue qmanagername`*
- `amqsbcgc` *`myqueue qmanagername`*

Dabei ist *myqueue* der Name der Warteschlange, in der die Nachrichten durchsucht werden, und *qmanagername* ist der Warteschlangenmanager, der Eigner von *myqueue*ist.

Es liest jede Nachricht aus der Warteschlange und schreibt Folgendes in 'stdout':

- Formatierte Nachrichtendeskriptorfelder
- Nachrichtendaten (gespeichert im hexadezimalen Format und, wenn möglich, im Zeichenformat)

Tabelle 162. Zulässige Werte für den Eigenschaftenparameter

Wert	Verhalten
0	Standardverhalten. Die Eigenschaften, die der Anwendung bereitgestellt werden, hängen vom Warteschlangenattribut **PropertyControl** ab, von dem die Nachricht abgerufen wird.

1	Ein Nachrichtenhandle wird erstellt und mit MQGET verwendet. Eigenschaften der Nachricht, ausgenommen jene, die im Nachrichtendeskriptor (oder in der Erweiterung) enthalten sind, werden dem Nachrichtendeskriptor in einer ähnlichen Weise angezeigt. For example:
	`****Message properties****`
	`  property name: property value`
	Falls keine Eigenschaften verfügbar sind:
	`****Message properties****`
	`  None`
	Numerische Werte werden mittels `printf` angezeigt, Zeichenfolgewerte stehen in einfachen Anführungszeichen und Bytefolgen stehen zwischen einem X und einfachen Anführungszeichen, wie für den Nachrichtendeskriptor.

2	MQGMO_NO_PROPERTIES wird angegeben, sodass nur Nachrichtendeskriptoreigenschaften zurückgegeben werden.
3	MQGMO_PROPERTIES_FORCE_MQRFH2 ist angegeben, sodass alle Eigenschaften in den Nachrichtendaten zurückgegeben werden.
4	MQGMO_PROPERTIES_FORCE_MQRFH2 ist angegeben, sodass alle Eigenschaften zurückgegeben werden können, wenn eine IBM MQ-Eigenschaft enthalten ist. Andernfalls werden die Eigenschaften gelöscht.

Das Programm druckt nur die ersten 65535 Zeichen einer Nachricht und schlägt beim Lesen einer längeren Nachricht mit der Ursache `truncated msg` (abgeschnittene Nachricht) fehl.

Ein Beispiel der Ausgabe dieses Dienstprogramms finden Sie unter Warteschlangen durchsuchen.

Das CICS Transaction-Beispielprogramm

Bereitstellung des CICS Transaction-Beispielprogramms 'amqscic0.ccs' für den Quellcode und des Beispielprogramms 'amqscic0' für die ausführbare Version. Sie können Transaktionen mit den Standardfunktionen von CICS erstellen.

Ausführliche Informationen zu den Befehlen für Ihre jeweilige Plattform finden Sie unter „Prozedurale Anwendung erstellen” auf Seite 1052.

Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE auf dem Standardwarteschlangenmanager und stellt sie in die lokale Warteschlange, deren Name im Übertragungsheader der Nachricht enthalten ist. Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICSgesendet.DLQ

Anmerkung: Verwenden Sie das MQSC-Beispielscript 'amqscic0.tst', um diese Warteschlangen und Beispieleingabewarteschlangen zu erstellen.

Das Connect-Beispielprogramm

Das Connect-Beispielprogramm ermöglicht Ihnen, den MQCONNX-Aufruf und seine Optionen von einem Client aus zu untersuchen. Das Beispielprogramm stellt mit dem MQCONNX-Aufruf eine Verbindung zum Warteschlangenmanager her, fragt den Namen des Warteschlangenmanagers mit dem MQINQ-Aufruf ab und zeigt ihn an. Zudem erhalten Sie in diesem Abschnitt Informationen zum Ausführen des Beispielprogramms 'amqscnxc'.

Anmerkung: Das Connect-Beispielprogramm ist ein Clientbeispiel. Sie können es kompilieren und auf einem Server ausführen, doch seine Funktionen sind nur auf einem Client sinnvoll. Es werden auch nur auf einem Client ausführbare Dateien bereitgestellt.

Beispielprogramm 'amqscnxc' ausführen

Die Befehlszeilensyntax des Connect-Beispielprogramms lautet:

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

Die Parameter sind optional und ihre Reihenfolge ist unwichtig - mit Ausnahme von 'QMgrName'. Dieser Parameter muss ganz zum Schluss angegeben werden. Die Parameter lauten wie folgt:

ConnName
Der TCP/IP-Verbindungsname des Server-Warteschlangenmanagers

Wenn Sie den TCP/IP-Verbindungsnamen nicht angeben, wird MQCONNX mit *ClientConnPtr* auf NULL ausgegeben.

SvrconnChannelName
Name des Serververbindungskanals

Wenn Sie den TCP/IP-Verbindungsnamen, aber keinen Serververbindungskanal angeben (umgekehrt ist dies nicht zulässig), verwendet das Beispielprogramm den Namen SYSTEM.DEF.SVRCONN.

User
Der für die Verbindungsauthentifizierung verwendete Benutzername

Nach dessen Angabe fragt das Programm nach dem zugehörigen Kennwort für diese Benutzer-ID.

QMgrName
Name des Zielwarteschlangenmanagers

Wenn Sie den Zielwarteschlangenmanager nicht angeben, stellt das Beispielprogramm eine Verbindung mit einem beliebigen Warteschlangenmanager her, der am angegebenen TCP/IP-Verbindungsnamen empfangsbereit ist.

Anmerkung: Wenn Sie als einzigen Parameter ein Fragezeichen eingeben oder wenn Sie falsche Parameter eingeben, erhalten Sie eine Nachricht mit einer Erklärung, wie das Programm zu verwenden ist.

Wenn Sie das Beispielprogramm ohne Befehlszeilenoptionen ausführen, werden anhand des Inhalts der Umgebungsvariablen MQSERVER die Verbindungsinformationen bestimmt. (In diesem Beispiel ist MQSERVER auf SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com gesetzt.) Sie sehen eine ähnliche Ausgabe wie die folgende:

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

Wenn Sie das Beispielprogramm ausführen und den Namen einer TCP/IP-Verbindung sowie eines Serververbindungskanals angeben, aber den Namen des Zielwarteschlangenmanagers nicht nennen, zum Beispiel:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

, wird der Name des Standardwarteschlangenmanagers verwendet und Sie erhalten eine Ausgabe ähnlich wie die folgende:

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

Wenn Sie das Beispielprogramm ausführen und den Namen einer TCP/IP-Verbindung sowie eines Zielwarteschlangenmanagers angeben, zum Beispiel:

```
amqscnxc -x machine.site.company.com MACHINE
```

, erhalten Sie eine ähnliche Ausgabe wie die folgende:

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

Das Data-Conversion-Beispielprogramm

Das Data-Conversion-Beispielprogramm ist ein Entwurf einer Datenkonvertierungsexitroutine. Dieser Abschnitt erläutert die Gestaltung des Beispielprogramms zur Datenkonvertierung.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Gestaltung des Data-Conversion-Beispielprogramms

Jede Datenkonvertierungsexitroutine konvertiert ein benanntes Nachrichtenformat. Dieser Entwurf ist als ein Wrapper für Codefragmente vorgesehen, die vom Dienstprogramm zur Generierung des Datenkonvertierungsexits generiert wurden.

Das Dienstprogramm erstellt ein Codefragment pro Datenstruktur; mehrere solcher Strukturen bilden ein Format, daher werden diesem Entwurf mehrere Codefragmente hinzugefügt, um eine Routine für die Datenkonvertierung des gesamten Format zu erstellen.

Das Programm überprüft dann, ob die Konvertierung erfolgreich war oder fehlgeschlagen ist und gibt die erforderlichen Werte an den Aufrufer zurück.

Beispiele zur Datenbankkoordination

Zwei Beispiele zur Veranschaulichung, wie IBM MQ Aktualisierungen von IBM MQ und Datenbankaktualisierungen in derselben Arbeitseinheit koordinieren kann.

Diese Beispiele sind:

1. AMQSXAS0 (in C) oder AMQ0XAS0 (in COBOL), aktualisiert eine einzelne Datenbank in einer IBM MQ-Arbeitseinheit.
2. AMQSXAG0 (in C) oder AMQ0XAG0 (in COBOL), AMQSXAB0 (in C) oder AMQ0XAB0 (in COBOL) und AMQSXAF0 (in C) oder AMQ0XAF0 (in COBOL), aktualisieren zusammen zwei Datenbanken in einer IBM MQ-Arbeitseinheit und zeigen den Zugriff auf mehrere Datenbanken. Diese Beispiele zeigen die Verwendung des MQBEGIN-Aufrufs sowie gemischter SQL- und IBM MQ-Aufrufe und erläutern, wo und wann eine Datenverbindung hergestellt wird.

Abbildung 128 auf Seite 1139 zeigt, wie Sie mit den bereitgestellten Beispielen Datenbanken aktualisieren:

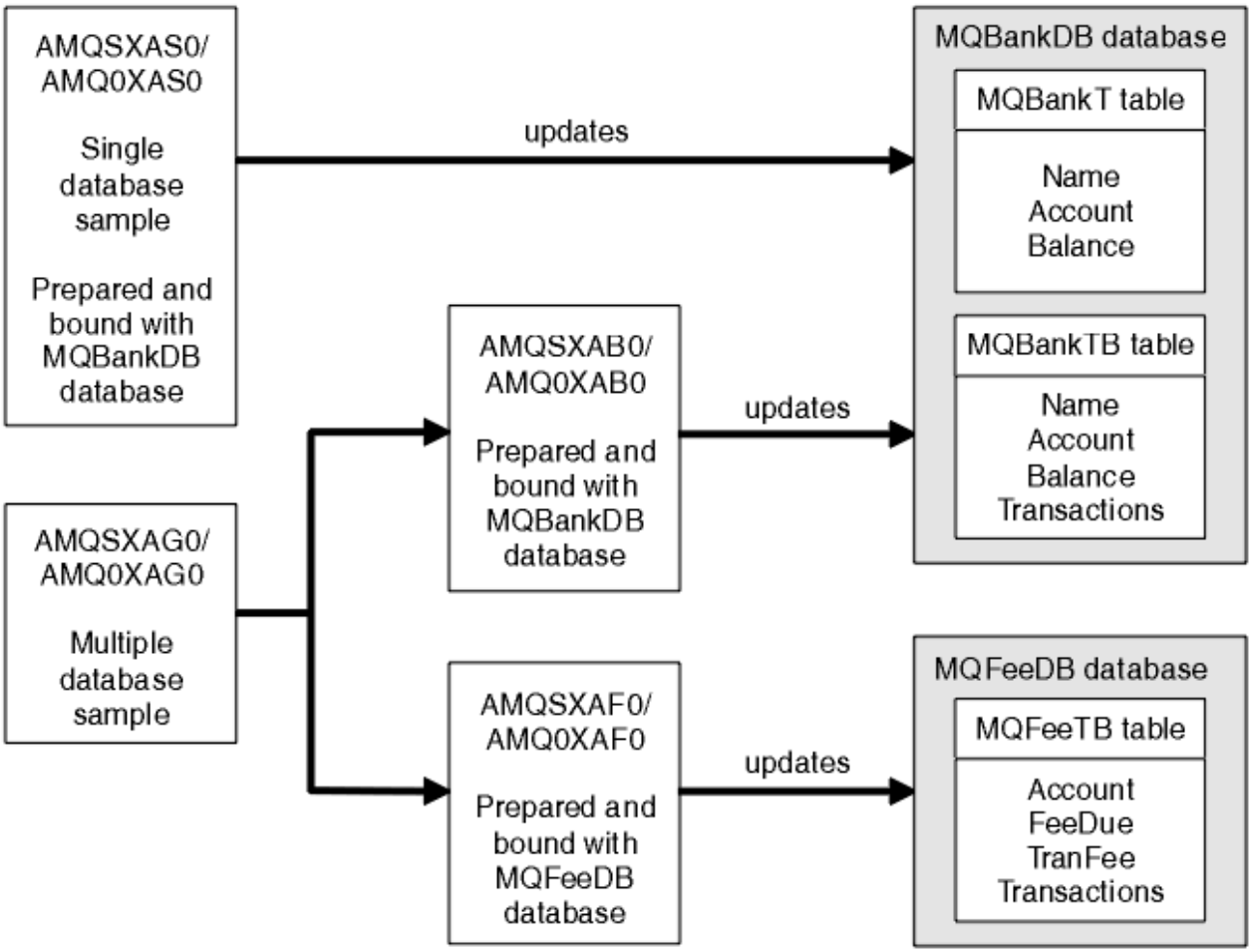


Abbildung 128. Beispiele zur Datenbankkoordination

Die Programme lesen eine Nachricht aus einer Warteschlange (unter Synchronisationspunkt) und rufen anschließend mithilfe der Informationen in der Nachricht die dazugehörigen Informationen aus der Datenbank ab und aktualisieren sie. Anschließend wird der neue Status der Datenbank ausgegeben.

Die Programmlogik gliedert sich wie folgt:

1. Verwendung des Namens der Eingabewarteschlange aus dem Programmargument

2. Verbindung zum Standardwarteschlangenmanager (oder optional zum bereitgestellten Namen in C) mittels MQCONN
3. Öffnen einer Warteschlange (mittels MQOPEN) zur Eingabe, wenn keine Fehler auftreten
4. Starten einer Arbeitseinheit mittels MQBEGIN
5. Abrufen der nächsten Nachricht (mittels MQGET) aus der Warteschlange unter Synchronisationspunkt
6. Abrufen von Informationen aus Datenbanken
7. Aktualisieren von Informationen aus Datenbanken
8. Festschreiben von Änderungen mittels MQCMIT
9. Ausgeben der aktualisierten Informationen (ist keine Nachricht verfügbar, zählt dies als Fehler und die Schleife wird beendet)
10. Schließen der Warteschlange mittels MQCLOSE
11. Trennen der Verbindung zur Warteschlange mittels MQDISC

In den Beispielen werden SQL-Cursor verwendet. Lesevorgänge aus den Datenbanken (d. h. mehrere Instanzen) werden während der Verarbeitung einer Nachricht gesperrt. Dadurch können mehrere Instanzen dieser Programme gleichzeitig ausgeführt werden. Die Cursor werden vom MQCMIT-Aufruf explizit geöffnet, jedoch implizit geschlossen.

Das Beispiel für eine einzelne Datenbank (AMQSXAS0 oder AMQ0XAS0) enthält keine SQL CONNECT-Anweisungen und die Verbindung zur Datenbank wird von IBM MQ implizit mit dem MQBEGIN-Aufruf hergestellt. Das Beispiel für mehrere Datenbanken (AMQSXAG0 oder AMQ0XAG0, AMQSXAB0 oder AMQ0XAB0 und AMQSXAF0 oder AMQ0XAF0) enthält SQL CONNECT-Anweisungen, da manche Datenbankprodukte nur eine aktive Verbindung zulassen. Für den Fall, dass dies für Ihr Datenbankprodukt nicht zutrifft oder Sie in mehreren Datenbankprodukten auf eine einzelne Datenbank zugreifen, können die SQL CONNECT-Anweisungen entfernt werden.

Die Beispiele wurden mit dem IBM Db2-Datenbankprodukt vorbereitet und müssen für andere Datenbankprodukte unter Umständen modifiziert werden.

Die SQL-Fehlerprüfung verwendet Routinen in UTIL.C und CHECKERR.CBL, bereitgestellt von Db2. Diese müssen vor der Kompilierung und Verknüpfung kompiliert oder ersetzt werden.

Anmerkung: Falls Sie zur SQL-Fehlerprüfung die COBOL-Quelle CHECKERR.MFC von Micro Focus verwenden, müssen Sie den Namen der Programm-ID zu Großbuchstaben ändern (also CHECKERR), damit AMQ0XAS0 ordnungsgemäße Verknüpfungen herstellen kann.

Datenbanken und Tabellen erstellen
Vor dem Kompilieren der Beispiele müssen Sie die Datenbanken und die Tabellen erstellen.

Erstellen Sie die Datenbanken mit der für Ihr Datenbankprodukt üblichen Methode, zum Beispiel:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

Erstellen Sie die Tabellen wie folgt mithilfe von SQL-Anweisungen:

In C:

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
```

```
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

In COBOL:

```
EXEC SQL CREATE TABLE
  MQBankT(Name         VARCHAR(40) NOT NULL,
          Account      INTEGER     NOT NULL,
          Balance      INTEGER     NOT NULL,
          PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name         VARCHAR(40) NOT NULL,
           Account      INTEGER     NOT NULL,
           Balance      INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER   NOT NULL,
          FeeDue       INTEGER   NOT NULL,
          TranFee      INTEGER   NOT NULL,
          Transactions INTEGER,
          PRIMARY KEY (Account))
  END-EXEC.
```

Geben Sie die Daten wie folgt mithilfe von SQL-Anweisungen in die Tabellen ein:

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
.
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
.
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
.
```

Anmerkung: Verwenden Sie für COBOL die gleichen SQL-Anweisungen, aber fügen Sie END_EXEC am Ende jeder Zeile hinzu.

Beispielprogramme vorkompilieren, kompilieren und verknüpfen
Dieser Abschnitt enthält Informationen zum Vorkompilieren, Kompilieren und Verknüpfen von Beispielprogrammen in C und COBOL.

Führen Sie eine Vorkompilierung der Dateien `.SQC` (in C) und `.SQB` (in COBOL) durch und binden Sie sie an die entsprechende Datenbank, um die Dateien `.C` bzw. `.CBL` zu erstellen. Verwenden Sie hierzu die für Ihre Datenbank übliche Methode.

Vorkompilierung in C

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

Vorkompilierung in COBOL

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```

Kompilieren und Verbinden

Die folgenden Beispielbefehle verwenden die Symbole *DB2TOP* und *MQ_INSTALLATION_PATH*. *DB2TOP* stellt das Installationsverzeichnis für das Db2-Produkt dar. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

- AIX Unter AIX lautet der Verzeichnispfad:

  ```
  /usr/lpp/db2_05_00
  ```

- Windows Auf Windows-Systemen richtet sich der Verzeichnispfad nach dem für die Installation des Produkts ausgewählten Pfad. Bei Auswahl der Standardeinstellungen lautet der Pfad wie folgt:

  ```
  c:\sqllib
  ```

Anmerkung: Bevor Sie auf Windows-Systemen den Verknüpfungsbefehl ausgeben, stellen Sie sicher, dass die Umgebungsvariable LIB die Pfade zu den Bibliotheken von Db2 und IBM MQ enthält.

Kopieren Sie die folgenden Dateien in ein temporäres Verzeichnis:

- Datei `amqsxag0.c` aus Ihrer IBM MQ-Installation

  Anmerkung: Diese Datei finden Sie in folgenden Verzeichnissen:

  – Linux AIX Auf Systemen mit AIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows Auf Systemen mit Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Dateien `.c`, die Sie durch die Vorkompilierung der Quellendateien `.sqc` (`amqsxas0.sqc`, `amqsxaf0.sqc`und `amqsxab0.sqc`) erhalten haben
- Dateien `util.c` und `util.h` aus Ihrer Db2-Installation

  Anmerkung: Diese Dateien finden Sie in folgendem Verzeichnis:

  ```
  DB2TOP/samples/c
  ```

Erstellen Sie die Objektdateien für alle Dateien `.c`; verwenden Sie hierzu den folgenden Compilerbefehl für die von Ihnen verwendete Plattform:

- AIX AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Windows Systeme mit Windows

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

Erstellen Sie die ausführbare Datei `amqsxag0` mit dem folgenden Verknüpfungsbefehl für die von Ihnen verwendete Plattform:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows Systeme mit Windows

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

Erstellen Sie die ausführbare Datei `amqsxas0` mit den folgenden Kompilierungs- und Verknüpfungsbefehlen für die von Ihnen verwendete Plattform:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Windows Systeme mit Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Weitere Informationen

AIX Wenn Sie unter AIX arbeiten und auf Oracle zugreifen möchten, verwenden Sie den Compiler 'xlc_r' und verknüpfen Sie mit 'libmqm_r.a'.

Beispielprogramme ausführen
In diesem Abschnitt erfahren Sie, wie Sie den Warteschlangenmanager konfigurieren, bevor Sie die Beispiele zur Datenbankkoordination unter C und COBOL ausführen.

Bevor Sie die Beispielprogramme ausführen, konfigurieren Sie den Warteschlangenmanager mit dem Datenbankprodukt, das Sie verwenden. Weitere Informationen hierzu erhalten Sie unter Szenario 1: Warteschlangenmanager führt die Koordination aus.

Die folgenden Titel stellen Informationen zum Ausführen der Beispielprogramme in C und COBOL bereit:

- „C-Beispiele” auf Seite 1143
- „COBOL-Beispiele” auf Seite 1144

C-Beispiele

Nachrichten müssen das folgende Format haben, um aus einer Warteschlange gelesen werden zu können:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

Mit AMQSPUT können Sie die Nachrichten in die Warteschlange einreihen.

Die Beispielprogramme zur Datenbankkoordination verwenden zwei Parameter:

1. Warteschlangenname (erforderlich)
2. Name des Warteschlangenmanagers (optional)

Angenommen, Sie haben einen Warteschlangenmanager für das Einzeldatenbankbeispiel 'singDBQM' mit einer Warteschlange namens 'singDBQ' erstellt und konfiguriert. Dann erhöhen Sie wie folgt das Konto von Fred Bloggs um 50:

```
AMQSPUT singDBQ singDBQM
```

Geben Sie nun die folgende Nachricht ein:

```
UPDATE Balance change=50 WHERE Account=1
```

Sie können mehrere Nachrichten in die Warteschlange einreihen.

```
AMQSXAS0 singDBQ singDBQM
```

Der aktualisierte Status von Fred Bloggs' Konto wird ausgegeben.

Angenommen, Sie haben einen Warteschlangenmanager für das Mehrfachdatenbankbeispiel 'multDBQM' mit einer Warteschlange namens 'multDBQ' erstellt und konfiguriert. Dann verringern Sie wie folgt das Konto von Mary Brown um 75:

```
AMQSPUT multDBQ multDBQM
```

Geben Sie nun die folgende Nachricht ein:

```
UPDATE Balance change=-75 WHERE Account=3
```

Sie können mehrere Nachrichten in die Warteschlange einreihen.

```
AMQSXAG0 multDBQ multDBQM
```

Der aktualisierte Status von Mary Browns Konto wird ausgegeben.

COBOL-Beispiele

Nachrichten müssen das folgende Format haben, um aus einer Warteschlange gelesen werden zu können:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

Aus Gründen der Einfachheit muss `Balance change` eine signierte achtstellige Zahl und `Account` muss eine achtstellige Zahl sein.

Mit dem Beispielprogramm AMQSPUT können Sie die Nachrichten in die Warteschlange einreihen.

Die Beispiele verwenden keine Parameter, sondern den Standardwarteschlangenmanager. Er kann so konfiguriert werden, dass er jederzeit eines der Beispiele ausführt. Angenommen, Sie haben den Standardwarteschlangenmanager für das Einzeldatenbankbeispiel mit einer Warteschlange namens 'singDBQ' konfiguriert. Dann erhöhen Sie wie folgt das Konto von Fred Bloggs um 50:

```
AMQSPUT singDBQ
```

Geben Sie nun die folgende Nachricht ein:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

Sie können mehrere Nachrichten in die Warteschlange einreihen:

```
AMQ0XAS0
```

Geben Sie den Namen der Warteschlange ein:

```
singDBQ
```

Der aktualisierte Status von Fred Bloggs' Konto wird ausgegeben.

Angenommen, Sie haben den Standardwarteschlangenmanager für das Mehrfachdatenbankbeispiel mit einer Warteschlange namens 'multDBQ' konfiguriert. Dann verringern Sie wie folgt das Konto von Mary Bloggs um 75:

```
AMQSPUT multDBQ
```

Geben Sie nun die folgende Nachricht ein:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

Sie können mehrere Nachrichten in die Warteschlange einreihen:

```
AMQ0XAG0
```

Geben Sie den Namen der Warteschlange ein:

```
multDBQ
```

Der aktualisierte Status von Mary Browns Konto wird ausgegeben.

Beispielprogramm für eine Steuerroutine der Warteschlange für nicht zustellbare Nachrichten

Eine Beispielsteuerroutine der Warteschlange für nicht zustellbare Nachrichten wird bereitgestellt. Der Name der ausführbaren Version lautet `amqsdlq`. Wenn Sie eine Steuerroutine der Warteschlange für nicht zustellbare Nachrichten verwenden möchten, die sich von **RUNMQDLQ**unterscheidet, können Sie die Quelle des Beispiels als Basis verwenden.

Das Beispiel ist vergleichbar mit der im Produkt bereitgestellten Steuerroutine der Warteschlange für nicht zustellbare Nachrichten, allerdings sind Trace- und Fehlerprotokollierung anders. Es stehen zwei Umgebungsvariablen zur Verfügung:

ODQ_TRACE
: Geben Sie YES oder `yes` an, um die Traceerstellung zu aktivieren.

ODQ_MSG
: Auf den Namen der Datei setzen, die Fehler- und Informationsnachrichten enthält. Die bereitgestellte Datei heißt `amqsdlq.msg`.

Sie müssen Ihrer Umgebung diese Variablen mit den plattformabhängigen Befehlen **`export`** bzw. **`set`** mitteilen; die Tracefunktion wird mit dem Befehl **`unset`** deaktiviert.

Sie können die Fehlernachrichtendatei `amqsdlq.msg`an Ihre eigenen Anforderungen anpassen. Das Beispielprogramm reiht Nachrichten **nicht** in die IBM MQ-Fehlerprotokolldatei ein, sondern in 'stdout'.

Weitere Informationen zur Funktionsweise der Steuerroutine der Warteschlange für nicht zustellbare Nachrichten und zu ihrer Ausführung finden Sie im Abschnitt Nachrichten in einer IBM MQ -Warteschlange für nicht zustellbare Nachrichten verarbeiten oder im Handbuch *Systemverwaltung* für Ihre Plattform.

Das Distribution List-Beispielprogramm

Das Distribution List-Beispielprogramm 'amqsptl0' veranschaulicht das Einreihen einer Nachricht in mehrere Nachrichtenwarteschlangen. Es basiert auf dem MQPUT-Beispielprogramm 'amqsput0'.

Distribution List-Beispielprogramm 'amqspt10' ausführen

Das Distribution List-Beispielprogramm wird ähnlich wie die Put-Beispielprogramme ausgeführt.

Es verwendet die folgenden Parameter:

- Namen der Warteschlangen
- Die Namen der WS-Manager

Diese Werte werden paarweise eingegeben. For example:

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

Die Warteschlangen werden mittels MQOPEN geöffnet, Nachrichten werden unter Verwendung von MQPUT in die Warteschlangen eingereiht. Wird der Name einer Warteschlange oder eines Warteschlangenmanagers nicht erkannt, werden Ursachencodes zurückgegeben.

Vergessen Sie nicht, Kanäle zwischen den Warteschlangenmanagern zu definieren, damit die Nachrichten zwischen ihnen fließen können. Diese Aufgabe wird nicht vom Beispielprogramm übernommen.

Gestaltung des Distribution List-Beispielprogramms

Datensätze von Put-Nachrichten (Put Message Records, MQPMRs) geben Nachrichtenattribute für jedes Ziel an. Das Beispielprogramm stellt Werte für *`MsgId`* und *`CorrelId`* bereit. Diese überschreiben die in der MQMD-Struktur angegebenen Werte.

Das Feld *`PutMsgRecFields`* in der MQPMO-Struktur gibt an, welche Felder in den MQPMRs vorhanden sind:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

Als Nächstes ordnet das Beispielprogramm die Antwortdatensätze und Objektdatensätze zu. Die Objektdatensätze (MQORs) benötigen mindestens ein Namenspaar für *`ObjectName`* und *`ObjectQMgrName`*, d. h. eine gerade Anzahl Namen.

Im nächsten Schritt werden die Warteschlangenmanager mittels MQCONN verbunden. Das Beispielprogramm versucht, eine Verbindung zu dem Warteschlangen herzustellen, der der ersten Warteschlange in der MQOR zugeordnet ist; schlägt dies fehl, durchläuft es die Objektdatensätze. Sie werden entsprechend informiert, wenn keine Verbindung zu einem Warteschlangenmanager hergestellt werden kann und das Programm beendet wird.

Die Zielwarteschlangen werden mittels MQOPEN geöffnet und die Nachricht wird unter Verwendung von MQPUT in diese Warteschlangen eingereiht. Probleme und Fehler werden in den Antwortdatensätzen (MQRRs) aufgelistet.

Zum Schluss werden die Zielwarteschlangen mittels MQCLOSE geschlossen und das Programm trennt die Verbindung zum Warteschlangenmanager mittels MQDISC. Für jeden Aufruf werden die gleichen Antwortdatensätze mit dem *`CompCode`* (Vergleichscode) und dem *`Reason`* (Grund) verwendet.

Die Echo-Beispielprogramme

Die Echo-Beispielprogramme melden eine Nachricht aus einer Nachrichtenwarteschlange an die Antwortwarteschlange zurück.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Die Programme wurden als Auslöserprogramme konzipiert.

Auf IBM i-, AIX, Linux, and Windows-Systemen ist ihre einzige Eingabe eine MQTMC2-Struktur (Auslösenachricht), die den Namen einer Zielwarteschlange und des Warteschlangenmanagers enthält. Die COBOL-Version verwendet den Standardwarteschlangenmanager.

IBM i Damit der Auslöseprozess unter IBM ifunktioniert, stellen Sie sicher, dass das Echo-Beispielprogramm, das Sie verwenden möchten, durch Nachrichten ausgelöst wird, die in der Warteschlange SYSTEM.SAMPLE.ECHO Geben Sie dazu den Namen des gewünschten Echo-Beispielprogramms im Feld *ApplId* der Prozessdefinition SYSTEM.SAMPLE.ECHOPROCESS ein. (Hierfür können Sie den Befehl CHGMQMPRC verwenden; ausführliche Informationen hierzu finden Sie unter Change MQ Process (CHGMQMPRC).) Die Beispielwarteschlange verfügt über den Auslösertyp FIRST, wenn also bereits Nachrichten in der Warteschlange enthalten sind, bevor Sie das Request-Beispiel ausführen, wird das Echo-Beispiel nicht von den Nachrichten ausgelöst, die Sie senden.

Wenn Sie die Definition ordnungsgemäß festgelegt haben, starten Sie zuerst AMQSERV4 in einem Job und dann AMQSREQ4 in einem anderen Job. Anstelle von AMQSERV4 könnten Sie AMQSTRG4 verwenden, doch potenzielle Jobübergabeverzögerungen könnten das Nachvollziehen der Vorgänge erschweren.

Verwenden Sie die Request-Beispielprogramme, um Nachrichten an die Warteschlange SYSTEM.SAMPLE.ECHO zu senden. Die Echo-Beispielprogramme senden eine Antwortnachricht, die die Daten in der Anforderungsnachricht enthält, an die Empfangswarteschlange für Antworten, die in der Anforderungsnachricht angegeben ist.

Gestaltung der Echo-Beispielprogramms

Das Programm öffnet die Warteschlange, die in der Auslösenachrichtstruktur genannt wurde, welche bei ihrem Start übergeben wurde. (Diese wird auch als Anforderungswarteschlange bezeichnet.) Das Programm verwendet den MQOPEN-Aufruf, um diese Warteschlange für die gemeinsame Eingabe zu öffnen.

Mit dem MQGET-Aufruf werden Nachrichten aus der Warteschlange entfernt. Dieser Aufruf verwendet die Optionen MQGMO_ACCEPT_TRUNCATED_MSG, MQGMO_CONVERT und MQGMO_WAIT mit einem Warteintervall von 5 Sekunden. Das Programm prüft den Deskriptor jeder einzelnen Nachricht, um festzustellen, ob es sich um eine Anforderungsnachricht handelt; ist dies nicht der Fall, verwirft das Programm die Nachricht und zeigt eine Warnung an.

Das Programm liest dann für jede Eingabezeile den Text in einen Puffer und verwendet den MQPUT1-Aufruf, um eine Anforderungsnachricht mit dem Text dieser Zeile in die Empfangswarteschlange für Antworten einzureihen.

Schlägt der MQGET-Aufruf fehl, reiht das Programm eine Berichtsnachricht in die Empfangswarteschlange für Antworten ein und setzt das Feld *Feedback* des Nachrichtendeskriptors auf den durch den MQGET-Aufruf zurückgegebenen Ursachencode.

Wenn in der Anforderungswarteschlange keine Nachrichten mehr vorhanden sind, schließt das Programm diese Warteschlange und trennt die Verbindung zum Warteschlangenmanager.

IBM i Unter IBM i kann das Programm auch auf Nachrichten antworten, die von anderen Plattformen als IBM MQ for IBM i an die Warteschlange gesendet wurden, obwohl für diesen Fall kein Beispielprogramm zur Verfügung steht. So entwerfen Sie ein funktionsfähiges ECHO-Programm:

- Schreiben Sie ein Programm mit den korrekt angegebenen Parametern **Format**, **Encoding** und **CCSID**, um Textanforderungsnachrichten zu senden.

Das ECHO-Programm fordert den Warteschlangenmanager gegebenenfalls auf, eine Konvertierung der Nachrichtendaten durchzuführen.

- Geben Sie im IBM MQ for IBM i-Sendekanal CONVERT(*YES) an, falls das von Ihnen geschriebene Programm keine vergleichbare Konvertierung für die Antwort bietet.

Die Get-Beispielprogramme

Mit den Get-Beispielprogrammen werden Nachrichten mithilfe des MQGET-Aufrufs aus einer Warteschlange abgerufen.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Gestaltung des Get-Beispielprogramms

Das Programm öffnet die Zielwarteschlange mit dem MQOPEN-Aufruf und der Option MQOO_INPUT_AS_Q_DEF. Wenn es die Warteschlange nicht öffnen kann, zeigt das Programm eine Fehlernachricht an, die den vom MQOPEN-Aufruf zurückgegebenen Ursachencode enthält.

Das Programm entfernt jede Nachricht mit dem MQGET-Aufruf aus der Warteschlange und zeigt dann die Daten an, die in der jeweiligen Nachricht enthalten sind. Der MQGET-Aufruf verwendet die MQGMO_WAIT-Option und gibt einen *WaitInterval* von 15 Sekunden an, sodass das Programm für diesen Zeitraum wartet, wenn keine Nachricht in der Warteschlange vorhanden ist. Wenn keine Nachricht eintrifft, bevor dieses Intervall abläuft, schlägt der Aufruf fehl und gibt den Ursachencode MQRC_NO_MSG_AVAILABLE zurück.

Das Programm veranschaulicht, wie die Felder *MsgId* und *CorrelId* der MQMD-Struktur nach jedem MQGET-Aufruf gelöscht werden müssen, da der Aufruf diese Felder auf die Werte setzt, die in den abgerufenen Nachrichten enthalten sind. Durch das Löschen dieser Felder rufen aufeinanderfolgende MQGET-Aufrufe die Nachrichten in der Reihenfolge ab, in der diese in der Warteschlange gehalten werden.

Der MQGET-Aufruf gibt einen Puffer mit einer festen Größe an. Wenn eine Nachricht länger als dieser Puffer ist, schlägt der Aufruf fehl und das Programm hält an.

Das Programm wird fortgesetzt, bis der MQGET-Aufruf entweder den Ursachencode MQRC_NO_MSG_AVAILABLE zurückgibt oder der MQGET-Aufruf fehlschlägt. Wenn der Aufruf fehlschlägt, zeigt das Programm eine Fehlernachricht mit dem Ursachencode an.

Das Programm schließt dann die Warteschlange mithilfe des MQCLOSE-Aufrufs.

Beispielprogramme 'amqsget' und 'amqsgetc' ausführen

Jedes dieser Programme verwendet folgende positionsgebundene Parameter:

1. Name der Quellenwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (optional)

   Wenn kein Warteschlangenmanager angegeben ist, stellt **amqsget** eine Verbindung zum Standardwarteschlangenmanager und **amqsgetc** eine Verbindung zu dem Warteschlangenmanager her, der durch die Umgebungsvariable *MQSERVER* oder die Clientkanaldefinitionsdatei angegeben ist.
3. Optionen zum Öffnen (optional)

   Sind keine Optionen zum Öffnen angegeben, verwendet das Beispielprogramm den Wert 8193. Dieser setzt sich aus diesen beiden Optionen zusammen:

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING
4. Optionen zum Schließen (optional)

   Sind keine Optionen zum Schließen angegeben, verwendet das Beispielprogramm den Wert 0 (MQCO_NONE).

Verwenden Sie die folgenden Umgebungsvariablen, um Berechtigungsnachweise für die Authentifizierung beim Warteschlangenmanager anzugeben:

MQSAMP_USER_ID
Wird auf die für die Verbindungsauthentifizierung zu verwendende Benutzer-ID gesetzt, wenn Sie eine Benutzer-ID und ein Kennwort für die Authentifizierung beim Warteschlangenmanager verwenden wollen. Das Programm fordert zur Eingabe des Kennworts für die Benutzer-ID auf.

MQSAMP_TOKEN
Legen Sie einen Wert fest, der nicht leer ist, wenn Sie ein Authentifizierungstoken für die Authentifizierung beim Warteschlangenmanager bereitstellen wollen. Das Programm fordert zur Eingabe des Authentifizierungstoken auf. Authentifizierungstoken können nur vom **amqsgetc** -Beispiel verwendet werden, das Clientbindungen verwendet.

Geben Sie zum Ausführen dieser Programme einen der folgenden Befehle ein:

- `amqsget` *`myqueue qmanagername`*
- `amqsgetc` *`myqueue qmanagername`*

Dabei steht *myqueue* für den Namen der Warteschlange, aus der das Programm die Nachrichten abruft, und *qmanagername* gibt den Warteschlangenmanager an, der Eigner von *myqueue* ist.

Verwendung von 'amqsget' und 'amqsgetc'

Beachten Sie, dass **amqsget** eine lokale Verbindung zum Warteschlangenmanager ausführt, indem gemeinsam genutzter Speicher für die Zuordnung zum Warteschlangenmanager verwendet wird, der nur auf dem System ausgeführt werden kann, auf dem sich der Warteschlangenmanager befindet, während **amqsgetc** eine Verbindung im Client-Stil ausführt (selbst wenn eine Verbindung zu einem Warteschlangenmanager auf demselben System hergestellt wird).

Wenn Sie **amqsgetc** verwenden, müssen Sie die Anwendungsdetails angeben, wie Sie den Warteschlangenmanager tatsächlich erreichen können, und zwar in Bezug auf den Warteschlangenmanager-Host oder die IP-Adresse und den Listener-Port des Warteschlangenmanagers.

Normalerweise erfolgt dies entweder über die Umgebungsvariable **MQSERVER** oder durch Definieren von Verbindungsdetails mithilfe einer Definitionstabelle für Clientkanäle, die auch über Umgebungsvariablen für **amqsgetc** bereitgestellt werden kann (siehe MQCCDTURL).

Im Folgenden sehen Sie ein Beispiel für die Verwendung von **MQSERVER**, das lokal eine Verbindung zu einem Warteschlangenmanager herstellt, bei dem ein Listener an Port 1414 ausgeführt wird und der den Standardserververbindungskanal verwendet:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

Beispielprogramme zur Hochverfügbarkeit

Die Beispielprogramme zur Hochverfügbarkeit - **amqsghac**, **amqsphac** und **amqsmhac** - verwenden eine automatisierte Verbindungswiederholung zur Wiederherstellungsdemonstration nach dem Ausfall eines Warteschlangenmanagers. **amqsfhac** prüft, ob ein Warteschlangenmanager, der einen vernetzten Speicher verwendet, nach einem Fehler die Nachrichtenintegrität beibehält.

Die Programme **amqsghac**, **amqsphac** und **amqsmhac** werden über die Befehlszeile gestartet und können nach dem Ausfall einer Instanz eines Mehrinstanz-Warteschlangenmanagers kombiniert zur Wiederherstellungsdemonstration verwendet werden.

Alternativ können Sie mit den Programmen **amqsghac**, **amqsphac** und **amqsmhac** auch die Client-Verbindungswiederholung zu Einzel-Instanz-Warteschlangenmanagern demonstrieren, die üblicherweise in einer Warteschlangenmanagergruppe konfiguriert sind.

Um das Beispiel aus Gründen der leichteren Konfiguration so einfach wie möglich zu halten, werden Ihnen die Beispielprogramme gezeigt, die die Verbindung zu einem Einzel-Instanz-Warteschlangenmanager wiederholen, der gestartet, gestoppt und dann erneut gestartet wird (siehe „Warteschlangenmanager konfigurieren und steuern” auf Seite 1152).

Verwenden Sie **amqsfhac** parallel zu **amqmfsck**, um die Integrität des Dateisystems zu prüfen. Weitere Informationen hierzu finden Sie in den Abschnitten **amqmfsck** (Dateisystemprüfung) und Verhalten eines gemeinsam genutzten Dateisystems prüfen.

amqsphac *Warteschlangenname* [*Warteschlangenmanagername*]

- **amqsphac** ist eine IBM MQ MQI client-Anwendung. Im Abstand von zwei Sekunden reiht sie Nachrichten aus einer Folge von Nachrichten in eine Warteschlange ein und zeigt die Ereignisse an, die an ihren Ereignishandler gesendet wurden.
- Zum Einreihen von Nachrichten in die Warteschlange wird kein Synchronisationspunkt verwendet.
- Die Verbindungswiederholung kann für jeden Warteschlangenmanager in derselben Warteschlangenmanagergruppe durchgeführt werden.

amqsghac *Warteschlangenname* [*Warteschlangenmanagername*]

- **amqsghac** ist eine IBM MQ MQI client-Anwendung. Sie ruft Nachrichten aus einer Warteschlange ab und zeigt Ereignisse an, die an ihren Ereignishandler gesendet wurden.
- Zum Abrufen von Nachrichten aus der Warteschlange wird kein Synchronisationspunkt verwendet.
- Die Verbindungswiederholung kann für jeden Warteschlangenmanager in derselben Warteschlangenmanagergruppe durchgeführt werden.

amqsmhac -s *Quellenwarteschlangenname* -t *Zielwarteschlangenname* [-m *Warteschlangenmanagername*] [-w *Warteintervall*]

- **amqsmhac** ist eine IBM MQ MQI client-Anwendung. Sie kopiert Nachrichten mit einem Standardwarteintervall von 15 Minuten, nachdem die letzte Nachricht empfangen wurde, aus einer Warteschlange in eine andere Warteschlange, bevor das Programm beendet wird.
- Die Nachrichten werden innerhalb des Synchronisationspunkts kopiert.
- Die Verbindung kann nur zum selben Warteschlangenmanager wiederholt werden.

amqsfhac *Warteschlangenmanagername Warteschlangenname Seitenwarteschlangename InTransaktionszähler Wiederholungszähler* (0 | 1 | 2)

- **amqsfhac** ist eine IBM MQ MQI client-Anwendung. Sie prüft, ob ein IBM MQ-Multi-Instanz-Warteschlangenmanager, der vernetzten Speicher verwendet (z. B. ein NAS- oder Clusterdateisystem), die Integrität der Daten beibehält. Führen Sie die unter Verhalten des gemeinsam genutzten Dateisystems überprüfenbeschriebenen Schritte aus, um **amqsfhac** auszuführen.
- Bei der Verbindungsherstellung zu *Warteschlangenmanagername* verwendet es die Option `MQCNO_RECONNECT_Q_MGR` und stellt die Verbindung bei Ausfall des Warteschlangenmanagers automatisch wieder her.
- Es reiht die persistenten Nachrichten *InTransaktionszähler*Wiederholungszähler* in *Warteschlangenname* ein. In dieser Zeit verursachen Sie den beliebig häufigen Ausfall des Warteschlangenmanagers. Jedes Mal stellt **amqsfhac** die Verbindung zum Warteschlangenmanager wieder her und setzt den Vorgang fort. Der Test soll sicherstellen, dass keine Nachrichten verloren gehen.
- Es wird jeweils die Anzahl von *InTransaktionszähler* Nachrichten innerhalb einer Transaktion eingereiht. Die Transaktion wird *Wiederholungszähler* Male wiederholt. Schlägt eine Transaktion fehl, macht **amqsfhac** sie rückgängig und übergibt sie erneut, wenn **amqsfhac** die Verbindung zum Warteschlangenmanager wieder herstellt.
- Zudem reiht es Nachrichten in *Seitenwarteschlangenname* ein. Mittels *Seitenwarteschlangenname* prüft es, ob alle Nachrichten aus *Warteschlangenname* erfolgreich festgeschrieben oder rückgängig gemacht wurden. Wird eine Inkonsistenz festgestellt, gibt es eine entsprechende Fehlernachricht aus.
- Variieren Sie die Anzahl des Ausgabe-Tracing von **amqsfhac**, indem Sie den letzten Parameter `(0|1|2)` entsprechend festlegen.

  0

  Geringste Ausgabe.

1
Mittlere Ausgabe.

2
Höchste Ausgabe.

Clientverbindung konfigurieren

Zur Ausführung der Beispielprogramme müssen Sie einen Client- und Serververbindungskanal konfigurieren. Die Clientprüfprozedur erläutert, wie eine Clienttestumgebung einzurichten ist.

Alternativ können Sie auch die Konfiguration aus dem folgenden Beispiel verwenden.

Beispiel mit Verwendung von amqsghac, amqsphac und amqsmhac

Das Beispiel zeigt wiederverbindbare Clients, die einen Einzel-Instanz-Warteschlangenmanager verwenden.

Mit dem Befehl **amqsphac** werden Nachrichten in die Warteschlange SOURCE gestellt, mittels **amqsmhac** an TARGET übertragen und mittels **amqsghac** aus TARGET abgerufen; siehe Abbildung 129 auf Seite 1151.

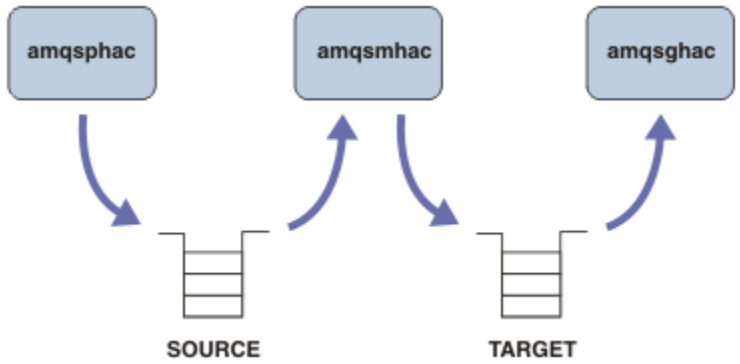


Abbildung 129. Beispiele zu wiederverbindbaren Clients

Gehen Sie zum Ausführen der Beispielprogramme wie folgt vor:

1. Erstellen Sie eine Datei namens `hasamples.tst`, die folgende Befehle enthält:

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. Geben Sie an der Eingabeaufforderung die folgenden Befehle ein:
   a. `crtmqm QM1`
   b. `strmqm QM1`
   c. `runmqsc QM1 < hasamples.tst`
3. Setzen Sie die Umgebungsvariable **MQCHLLIB** auf den Pfad zur `AMQCLCHL.TAB`-Clientkanaldefinitionsdatei, z. B. `SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc`.
4. Öffnen Sie drei neue Fenster mit festgelegtem Parameter **MQCHLLIB**. Geben Sie beispielsweise unter Windows drei Mal den Befehl **start** an der vorherigen Eingabeaufforderung ein, um jedes Programm in einem der Fenster zu starten. Informationen hierzu finden Sie unter Schritt „5” auf Seite 1152 in „Warteschlangenmanager konfigurieren und steuern” auf Seite 1152.)

5. Geben Sie den Befehl endmqm -r -p QM1 ein, um den Warteschlangenmanager zu stoppen, und lassen Sie dann zu, dass die Clients die Verbindung wiederherstellen.
6. Geben Sie den Befehl strmqm QM1 ein, um den Warteschlangenmanager neu zu starten.

Die Ergebnisse der Ausführung der Beispielprogramme **amqsghac**, **amqsphac** und **amqsmhac** unter Windows werden in den nachfolgenden Beispielen gezeigt.

Warteschlangenmanager konfigurieren und steuern

1. Erstellen Sie den Warteschlangenmanager.

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Merken Sie sich das Datenverzeichnis, um die Variable **MQCHLLIB** später festzulegen.
2. Starten Sie den Warteschlangenmanager.

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. Erstellen Sie die Warteschlangen und Kanäle, ändern Sie den Listener-Port und starten Sie Listener und Kanal.

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN) RE
PLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. Machen Sie den Clients die Clientkanaltabelle bekannt.

   Verwenden Sie das Datenverzeichnis, das in Schritt „1” auf Seite 1152 vom Befehl **crtmqm** zurückgegeben wurde, und fügen Sie ihm das Verzeichnis @ipcc hinzu, um die Variable **MQCHLLIB** festzulegen.

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. Starten Sie die Beispielprogramme in den anderen Fenstern.

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. Beenden Sie den Warteschlangenmanager und starten Sie ihn erneut.

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

Zugehörige Tasks

Verhalten eines gemeinsam genutzten Dateisystems überprüfen

Zugehörige Verweise

amqmfsck (Dateisystemprüfung)

Die Inquire-Beispielprogramme

Die Inquire-Beispielprogramme fragen über einen MQINQ-Aufruf einige Attribute der Warteschlange ab.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Diese Programme wurden als Auslöserprogramme konzipiert; ihre einzige Eingabe ist daher eine MQTMC2-Struktur (Auslösenachricht) für IBM MQ for Multiplatforms. Diese Struktur enthält den Namen einer Zielwarteschlange, deren Attribute abzufragen sind. Die C-Version verwendet auch den Namen des Warteschlangenmanagers. Die COBOL-Version verwendet den Standardwarteschlangenmanager.

Damit der Auslöseprozess funktioniert, stellen Sie sicher, dass das gewünschte Inquiry-Beispielprogramm durch Nachrichten ausgelöst wird, die in der Warteschlange SYSTEM.SAMPLE.INQ eintreffen. Geben Sie dazu den Namen des gewünschten Abfragebeispielprogramms im Feld *ApplicId* der Prozessdefinition SYSTEM.SAMPLE.INQPROCESS ein. IBM i Für IBM i können Sie den Befehl CHGMQMPRC verwenden; ausführliche Informationen hierzu finden Sie unter MQ-Prozess (CHGMQMPRC) ändern. Die Beispielwarteschlange verfügt über den Auslösertyp FIRST; wenn also bereits Nachrichten in der Warteschlange enthalten sind, bevor Sie das Request-Beispiel ausführen, wird das Inquiry-Beispiel nicht von den Nachrichten ausgelöst, die Sie senden.

Wenn Sie die Definition ordnungsgemäß festlegen müssen:

- ALW Starten Sie unter AIX, Linux, and Windows das Programm **runmqtrm** in einer Sitzung und anschließend das Programm `amqsreq` in einer anderen Sitzung.
- IBM i Starten Sie unter IBM i das Programm AMQSERV4 in einer Sitzung und anschließend das Programm AMQSREQ4 in einer anderen Sitzung. Anstelle von AMQSERV4 könnten Sie AMQSTRG4 verwenden, doch potenzielle Jobübergabeverzögerungen könnten das Nachvollziehen der Vorgänge erschweren.

Verwenden Sie die Request-Beispielprogramme, um Anforderungsnachrichten - jede mit nur einem Warteschlangennamen - an die Warteschlange SYSTEM.SAMPLE.INQ zu senden. Für jede Anforderungsnachricht senden die Inquiry-Beispielprogramme eine Antwortnachricht mit Informationen zur Warteschlange, die in der Anforderungsnachricht angegeben ist. Die Antworten werden an die Empfangswarteschlange für Antworten gesendet, die in der Anforderungsnachricht angegeben ist.

IBM i Wenn unter IBM i das Element QMQMSAMP.AMQSDATA(INQ) der Beispieleingabedatei verwendet wird, ist die zuletzt genannte Warteschlange nicht vorhanden, sodass das Beispiel eine Berichtsnachricht mit einem Ursachencode für den Fehler zurückgibt.

Gestaltung des Inquire-Beispielprogramms

Das Programm öffnet die Warteschlange, die in der Auslösenachrichtstruktur genannt wurde, welche bei ihrem Start übergeben wurde. (Zur Verdeutlichung nennen wir diese die *Anforderungswarteschlange*.) Das Programm verwendet den MQOPEN-Aufruf, um diese Warteschlange für die gemeinsame Eingabe zu öffnen.

Mit dem MQGET-Aufruf werden Nachrichten aus der Warteschlange entfernt. Dieser Aufruf verwendet die Optionen MQGMO_ACCEPT_TRUNCATED_MSG und MQGMO_WAIT mit einem Warteintervall von 5 Sekunden. Das Programm prüft den Deskriptor jeder einzelnen Nachricht, um festzustellen, ob es sich um eine Anforderungsnachricht handelt; ist dies nicht der Fall, verwirft das Programm die Nachricht und zeigt eine Warnung an.

Für jede aus der Anforderungswarteschlange entfernte Anforderungsnachricht liest das Programm den Namen der in den Daten enthaltenen Warteschlange (die *Zielwarteschlange*) und öffnet diese Warteschlange mit dem MQOPEN-Aufruf ohne MQOO_INQ-Option. Das Programm verwendet dann den Aufruf MQINQ, um die Werte der Attribute *InhibitGet*, **CurrentQDepth** und **OpenInputCount** der Zielwarteschlange abzufragen.

Ist der MQINQ-Aufruf erfolgreich, reiht das Programm mit dem MQPUT1-Aufruf eine Antwortnachricht in die Empfangswarteschlange für Antworten ein. Diese Nachricht enthält die Werte der drei Attribute.

Ist der MQINQ-Aufruf nicht erfolgreich, reiht das Programm mit dem MQPUT1-Aufruf eine Berichtsnachricht in die Empfangswarteschlange für Antworten ein. Das Feld *Feedback* des Nachrichtendeskriptors dieser Berichtsnachricht enthält den Ursachencode, der (je nachdem, welcher Aufruf fehlgeschlagen ist) vom MQOPEN- bzw. MQINQ-Aufruf zurückgegeben wurde.

Nach dem MQINQ-Aufruf schließt das Programm die Zielwarteschlange mithilfe des Aufrufs MQCLOSE.

Wenn in der Anforderungswarteschlange keine Nachrichten mehr vorhanden sind, schließt das Programm diese Warteschlange und trennt die Verbindung zum Warteschlangenmanager.

Beispielprogramm zum Abfragen der Eigenschaften eines Nachrichtenhandles

AMQSIQMA ist ein C-Beispielprogramm für die Abfrage von Eigenschaften einer Nachrichtenkennung aus einer Nachrichtenwarteschlange und bietet ein Beispiel für die Verwendung des Aufrufs der API MQINQMP.

Dieses Beispiel erstellt ein Nachrichtenhandle und gibt es im Feld 'MsgHandle' der MQGMO-Struktur an. Dann ruft das Beispiel eine Nachricht ab, fragt alle Eigenschaften ab, mit denen das Nachrichtenhandle aufgefüllt wurde, und zeigt dies an.

```
C:\Programme\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

Publish/Subscribe-Beispielprogramme

Die Publish/Subscribe-Beispielprogramme veranschaulichen die Verwendung der Publish- und Subscribe-Funktionen in IBM MQ.

Drei Beispielprogramme in der Programmiersprache C veranschaulichen die Programmierung der Publish/Subscribe-Schnittstelle von IBM MQ. Einige in der Programmiersprache C geschriebene Beispielprogramme verwenden ältere Schnittstellen, zudem gibt es auch Java-Beispielprogramme. Die Java-Beispielprogramme verwenden die Publish/Subscribe-Schnittstelle von IBM MQ 'com.ibm.mq.jar' und die Publish/Subscribe-Schnittstelle von JMS in 'com.ibm.mqjms'. Die JMS-Beispielprogramm werden in diesem Abschnitt nicht behandelt.

C

Das Publisher-Beispielprogramm `amqspub` finden Sie im Beispielordner `C`. Führen Sie es mit einem beliebigen Namen als ersten Parameter aus, gefolgt von einem optionalen Warteschlangenmanagernamen. Beispiel: `amqspub mytopic QM3` . Es steht auch eine Clientversion namens `amqspubc` zur Verfügung. Wenn Sie die Clientversion ausführen möchten, lesen Sie zunächst die Informationen unter „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.

Der Publisher stellt eine Verbindung zum Standardwarteschlangenmanager her und antwortet mit der Ausgabe `target topic is mytopic` . Jede Zeile, die Sie ab sofort in dieses Fenster eingeben, wird in `mytopic` veröffentlicht .

Öffnen Sie ein weiteres Befehlsfenster in demselben Verzeichnis und führen Sie das Subskribentenprogramm `amqssub` aus. Geben Sie dabei denselben Themennamen und einen optionalen Warteschlangenmanagernamen an, zum Beispiel `amqssub mytopic QM3` .

Der Subskribent antwortet mit der Ausgabe `Calling MQGET : 30 seconds wait time` . Ab sofort erscheinen die Zeilen, die in im Publisher eingeben, in der Ausgabe des Subskribenten.

Starten Sie einen anderen Subskribenten in einem anderen Befehlsfenster und beobachten Sie, wie beide Subskribenten Veröffentlichungen erhalten.

Eine ausführliche Dokumentation der Parameter einschließlich Einstellungsoptionen finden Sie im Quellcode des Beispielprogramms. Die Werte für die Subskribentoptionsfelder werden im Abschnitt Options (MQLONG) erläutert.

Es gibt ein weiteres Subskribentenbeispielprogramm (amqssbx), das zusätzliche Subskriptionsoptionen als Befehlszeilenoptionen anbietet.

Geben Sie `amqssbx -d mysub -t mytopic -k` ein, um den Subskribenten mittels permanenter Subskriptionen aufzurufen, die nach Beendigung des Subskribenten beibehalten und aufbewahrt werden.

Testen Sie die Subskription, indem Sie einen anderen Artikel mithilfe des Publishers veröffentlichen. Warten Sie 30 Sekunden, bis der Subskribent beendet wird. Veröffentlichen Sie einige weitere Artikel unter demselben Thema. Starten Sie den Subskribenten neu. Der Artikel, der zuletzt bei nicht aktivem Subskribent veröffentlicht wurde, wird sofort nach Neustart des Subskribenten angezeigt.

Ältere C-Programme

Zudem gibt es eine Reihe von Beispielprogrammen in der Programmiersprache C zur Veranschaulichung von Befehlen in Warteschlangen. Einige dieser Beispielprogramme waren ursprünglich Bestandteil des MQ0C-SupportPacs. Aus Kompatibilitätsgründen werden die Funktionen der Beispielprogramme vollständig unterstützt.

Wir empfehlen Ihnen, die Schnittstelle der Befehle in Warteschlangen zu verwenden. Sie ist weitaus komplexer als die Publish/Subscribe-API und es gibt keinen zwingenden funktionsbedingten Grund, komplexe Befehle in Warteschlangen zu programmieren. Möglicherweise sagt Ihnen dieser Ansatz jedoch mehr zu, da Sie die Schnittstelle bereits verwenden oder Ihre Programmierumgebung die Erstellung komplexer Nachrichten und Aufrufe eines generischen MQPUT-Befehls erleichtert, statt verschiedene Aufrufe von MQSUB zu erstellen.

Die zusätzlichen Beispielprogramme finden Sie im Unterverzeichnis pubsub des Ordners `samples`.

In Tabelle 163 auf Seite 1156 sind sechs Beispielprogrammtypen aufgeführt.

Tabelle 163. Kategorien veralteter Publish/Subscribe-Beispielprogramme in Programmiersprache C

Kategorie	Programme	Kommentare
RFH1	`amqssr1a.c` `amqspr1a.c`	Einfaches Publish/Subscribe-Beispiel, erstellt mittels RFH1-Formatnachrichten.
RFH2	`amqssr2a.c` `amqspr2a.c`	Einfaches Publish/Subscribe-Beispiel, erstellt mittels RFH2-Formatnachrichten.
MQAI-Beispielprogramme	`amqsppca.c` `amqsspca.c`	Einfaches Publish/Subscribe-Beispielprogramm, erstellt mittels PCF-Befehlen und der MQAI-Befehlsschnittstelle.
MA0C-Ergebnisdienst unter Verwendung von RFH1	`amqsgama.c` `amqsresa.c`	Ergebnisdienst, erstellt mit RFH1-Headern 1. Erfordert die Definition der Warteschlangen in `amqsgama.tst` und `amqsresa.tst`. 2. `amqsresa` muss vor `amqsgama` gestartet werden.
MA0C-Ergebnisdienst unter Verwendung von RFH2	`amqsgr2a.c` `amqsrr2a.c`	Ergebnisdienst, erstellt mit RFH2-Headern 1. Erfordert die Definition der Warteschlangen in `amqsgama.tst` und `amqsresa.tst`. 2. `amqsresa` muss vor `amqsgama` gestartet werden.

Tabelle 163. Kategorien veralteter Publish/Subscribe-Beispielprogramme in Programmiersprache C (Forts.)

Kategorie	Programme	Kommentare
Beispielprogramm für Publish/Subscribe in einem Routing-Exit	`amqspsra.c`	Veranschaulicht die Änderung des Ziels der Warteschlange bzw. des Warteschlangenmanagers für eine Publish/Subscribe-Nachricht in einem Routing-Exit.

Beispielprogramm für Java

Das Java-Beispielprogramm `MQPubSubApiSample.java` kombiniert Publisher und Subskribenten in einem Programm. Seine Quellenklassendateien und kompilierten Klassendateien finden Sie im Beispielordner `wmqjava`.

Wenn Sie die Ausführung im Clientmodus wünschen, lesen Sie zunächst die Informationen unter „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123.

Führen Sie das Beispielprogramm über den Befehl Java in der Befehlszeile aus, wenn Sie eine konfigurierte Java-Umgebung verwenden. Sie können das Beispielprogramm auch über den IBM MQ Explorer-Eclipse-Arbeitsbereich ausführen. Dieser verfügt über eine bereits eingerichtete Java-Programmierungs-Workbench.

Einige der Eigenschaften des Beispielprogramms müssen unter Umständen geändert werden, um das Programm ausführen zu können. Geben Sie dazu die Parameter in der JVM an oder bearbeiten Sie die Quelle.

Befolgen Sie die Anweisungen in „Java-Beispielprogramm MQPubSubApiSample ausführen” auf Seite 1157, um das Beispielprogramm im Eclipse-Arbeitsbereich auszuführen.

Java-Beispielprogramm MQPubSubApiSample ausführen
In diesem Abschnitt wird erläutert, wie Sie 'MQPubSubApiSample' mithilfe des Java Development Tools auf der Eclipse-Plattform ausführen.

Vorbereitende Schritte

Öffnen Sie die Eclipse-Workbench. Erstellen Sie ein neues Arbeitsbereichsverzeichnis und wählen Sie es aus. Schließen Sie das Begrüßungsfenster.

Vor der Ausführung als Client führen Sie die Schritte unter „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1123 aus.

Informationen zu diesem Vorgang

Das Publish/Subscribe-Beispielprogramm von Java ist ein IBM MQ MQI client Java-Programm. Das Beispielprogramm wird ohne Änderung unter Verwendung eines Standardwarteschlangenmanagers ausgeführt, der auf Port 1414 empfangsbereit ist. Die Task beschreibt diesen einfachen Fall und erläutert, wie Sie Parameter bereitstellen und das Beispielprogramm an verschiedene IBM MQ-Konfigurationen anpassen. Das dargestellte Beispiel wird unter Windows ausgeführt. Die Dateipfade variieren je nach verwendetet Plattform.

Vorgehensweise

1. Importieren Sie die Java-Beispielprogramme.

a) Klicken Sie in der Workbench auf **Window** > **Open perspective** > **Other** > **Java** (Fenster > Perspektive öffnen > Sonstige) und klicken Sie anschließend auf **OK**.

b) Wechseln Sie zur Ansicht **Package Explorer** (Paketexplorer).

c) Klicken Sie mit der rechten Maustaste in den Leerraum in der Ansicht **Package Explorer** (Paketexplorer). Klicken Sie auf **New** > **Java project** (Neu > Java-Projekt).

d) Geben Sie im Feld **Project name** (Projektname) `MQ Java Samples` ein. Klicken Sie auf **Next** (Weiter).

e) Wechseln Sie in der Anzeige **Java Settings** (Java-Einstellungen) zur Registerkarte **Libraries** (Bibliotheken).

f) Klicken Sie auf **Add External JARs** (Externe JARs hinzufügen).

g) Navigieren Sie zu *MQ_INSTALLATION_PATH* `\java\lib` (hierbei steht *MQ_INSTALLATION_PATH* für den IBM MQ-Installationsordner) und wählen Sie `com.ibm.mq.jar` und `com.ibm.mq.jmqi.jar` aus.

h) Klicken Sie auf **Open** > **Finish** (Öffnen > Fertigstellen).

i) Klicken Sie in der Ansicht **Package Explorer** (Paketexplorer) mit der rechten Maustaste auf `src`.

j) Wählen Sie **Import...** > **Allgemein** > **Dateisystem** > **Weiter** > **Durchsuchen**... Navigieren Sie zum Pfad *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` . Dabei ist *MQ_INSTALLATION_PATH* das Installationsverzeichnis von IBM MQ .

k) Klicken Sie in der Anzeige **Import** (siehe Abbildung 130 auf Seite 1159) auf `samples` (wählen Sie nicht das Kontrollkästchen aus).

l) Wählen Sie `MQPubSubApiSample.java` aus. Das Feld **Into folder** (In Ordner) sollte `MQ Java Samples/src` enthalten. Klicken Sie auf **Fertigstellen**.

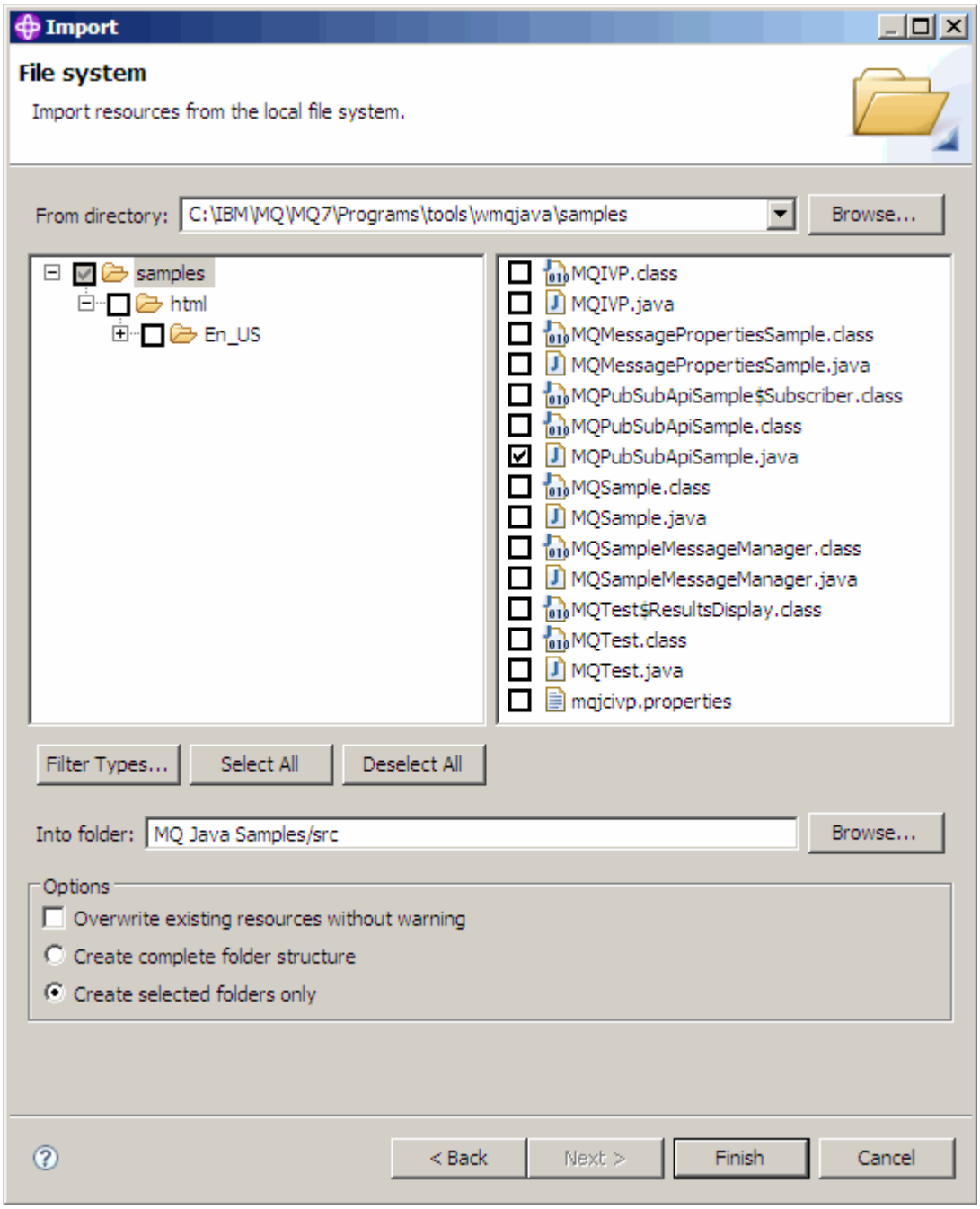


Abbildung 130. Dateisystemimport

2. Führen Sie das Publish/Subscribe-Beispielprogramm aus.

   Je nachdem, ob Sie die Standardparameter ändern müssen, kann das Programm auf zwei Arten ausgeführt werden.

   - So führen Sie das Programm aus, wenn keine Änderungen erforderlich sind:
     - Erweitern Sie im Hauptmenü des Arbeitsbereichs den Ordner `src`. Klicken Sie mit der rechten Maustaste auf **MQPubSubApiSample.java Run-as** > **1 Java Anwendung**
   - So führen Sie das Programm mit Parametern oder mit an Ihre Umgebung angepasstem Quellcode aus:
     - Öffnen Sie die Datei `MQPubSubApiSample.java` und prüfen Sie den Konstruktor `MQPubSubApiSample`.

– Ändern Sie die Attribute des Programms.

  Diese Attribute können mit dem -D JVM-Switch geändert werden oder indem Sie einen Standardwert für die System p-Eigenschaft angeben. Dazu muss der Quellcode bearbeitet werden.

  - topicObject
  - queueManagerName
  - subscriberCount

  Diese Attribute können nur durch Bearbeiten des Quellcodes im Konstruktor geändert werden.

  - Hostname
  - port
  - Kanal

  Zum Festlegen der System p-Eigenschaften codieren Sie einen Standardwert im Zugriffsobjekt, beispielsweise:

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

  Alternativ können Sie der JVM den Parameter über die Option `-D` bereitstellen, wie nachfolgend gezeigt:

a. Kopieren Sie den vollständigen Namen der gewünschten Systemeigenschaft (System.Property), beispielsweise: `com.ibm.mq.pubSubSample.queueManagerName`.
b. Klicken Sie im Arbeitsbereich mit der rechten Maustaste auf **Run** > **Open Run Dialog** (Ausführen > Ausführungsdialog öffnen). Klicken Sie doppelt auf Java Anwendung in **Anwendungen erstellen, verwalten und ausführen** und klicken Sie auf die Registerkarte **(x) = Argumente** .
c. Geben Sie im Fensterbereich **VM arguments:** (VM-Argumente) `-D` ein und fügen Sie den System.property-Namen `com.ibm.mq.pubSubSample.queueManagerName`, gefolgt von `=QM3`, ein. Klicken Sie auf **Apply** > **Run** (Anwenden > Ausführen).
d. Fügen Sie weitere Argumente als eine durch Kommas getrennte Liste oder als zusätzliche Zeilen im Teilfenster (ohne Kommatrennzeichen) hinzu.

   Beispiel: `-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3, -Dcom.ibm.mq.pubSubSample.subscriberCount=6`.

Das Publish Exit-Beispielprogramm

AMQSPSE0 ist das C-Beispielprogramm eines Exits zum Abfangen einer Veröffentlichung, bevor sie einem Subskribenten bereitgestellt wird. Der Exit kann beispielsweise die Nachrichtenheader, Nutzdaten oder das Ziel ändern oder verhindern, dass die Nachricht für einen Subskribenten veröffentlicht wird.

Gehen Sie zum Ausführen des Beispielprogramms wie folgt vor:

1. Konfigurieren Sie den Warteschlangenmanager:
   - Linux AIX Fügen Sie auf AIX and Linux-Systemen der Datei `qm.ini` eine Zeilengruppe wie die nachfolgende hinzu:

```
PublishSubscribe:
PublishExitPath=Module
PublishExitFunction=EntryPoint
```

     Dabei ist das Modul *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.
   - Windows Legen Sie unter Windows die entsprechenden Attribute in der Registry fest.
2. Stellen Sie sicher, dass IBM MQ auf das Modul zugreifen kann.
3. Starten Sie den Warteschlangenmanager neu, damit dieser die Konfiguration übernimmt.

4. Beschreiben Sie in dem zu verfolgenden Anwendungsprozess den Pfad, in den die Tracedateien geschrieben werden sollen. For example:

   - Linux AIX Stellen Sie auf AIX and Linux -Systemen sicher, dass das Verzeichnis `/var/mqm/trace` vorhanden ist, und exportieren Sie die Umgebungsvariable **MQPSE_TRACE_LOGFILE** :

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows Stellen Sie unter Windowssicher, dass das Verzeichnis `C:\temp` vorhanden ist, und legen Sie die Umgebungsvariable **MQPSE_TRACE_LOGFILE** fest:

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

Die Put-Beispielprogramme

Mit den Put-Beispielprogrammen werden Nachrichten mithilfe des MQPUT-Aufrufs in eine Warteschlange eingereiht.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Gestaltung des Put-Beispielprogramms

Das Programm verwendet den MQOPEN-Aufruf mit der Option MQOO_OUTPUT, um die Zielwarteschlange zum Einreihen von Nachrichten zu öffnen.

Wenn es die Warteschlange nicht öffnen kann, zeigt das Programm eine Fehlernachricht an, die den vom MQOPEN-Aufruf zurückgegebenen Ursachencode enthält. Um das Programm einfach zu halten, verwendet es für diesen und für nachfolgende MQI-Aufrufe die Standardwerte für die meisten Optionen.

Das Programm liest dann für jede Eingabezeile den Text in einen Puffer und verwendet den MQPUT-Aufruf, um eine Datagrammnachricht mit dem Text dieser Zeile zu erstellen. Das Programm wird fortgesetzt, bis es entweder das Ende der Eingabe erreicht oder bis der MQPUT-Aufruf fehlschlägt. Wenn das Programm das Ende der Warteschlange erreicht, schließt es die Warteschlange unter Verwendung des MQCLOSE-Aufrufs.

Put-Beispielprogramme ausführen

Beispielprogramme 'amqsput' und 'amqsputc' ausführen

ALW

Das Beispiel **amqqsput** ist das Programm zum Einreihen von Nachrichten unter Verwendung lokaler Bindungen und das Beispiel **amqsputc** ist das Programm zum Einreihen von Nachrichten unter Verwendung von Clientbindungen. Jedes dieser Programme verwendet folgende positionsgebundene Parameter:

1. Name der Zielwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (optional)

   Wenn kein Warteschlangenmanager angegeben ist, stellt **amqsput** eine Verbindung zum Standardwarteschlangenmanager und **amqsputc** eine Verbindung zu dem Warteschlangenmanager her, der durch die Umgebungsvariable *MQSERVER* oder die Clientkanaldefinitionsdatei angegeben ist.
3. Optionen zum Öffnen (optional)

   Sind keine Optionen zum Öffnen angegeben, verwendet das Beispielprogramm den Wert 8208. Dieser setzt sich aus diesen beiden Optionen zusammen:

   - MQOO_OUTPUT
   - MQOO_FAIL_IF_QUIESCING

4. Optionen zum Schließen (optional)

   Sind keine Optionen zum Schließen angegeben, verwendet das Beispielprogramm den Wert 0 (MQCO_NONE).

5. Name des Zielwarteschlangenmanagers (optional)

   Ist kein Zielwarteschlangenmanager angegeben, bleibt das Feld `ObjectQMgrName` im MQOD leer.

6. Name der dynamischen Warteschlange (optional)

   Ist kein Name einer dynamischen Warteschlange angegeben, bleibt das Feld `DynamicQName` im MQOD leer.

Verwenden Sie die folgenden Umgebungsvariablen, um Berechtigungsnachweise für die Authentifizierung beim Warteschlangenmanager anzugeben:

MQSAMP_USER_ID
Wird auf die für die Verbindungsauthentifizierung zu verwendende Benutzer-ID gesetzt, wenn Sie eine Benutzer-ID und ein Kennwort für die Authentifizierung beim Warteschlangenmanager verwenden wollen. Das Programm fordert zur Eingabe des Kennworts für die Benutzer-ID auf.

Linux V 9.4.0 AIX **MQSAMP_TOKEN**
Legen Sie einen Wert fest, der nicht leer ist, wenn Sie ein Authentifizierungstoken für die Authentifizierung beim Warteschlangenmanager bereitstellen wollen. Das Programm fordert zur Eingabe des Authentifizierungstoken auf. Authentifizierungstoken können nur vom **amqsputc** -Beispiel verwendet werden, das Clientbindungen verwendet.

Geben Sie zum Ausführen dieser Programme einen der folgenden Befehle ein:

- `amqsput` *`myqueue qmanagername`*
- `amqsputc` *`myqueue qmanagername`*

Dabei steht *myqueue* für den Namen der Warteschlange, in die die Nachrichten eingereiht werden, und *qmanagername* gibt den Warteschlangenmanager an, der Eigner von *myqueue* ist.

Beispielprogramm 'amq0put' ausführen

ALW

Die COBOL-Version hat keine Parameter. Sie stellt eine Verbindung zum Standardwarteschlangenmanager her und gibt beim Ausführen folgende Eingabeaufforderungen aus:

```
Please enter the name of the target queue
```

Es übernimmt die Eingaben von 'StdIn' und fügt der Zielwarteschlange alle Eingabezeilen hinzu. Eine leere Zeile zeigt an, dass keine weiteren Daten vorhanden sind.

AMQSPUT4-Beispielprogramm in C ausführen (IBM i)

IBM i

Das in C geschriebene Programm AMQSPUT4 steht nur für die IBM i-Plattform zur Verfügung und erstellt Nachrichten, indem es Daten aus der Teildatei einer Quellendatei liest.

Beim Programmstart müssen Sie den Namen der Datei als Parameter angeben. Die Struktur der Datei muss wie folgt aussehen:

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

Ein Eingabebeispiel für die Put-Beispielprogramme steht in der Bibliothek QMQMSAMP, Datei AMQSDATA, Teildatei PUT, zur Verfügung.

Anmerkung: Bei den Namen der Warteschlangen muss die Groß-/Kleinschreibung beachtet werden. Alle Namen der Warteschlangen, die vom Beispieldatei-Erstellungsprogramm AMQSAMP4 erstellt werden, sind in Großbuchstaben geschrieben.

Das C-Programm reiht Nachrichten in die Warteschlange ein, die in der ersten Zeile der Datei angegeben ist; Sie können die bereitgestellte Warteschlange SYSTEM.SAMPLE.LOCAL. verwenden. Das Programm reiht den Text jeder der folgenden Zeilen der Datei in separate Datagrammnachrichten ein und stoppt, wenn es am Ende der Datei eine leere Zeile liest.

Der Befehl für die Verwendung der Beispieldatendatei lautet:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

AMQ0PUT4-Beispielprogramm in COBOL ausführen (IBM i)

IBM i

Das COBOL-Programm AMQ0PUT4 steht nur auf der IBM i-Plattform zur Verfügung und erstellt Nachrichten aus über die Tastatur eingegebenen Daten.

Rufen Sie das Programm auf, um es zu starten, und geben Sie den Namen der Zielwarteschlange als Programmparameter an. Das Programm speichert die Tastatureingabe in einem Puffer und erstellt eine Datagrammnachricht für jede Textzeile. Das Programm stoppt, wenn Sie eine leere Zeile über die Tastatur eingeben.

Die Reference Message-Beispielprogramme

Die Reference Message-Beispielprogramme ermöglichen die Übertragung eines großen Objekts zwischen zwei Knoten (i.d.R. auf verschiedenen Systemen), ohne das Objekt in IBM MQ-Warteschlangen in den Quell- oder Zielknoten speichern zu müssen.

Mehrere Beispielprogramme veranschaulichen, wie Referenznachrichten in eine Warteschlange eingereiht, von Nachrichtenexits empfangen und aus einer Warteschlange abgerufen werden können. Zum Verschieben der Dateien verwenden die Beispielprogramme Referenznachrichten. Wenn Sie andere Objekte, z. B. Datenbanken, verschieben oder Sicherheitsprüfungen durchführen möchten, definieren Sie auf Grundlage des Beispielprogramms 'amqsxrm' einen eigenen Exit.

Die Version des zu verwendenden Reference Message-Exit-Beispielprogramms hängt von der Plattform ab, auf der der Kanal aktiv ist:

- Verwenden Sie auf allen Plattformen 'amqsxrma' auf der Sendeseite.
- Verwenden Sie 'amqsxrma' auf der Empfangsseite, wenn der Empfänger auf einer anderen Plattform als IBM i aktiv ist.
- IBM i Wenn der Empfänger unter IBM i ausgeführt wird, verwenden Sie 'amqsxrm4'.

.

IBM i *Hinweise für IBM i-Benutzer*

Wenn Sie eine Referenznachricht mit dem Beispielnachrichtenexit empfangen möchten, geben Sie im IFS-Stammdateisystem oder in einem beliebigen Unterverzeichnis eine Datei an, damit eine Datenstromdatei erstellt werden kann.

Das Beispielnachricht-Exit unter IBM i erstellt die Datei, konvertiert die Daten in EBCDIC und setzt die Codepage auf die Codepage Ihres Systems. Sie können diese Datei dann mit dem Befehl CPYFRMSTMF in das QSYS.LIB-Dateisystem kopieren. For example:

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
```

```
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

Mit dem Befehl CPYFRMSTMF können Sie die Datei nicht erstellen. Bevor Sie diesen Befehl ausführen, müssen Sie erst die Datei erstellen.

Wenn Sie eine Datei aus QSYS.LIB senden, sind keine Änderungen an den Beispielprogrammen erforderlich. Stellen Sie ein anderes Dateisystem verwenden, stellen Sie sicher, dass die im Feld 'CodedCharSetId' der MQRMH-Struktur angegebene CCSID mit den Massendaten übereinstimmt, die Sie senden.

Wenn Sie das integrierte Dateisystem verwenden, erstellen Sie Programmmodule mit der Optionsgruppe SYSIFCOPT(*IFSIO). Wenn Sie Datenbankdateien oder Datensatzdateien mit fester Länge verschieben möchten, definieren Sie Ihren eigenen Exit auf Grundlage des Beispielprogramms AMQSXRM4.

Zur Übertragung einer Datenbankdatei wird empfohlen, die Datei mithilfe des Befehls CPYTOSTMF in die IFS-Struktur zu konvertieren und dann die Referenznachricht mit angehängter IFS-Datei zu senden. Wenn Sie eine Datenbankdatei übertragen, indem Sie in IFS darauf verweisen, aber die Datei nicht in die IFS-Struktur konvertieren, müssen Sie den Teildateinamen angeben. Bei Verwendung dieser Methode wird keine Datenintegrität garantiert.

Referenznachrichtenbeispiel ausführen

In diesem Beispiel wird veranschaulicht, wie die Referenznachrichten-Beispielanwendung AMQSPRM unter AIX, Linux, and Windows oder AMQSPRMA unter IBM i ausgeführt wird. Dieses Beispiel zeigt, wie Referenznachrichten in eine Warteschlange eingereiht, von Nachrichtenexits empfangen und aus einer Warteschlange abgerufen werden können.

Die Referenznachrichtenbeispiele werden wie folgt ausgeführt:

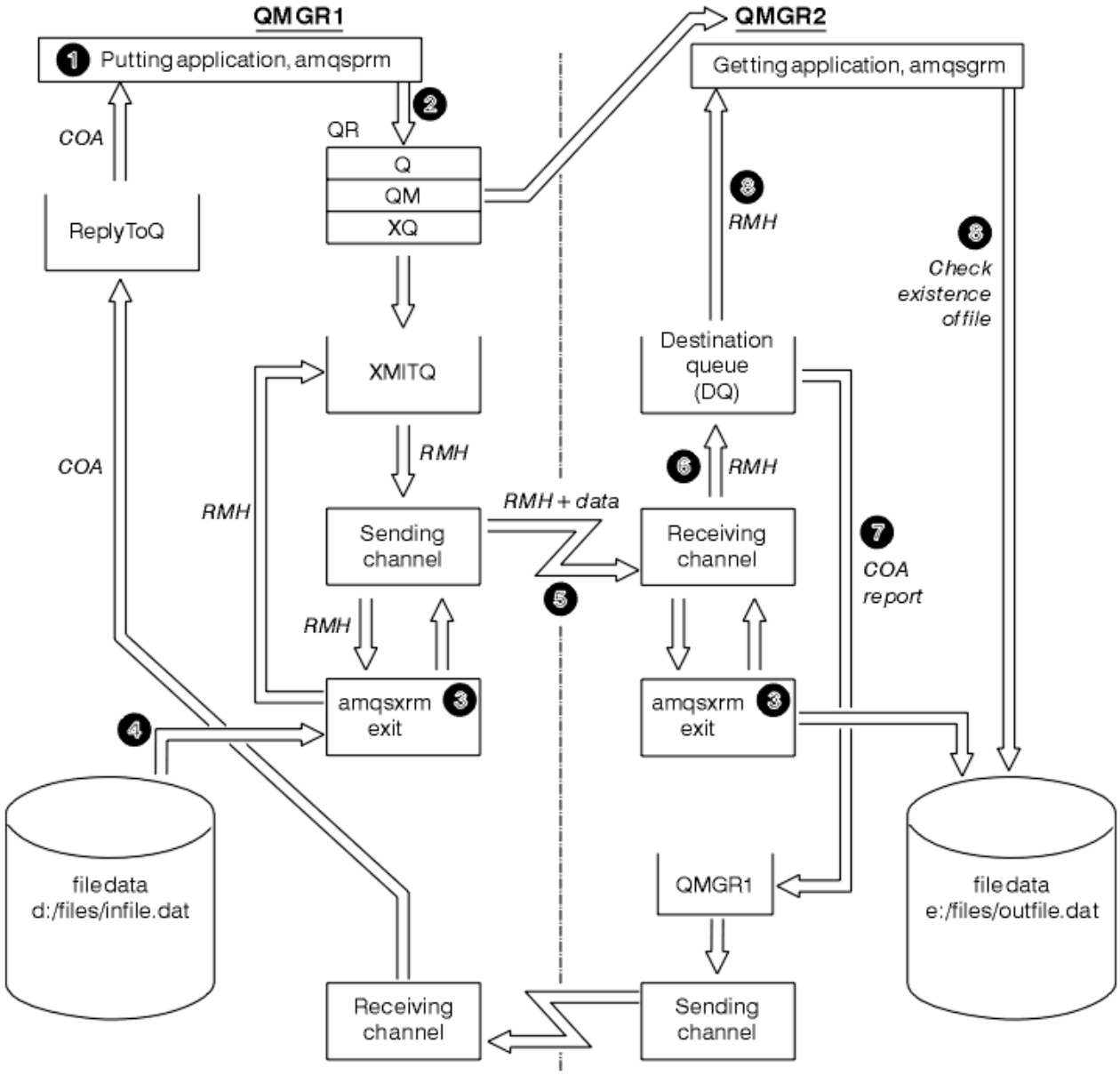


Abbildung 131. Referenznachrichtenbeispiel ausführen

1. Richten Sie die Umgebung zum Starten der Empfangsprogramme, Kanäle und Auslösemonitore ein und definieren Sie Ihre Kanäle und Warteschlangen.

   Bei der Erläuterung der Konfiguration des Referenznachrichtenbeispiels werden die sendende Maschine als MACHINE1 und deren Warteschlangenmanager als QMGR1 sowie die empfangende Maschine als MACHINE2 und deren Warteschlangenmanager als QMGR2 bezeichnet.

   Anmerkung: Mithilfe der folgenden Definitionen kann eine Referenznachricht erstellt werden, mit der eine Datei mit dem Objekttyp FLATFILE vom Warteschlangenmanager QMGR1 an den Warteschlangenmanager QMGR2 gesendet wird, und die Datei kann gemäß der Definition im AMQSPRM-Aufruf (bzw. AMQSPRMA unter IBM i) erneut erstellt werden. Die Referenznachricht (einschließlich der Dateidaten) wird mithilfe des Kanals CHL1 und der Übertragungswarteschlange XMITQ gesendet und in die Warteschlange DQ gestellt. Ausnahmeberichte sowie Berichte mit Bestätigung bei Eingang werden über den Kanal REPORT und die Übertragungswarteschlange QMGR1 an QMGR1 zurückgesendet.

   Die Anwendung, welche die Referenznachricht empfängt (AMQSGRM oder AMQSGRMA unter IBM i), wird über die Initialisierungswarteschlange INITQ und den Prozess PROC ausgelöst. Stellen Sie sicher, dass die CONNAME-Felder richtig festgelegt sind und dass das Feld MSGEXIT Ihre Verzeichnisstruktur entsprechend des Maschinentyps und des Installationspfads des IBM MQ-Produkts wiedergibt.

IBM i In den MQSC-Definitionen wurde ein AIX-Stil zur Definition der Exits verwendet; wenn Sie MQSC unter IBM i verwenden, müssen Sie die Definitionen entsprechend anpassen. Wichtiger Hinweis: Bei den Nachrichtendaten FLATFILE muss die Groß-/Kleinschreibung beachtet werden. Das Beispiel funktioniert nur, wenn der Name in Großbuchstaben geschrieben ist.

Auf Maschine MACHINE1, Warteschlangenmanager QMGR1

MQSC-Syntax

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i **IBM i-Befehlssyntax**

Anmerkung: Wenn Sie keinen Warteschlangenmanagernamen angeben, verwendet das System den Standardwarteschlangenmanager.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
            REPLACE(*YES) TRPTYPE(*TCP) +
            CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
            MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ   QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
            REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*RCVR) +
            MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ   QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
            REPLACE(*YES) RMTQNAME(DQ) +
            RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

Auf Maschine MACHINE2, Warteschlangenmanager QMGR2

MQSC-Syntax

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i **IBM i-Befehlssyntax**

Anmerkung: IBM i Wenn Sie unter IBM i keinen Warteschlangenmanagernamen angeben, verwendet das System den Standardwarteschlangenmanager.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
```

```
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. Nachdem die IBM MQ-Objekte erstellt wurden:
   a. Starten Sie je nach Plattform das Empfangsprogramm für die sendenden und empfangenden Warteschlangenmanager.
   b. Starten Sie die Kanäle CHL1 und REPORT.
   c. Starten Sie auf dem empfangenden Warteschlangenmanager den Auslösemonitor für die Initialisierungswarteschlange INITQ
3. Rufen Sie das Referenznachricht-Beispielprogramm AMQSPRM (bzw. AMQSPRMA unter IBM i) mit den folgenden Parametern aus der Befehlszeile auf:

   -m
   Name des lokalen Warteschlangenmanagers; standardmäßig ist dies der Standardwarteschlangenmanager

   -i
   Name und Speicherort der Quellendatei

   -o
   Name und Speicherort der Zieldatei

   -q
   Den Namen der Warteschlange

   -g
   Name des Warteschlangenmanagers mit der Warteschlange, die im Parameter -q definiert ist. Dies ist standardmäßig der Warteschlangenmanager, der im Parameter -m definiert ist.

   -t
   Objekttyp

   -w
   Warteintervall, also die Wartezeit für Ausnahmeberichte und Berichte mit Bestätigung bei Eingang des empfangenden Warteschlangenmanagers

   Wenn Sie beispielsweise das Beispiel mit den zuvor definierten Objekten verwenden möchten, geben Sie folgende Parameter an:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   Eine längere Wartezeit ermöglicht das Senden einer großen Datei über das Netz, bevor das Programm ein Überschreiten des Zeitlimits der Nachrichten meldet.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

   Benutzer von IBM i: 

Führen Sie unter IBM i folgende Schritte aus:

   a. Verwenden Sie folgenden Befehl:

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
```

```
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

Dies setzt voraus, dass die ursprüngliche Datei `rmsg1` im IFS-Verzeichnis `/refmsgs` gespeichert ist und Sie möchten, dass die Zieldatei im IFS-Verzeichnis `/refmsgs` auf dem Zielsystem `rmsgx` lauten soll.

b. Erstellen Sie mit dem Befehl CRTDIR ein eigenes Verzeichnis, anstatt das Stammverzeichnis zu verwenden.

c. Beachten Sie beim Aufrufen des Programms, das Daten einreiht, dass der Name der Ausgabedatei der IFS-Namenskonvention entsprechen muss; so wird zum Beispiel mit /TEST/DATEINAME eine Datei namens DATEINAME im Verzeichnis TEST erstellt.

Anmerkung:

IBM i Unter IBM i können Sie bei der Angabe von Parametern einen Schrägstrich (/) oder einen Gedankenstrich (-) verwenden. For example:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX Auf AIX and Linux-Plattformen müssen Sie das Verzeichnis der Zieldatei mit zwei umgekehrten Schrägstrichen (\\) statt mit nur einem bezeichnen. Der Befehl **amqsprm** sieht daher folgendermaßen aus:

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

Das Ausführen des Referenznachrichtenprogramms zum Einreihen bewirkt Folgendes:

- Die Referenznachricht wird in die Warteschlange QR im Warteschlangenmanager QMGR1 eingereiht.
- Die Quellendatei und der Quellenpfad lauten `d:\files\infile.dat` und sind auf dem System vorhanden, auf dem der Beispielbefehl ausgegeben wird.
- Handelt es sich bei der Warteschlange QR um eine ferne Warteschlange, wird die Referenznachricht an einen anderen Warteschlangenmanager auf einem anderen System gesendet, auf dem eine Datei mit dem Namen und Pfad `e:\files\outfile.dat` erstellt wird. Der Inhalt dieser Datei ist mit dem der Quellendatei identisch.
- 'amqsprm' wartet 30 Sekunden lang auf einen Bericht mit Bestätigung bei Eingang des Zielwarteschlangenmanagers.
- Der Objekttyp lautet `flatfile`. Daher muss der Kanal, mit dem Nachrichten aus der Warteschlange QR verschoben werden, dies im Feld *MsgData* angeben.

4. Wenn Sie Ihre Kanäle definieren, wählen Sie sowohl am sendenden als auch am empfangenden Enden das Nachrichtenexit 'amqsxrm' aus.

Windows Dies wird unter Windows wie folgt definiert:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX Dies wird unter AIX and Linux wie folgt definiert:

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

Geben Sie immer den vollständigen Pfadnamen an. Wenn Sie ihn weglassen, wird angenommen, dass sich das Programm in dem in der Datei `qm.ini` angegebenen Pfad befindet (bzw. unter IBM MQ for Windows in dem Pfad, der in der Registry angegeben ist).

5. Der Kanalexit liest den Referenznachrichten-Header und sucht nach der Datei, auf die der Header verweist.
6. Anschließend kann der Kanalexit die Datei segmentieren, bevor er sie zusammen mit dem Header über den Kanal sendet.

   Linux AIX Ändern Sie unter AIX and Linux den Gruppeneigner des Zielverzeichnisses in 'mqm', damit das Exit der Beispielnachricht die Datei in diesem Verzeichnis erstellen kann. Ändern Sie außerdem die Berechtigungen des Zielverzeichnisses, damit Mitglieder der Gruppe 'mqm' Schreibzugriff auf dieses Verzeichnis erhalten. Die Dateidaten werden nicht in den IBM MQ-Warteschlangen gespeichert.
7. Wenn das letzte Segment der Datei vom empfangenden Nachrichtenexit verarbeitet wird, wird die Referenznachricht in die von 'amqsprm' angegebene Zielwarteschlange eingereiht. Wenn diese Warteschlange ausgelöst wird (d. h., die Definition gibt die Warteschlangenattribute **Trigger**, **InitQ** und **Process** an), wird das vom Parameter PROC der Zielwarteschlange angegebene Programm ausgelöst. Das auszulösende Programm muss im Feld ApplId des Attributs **Process** definiert werden.
8. Wenn die Referenznachricht die Zielwarteschlange (DQ) erreicht (DQ), wird ein Bericht mit Bestätigung bei Eingang an die einreihende Anwendung (amqsprm) zurückgesendet.
9. Das Beispiel 'amqsgrm' zum Abrufen von Referenznachrichten ruft Nachrichten aus der in der Eingabeauslösenachricht angegebenen Warteschlange ab und prüft die Existenz der Datei.

Gestaltung des Put Reference Message-Beispielprogramms (amqsprma.c, AMQSPRM4)
In diesem Abschnitt wird das Put Reference Message-Beispielprogramm ausführlich erläutert.

Dieses Beispielprogramm erstellt eine Referenznachricht, die auf eine Datei verweist und sie in eine angegebene Warteschlange einreiht:

1. Das Beispielprogramm stellt mittels MQCONN eine Verbindung zu einem lokalen Warteschlangenmanager her.
2. Dann öffnet es (MQOPEN) eine Modellwarteschlange, mit der Berichtsnachrichten empfangen werden.
3. Das Beispielprogramm erstellt eine Referenznachricht, welche die Werte enthält, die zum Verschieben der Datei erforderlich sind, z. B. die Namen der Quellen- und Zieldatei und der Objekttyp.
   Beispiel: Das mit IBM MQ ausgelieferte Beispielprogramm erstellt eine Referenznachricht, um die Datei `d:\x\file.in` aus `QMGR1` in `QMGR2` zu senden und die Datei mittels folgender Parameter als `d:\y\file.out` erneut zu erstellen:

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   Dabei steht `QR` für die Definition einer fernen Warteschlange, die auf eine Zielwarteschlange unter `QMGR2` verweist.

   Anmerkung: Auf AIX and Linux-Plattformen müssen Sie das Verzeichnis der Zieldatei mit zwei umgekehrten Schrägstrichen (\\) statt mit nur einem bezeichnen. Der Befehl **amqsprm** sieht daher folgendermaßen aus:

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```
4. Die Referenznachricht wird (ohne Dateidaten) in die vom Parameter '/q' angegebene Warteschlange eingereiht. Handelt es sich dabei um eine ferne Warteschlange, wird die Nachricht in die entsprechende Übertragungswarteschlange eingereiht.
5. Das Beispielprogramm wartet für die Dauer des im Parameter '/w' angegebenen Zeitraums (Standardwert 15 Sekunden) auf Berichte mit Bestätigung bei Eingang, die gemeinsam mit Ausnahmeberichten an die dynamische Warteschlange zurückgesendet werden, die im lokalen Warteschlangenmanager erstellt wurde(QMGR1).

Entwurf des Reference Message Exit-Beispielprogramms (amqsxrma.c, AMQSXRM4)
Dieses Beispielprogramm erkennt Referenznachrichten mit einem Objekttyp, der dem Objekttyp im Benutzerdatenfeld für den Nachrichtenexit in der Kanaldefinition entspricht.

Diese Nachrichten werden wie folgt bearbeitet:

- Beim Sender- oder Serverkanal wird die angegebene Länge der Daten aus dem angegebenen Offset der angegebenen Datei in den restlichen Speicherbereich im Agentenpuffer nach der Referenznachricht kopiert. Wird das Ende der Datei nicht erreicht, wird die Referenznachricht wieder in die Übertragungswarteschlange eingereiht, nachdem das Feld *DataLogicalOffset* aktualisiert wurde.
- Wenn das Feld *DataLogicalOffset* auf dem Requester- oder Empfängerkanal null ist und die angegebene Datei nicht vorhanden ist, wird sie erstellt. Die auf die Referenznachricht folgenden Daten werden am Ende der angegebenen Datei hinzugefügt. Bezieht sich die Referenznachricht nicht auf die letzte angegebene Datei, wird sie gelöscht. Andernfalls wird sie ohne die angehängten Daten an den Kanalexit zurückgegeben, um in die Zielwarteschlange eingereiht zu werden.

Wenn für Sender- und Serverkanäle das Feld *DataLogicalLength* in der Referenznachricht null ist, wird der verbleibende Teil der Datei, von *DataLogicalOffset* bis zum Ende der Datei, entlang des Kanals gesendet. Ist der Wert nicht null, wird nur die angegebene Länge gesendet.

Wenn ein Fehler auftritt (das Beispielprogramm beispielsweise keine Datei öffnen kann), wird MQCXP *ExitResponse* wird auf MQXCC_SUPPRESS_FUNCTION gesetzt, sodass die Nachricht, die verarbeitet wird, in die Warteschlange für nicht zustellbare Nachrichten eingereiht und nicht in die Zielwarteschlange eingereiht wird. In MQCXP wird ein Rückkopplungscode zurückgegeben. *Feedback* und an die Anwendung zurückgegeben, die die Nachricht in das Feld *Feedback* des Nachrichtendeskriptors einer Berichtsnachricht eingereiht hat. Dies liegt daran, dass die einreihende Anwendung Ausnahmeberichte angefordert hat, indem sie MQRO_EXCEPTION im Feld *Report* des MQMD festgelegt hat.

Wenn die Codierung oder *CodedCharacterSetId* (CCSID) der Referenznachricht von der des Warteschlangenmanagers abweicht, wird die Referenznachricht in die lokale Codierung und CCSID konvertiert. Unser Beispielprogramm 'amqsprm' hat das Objektformat MQFMT_STRING. 'amqsxrm' konvertiert also die Objektdaten auf der Empfangsseite in die lokale CCSID, bevor die Daten in die Datei geschrieben werden.

Geben Sie das Format der zu übertragenden Datei nicht als MQFMT_STRING an, wenn die Datei Mehrbytezeichen enthält (zum Beispiel DBCS oder Unicode), da ein Mehrbytezeichen nicht aufgeteilt werden kann, wenn die Datei auf der Sendeseite segmentiert wird. Geben Sie zum Übertragen und Konvertieren einer solchen Datei ein anderes Format als MQFMT_STRING an, damit sie vom Referenznachricht-Exit nicht konvertiert wird, und konvertieren Sie die Datei nach beendeter Übermittlung auf der Empfangsseite.

Reference Message Exit-Beispielprogramm kompilieren
Verwenden Sie zum Kompilieren des Beispielprogramms 'Reference Message Exit' den Befehl für die Plattform, auf der IBM MQ installiert ist.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Kompilieren Sie 'amqsxrma' mit folgenden Befehlen:

unter AIX

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

unter IBM i

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

Anmerkung:

1. Um das Modul so zu erstellen, dass es das IFS-Dateisystem verwendet, fügen Sie die Option SYSIFCOPT(*IFSIO) hinzu.
2. Um das Programm mit Nicht-Thread-Kanälen verwenden zu können, erstellen Sie es mit folgendem Befehl: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)
3. Um das Programm mit Thread-Kanälen verwenden zu können, erstellen Sie es mit folgendem Befehl: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)

unter Linux


```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

unter Windows

IBM MQ stellt nun die Bibliothek 'mqm' mit Clientpaketen und mit Serverpaketen bereit. Das folgende Beispiel verwendet daher mqm.lib anstatt mqmvx.lib:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Zugehörige Konzepte

„Kanalexitprogramme schreiben” auf Seite 1016
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme schreiben.

Gestaltung des Get Reference Message-Beispielprogramms (amqsgrma.c, AMQSGRM4)
In diesem Abschnitt wird die Gestaltung des Get Reference Message-Beispielprogramms erläutert.

Die Programmlogik gliedert sich wie folgt:

1. Das Beispiel wird ausgelöst und extrahiert die Namen der Warteschlange und des Warteschlangenmanagers aus der Eingabeauslösenachricht.
2. Dann stellt es mit MQCONN eine Verbindung zum angegebenen Warteschlangenmanager her und öffnet mittels MQOPEN die angegebene Warteschlange.
3. Das Beispiel gibt MQGET mit einem Warteintervall von 15-Sekunden innerhalb einer Schleife aus, um Nachrichten aus der Warteschlange abzurufen.
4. Handelt es sich bei einer Nachricht um eine Referenznachricht, prüft das Beispiel die Existenz der Datei, die übertragen wurde.
5. Danach schließt es die Warteschlange und trennt die Verbindung zum Warteschlangenmanager.

Die Request-Beispielprogramme

Die Request-Beispielprogramme veranschaulichen die Client/Serververarbeitung. Die Beispiele agieren als die Clients, die Anforderungsnachrichten in eine Zielserverwarteschlange einreihen, die von einem Serverprogramm verarbeitet wird. Sie warten darauf, dass das Serverprogramm eine Antwortnachricht in die Empfangswarteschlange für Antworten einreiht.

Die Request-Beispielprogramme reihen mehrere Anforderungsnachrichten mit dem MQPUT-Aufruf in die Zielserverwarteschlange ein. Diese Nachrichten geben die lokale Warteschlange SYSTEM.SAMPLE.REPLY als die Empfangswarteschlange für Antworten an, die eine lokale oder eine ferne Warteschlange sein kann. Die Programme warten auf Antwortnachrichten und zeigen sie dann an. Antworten werden nur gesendet, wenn die Zielserverwarteschlange von einer Serveranwendung verarbeitet wird oder wenn zu diesem Zweck eine Anwendung ausgelöst wird (die Beispielprogramme 'Inquire', 'Set' und 'Echo' müssen ausgelöst werden). Das C-Beispielprogramm wartet eine Minute (die COBOL-Version 5 Minuten) auf den Eingang der ersten Antwort (damit die Serveranwendung ausgelöst werden kann) sowie 15 Sekunden für nachfolgende Antworten. Beide Beispielprogramme können jedoch beendet werden, ohne Antworten

erhalten zu haben. Die Namen der Request-Beispielprogramme sind unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114 aufgeführt.

Request-Beispielprogramme ausführen

Beispielprogramme 'amqsreq0.c', 'amqsreq' und 'amqsreqc' ausführen

Die C-Version des Programms verwendet drei Parameter:

1. Name der Zielserverwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (optional)
3. Antwortwarteschlange (optional)

Geben Sie zum Beispiel einen der folgenden Befehle ein:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

Dabei steht `myqueue` für den Namen der Zielserverwarteschlange, `qmanagername` für den Namen des Warteschlangenmanagers, der Eigner von `myqueue` ist, und `replyqueue` für den Namen der Antwortwarteschlange.

Wenn Sie den Namen des Warteschlangenmanagers übergehen, wird angenommen, dass der Standardwarteschlange der Eigner der Warteschlange ist. Wenn Sie den Namen der Antwortwarteschlange nicht angeben, wird die Standardantwortwarteschlange verwendet.

Beispielprogramm 'amq0req0.cbl' ausführen

Die COBOL-Version hat keine Parameter. Sie stellt eine Verbindung zum Standardwarteschlangenmanager her und gibt beim Ausführen folgende Eingabeaufforderungen aus:

```
Please enter the name of the target server queue
```

Das Programm fügt die Eingabe von 'StdIn' zeilenweise in die Zielserverwarteschlange ein. Dabei wird jede Textzeile für den Inhalt einer Anforderungsnachricht verwendet. Das Programm wird beendet, wenn eine Nullzeile gelesen wird.

Beispielprogramm AMQSREQ4 ausführen

Das C-Programm erstellt Nachrichten anhand der Daten von 'StdIn' (die Tastatur) mit einer leeren Eingabe zum Ende der Wartezeit. Das Programm verwendet bis zu drei Parameter: Name der Zielwarteschlange (erforderlich), Name des Warteschlangenmanagers (optional) und Name der Empfangswarteschlange für Antworten (optional). Wird kein Warteschlangenmanagername angegeben, wird der Standardwarteschlange verwendet. Wenn keine Empfangswarteschlange für Antworten angegeben wird, wird die Warteschlange SYSTEM.SAMPLE.REPLY verwendet.

Nachfolgend sehen Sie ein Beispiel zum Aufrufen des C-Beispielprogramms. Dabei wird die Empfangswarteschlange für Antworten angegeben, aber der Standardwarteschlangenmanager verwendet:

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

Anmerkung: Bei den Namen der Warteschlangen muss die Groß-/Kleinschreibung beachtet werden. Alle Namen der Warteschlangen, die vom Beispieldatei-Erstellungsprogramm AMQSAMP4 erstellt werden, sind in Großbuchstaben geschrieben.

Beispielprogramm AMQ0REQ4 ausführen

Das COBOL-Programm erstellt Nachrichten durch Akzeptieren der über die Tastatur eingegebenen Daten. Rufen Sie das Programm auf, um es zu starten, und geben Sie den Namen Ihrer Zielwarteschlange als Parameter an. Das Programm speichert die Tastatureingabe in einem Puffer und erstellt für jede Textzeile eine Anforderungsnachricht. Das Programm stoppt, wenn Sie eine leere Zeile über die Tastatur eingeben.

Request-Beispielprogramme mittels Triggering ausführen

Wenn das Beispiel mit Triggering und einem der Beispielprogramme 'Inquire', 'Set' oder 'Echo' verwendet wird, muss die Eingabezeile den Namen der Warteschlange enthalten, auf die das ausgelöste Programm zugreifen soll.

ALW *Request-Beispielprogramm mittels Triggering unter AIX, Linux, and Windows ausführen*

Starten Sie unter AIX, Linux, and Windows das Auslösemonitorprogramm RUNMQTRM in einer Sitzung und anschließend das Programm 'amqsreq' in einer anderen Sitzung.

So führen Sie die Beispielprogramme mittels Triggering aus:

1. Starten Sie das Auslösemonitorprogramm RUNMQTRM in einer Sitzung (die Initialisierungswarteschlange SYSTEM.SAMPLE.TRIGGER ist zur Verwendung verfügbar).
2. Starten Sie das Programm 'amqsreq' in einer anderen Sitzung.
3. Stellen Sie sicher, dass Sie eine Zielserverwarteschlange definiert haben.

   Als Zielserverwarteschlange zum Einreihen von Nachrichten im Request-Beispielprogramm stehen Ihnen folgende Beispielwarteschlangen zur Verfügung:

   - SYSTEM.SAMPLE.INQ - für das Inquire-Beispielprogramm
   - SYSTEM.SAMPLE.SET - für das Set-Beispielprogramm
   - SYSTEM.SAMPLE.ECHO - für das Echo-Beispielprogramm

   Diese Warteschlangen verfügen über den Auslösertyp FIRST, wenn also bereits Nachrichten in den Warteschlangen enthalten sind, bevor Sie das Request-Beispiel ausführen, werden Serveranwendungen nicht von den Nachrichten, die Sie senden, ausgelöst.
4. Stellen Sie sicher, dass Sie eine Warteschlange für die Beispielprogramme 'Inquire', 'Set' bzw. 'Echo' definiert haben.

Der Auslösemonitor ist dann bereit, wenn das Request-Beispielprogramm eine Nachricht sendet.

Anmerkung: Die mit mit RUNMQSC erstellten Beispielprozessdefinitionen und der Dateiauslöser 'amqscos0.tst' lösen die Beispielprogramme in C aus. Ändern Sie die Prozessdefinitionen in 'amqscos0.tst' und verwenden Sie RUNMQSC mit dieser aktualisierten Datei, wenn Sie die COBOL-Versionen verwenden möchten.

Abbildung 132 auf Seite 1174 veranschaulicht die gemeinsame Verwendung der Request- und Inquire-Beispielprogramme.

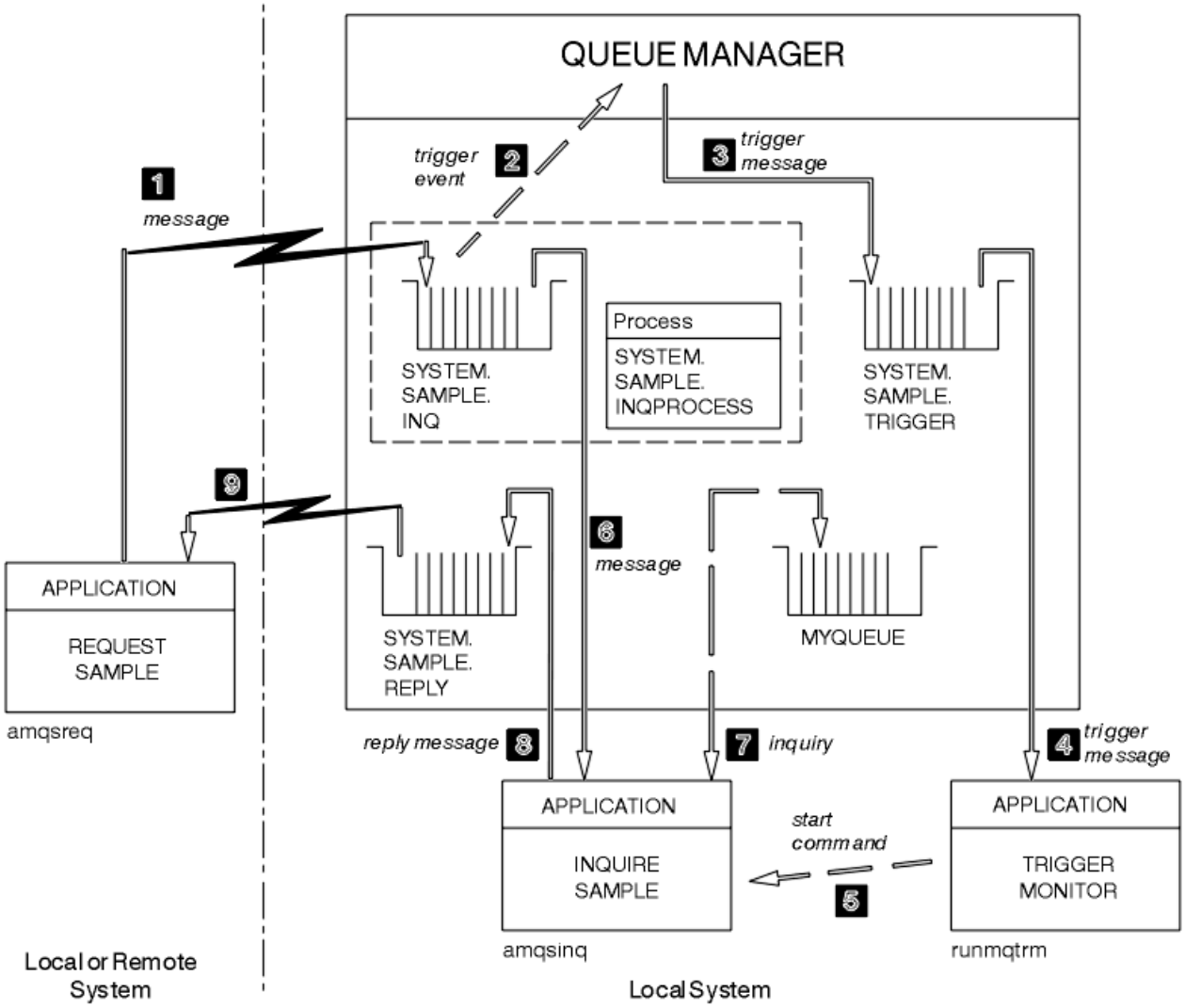


Abbildung 132. Request- und Inquire-Beispielprogramme mittels Triggering ausführen

In Abbildung 132 auf Seite 1174 reiht das Request-Beispielprogramm Nachrichten in die Zielserverwarteschlange SYSTEM.SAMPLE.INQ ein und das Inquire-Beispielprogramm fragt die Warteschlange MYQUEUE ab. Alternativ können Sie für das Inquire-Beispielprogramm eine der beim Ausführen von 'amqscos0.tst' definierten Beispielwarteschlangen oder jede andere definierte Warteschlange verwenden.

Anmerkung: Die Zahlen in Abbildung 132 auf Seite 1174 zeigen die Abfolge von Ereignissen.

So führen Sie die Request- und Inquire-Beispielprogramme mittels Triggering aus:

1. Stellen Sie sicher, dass die Warteschlangen, die Sie verwenden möchten, definiert sind. Führen Sie 'amqscos0.tst' zum Definieren der Beispielwarteschlangen aus und definieren Sie eine MYQUEUE-Warteschlange.
2. Führen Sie den Auslösemonitorbefehl RUNMQTRM aus:

   ```
   RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
   ```

3. Führen Sie das Request-Beispielprogramm aus:

   ```
   amqsreq SYSTEM.SAMPLE.INQ
   ```

   Anmerkung: Das Prozessobjekt definiert, was ausgelöst werden soll. Wenn Client und Server nicht auf derselben Plattform ausgeführt werden, muss in allen durch den Auslösemonitor gestarteten

Prozessen der Anwendungstyp (*ApplType*) definiert werden. Andernfalls verwendet der Server seine Standarddefinitionen (d. h. den Anwendungstyp, der normalerweise der Servermaschine zugeordnet ist) und verursacht einen Fehler.

Eine Liste mit den Anwendungstypen finden Sie unter ApplType.

4. Geben Sie den Namen der Warteschlange ein, die das Inquire-Beispielprogramm verwenden soll:

```
MYQUEUE
```

5. Geben Sie eine Leerzeile ein (um das Request-Programm zu beenden).
6. Das Request-Beispielprogramm zeigt daraufhin eine Nachricht mit den Daten an, die das Inquire-Programm aus der MYQUEUE-Warteschlange abgerufen hat.

Sie können mehrere Warteschlangen verwenden; geben Sie in diesem Fall die Namen der anderen Warteschlangen unter Schritt „4" auf Seite 1175 ein.

Weitere Informationen zum Triggering finden Sie unter „IBM MQ-Anwendungen durch Auslöser starten" auf Seite 913.

IBM i *Request-Beispielprogramm mittels Triggering unter IBM i ausführen*

Starten Sie unter IBM i den Trigger-Server AMQSERV4 des Beispielprogramms in einem Job und anschließend AMQSREQ4 in einem anderen Job. Der Trigger-Server ist dann bereit, wenn das Request-Beispielprogramm eine Nachricht sendet.

Anmerkung:

1. Die mit AMQSAMP4 erstellten Beispieldefinitionen lösen die C-Versionen der Beispielprogramme aus. Wenn Sie die COBOL-Versionen auslösen möchten, ändern Sie die Prozessdefinitionen SYSTEM.SAMPLE.ECHOPROCESS, SYSTEM.SAMPLE.INQPROCESS und SYSTEM.SAMPLE.SETPROCESS. Dazu können Sie entweder den Befehl CHGMQMPRC verwenden (ausführliche Informationen hierzu unter Details Change MQ Process (CHGMQMPRC)) verwenden oder Sie bearbeiten Ihre eigene AMQSAMP4-Version und führen diese aus.
2. Der Quellcode für AMQSERV4 steht nur in der Programmiersprache C zur Verfügung. Allerdings finden Sie eine kompilierte Version (für die COBOL-Beispielprogramme) in der Bibliothek QMQM.

Sie könnten Ihre Anforderungsnachrichten in diese Beispielserverwarteschlangen einreihen:

- SYSTEM.SAMPLE.ECHO (für die Echo-Beispielprogramme)
- SYSTEM.SAMPLE.INQ (für die Inquire-Beispielprogramme)
- SYSTEM.SAMPLE.SET (für die Set-Beispielprogramme)

Ein Ablaufdiagramm für das Programm SYSTEM.SAMPLE.ECHO wird in Abbildung 133 auf Seite 1177 dargestellt. Der Befehl zur Ausgabe der C-Programmanforderung an diesen Server (bei Verwendung der Beispieldatendatei) lautet:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

Anmerkung: Diese Musterwarteschlange verfügt über den Auslösertyp FIRST, wenn also bereits Nachrichten in der Warteschlange enthalten sind, bevor Sie das Request-Beispiel ausführen, werden Serveranwendungen nicht von den Nachrichten, die Sie senden, ausgelöst.

Wenn Sie weitere Beispiele ausprobieren möchten, versuchen Sie es mit folgenden Varianten:

- Verwenden Sie AMQSTRG4 (oder dessen Befehlszeilenäquivalent STRMQMTRM; ausführliche Informationen hierzu unter Start MQ Trigger Monitor (STRMQMTRM)) anstelle von AMQSERV4, um den Job zu übergeben. Potenzielle Jobübergabeverzögerungen könnten das Nachvollziehen der Vorgänge jedoch erschweren.
- Führen Sie die Beispielprogramme SYSTEM.SAMPLE.INQUIRE und SYSTEM.SAMPLE.SET aus. Die Befehle zur Ausgabe der C-Programmanforderungen an diese Server (bei Verwendung der Beispieldatendatei) lauten jeweils wie folgt:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

Diese Musterwarteschlangen verfügen ebenfalls über den Auslösertyp FIRST.

Gestaltung des Request-Beispielprogramms
Das Programm öffnet die Zielserverwarteschlange, damit Nachrichten eingereiht werden können. Es verwendet den MQOPEN-Aufruf mit der Option MQOO_OUTPUT. Wenn es die Warteschlange nicht öffnen kann, zeigt das Programm eine Fehlernachricht an, die den vom MQOPEN-Aufruf zurückgegebenen Ursachencode enthält.

Das Programm öffnet dann die Empfangswarteschlange für Antworten mit dem Namen SYSTEM.SAMPLE.REPLY, sodass es Antwortnachrichten abrufen kann. Dazu verwendet das Programm den MQOPEN-Aufruf mit der Option MQOO_INPUT_EXCLUSIVE. Wenn es die Warteschlange nicht öffnen kann, zeigt das Programm eine Fehlernachricht an, die den vom MQOPEN-Aufruf zurückgegebenen Ursachencode enthält.

Das Programm liest dann für jede Eingabezeile den Text in einen Puffer und verwendet den MQPUT-Aufruf, um eine Anforderungsnachricht mit dem Text dieser Zeile zu erstellen. Bei diesem Aufruf verwendet das Programm die Berichtsoption MQRO_EXCEPTION_WITH_DATA, um anzufordern, dass alle Berichtsnachrichten, die bezüglich der Anforderungsnachricht gesendet wurden, die ersten 100 Bytes der Nachrichtendaten enthalten. Das Programm wird fortgesetzt, bis es entweder das Ende der Eingabe erreicht oder bis der MQPUT-Aufruf fehlschlägt.

Dann verwendet das Programm den MQGET-Aufruf, um Antwortnachrichten aus der Warteschlange zu entfernen, und zeigt die in den Antworten enthalten Nachrichten an. Der MQGET-Aufruf verwendet die Optionen MQGMO_WAIT, MQGMO_CONVERT und MQGMO_ACCEPT_TRUNCATED. Bei der ersten Antwort beträgt das Warteintervall *WaitInterval* in der COBOL-Version 5 Minuten und in der C-Version 1 Minute (damit die Serveranwendung ausgelöst werden kann) und bei allen weiteren Antworten jeweils 15 Sekunden. Das Programm wartet diese Zeiträume ab, wenn in der Warteschlange keine Nachricht vorhanden ist. Wenn keine Nachricht eintrifft, bevor dieses Intervall abläuft, schlägt der Aufruf fehl und gibt den Ursachencode MQRC_NO_MSG_AVAILABLE zurück. Der Aufruf verwendet auch die Option MQGMO_ACCEPT_TRUNCATED_MSG, sodass Nachrichten abgeschnitten werden, die länger sind als die deklarierte Puffergröße.

Das Programm veranschaulicht, wie die Felder *MsgId* und *CorrelId* der MQMD-Struktur nach jedem MQGET-Aufruf gelöscht werden, da der Aufruf diese Felder auf die Werte setzt, die in der empfangenen Nachricht enthalten sind. Durch das Löschen dieser Felder rufen aufeinanderfolgende MQGET-Aufrufe die Nachrichten in der Reihenfolge ab, in der diese in der Warteschlange gehalten werden.

Das Programm wird fortgesetzt, bis der MQGET-Aufruf entweder den Ursachencode MQRC_NO_MSG_AVAILABLE zurückgibt oder der MQGET-Aufruf fehlschlägt. Wenn der Aufruf fehlschlägt, zeigt das Programm eine Fehlernachricht mit dem Ursachencode an.

Das Programm schließt dann die Zielserverwarteschlange und die Empfangswarteschlange für Antworten mithilfe des MQCLOSE-Aufrufs.

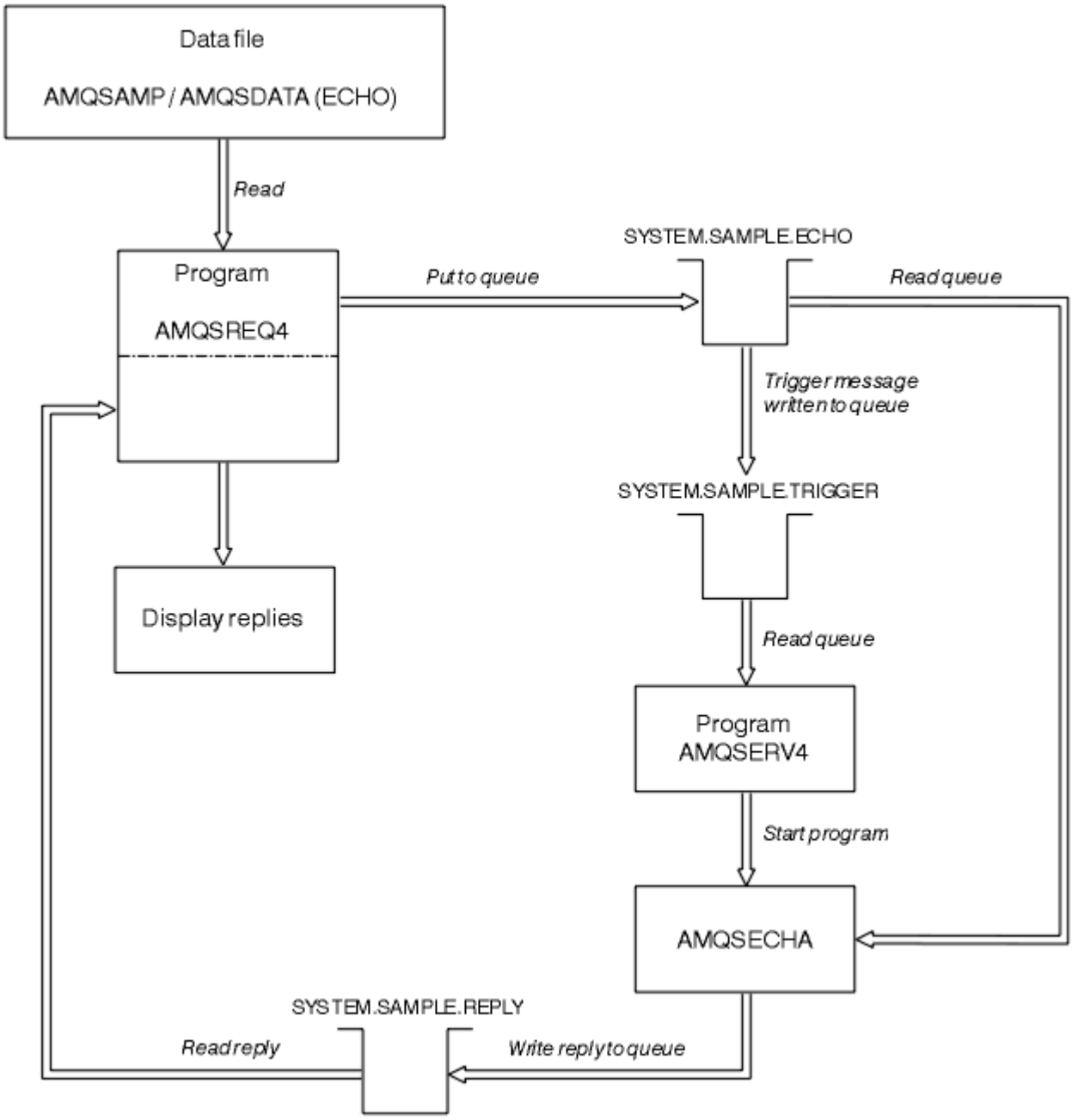


Abbildung 133. Ablaufdiagramm des IBM i Client/Server (Echo)-Beispielprogramms

Die Set--Beispielprogramme

Die Set-Beispielprogramme blockieren Put-Operationen in einer Warteschlange, indem sie mit dem MQSET-Aufruf das Attribut **InhibitPut** der Warteschlange ändern. Zudem erhalten Sie in diesem Abschnitt Informationen zur Gestaltung der Set-Beispielprogramme.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Die Programme sollen als Auslöserprogramme ausgeführt werden. Ihre einzige Eingabe ist also eine MQTMC2-Struktur (Auslösenachricht), die den Namen einer Zielwarteschlange mit Attributen enthält, die abgefragt werden. Die C-Version verwendet auch den Namen des Warteschlangenmanagers. Die COBOL-Version verwendet den Standardwarteschlangenmanager.

Damit der Auslöseprozess funktioniert, stellen Sie sicher, dass das gewünschte Set-Beispielprogramm durch Nachrichten ausgelöst wird, die in der Warteschlange SYSTEM.SAMPLE.SET eintreffen. Geben Sie

dazu den Namen des gewünschten Set-Beispielprogramms im Feld *ApplicId* der Prozessdefinition SYSTEM.SAMPLE.SETPROCESS ein. Die Beispielwarteschlange verfügt über den Auslösertyp FIRST; wenn also bereits Nachrichten in der Warteschlange enthalten sind, bevor Sie das Request-Beispiel ausführen, wird das Set-Beispiel nicht von den Nachrichten ausgelöst, die Sie senden.

Wenn Sie die Definition ordnungsgemäß festlegen müssen:

- ALW Starten Sie unter AIX, Linux, and Windows das Programm **runmqtrm** in einer Sitzung und anschließend das Programm 'amqsreq' in einer anderen Sitzung.
- IBM i Starten Sie unter IBM i das Programm AMQSERV4 in einer Sitzung und anschließend das Programm AMQSREQ4 in einer anderen Sitzung. Anstelle von AMQSERV4 könnten Sie AMQSTRG4 verwenden, doch potenzielle Jobübergabeverzögerungen könnten das Nachvollziehen der Vorgänge erschweren.

Verwenden Sie die Request-Beispielprogramme, um Anforderungsnachrichten - jede mit nur einem Warteschlangennamen - an die Warteschlange SYSTEM.SAMPLE.SET zu senden. Für jede Anforderungsnachricht senden die Set-Beispielprogramme eine Antwortnachricht mit einer Bestätigung, dass Put-Operationen in der angegebenen Warteschlange unterdrückt wurden. Die Antworten werden an die Empfangswarteschlange für Antworten gesendet, die in der Anforderungsnachricht angegeben ist.

Gestaltung des Set-Beispielprogramms

Das Programm öffnet die Warteschlange, die in der Auslösenachrichtstruktur genannt wurde, welche bei ihrem Start übergeben wurde. (Zur Verdeutlichung nennen wir diese die *Anforderungswarteschlange*.) Das Programm verwendet den MQOPEN-Aufruf, um diese Warteschlange für die gemeinsame Eingabe zu öffnen.

Mit dem MQGET-Aufruf werden Nachrichten aus der Warteschlange entfernt. Dieser Aufruf verwendet die Optionen MQGMO_ACCEPT_TRUNCATED_MSG und MQGMO_WAIT mit einem Warteintervall von 5 Sekunden. Das Programm prüft den Deskriptor jeder einzelnen Nachricht, um festzustellen, ob es sich um eine Anforderungsnachricht handelt; ist dies nicht der Fall, verwirft das Programm die Nachricht und zeigt eine Warnung an.

Für jede aus der Anforderungswarteschlange entfernte Anforderungsnachricht liest das Programm den Namen der in den Daten enthaltenen Warteschlange (die *Zielwarteschlange*) und öffnet diese Warteschlange mit dem MQOPEN-Aufruf und der MQOO_SET-Option. Anschließend setzt das Programm mithilfe des MQSET-Aufrufs den Wert des Attributs **InhibitPut** der Zielwarteschlange auf MQQA_PUT_INHIBITED.

Ist der MQSET-Aufruf erfolgreich, reiht das Programm mit dem MQPUT1-Aufruf eine Antwortnachricht in die Empfangswarteschlange für Antworten ein. Diese Nachricht enthält die Zeichenfolge `PUT inhibited`.

Wenn der MQOPEN-oder MQSET-Aufruf nicht erfolgreich ist, reiht das Programm mithilfe des MQPUT1 -Aufrufs eine `report` -Nachricht in die Empfangswarteschlange für Antworten ein. Das Feld *Feedback* des Nachrichtendeskriptors dieser Berichtsnachricht enthält den Ursachencode, der (je nachdem, welcher Aufruf fehlgeschlagen ist) vom MQOPEN- bzw. MQSET-Aufruf zurückgegeben wurde.

Nach dem MQSET-Aufruf schließt das Programm die Zielwarteschlange mithilfe des Aufrufs MQCLOSE.

Wenn in der Anforderungswarteschlange keine Nachrichten mehr vorhanden sind, schließt das Programm diese Warteschlange und trennt die Verbindung zum Warteschlangenmanager.

TLS-Beispielprogramm

AMQSSSLC ist ein C-Beispielprogramm, das veranschaulicht, wie mithilfe der MQCNO- und MQSCO-Strukturen TLS-Clientverbindungsinformationen im MQCONNX-Aufruf bereitgestellt werden. Damit kann eine MQI-Clientanwendung während der Laufzeit die Definition ihres Clientverbindungskanals und die TLS-Einstellungen ohne eine Definitionstabelle für Clientkanäle (CCDT) bereitstellen.

Bei Angabe eines Verbindungsnamens erstellt das Programm in einer MQCD-Struktur die Definition für einen Clientverbindungskanal.

Wenn der Stammname der Schlüsselrepositorydatei bereitgestellt wird, erstellt das Programm eine MQSCO-Struktur; wird zusätzlich noch eine OCSP-Responder-URL bereitgestellt, erstellt das Programm eine MQAIR-Struktur mit einem Authentifizierungsdatensatz.

Anschließend stellt das Programm über den MQCONNX-Aufruf eine Verbindung zum Warteschlangenmanager her. Es fragt den Namen des Warteschlangenmanagers ab, mit dem es verbunden ist, und gibt diesen aus.

Für dieses Programm ist eigentlich eine Verknüpfung als MQI-Clientanwendung vorgesehen. Sie kann jedoch als reguläre MQI-Anwendung verbunden sein. In diesem Fall stellt sie einfach eine Verbindung zu einem lokalen Warteschlangenmanager her und ignoriert die Clientverbindungsinformationen.

Wenn die Kennphrase für den Zugriff auf das Schlüsselrepository nicht verdeckt in einer Datei gespeichert ist, müssen Sie die Kennphrase für **amqssslc** angeben, wenn die Anwendung ausgeführt wird. Sie können die Kennphrase wie folgt angeben:

- **amqssslc** wird aufgefordert, zur Eingabe der Kennphrase aufzufordern, oder
- Verwenden Sie die Umgebungsvariable *MQKEYRPWD* oder
- Attribut **SSLKeyRepositoryPassword** in der Clientkonfigurationsdatei verwenden

Weitere Informationen zur Angabe des Kennworts für das Schlüsselrepository für IBM MQ MQI client -Anwendungen finden Sie unter Kennwort für das Schlüsselrepository für einen IBM MQ MQI client unter AIX, Linux, and Windowsangeben .

amqssslc akzeptiert die folgenden Parameter, die alle optional sind:

-m QmgrName
Der Name des Warteschlangenmanagers, zu dem eine Verbindung hergestellt werden soll.

-c ChannelName
Der Name des Kanals, der verwendet werden soll.

-x ConnName
Der Name der Serververbindung.

TLS-Parameter:

-k KeyReposFileName
Der Name der Schlüsselrepositorydatei. Wenn die Dateierweiterung nicht angegeben wird, wird angenommen, dass sie **.**kdbist. For example:

```
/home/user/client.kdb
C:\User\client.p12
```

-s CipherSpec
Die CipherSpec -Zeichenfolge des TLS-Kanals, die der **SSLCIPH** in der SVRCONN-Kanaldefinition auf dem Warteschlangenmanager entspricht.

-f
Gibt an, dass nur FIPS 140-2-zertifizierte Algorithmen verwendet werden dürfen.

-b WERT1[,WERT2...]
Gibt an, dass nur mit Suite B konforme Algorithmen verwendet werden dürfen. Dieser Parameter wird als eine durch Kommas getrennte Liste mit einem oder mehreren der folgenden Werte angegeben: NONE,128_BIT,192_BIT. Diese Werte haben dieselbe Bedeutung wie die für die Umgebungsvariable **MQSUITEB** und die entsprechende Einstellung **EncryptionPolicySuiteB** in der SSL-Zeilengruppe der Clientkonfigurationsdatei.

-p Policy
Gibt die Prüfrichtlinie für Zertifikate an, die verwendet werden soll. Folgende Werte sind möglich:

ANY
Es werden alle Zertifikatsprüfrichtlinien verwendet, die durch die Secure Sockets-Bibliothek unterstützt werden. Die Zertifikatskette wird akzeptiert, wenn eine der Richtlinien die Zertifikatskette als gültig bewertet. Diese Einstellung kann verwendet werden, um bei älteren digitalen Zertifika-

ten, die nicht den modernen Standards für Zertifikate entsprechen, ein Maximum an Abwärtskompatibilität zu erreichen.

RFC5280
Es wird nur die Zertifikatsprüfrichtlinie verwendet, die dem Standard RFC 5280 entspricht. Bei dieser Einstellung erfolgt eine strengere Prüfung als bei der Einstellung "ANY", es werden aber einige ältere digitale Zertifikate zurückgewiesen.

Der Standardwert ist ANY.

-l CertLabel
Die Zertifikatsbezeichnung, die für die sichere Verbindung verwendet werden soll.

Anmerkung: Der Wert muss in Kleinbuchstaben angegeben werden.

-w
Gibt an, dass **amqssslc** die Angabe der Kennphrase für das Schlüsselrepository anfordert.

-i
Gibt an, dass **amqssslc** zur Eingabe des Anfangsschlüssels auffordert, der zum Verschlüsseln der bereitgestellten Kennphrase für das Schlüsselrepository verwendet wird.

Geben Sie diese Option an, wenn eine Anfangsschlüsseldatei angegeben wurde, als die Kennphrase des Schlüsselrepositorys mit dem Dienstprogramm **runmqicred** verschlüsselt wurde.

Parameter für den OCSP-Zertifikatswiderruf:

-o URL
Die URL des OCSP-Responders.

Sie können auch eine der folgenden Umgebungsvariablen festlegen, um Berechtigungsnachweise für die Authentifizierung beim Warteschlangenmanager anzugeben:

MQSAMP_USER_ID
Wird auf die für die Verbindungsauthentifizierung zu verwendende Benutzer-ID gesetzt, wenn Sie eine Benutzer-ID und ein Kennwort für die Authentifizierung beim Warteschlangenmanager verwenden wollen. Das Programm fordert zur Eingabe des Kennworts für die Benutzer-ID auf.

Linux V 9.4.0 AIX **MQSAMP_TOKEN**
Legen Sie einen Wert fest, der nicht leer ist, wenn Sie ein Authentifizierungstoken für die Authentifizierung beim Warteschlangenmanager bereitstellen wollen. Das Programm fordert zur Eingabe des Authentifizierungstoken auf.

TLS-Beispielprogramm ausführen
Um das TLS-Beispielprogramm auszuführen, müssen Sie zuerst Ihre TLS-Umgebung einrichten. Anschließend wird das Beispielprogramm unter Angabe mehrerer Parameter über die Befehlszeile ausgeführt.

Informationen zu diesem Vorgang

Bei der folgenden Vorgehensweise wird das Beispielprogramm unter Verwendung persönlicher Zertifikate ausgeführt. Über Änderungen am Befehl können Sie beispielsweise CA-Zertifikate verwenden (Zertifikate einer Zertifizierungsstelle) und deren Status mithilfe eines OCSP-Responders überprüfen. Weitere Anweisungen finden Sie im Beispiel.

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager mit dem Namen QM1. Weitere Informationen finden Sie unter crtmqm.
2. Erstellen Sie für den Warteschlangenmanager ein Schlüsselrepository. Weitere Informationen finden Sie unter Schlüsselrepository unter AIX, Linux, and Windows einrichten.
3. Erstellen Sie für den Client ein Schlüsselrepository und weisen Sie diesem den Namen *clientkey.kdb* zu.

   Versteck das Kennwort für das Schlüsselrepository in einer Datei, wenn Sie das Schlüsselrepository erstellen

4. Erstellen Sie für den Warteschlangenmanager ein persönliches Zertifikat. Weitere Informationen finden Sie im Abschnitt Selbst signiertes persönliches Zertifikat unter AIX, Linux, and Windows erstellen.
5. Erstellen Sie für den Client ein persönliches Zertifikat.
6. Rufen Sie das persönliche Zertifikat aus dem Serverschlüsselrepository ab und fügen Sie es dem Client-Repository hinzu. Weitere Informationen finden Sie im Abschnitt Öffentlicher Teil eines selbst signierten Zertifikats aus einem Schlüsselrepository unter AIX, Linux, and Windows extrahieren und Zertifikat einer Zertifizierungsstelle (oder den öffentlichen Teil eines selbst signierten Zertifikats) in einem Schlüsselrepository auf AIX, Linux, and Windows-Systemen hinzufügen.
7. Rufen Sie das persönliche Zertifikat aus dem Clientschlüsselrepository ab und fügen Sie es dem Serverschlüsselrepository hinzu.
8. Erstellen Sie mit dem folgenden MQSC-Befehl einen Serververbindungskanal:

   ```
   DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
   ```

   Weitere Informationen finden Sie unter Serververbindungskanal.
9. Definieren und starten Sie auf dem Warteschlangenmanager einen Kanallistener. Weitere Informationen finden Sie unter DEFINE LISTENER und START LISTENER.
10. Führen Sie das Beispielprogramm mit dem folgenden Befehl aus:

    ```
    AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
    -k "C:\Programme\IBM\MQ\clientkey.kdb" -s TLS_RSA_WITH_AES_128_CBC_SHA256
    -o http://dummy.OCSP.responder
    ```

Ergebnisse

Das Beispielprogramm führt Folgendes aus:

1. Es stellt unter Verwendung der angegebenen Optionen eine Verbindung zu den angegebenen Warteschlangenmanagern bzw. zu dem Standardwarteschlangenmanager her.
2. Es öffnet den Warteschlangenmanager und fragt dessen Namen ab.
3. Es schließt den Warteschlangenmanager.
4. Es beendet die Verbindung zum Warteschlangenmanager.

Bei einer erfolgreichen Ausführung des Beispielprogramms wird eine Ausgabe ähnlich der folgenden angezeigt:

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Programme\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

Tritt bei der Ausführung des Beispielprogramms ein Fehler auf, wird eine entsprechende Fehlernachricht angezeigt; so erhalten Sie beispielsweise bei Angabe einer ungültigen OCSP-Responder-URL die folgende Nachricht:

```
MQCONNX ended with reason code 2553
```

Eine Liste der Ursachencodes finden Sie unter API-Beendigungs-und Ursachencodes.

Auslösebeispielprogramme

Bei der im Auslösebeispiel bereitgestellten Funktion handelt es sich um einen Teil der Funktion, die im Programm **runmqtrm** im Auslösemonitor bereitstellt wird.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1114.

Design des Auslösebeispiels

Das Auslösebeispielprogramm öffnet die Initialisierungswarteschlange mit dem MQOPEN-Aufruf unter Angabe der Option MQOO_INPUT_AS_Q_DEF. Es ruft mithilfe des MQGET-Aufrufs mit den Optionen MQGMO_ACCEPT_TRUNCATED_MSG und MQGMO_WAIT Nachrichten von der Initialisierungswarteschlange ab, die ein unbegrenztes Warteintervall angeben. Das Programm löscht vor jedem MQGET-Aufruf den Inhalt der Felder *MsgId* und *CorrelId*, um nacheinander Nachrichten abzurufen.

Nach dem Abruf einer Nachricht aus der Initialisierungswarteschlange überprüft das Programm die Größe der Nachricht, um sicherzustellen, dass sie dieselbe Größe wie eine MQTM-Struktur hat. Schlägt dieser Test fehl, wird vom Programm eine Warnung angezeigt.

Bei gültigen Auslösenachrichten kopiert das Auslösemuster Daten aus diesen Feldern: *ApplicId*, *EnvrData*, *Version* und *ApplType*. Bei den letzten beiden Feldern handelt es sich um numerische Felder, daher erstellt das Programm Ersatzzeichen zur Verwendung in einer MQTMC2-Struktur für IBM i-, AIX, Linux, and Windows-Systeme.

Das Auslösemuster gibt einen Startbefehl an die im Feld *ApplicId* der Auslösenachricht angegebene Anwendung aus und übergibt eine MQTMC2- oder MQTMC-Struktur (eine Zeichenversion der Auslösenachricht).

- ALW Unter AIX, Linux, and Windows wird das Feld *EnvData* als Erweiterung zur aufrufenden Befehlszeichenfolge verwendet.
- IBM i In IBM i wird es als Jobübergabeparameter verwendet, beispielsweise zur Angabe der Jobpriorität oder als Jobbeschreibung.

Am Ende schließt das Programm die Initialisierungswarteschlange.

Beenden der auslösenden Beispielprogramme unter IBM i

IBM i

Ein Auslösemonitorprogramm kann durch die sysrequest-Option 2 (ENDRQS) oder durch Sperren der Abrufe aus der Auslösewarteschlange beendet werden.

Bei Verwendung der Beispielauslösewarteschlange wird der folgende Befehl ausgegeben:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

Wichtig: Vor einer erneuten Initialisierung der Auslösefunktion für diese Warteschlange müssen Sie folgenden Befehl eingeben:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Auslösebeispielprogramme ausführen

Dieser Abschnitt enthält Informationen zum Ausführen von Auslösebeispielprogrammen.

Beispielprogramme 'amqstrg0.c', 'amqstrg' und 'amqstrgc' ausfühen

Für das Programm können zwei Parameter angegeben werden:

1. De Name der Initialisierungswarteschlange (obligatorisch)
2. Name des Warteschlangenmanagers (optional)

Wird kein Warteschlangenmanager angegeben, wird der Standardwarteschlangenmanager verwendet. Wenn Sie 'amqscos0.tst' ausgeführt haben, wurde eine Beispielinitialisierungswarteschlange mit dem Namen SYSTEM.SAMPLE.TRIGGER definiert; diese Warteschlangen können Sie bei der Ausführung dieses Programms verwenden.

Anmerkung: Die Funktion in diesem Beispiel ist ein Teil der vollständigen Triggering-Funktion, die im Programm 'runmqtrm' bereitgestellt wird.

Beispiel AMQSTRG4 ausführen

IBM i

Dies ist ein Auslösemonitor für die IBM i-Umgebung. Er übergibt für jede Anwendung, die gestartet werden soll, genau einen IBM i-Job. Dies bedeutet für jede Auslösenachricht zusätzlichen Verarbeitungsaufwand.

Für AMQSTRG4 (in QCSRC) können zwei Parameter angegeben werden: der Name der Initialisierungswarteschlange, die bedient werden muss, und der Name des Warteschlangenmanagers (optional). AMQSAMP4 (in QCLSRC) definiert eine Beispielinitialisierungswarteschlange (SYSTEM.SAMPLE.TRIGGER), die Sie beim Ausführen der Beispielprogramme verwenden können.

Bei Verwendung der Beispielauslösewarteschlange wird der folgende Befehl ausgegeben:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Alternativ können Sie den entsprechenden Befehlszeilenbefehl STRMQMTRM verwenden; Details finden Sie unter MQ-Auslösemonitor starten (STRMQMTRM).

Beispielprogramm AMQSERV4 ausführen

IBM i

Dies ist ein Auslöseserver für die IBM i-Umgebung. Für jede Auslösenachricht führt dieser Server den Startbefehl in seinem eigenen Job aus, um die angegebene Anwendung zu starten. Der Auslöseserver kann CICS-Transaktionen aufrufen.

Für AMQSERV4 können zwei Parameter angegeben werden: der Name der Initialisierungswarteschlange, die bedient werden muss, und der Name des Warteschlangenmanagers (optional). AMQSAMP4 definiert eine Beispiel-Initialisierungswarteschlange, SYSTEM.SAMPLE.TRIGGER, die Sie zum Ausprobieren von Beispielprogrammen verwenden können.

Unter Verwendung der Beispiel-Auslösewarteschlange ist folgender Befehl aufzurufen:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Konstruktion des Auslöseservers
Die Funktionsweise des Auslöseservers ähnelt der des Auslösemonitors, mit einigen Ausnahmen.

Die Funktionsweise des Auslöseservers ähnelt der des Auslösemonitors, nur gilt beim Auslöseserver Folgendes:

- Er ermöglicht sowohl MQAT_CICS- als auch MQAT_OS400-Anwendungen.
- IBM i Ruft IBM i -Anwendungen im eigenen Job auf (oder verwendet STRCICSUSR zum Starten von CICS -Anwendungen), anstatt einen IBM i -Job zu übergeben.
- Bei CICS-Anwendungen ersetzt er *EnvData* (beispielsweise zur Angabe der CICS-Region) über die Auslösenachricht im Befehl STRCICSUSR.
- Er öffnet die Initialisierungswarteschlange für die gemeinsam genutzte Eingabe, sodass viele Auslöseserver gleichzeitig ausgeführt werden können.

Anmerkung: Programme die von AMQSERV4 gestartet werden, dürfen keine MQDISC-Aufrufe verwenden, da der Auslöseserver sonst beendet wird. Wenn über AMQSERV4 gestartete Programme den MQCONN-Aufruf verwenden, wird der Ursachencode MQRC_ALREADY_CONNECTED zurückgegeben.

ALW *TUXEDO-Beispiele unter AIX, Linux, and Windows verwenden*

Diese Abschnitte enthalten Informationen zu den Put- und Get-Beispielprogrammen für TUXEDO sowie zum Erstellen der Serverumgebung für TUXEDO.

Vorbereitende Schritte

Sie müssen vor dem Ausführen dieser Beispielprogramme zunächst die Serverumgebung erstellen.

Informationen zu diesem Vorgang

Anmerkung: In diesem Abschnitt wird durchgängig ein Backslash (\) zur Aufteilung langer Befehle auf mehrere Zeile verwendet. Dieser Backslash gehört nicht zum Befehl und darf daher nicht eingegeben werden. Geben Sie die einzelnen Befehle jeweils in eine einzige Zeile ein.

ALW *Serverumgebung erstellen*

In diesen Abschnitten finden Sie Informationen zum Erstellen der Serverumgebung für IBM MQ auf verschiedenen Plattformen.

Vorbereitende Schritte

Es wird davon ausgegangen, dass eine funktionierende TUXEDO-Umgebung vorhanden ist.

AIX *Serverumgebung für AIX (32-Bit) erstellen*

Einrichtung der Serverumgebung für IBM MQ for AIX (32 Bit).

Vorgehensweise

1. Erstellen Sie ein Verzeichnis (beispielsweise APPDIR), in dem die Serverumgebung erstellt werden soll, und führen Sie alle Befehle in diesem Verzeichnis aus.
2. Exportieren Sie die folgenden Umgebungsvariablen, wobei TUXDIR für das TUXEDO-Stammverzeichnis und *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis steht, in dem IBM MQ installiert ist:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
```

3. Fügen Sie in der TUXEDO-Datei udataobj/RM die folgende Zeile hinzu:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. Führen Sie die folgenden Befehle aus:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
```

```
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. Bearbeiten Sie die Datei 'ubbstxcx.cfg' und fügen Sie ihr bei Bedarf Informationen wie den Maschinennamen, die Arbeitsverzeichnisse und den Warteschlangenmanager hinzu:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Erstellen Sie die TLOGDEVICE:

```
$tmadmin -c
```

   Daraufhin wird eine Eingabeaufforderung angezeigt. Geben Sie an dieser Eingabeaufforderung Folgendes ein:

```
> crdl -z /APPDIR/TLOG1
```

7. Starten Sie den Warteschlangenmanager:

```
$ strmqm
```

8. Starten Sie TUXEDO:

```
$ tmboot -y
```

Nächste Schritte

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

AIX *Serverumgebung für AIX (64-Bit) erstellen*

Einrichtung der Serverumgebung für IBM MQ for AIX (64 Bit).

Vorgehensweise

1. Erstellen Sie ein Verzeichnis (beispielsweise APPDIR), in dem die Serverumgebung erstellt werden soll, und führen Sie alle Befehle in diesem Verzeichnis aus.
2. Exportieren Sie die folgenden Umgebungsvariablen, wobei TUXDIR für das TUXEDO-Stammverzeichnis und *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis steht, in dem IBM MQ installiert ist:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Fügen Sie in der TUXEDO-Datei udataobj/RM die folgende Zeile hinzu:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. Führen Sie die folgenden Befehle aus:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
```

```
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. Bearbeiten Sie die Datei 'ubbstxcx.cfg' und fügen Sie ihr bei Bedarf Informationen wie den Maschinennamen, die Arbeitsverzeichnisse und den Warteschlangenmanager hinzu:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Erstellen Sie die TLOGDEVICE:

```
$tmadmin -c
```

   Daraufhin wird eine Eingabeaufforderung angezeigt. Geben Sie an dieser Eingabeaufforderung Folgendes ein:

```
> crdl -z /APPDIR/TLOG1
```

7. Starten Sie den Warteschlangenmanager:

```
$ strmqm
```

8. Starten Sie TUXEDO:

```
$ tmboot -y
```

Nächste Schritte

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

Windows *Serverumgebung für Windows (32-Bit) erstellen*
Einrichtung der Serverumgebung für IBM MQ for Windows (32 Bit).

Informationen zu diesem Vorgang

Anmerkung: Geben Sie in den nachfolgend als *VARIABLES* angegebenen Feldern die Verzeichnispfade an:

Tabelle 164. Felder, für die Verzeichnispfade angegeben werden müssen

Feld	Verzeichnispfad
MQMDIR	Der Verzeichnispfad, der bei der Installation von IBM MQ angegeben wurde, z. B. `g:\Program Files\IBM\MQ`.
TUXDIR	Der Verzeichnispfad, der bei der Installation von TUXEDO angegeben wurde, z. B. `f:\tuxedo`.
APPDIR	Der Verzeichnispfad, der für die Beispielanwendung verwendet werden soll, z. B. `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Abbildung 134. Beispieldatei 'ubbstxcn.cfg' für IBM MQ for Windows

Anmerkung: Geben Sie für *MachineName* und die Verzeichnispfade die entsprechenden Werte Ihrer Installation ein. Geben Sie außerdem für *MYQUEUEMANAGER* den Namen des Warteschlangenmanagers ein, zu dem eine Verbindung hergestellt werden soll.

Die Beispieldatei `ubbconfig` für IBM MQ for Windows ist in Abbildung 134 auf Seite 1187 aufgeführt. Sie wird im IBM MQ-Verzeichnis 'samples' als `ubbstxcn.cfg` bereitgestellt.

Die für IBM MQ for Windows bereitgestellte Beispiel-Makefile (siehe Abbildung 135 auf Seite 1188) heißt `ubbstxmn.mak`; sie ist im IBM MQ-Verzeichnis 'samples' enthalten.

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Abbildung 135. TUXEDO-Beispiel-Makefile für IBM MQ for Windows

Führen Sie zur Erstellung der Serverumgebung und Beispiele folgende Schritte aus.

Vorgehensweise

1. Erstellen Sie ein Anwendungsverzeichnis, in dem die Beispielanwendung erstellt werden soll; Beispiel:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Kopieren Sie die Beispieldateien aus dem IBM MQ-Verzeichnis mit den Beispielen in das Anwendungsverzeichnis:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Bearbeiten Sie diese Dateien, indem Sie die in Ihrer Installation verwendeten Verzeichnisnamen und Verzeichnispfade angeben.
4. Bearbeiten Sie die Datei `ubbstxcn.cfg` (siehe Abbildung 134 auf Seite 1187), indem Sie den Maschinennamen und den Warteschlangenmanager, zu dem eine Verbindung hergestellt werden soll, hinzufügen.
5. Fügen Sie in der TUXEDO-Datei *TUXDIR*`udataobj\rm` die folgende Zeile hinzu:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   Der neue Eintrag darf nur aus einer Zeile bestehen.
6. Setzen Sie die folgenden Umgebungsvariablen:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Erstellen Sie eine TLOG-Einheit (TLOGDEVICE) für TUXEDO.

Rufen Sie dazu `tmadmin -c` auf und geben Sie den folgenden Befehl ein:

```
crdl -z APPDIR\TLOG
```

8. Legen Sie *APPDIR* als aktuelles Verzeichnis fest und rufen Sie die Beispiel-Makefile (`amqstxmn.mak`) als externe Projekt-Makefile auf. Geben Sie hierzu beispielsweise in Microsoft Visual C++ den folgenden Befehl aus:

   ```
   msvc amqstxmn.mak
   ```

   Wählen Sie **build**, um alle Beispielprogramme zu erstellen.

Serverumgebung für Windows (64-Bit) erstellen
Einrichtung der Serverumgebung für IBM MQ for Windows (64 Bit).

Informationen zu diesem Vorgang

Anmerkung: Geben Sie in den nachfolgend als *VARIABLES* angegebenen Feldern die Verzeichnispfade an:

Tabelle 165. Felder, für die Verzeichnispfade angegeben werden müssen

Feld	**Verzeichnispfad**
MQMDIR	Der Verzeichnispfad, der bei der Installation von IBM MQ angegeben wurde, z. B. `g:\Program Files\IBM\MQ`.
TUXDIR	Der Verzeichnispfad, der bei der Installation von TUXEDO angegeben wurde, z. B. `f:\tuxedo`.
APPDIR	Der Verzeichnispfad, der für die Beispielanwendung verwendet werden soll, z. B. `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Abbildung 136. Beispieldatei 'ubbstxcn.cfg' für IBM MQ for Windows

Anmerkung: Geben Sie für *MachineName* und die Verzeichnispfade die entsprechenden Werte Ihrer Installation ein. Geben Sie außerdem für *MYQUEUEMANAGER* den Namen des Warteschlangenmanagers ein, zu dem eine Verbindung hergestellt werden soll.

Die Beispieldatei 'ubbconfig' für IBM MQ for Windows ist in Abbildung 136 auf Seite 1190 aufgeführt. Sie wird im IBM MQ-Verzeichnis 'samples' als `ubbstxcn.cfg` bereitgestellt.

Die für IBM MQ for Windows bereitgestellte Beispiel-Makefile (siehe Abbildung 137 auf Seite 1191) heißt `ubbstxmn.mak`; sie ist im IBM MQ-Verzeichnis 'samples' enthalten.

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Abbildung 137. TUXEDO-Beispiel-Makefile für IBM MQ for Windows

Führen Sie zur Erstellung der Serverumgebung und Beispiele folgende Schritte aus.

Vorgehensweise

1. Erstellen Sie ein Anwendungsverzeichnis, in dem die Beispielanwendung erstellt werden soll; Beispiel:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Kopieren Sie die Beispieldateien aus dem IBM MQ-Verzeichnis mit den Beispielen in das Anwendungsverzeichnis:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Bearbeiten Sie diese Dateien, indem Sie die in Ihrer Installation verwendeten Verzeichnisnamen und Verzeichnispfade angeben.
4. Bearbeiten Sie die Datei `ubbstxcn.cfg` (siehe Abbildung 136 auf Seite 1190), indem Sie den Maschinennamen und den Warteschlangenmanager, zu dem eine Verbindung hergestellt werden soll, hinzufügen.
5. Fügen Sie in der TUXEDO-Datei *TUXDIR*`udataobj\rm` die folgende Zeile hinzu:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   Der neue Eintrag darf nur aus einer Zeile bestehen.
6. Setzen Sie die folgenden Umgebungsvariablen:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Erstellen Sie eine TLOG-Einheit (TLOGDEVICE) für TUXEDO. Rufen Sie dazu `tmadmin -c` auf und geben Sie den folgenden Befehl ein:

```
crdl -z APPDIR\TLOG
```

8. Legen Sie *APPDIR* als aktuelles Verzeichnis fest und rufen Sie die Beispiel-Makefile (`amqstxmn.mak`) als externe Projekt-Makefile auf. Geben Sie hierzu beispielsweise in Microsoft Visual C++ den folgenden Befehl aus:

   ```
   msvc amqstxmn.mak
   ```

   Wählen Sie **build**, um alle Beispielprogramme zu erstellen.

ALW *Serverbeispielprogramm für TUXEDO*

Das Serverbeispielprogramm (amqstxsx) ist für die gemeinsame Ausführung mit dem Put-Beispielprogramm (amqstxpx.c) und dem Get-Beispielprogramm (amqstxgx.c) vorgesehen. Das Beispielserverprogramm wird beim Start von TUXEDO automatisch ausgeführt.

Anmerkung: Vor dem Start von TUXEDO müssen Sie Ihren Warteschlangenmanager starten.

Der Beispielserver stellt die beiden TUXEDO-Services MPUT1 und MGET1 bereit:

- Der Service MPUT1 wird vom Put-Beispiel gesteuert und verwendet den MQPUT1-Aufruf unter Synchronisationspunktsteuerung, um eine Nachricht innerhalb einer von TUXEDO gesteuerten Arbeitseinheit einzureihen. Dieser Service hat als Parameter den Namen des Warteschlangenmanagers und den Nachrichtentext, die beide vom Put-Beispiel bereitgestellt werden.
- Der Service MGET1 öffnet und schließt beim Empfang einer Nachricht die Warteschlange. Dieser Service hat als Parameter den Namen der Warteschlange und den Nachrichtentext, die beide vom Get-Beispiel bereitgestellt werden.

Fehlernachrichten, Ursachencodes und Statusnachrichten werden in die TUXEDO-Protokolldatei geschrieben.

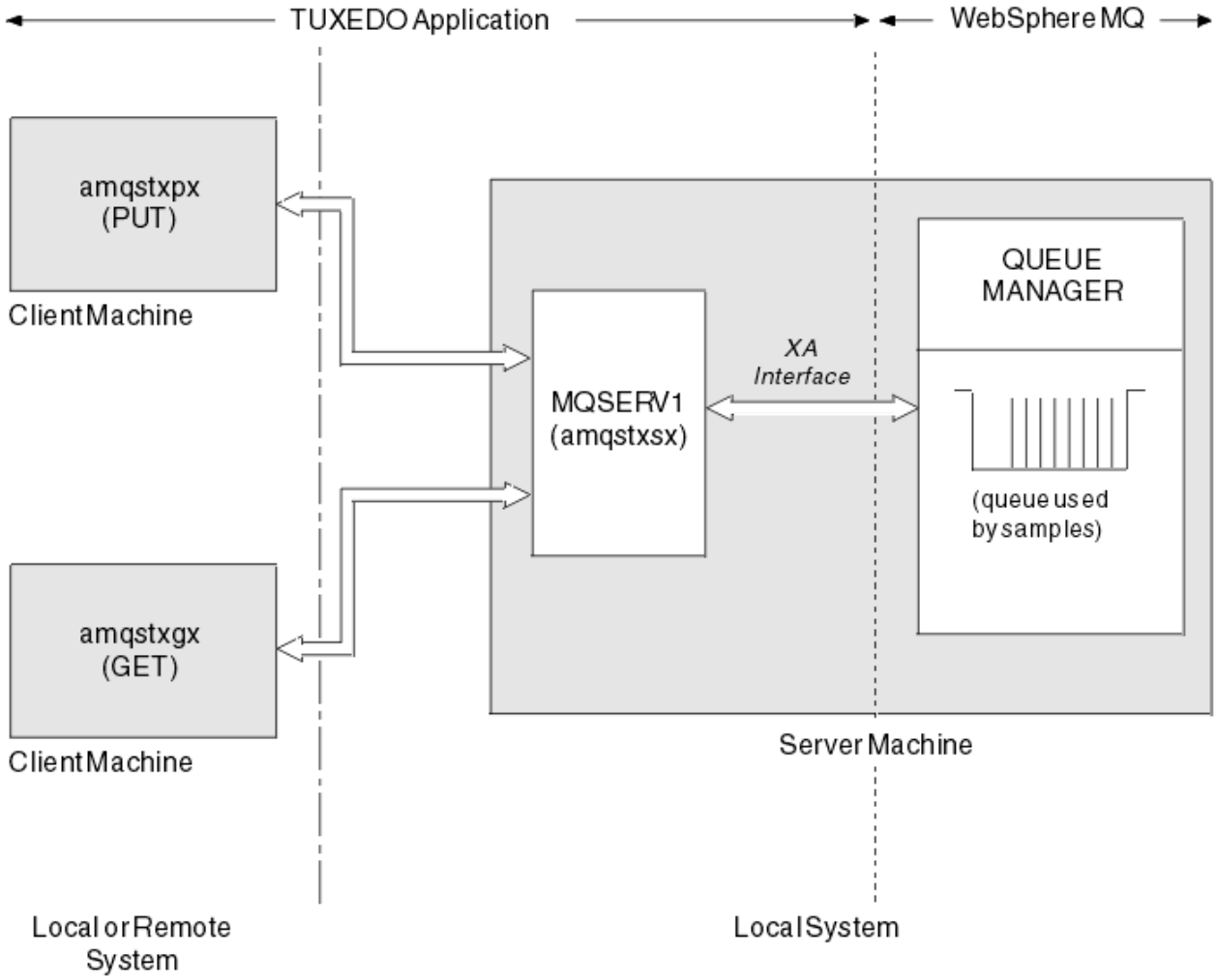


Abbildung 138. Zusammenspiel der TUXEDO-Beispiele

ALW *Das Put-Beispielprogramm für TUXEDO*

Mit diesem Beispiel können Sie eine Nachricht mehrmals (im Stapelbetrieb) in eine Warteschlange einreihen; damit wird die Verwendung der Synchronisationspunktsteuerung mit TUXEDO als Ressourcenmanager veranschaulicht.

Das Serverbeispielprogramm `amqstxsx` muss aktiv sein, damit das Put-Beispiel erfolgreich ausgeführt werden kann; das Serverbeispielprogramm stellt eine Verbindung zum Warteschlangenmanager her und verwendet die XA-Schnittstelle. Führen Sie das Beispiel aus, indem Sie Folgendes eingeben:

- `doputs -n queuename -b batchsize -c trancount -t message`

For example:

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

Mit diesem Befehl werden 30 Nachrichten (in sechs Stapeln mit je fünf Nachrichten) in die Warteschlange `myqueue` eingereiht. Bei Auftreten von Problemen wird ein Nachrichtenstapel zurückgesetzt, andernfalls wird er festgeschrieben.

Fehlernachrichten werden in die TUXEDO-Protokolldatei und in die Standard-Fehlerausgabe geschrieben, Ursachencodes werden in die Standard-Fehlerausgabe geschrieben.

ALW *Das Get-Beispielprogramm für TUXEDO*

Mit diesem Beispiel können Sie Nachrichten im Stapelbetrieb aus einer Warteschlange abrufen.

Das Serverbeispielprogramm `amqstxsx` muss zur erfolgreichen Ausführung des Get-Beispiels aktiv sein. Das Beispielprogramm stellt eine Verbindung zum Warteschlangenmanager her und verwendet die XA-Schnittstelle. Geben Sie zur Ausführung des Beispiels folgenden Befehl ein:

- `dogets -n queuename -b batchsize -c trancount`

For example:

- `dogets -n myqueue -b 6 -c 4`

Mit diesem Befehl werden 24 Nachrichten (in sechs Stapeln mit je vier Nachrichten) aus der Warteschlange `myqueue` abgerufen. Wird dieses Beispiel im Anschluss an das Put-Beispiel ausgeführt, mit dem 30 Nachrichten in die Warteschlange `myqueue` eingereiht werden, enthält die Warteschlange `myqueue` anschließend nur noch sechs Nachrichten. Die Anzahl der Stapel und die Stapelgröße können beim Einreihen und beim Abrufen der Nachrichten variieren.

Fehlernachrichten werden in die TUXEDO-Protokolldatei und in die Standard-Fehlerausgabe geschrieben, Ursachencodes werden in die Standard-Fehlerausgabe geschrieben.

Windows *SSPI-Sicherheitsexit unter Windows verwenden*

In diesem Abschnitt wird die Verwendung des SSPI-Kanalexitprogramms auf Windows-Systemen beschrieben. Der Exit-Code wird als Objektcode und als Quellcode bereitgestellt.

Objektcode

Die Objektcodedatei heißt `amqrspin.dll`. Sie wird für Server und Client als standardmäßiger Bestandteil von IBM MQ for Windows im Ordner *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME* installiert. Beispiel: `C:\Programme\IBM\MQ\exits\installation2`. Die Datei wird als Standardbenutzerexit geladen. Sie können den bereitgestellten Sicherheitskanalexit ausführen und in Ihrer Definition des Kanals Authentifizierungsservices verwenden.

Hierzu haben Sie die folgenden beiden Eingabemöglichkeiten:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

Soll Unterstützung für einen eingeschränkten Kanal bereitgestellt werden, müssen Sie für den Serververbindungskanal (SRVCONN) Folgendes eingeben:

```
SCYDATA('remote_principal_name')
```

Dabei wird *Name_des_fernen_Principals* im Format *DOMAIN**Benutzer* angegeben. Der sichere Kanal wird nur eingerichtet, wenn der Name des fernen Principals mit *Name_des_fernen_Principals* übereinstimmt.

Damit die bereitgestellten Kanalexitprogramme zwischen Systemen verwendet werden können, die innerhalb einer Kerberos-Sicherheitsdomäne betrieben werden, müssen Sie für den Warteschlangenmanager einen SPN (**servicePrincipalName**) erstellen.

Quellcode

Die Exitquellcodedatei heißt `amqsspin.c`. Sie befindet sich im Verzeichnis `C:\Programme\IBM\MQ\Tools\c\Samples`.

Wenn Sie den Quellcode ändern, muss die geänderte Quelle erneut kompiliert werden.

Sie wird auf dieselbe Weise wie jeder andere Kanalexit für die jeweilige Plattform kompiliert und verknüpft, außer dass beim Kompilieren auf SSPI-Header zugegriffen werden muss und beim Verknüpfen auf die SSPI-Sicherheitsbibliotheken sowie auf alle anderen empfohlenen Bibliotheken, die dazu gehören.

Vor der Ausführung des folgenden Befehls müssen Sie sicherstellen, dass `cl.exe`, die Visual C++-Bibliothek sowie der Ordner `include` in Ihrem Pfad verfügbar sind. For example:

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

Anmerkung: Der Quellcode bietet keine Möglichkeit für Tracing oder eine Fehlerbehandlung. Wenn Sie den Quellcode ändern und dann verwenden, fügen Sie Ihre eigenen Routinen für die Tracefunktion und die Fehlerbehandlung hinzu.

Beispiele unter Verwendung ferner Warteschlangen ausführen

Die Ausführung der Beispiele auf verbundenen Warteschlangenmanagern veranschaulicht die Steuerung ferner Warteschlangen.

Das Programm 'amqscos0.tst' stellt die lokale Definition einer fernen Warteschlange (SYSTEM.SAMPLE.REMOTE) bereit, die den fernen Warteschlangenmanager OTHER verwendet. Soll diese Beispieldefinition verwendet werden, müssen Sie OTHER in den Namen des zweiten Warteschlangenmanagers ändern, der verwendet werden soll. Darüber hinaus müssen Sie zwischen den beiden Warteschlangenmanagern einen Nachrichtenkanal einrichten; Informationen hierzu finden Sie unter Defining the channels.

Die Request-Beispielprogramme fügen den Namen des eigenen lokalen Warteschlangenmanagers in das Feld *ReplyToQMgr* der Nachrichten ein, die sie versenden. Die Inquire- und Set-Beispiele senden Antwortnachrichten an die Warteschlange und den Warteschlangenmanager, die in den Feldern *ReplyToQ* und *ReplyToQMgr* der Anforderungsnachrichten angegeben sind, die von ihnen verarbeitet werden.

Das Beispielprogramm 'Clusterwarteschlangenüberwachung' (AMQSCLM)

Dieses Beispiel leitet Nachrichten mithilfe integrierter IBM MQ-Funktionen für eine gleichmäßige Clusterauslastung an Warteschlangeninstanzen weiter, die mit konsumierenden Anwendungen verbunden sind. Durch diese automatische Weiterleitung wird verhindert, dass sich Nachrichten in einer Clusterwarteschlange ansammeln, die mit keiner konsumierenden Anwendung verbunden ist.

Übersicht

Sie können einen Cluster einrichten, der für ein und dieselbe Warteschlange auf verschiedenen Warteschlangenmanagern mehrere Definitionen enthält. Diese Konfiguration ermöglicht eine höhere Verfügbarkeit und einen besseren Lastausgleich. Allerdings verfügt IBM MQ über keine integrierte Funktion, die entsprechend dem Status der verbundenen Anwendungen eine dynamische Anpassung der Nachrichtenverteilung im Cluster ermöglicht. Um sicherzustellen, dass Nachrichten verarbeitet werden, muss eine konsumierende Anwendung daher immer mit jeder Instanz einer Warteschlange verbunden sein.

Das Beispielprogramm zur Überwachung von Clusterwarteschlangen überwacht den Status der verbundenen Anwendungen. Es passt die integrierte Konfiguration für den Lastausgleich dynamisch an, damit Nachrichten an Clusterwarteschlangeninstanzen weitergeleitet werden, die mit konsumierenden Anwendungen verbunden sind. In bestimmten Fällen kann mit diesem Programm die Notwendigkeit, dass eine konsumierende Anwendung immer mit jeder Instanz einer Warteschlange verbunden sein muss, in gewisser Weise umgangen werden. Darüber hinaus werden mit diesem Programm Nachrichten, die in eine Warteschlangeninstanz eingereiht wurden, die mit keiner konsumierenden Anwendung verbunden ist, erneut gesendet; dadurch können Nachrichten an einer konsumierenden Anwendung vorbei geleitet werden, die vorübergehend inaktiviert wurde.

Das Programm ist für eine Verwendung in Szenarios mit konsumierenden Anwendungen gedacht, die über einen längeren Zeitraum aktiv sind, nicht für Szenarios, in denen Anwendungen häufig verbunden und getrennt werden.

Das Beispielprogramm zur Überwachung von Clusterwarteschlangen ist das kompilierte ausführbare Programm der C-Beispieldatei `amqsclma.c`.

Weitere Informationen zu Clustern und Auslastung finden Sie im Abschnitt Cluster für das Workload-Management verwenden.

AMQSCLM: Design des Beispielprogramms sowie Planung für die Verwendung
Dieser Abschnitt enthält Informationen zur Funktionsweise des Beispielprogramms zur Überwachung von Clusterwarteschlangen, zu den Aspekten, die bei der Konfiguration der Systeme beachtet werden müssen, auf denen das Beispielprogramm ausgeführt werden soll, sowie zu den Änderungen, die am Quellcode des Beispielprogramms vorgenommen werden können.

Design

Mit dem Beispielprogramm zur Überwachung von Clusterwarteschlangen werden lokale Clusterwarteschlangen überwacht, die mit konsumierenden Anwendungen verbunden sind. Das Programm überwacht die vom Benutzer angegebenen Warteschlangen. Dabei kann ein bestimmter Warteschlangenmanager (beispielsweise APP.TEST01) oder ein generischer Warteschlangenname angegeben werden. Generische Namen müssen in einem PCF-kompatiblen Format (Programmable Command Format) angegeben werden. Beispiele für generische Namen sind APP.TEST* oder APP*.

Jeder Warteschlangenmanager im Cluster, der eine Instanz einer lokalen Warteschlange enthält, die überwacht werden soll, muss mit einer Instanz des Beispielprogramms zur Überwachung von Clusterwarteschlangen verbunden sein.

Dynamisches Nachrichtenrouting

Das Beispielprogramm zur Überwachung von Clusterwarteschlangen stellt anhand des Werts von **IPPROCS** (Anzahl der Anwendungen, die gerade zum Abruf von Nachrichten mit der Warteschlange verbunden sind) fest, ob konsumierende Anwendungen mit der Warteschlange verbunden sind. Bei einem Wert größer als 0 ist die Warteschlange mit mindestens einer konsumierenden Anwendung verbunden (Solche Warteschlangen sind aktiv). Der Wert 0 gibt an, dass keine konsumierenden Programme mit der Warteschlange verbunden sind (solche Warteschlangen sind inaktiv).

Bei Clusterwarteschlangen mit mehreren Instanzen in einem Cluster ermittelt IBM MQ anhand der Eigenschaft **CLWLPRTY** (zur Priorisierung bei der Clusterauslastung) der einzelnen Warteschlangeninstanzen die Instanzen, an die Nachrichten gesendet werden sollen. IBM MQ sendet Nachrichten an die verfügbaren Instanzen einer Warteschlange mit dem höchsten Wert für **CLWLPRTY**.

Das Beispielprogramm zur Überwachung von Clusterwarteschlangen aktiviert eine Clusterwarteschlange, indem es die Eigenschaft **CLWLPRTY** dieser Warteschlange lokal auf 1 setzt; soll eine Clusterwarteschlange inaktiviert werden, wird die Eigenschaft **CLWLPRTY** für sie auf 0 gesetzt.

Mit der Clustertechnologie von IBM MQ wird die aktualisierte Eigenschaft **CLWLPRTY** einer Clusterwarteschlange an alle betreffenden Warteschlangenmanager im Cluster weitergegeben. Beispiel:

- An einen Warteschlangenmanager, der mit einer Anwendung verbunden ist, die Nachrichten in die Warteschlange einreiht.
- An einen Warteschlangenmanager, der in demselben Cluster über eine lokale Warteschlange desselben Namens verfügt.

Die Weitergabe erfolgt über die Warteschlangenmanager mit vollständigem Repository im Cluster. Neue Nachrichten für die Clusterwarteschlange werden an die Instanzen mit dem höchsten Wert für **CLWLPRTY** im Cluster übertragen.

Übertragung eingereihter Nachrichten

Die dynamische Änderung des Werts von **CLWLPRTY** beeinflusst die Weiterleitung neuer Nachrichten. Auf Nachrichten, die sich bereits in einer Warteschlange befinden, die mit keinen konsumierenden Anwendungen verbunden ist, oder auf Nachrichten, die über den Mechanismus für den Lastausgleich übertragen wurden, bevor ein geänderter **CLWLPRTY**-Wert im Cluster weitergegeben wurde, hat diese dynamische Änderung keine Auswirkung. Dies bedeutet, dass die Nachrichten in den inaktiven Warteschlangen verbleiben und daher von keiner konsumierenden Anwendung verarbeitet werden. Um dieses Problem zu beheben, kann das Beispielprogramm zur Überwachung von Clusterwarteschlangen Nachrichten aus einer lokalen Warteschlange, die mit keinen konsumierenden Anwendungen verbunden ist, abrufen und

an ferne Instanzen derselben Warteschlange senden, die mit konsumierenden Anwendungen verbunden sind.

Das Beispielprogramm zur Überwachung von Clusterwarteschlangen überträgt Nachrichten aus einer inaktiven lokalen Warteschlange an eine oder mehrere ferne aktive Warteschlangen, indem es Nachrichten mithilfe des **MQGET**-Aufrufs abruft und mithilfe des **MQPUT**-Aufrufs in dieselbe Clusterwarteschlange einreiht. Diese Übertragung hat zur Folge, dass die IBM MQ-Funktion für einen Lastausgleich im Cluster eine andere Zielinstanz auswählt, deren CLWLPRTY-Wert höher ist als der der lokalen Warteschlangeninstanz. Nachrichtenpersistenz und -kontext werden bei der Nachrichtenübertragung beibehalten, die Nachrichtenreihenfolge sowie alle eventuell angegebenen Bindungsoptionen hingegen nicht.

Planung

Wenn sich die Verbindung von konsumierenden Anwendungen ändert, modifiziert das Beispielprogramm zur Überwachung von Clusterwarteschlangen die Clusterkonfiguration entsprechend. Änderungen werden von den Warteschlangenmanagern, auf denen das Beispielprogramm Warteschlangen überwacht, an die Manager mit vollständigem Repository im Cluster übertragen. Die Warteschlangenmanager mit vollständigem Repository verarbeiten die Konfigurationsaktualisierungen und senden sie erneut an alle betreffenden Warteschlangenmanager im Cluster. Zu diesen betreffenden Warteschlangenmanagern gehören Warteschlangenmanager, die über eigene Clusterwarteschlangen desselben Namens verfügen (auf denen eine Instanz des Beispielprogramms zur Überwachung von Clusterwarteschlangen aktiv ist) sowie alle Warteschlangenmanager, auf denen eine Anwendung innerhalb der letzten 30 Tage eine Clusterwarteschlange geöffnet hat, um Nachrichten einzureihen.

Änderungen werden asynchron im Cluster verarbeitet. Daher enthalten Warteschlangenmanager im Cluster nach jeder Änderung unter Umständen vorübergehend unterschiedliche Konfigurationen.

Das Beispielprogramm zur Überwachung von Clusterwarteschlangen eignet sich nur für Systeme, auf denen Anwendungen nicht allzu häufig verbunden oder getrennt werden, auf denen also beispielsweise konsumierende Anwendungen mit langer Laufzeit zum Einsatz kommen. Wird dieses Beispielprogramm zur Überwachung von Systemen verwendet, auf denen konsumierende Anwendungen nur kurze Zeit verbunden werden, kann die anfallende Latenz bei der Verteilung von Konfigurationsaktualisierungen dazu führen, dass die Warteschlangenmanager im Cluster nicht richtig darüber informiert sind, welche Warteschlangen mit konsumierenden Anwendungen verbunden sind. Diese Latenz kann zu falsch weitergeleiteten Nachrichten führen.

Bei der Überwachung einer großen Anzahl von Warteschlangen können schon wenige Änderungen an den verbundenen Konsumenten aller Warteschlangen zu einem erhöhten Datenaufkommen in Zusammenhang mit der Clusterkonfiguration im Cluster führen. Ein erhöhtes Datenaufkommen in Zusammenhang mit der Clusterkonfiguration kann eine übermäßige Belastung einer oder mehrerer der folgenden Warteschlangenmanager verursachen:

- Warteschlangenmanager, auf denen das Beispielprogramm zur Überwachung von Clusterwarteschlangen aktiv ist
- Die Warteschlangenmanager mit einem vollständigen Repository
- Warteschlangenmanager, die mit einer Anwendung verbunden sind, die Nachrichten in die Warteschlange einreiht
- Warteschlangenmanager, die in demselben Cluster über eine lokale Warteschlange desselben Namens verfügen

Die Prozessorauslastung auf den Warteschlangenmanagern mit vollständigem Repository muss überprüft werden. Eine zusätzliche Prozessorauslastung lässt sich an den Nachrichtenübertragungen in der Warteschlange SYSTEM.CLUSTER.COMMAND.QUEUE für Übertragungen an das vollständige Repository ablesen. Wenn sich in dieser Warteschlange mehr und mehr Nachrichten ansammeln, deutet dies darauf hin, dass die Warteschlangenmanager mit vollständigem Repository die Änderungen an der Clusterkonfiguration im System nicht mehr handhaben können.

Wenn das Beispielprogramm zur Überwachung von Clusterwarteschlangen für die Überwachung einer großen Anzahl von Warteschlangen eingesetzt wird, fällt für das Beispielprogramm und den Warteschlangenmanager ein bestimmter Arbeitsaufwand an. Dieser Arbeitsaufwand fällt immer an, auch wenn es

keine Änderungen an den verbundenen Konsumenten gibt. Es besteht die Möglichkeit, durch eine Änderung des Arguments **-i** die Prozessorauslastung durch das Beispielprogramm auf dem lokalen System zu reduzieren, indem für die Frequenz des Überwachungszyklus ein niedrigerer Wert angegeben wird.

Damit eine übermäßige Aktivität festgestellt werden kann, werden vom Beispielprogramm zur Überwachung von Clusterwarteschlangen die durchschnittliche Verarbeitungszeit pro Abfrageintervall, die verstrichene Verarbeitungszeit sowie die Anzahl der Konfigurationsänderungen gemeldet. Diese Berichte werden alle 30 Minuten oder immer nach 600 Abfrageintervallen (je nachdem, welches der beiden Intervalle kürzer ist) in der Informationsnachricht **CLM0045I** übermittelt.

Voraussetzungen für die Verwendung der Überwachung von Clusterwarteschlangen

Für das Beispielprogramm zur Überwachung von Clusterwarteschlangen gelten Voraussetzungen und Einschränkungen. Sie können den bereitgestellten Quellcode modifizieren, um einige dieser Einschränkungen in Bezug auf die Verwendungsmöglichkeit des Programms zu ändern. Die möglichen Änderungen werden anhand von Beispielen in diesem Abschnitt veranschaulicht.

- Das Beispielprogramm zur Überwachung von Clusterwarteschlangen ist für die Überwachung von Warteschlangen gedacht, die mit konsumierenden Anwendungen verbunden bzw. nicht verbunden sind. Wenn auf dem System konsumierende Anwendungen vorhanden sind, die häufig verbunden und getrennt werden, kann das Beispielprogramm unter Umständen ein hohes Datenaufkommen in Zusammenhang mit der Clusterkonfiguration im gesamten Cluster verursachen. Die Leistung der Warteschlangenmanager im Cluster kann dadurch beeinträchtigt werden.
- Das Beispielprogramm zur Überwachung von Clusterwarteschlangen ist von der System- und Clustertechnologie von IBM MQ abhängig. Die Anzahl der Warteschlangen, die überwacht werden, die Häufigkeit der Überwachungszyklen und der Statusänderungen der einzelnen Warteschlangen wirken sich auf das System insgesamt aus. Diese Faktoren müssen bei der Auswahl der Warteschlangen, die überwacht werden sollen, und bei der Einstellung des Abfrageintervalls für die Überwachung berücksichtigt werden.
- Eine Instanz des Beispielprogramms zur Überwachung von Clusterwarteschlangen muss mit jedem Warteschlangenmanager im Cluster verbunden sein, der eine Instanz einer Warteschlange enthält, die überwacht werden soll. Warteschlangenmanager im Cluster, die über keine Warteschlangen verfügen, müssen nicht mit dem Beispielprogramm verbunden werden.
- Das Beispielprogramm zur Überwachung von Clusterwarteschlangen muss mit der entsprechenden Berechtigung ausgeführt werde, damit ein Zugriff auf alle erforderlichen IBM MQ-Ressourcen möglich ist. Beispiel:
  - Auf den Warteschlangenmanager, zu dem eine Verbindung hergestellt werden soll
  - Auf die Warteschlange SYSTEM.ADMIN.COMMAND.QUEUE
  - Auf alle Warteschlangen, die bei einer Nachrichtenübertragung überwacht werden sollen
- Der Befehlsserver muss für jeden Warteschlangenmanager aktiv sein, der mit dem Beispielprogramm zur Überwachung von Clusterwarteschlangen verbunden ist.
- Für jede Instanz des Beispielprogramms zur Überwachung von Clusterwarteschlangen ist die exklusive Nutzung einer lokalen Warteschlange (keiner Clusterwarteschlange) auf dem Warteschlangenmanager erforderlich, mit dem es verbunden ist. Diese lokale Warteschlange wird zur Steuerung des Beispielprogramms und zum Empfang von Antwortnachrichten für Anfragen verwendet, die an den Befehlsserver des Warteschlangenmanagers gesendet wurden.
- Alle Warteschlangen, die von einer einzigen Instanz des Beispielprogramms zur Überwachung von Clusterwarteschlangen überwacht werden, müssen sich in demselben Cluster befinden. Verfügt ein Warteschlangenmanager über Warteschlangen in mehreren Clustern und sollen diese Warteschlangen überwacht werden, sind mehrere Instanzen des Beispielprogramms erforderlich. Für jede Instanz muss eine lokale Warteschlange für die Steuerung und für den Empfang von Antwortnachrichten vorhanden sein.
- Alle Warteschlangen, die überwacht werden sollen, müssen sich in einem einzigen Cluster befinden. Warteschlangen, die für die Verwendung einer Clusternamensliste konfiguriert sind, werden nicht überwacht.

- Die Aktivierung der Übertragung von Nachrichten aus inaktiven Warteschlangen ist optional. Sie gilt für alle Warteschlangen, die von der Instanz des Beispielprogramms zur Überwachung von Clusterwarteschlangen überwacht werden. Ist eine Aktivierung der Nachrichtenübertragung nur für einige der überwachten Warteschlangen erforderlich, werden zwei Instanzen des Beispielprogramms zur Überwachung von Clusterwarteschlangen benötigt. Dabei ist bei einem Beispielprogramm die Nachrichtenübertragung aktiviert, bei dem anderen inaktiviert. Für jede Instanz des Beispielprogramms muss eine lokale Warteschlange für die Steuerung und für den Empfang von Antwortnachrichten vorhanden sein.
- Die IBM MQ-Funktion für einen Lastausgleich im Cluster sendet standardmäßig Nachrichten an Instanzen von Clusterwarteschlangen, die sich auf demselben Warteschlangenmanager befinden, mit dem auch eine Anwendung verbunden ist, die Nachrichten einreiht. Diese Einstellung muss in den folgenden Fällen inaktiviert werden, solange die lokale Warteschlange inaktiv ist:
  - Anwendungen, die Nachrichten einreihen, stellen eine Verbindung zu Warteschlangenmanagern her, die über Instanzen einer inaktiven Warteschlange verfügen, die überwacht werden.
  - Eingereihte Nachrichten werden aus inaktiven Warteschlangen in aktive Warteschlangen übertragen.

  Die lokale Einstellung für den Lastausgleich für die Warteschlange kann statisch inaktiviert werden, indem der Wert für `CLWLUSEQ` auf `ANY` gesetzt wird. Bei dieser Konfiguration werden Nachrichten, die in lokale Warteschlangen eingereiht werden, an lokale und ferne Warteschlangeninstanzen verteilt, um eine gleichmäßige Lastverteilung zu erreichen; dies ist auch der Fall, wenn lokale konsumierende Anwendungen vorhanden sind. Alternativ kann das Beispielprogramm zur Überwachung von Clusterwarteschlangen so konfiguriert werden, dass der Wert von **CLWLUSEQ** vorübergehend auf `ANY` gesetzt wird, solange die Anwendung mit keinem Konsumenten verbunden ist; in diesem Fall werden nur lokale Nachrichten an lokale Instanzen einer Warteschlange übertragen, während diese Warteschlange aktiv ist.
- Das IBM MQ-System und Anwendungen dürfen die Eigenschaft **CLWLPRTY** nicht für Warteschlangen verwenden, die überwacht werden sollen, oder für Kanäle, die verwendet werden. Andernfalls können die Aktionen des Beispielprogramms zur Überwachung von Clusterwarteschlangen in Zusammenhang mit den **CLWLPRTY**-Warteschlangenattributen unerwünschte Auswirkungen haben.
- Das Beispielprogramm zur Überwachung von Clusterwarteschlangen protokolliert Laufzeitinformationen in einer Reihe von Berichtsdateien. Zum Speichern dieser Berichte ist ein Verzeichnis erforderlich; das Beispielprogramm zur Überwachung von Clusterwarteschlangen muss Schreibzugriff auf dieses Verzeichnis haben.

AMQSCLM: Beispiel vorbereiten und ausführen
Das Beispielprogramm für die Überwachung von Clusterwarteschlangen kann lokal über einen verbundenen Warteschlangenmanager oder als Client, der über einen Kanal verbunden ist, ausgeführt werden. Das Beispielprogramm sollte, solange der Warteschlangenmanager aktiv ist, ausgeführt werden. Bei lokaler Ausführung kann das Programm als Warteschlangenmanagerservice konfiguriert werden, damit es für den betreffenden Warteschlangenmanager automatisch gestartet und gestoppt wird.

Vorbereitende Schritte

Vor der Ausführung des Beispielprogramms zur Überwachung von Clusterwarteschlangen müssen die folgenden Schritte ausgeführt werden.

1. Erstellen Sie auf jedem Warteschlangenmanager eine Arbeitswarteschlange zur internen Verwendung des Beispiels.

   Für jede Instanz des Beispielprogramms ist eine lokale Warteschlange (bei der es sich nicht um eine Clusterwarteschlange handeln darf) zur exklusiven internen Verwendung erforderlich. Sie können einen eigenen Namen für die Warteschlange angeben. Das Beispielprogramm verwendet `AMQSCLM.CONTROL.QUEUE` als Name. Unter Windows beispielsweise können Sie diese Warteschlange mit dem folgenden **MQSC**-Befehl erstellen:

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   Für **MAXDEPTH** und **MAXMSGL** können Sie die Standardwerte übernehmen.

2. Erstellen Sie ein Verzeichnis für die Protokolle mit den Fehler- und Informationsnachrichten.

   Das Beispiel schreibt Diagnosenachrichten in Berichtsdateien. Sie müssen ein Verzeichnis angeben, in dem die Dateien gespeichert werden sollen. Unter Windows können Sie ein Verzeichnis beispielsweise mit dem folgenden Befehl erstellen:

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   Für die vom Beispiel erstellten Berichtsdateien wird die folgende Namenskonvention verwendet:

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```

3. (Optional) Definieren Sie das Beispiel zur Überwachung von Clusterwarteschlangen als IBM MQ-Service.

   Das Beispielprogramm muss immer aktiv sein, damit Warteschlangen überwacht werden. Um sicherzustellen, dass das Beispiel zur Überwachung von Clusterwarteschlangen immer aktiv ist, können Sie es als Warteschlangenmanagerservice definieren. Wenn das Beispiel als Service definiert ist, wird AMQSCLM beim Start des Warteschlangenmanagers gestartet. Mit dem folgenden Beispiel können Sie das Beispielprogramm zur Überwachung von Clusterwarteschlangen als IBM MQ-Service definieren.

   ```
   define service(AMQSCLM) +
         descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
         control(qmgr) +
         servtype(server) +
         startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
         startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
   c:\AMQSCLM\rpts') +
         stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
         stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
   ```

Definition	Beschreibung
service	Gibt den Namen des Service an. Sie können den Servicenamen selbst wählen.
descr	Eine Beschreibung des Service.
control	Gibt an, dass der Service zusammen mit dem Warteschlangenmanager gestartet bzw. gestoppt wird.
servtype	Gibt ein Serverserviceobjekt an; dies bedeutet, dass nur jeweils eine Instanz für diesen Warteschlangenmanager ausgeführt werden kann.
startcmd	Gibt den Namen des Programms an und wo es sich befindet.
startarg	Gibt die Argumente für das Beispiel an. Beachten Sie, dass im Beispiel oben *+QMNAME+* verwendet wird; der Name des Warteschlangenmanagers wird automatisch eingesetzt.
stdout	Der vollständig qualifizierte Name der Datei, in die die Standardausgabe weitergeleitet wird. Das Beispiel speichert in dieser Datei nur Nachrichten, die bestätigen, dass das Beispiel beendet wurde. Dem liegt zugrunde, dass die Standardfehlerdatei bereits in einem früheren Stadium des Beendigungsprozesses des Beispiels geschlossen wurde.
stderr	Der vollständig qualifizierte Name der Datei, in die die Standard-Fehlerausgabe weitergeleitet wird. Das Beispiel speichert sämtliche Fehlernachrichten in der Standardfehlerdatei, die vor Beendigung des Beispiels aufgetreten sind.

Informationen zu diesem Vorgang

Mit dieser Task haben Sie mehrere Möglichkeiten, das Beispielprogramm zur Überwachung von Clusterwarteschlangen zu starten bzw. zu stoppen. Außerdem ermöglicht sie Ihnen die Ausführung des Beispielprogramms in einem Modus, in dem Berichtsdateien mit statistischen Informationen zu den Warteschlangen generiert werden, die überwacht werden.

Das Beispielprogramm kann mit dem folgenden Befehl ausgeführt werden:

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

In der Tabelle sind die Argumente aufgeführt, die für das Beispielprogramm zur Überwachung von Clusterwarteschlangen verwendet werden können, sowie weitere Informationen zu diesen Argumenten.

Argument	Variable	Weitere Informationen
-m	**QMgrName**	Der Warteschlangenmanager, der überwacht werden soll.
-c	**ClusterName**	Der Cluster, in dem sich die Warteschlangen befinden, die überwacht werden sollen.
-q	**QNameMask**	Die Warteschlange bzw. Warteschlangen, die überwacht werden soll(en). Bei Angabe eines abschließenden Sterns (*) werden alle Warteschlangen überwacht, deren Name mit null oder mehr abschließenden Zeichen übereinstimmt.
-f	**QListFile**	Der vollständige Pfad und Dateiname der Datei mit den zu überwachenden Warteschlangennamen bzw. Warteschlangen-Namensmasken. Die Datei darf nur jeweils eine Warteschlange oder Maske pro Zeile enthalten. Sie können **-q** oder **-f** angeben, nicht jedoch beides.
-r	**MonitorQName**	Die lokale Warteschlange, die nur vom Beispielprogramm verwendet wird.
-l	**ReportDir**	Der Verzeichnispfad, in dem protokollierte Informationsnachrichten in einer Gruppe gespeichert werden sollen.[9]Berichtsdateien geschrieben werden.
-t		(Optional) Aktiviert die Übertragung eingereihter Nachrichten aus inaktiven lokalen Warteschlangen in aktive Warteschlangen. Ist diese Option nicht aktiviert, werden nur neue Nachrichten, die im Cluster eingehen, dynamisch an die aktiven Instanzen einer Warteschlange weitergeleitet.
-u	**ActiveVal**	(Optional) Bewirkt, dass die Eigenschaft **CLWLUSEQ** einer überwachten Warteschlangeninstanz automatisch auf ANY gesetzt wird, wenn diese Instanz inaktiv ist, und auf den Wert der Eigenschaft **ActiveVal**, wenn sie aktiv ist. **ActiveVal** kann den Wert LOCAL oder QMGR haben. Wird dieses Argument in einem System, in dem Anwendungen, die Nachrichten einreihen, eine Verbindung zu demselben Warteschlangenmanager herstellen oder in dem die Nachrichtenübertragung aktiviert ist, nicht gesetzt, muss die Eigenschaft **CLWLUSEQ** der überwachten Warteschlangen auf ANY gesetzt sein oder aber auf QMGR, wobei der Warteschlangenmanager auf ANY gesetzt ist.
-i	**Interval**	(Optional) Das Zeitintervall in Sekunden, in dem die Warteschlangen vom Überwachungsprogramm überprüft werden. Standardwert ist 300 Sekunden (5 Minuten),
-d		(Optional) Aktiviert zusätzliche Diagnosenachrichten. Die Debugausgabe ist unter Umständen bei der Erstkonfiguration des Systems oder bei der Verwendung des Beispielcodes hilfreich.
-s		(Optional) Aktiviert eine minimale statistische Ausgabe pro Intervall.
-v		(Optional) Berichtsinformationen werden nicht nur in Berichtsdateien, sondern zusätzlich noch in `standard out` aufgezeichnet.

[9] Für jede Kombination aus Warteschlangenmanager und Warteschlange wird eine Protokolldatei mit fester Größe generiert; ist die Datei voll, wird sie überschrieben. Die Protokollfunktion schreibt Daten immer in dieselbe Datei und bewahrt außerdem die beiden vorherigen Versionen der Datei auf.

Beispiele für Argumentlisten:

```
-m QMGR1 -c CLUS1   -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1   -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

Beispiel für eine Datei mit Warteschlangen:

```
Q1
QUEUE.*
ABC
ABD
```

Vorgehensweise

1. Starten Sie das Beispielprogramm zur Überwachung von Clusterwarteschlangen. Sie haben folgende Möglichkeiten, das Beispiel zu starten:
   - Über eine Eingabeaufforderung mit den entsprechenden Benutzerberechtigungen.
   - Mit dem MQSC-Befehl **START SERVICE** (wenn das Beispiel als IBM MQ-Service konfiguriert ist).

   In beiden Fällen ist die Argumentliste dieselbe.

   Das Beispielprogramm beginnt mit der Überwachung der Warteschlangen erst 10 Sekunden nach seiner Initialisierung. Durch diese Verzögerung können konsumierende Anwendungen zuerst eine Verbindung zu den überwachten Warteschlangen herstellen, sodass unnötige Änderungen am aktiven Status der Warteschlangen vermieden werden.
2. Stoppen Sie das Beispielprogramm zur Überwachung von Clusterwarteschlangen. Das Beispiel wird automatisch gestoppt, wenn der Warteschlangenmanager beendet wurde bzw. beendet oder in den Quiescemodus versetzt wird, oder wenn die Verbindung zum Warteschlangenmanager unterbrochen wird. Es ist möglich, das Beispiel zu stoppen, ohne den Warteschlangenmanager zu beenden:
   - Konfigurieren Sie die lokale Warteschlange für die exklusive Nutzung durch das Beispielprogramm, um die Abruffunktion zu inaktivieren.
   - Senden Sie eine Nachricht mit `"STOP CLUSTER MONITOR\0\0\0\0"` als **CorrelId** an die ausschließlich vom Beispiel verwendete lokale Warteschlange.
   - Beenden Sie den Beispielprozess. Dies kann unter Umständen zum Verlust nicht persistenter Nachrichten führen, die gerade an aktive Warteschlangen übertragen werden. Ebenso kann der Fall eintreten, dass die vom Beispiel verwendete lokale Warteschlange nach der Beendigung noch einige Sekunden geöffnet bleibt. Dadurch wird verhindert, dass sofort eine neue Instanz des Beispielprogramms zur Überwachung von Clusterwarteschlangen gestartet wird.

   Wurde das Beispiel als IBM MQ-Service gestartet, ist der Befehl **STOP SERVICE** wirkungslos. Es ist möglich, eine der beschriebenen Beendigungsmethoden als einen konfigurierten **STOP SERVICE**-Mechanismus im Warteschlangenmanager zu verwenden.

Nächste Schritte

Überprüfen Sie den Status des Beispielprogramms.

Wenn die Berichterstellung aktiviert ist, können Sie den Status anhand der Berichtsdateien überprüfen. Mit dem folgenden Befehl können Sie die aktuellste Berichtsdatei überprüfen:

```
QMgrName.ClusterName.RPT01.LOG
```

Ältere Berichte können mit den folgenden Befehlen überprüft werden:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

Berichtsdateien können eine maximale Größe von ungefähr 1 MB erreichen. Wenn die Datei RPT01 voll ist, wird eine neue Datei RPT01 erstellt. Die ursprüngliche RPT01-Datei wird in RPT02 umbenannt. RPT02 wird in RPT03 umbenannt Die alte Datei RPT03 wird gelöscht.

Das Beispielprogramm erstellt in den folgenden Fällen Informationsnachrichten:

- Beim Starten
- Beim Beenden
- Wenn es eine Warteschlange als **ACTIVE** oder **INACTIVE** kennzeichnet
- Wenn es Nachrichten aus einer inaktiven Warteschlange in aktive Instanzen einreiht

Das Beispielprogramm erstellt Fehlernachrichten im Format *CLMnnnnE*, mit denen Probleme gemeldet werden, die eingehender untersucht werden müssen.

Alle 30 Minuten meldet das Beispielprogramm die durchschnittliche Verarbeitungszeit pro Abfrageintervall sowie die bereits verstrichene Verarbeitungszeit. Diese Informationen werden in der Nachricht CLM0045I gemeldet.

Wurden statistische Nachrichten aktiviert (**-s**), meldet das Beispielprogramm die folgenden statistischen Informationen zu jeder Warteschlangenprüfung:

- Wie lange die Verarbeitung der Warteschlangen gedauert hat (in Millisekunden)
- Die Anzahl der Warteschlangen, die überprüft wurden
- Die Anzahl der Statusänderungen (in den aktiven bzw. inaktiven Status), die vorgenommen wurden
- Die Anzahl der Nachrichten, die übertragen wurden

Diese Informationen werden in der Nachricht CLM0048I gemeldet.

Berichtsdateien können sich im Debugmodus rasch füllen und in schneller Folge umbenannt und überschrieben werden. In einem solchen Fall kann das Größenlimit von 1 MB bei einzelnen Dateien unter Umständen überschritten werden.

AMQSCLM: Fehlerbehebung
Die folgenden Abschnitte enthalten Informationen zu Szenarios, zu denen es bei der Verwendung des Beispielprogramms kommen kann. Hinweise zu möglichen Ursachen für ein Szenario sowie Möglichkeiten, das Problem zu beheben, werden ebenfalls bereitgestellt.

Szenario: AMQSCLM startet nicht

Mögliche Ursache: Falsche Syntax.

Maßnahme: Überprüfen Sie die Standard-Fehlerausgabe auf eine korrekte Syntax.

Mögliche Ursache: Der Warteschlangenmanager ist nicht verfügbar.

Maßnahme: Überprüfen Sie die Berichtsdatei auf die Nachrichten-ID CLM0010E.

Mögliche Ursache: Eine oder mehrere Berichtsdateien können nicht geöffnet oder erstellt werden.

Maßnahme: Überprüfen Sie die Standard-Fehlerausgabe auf Fehlernachrichten, die während der Initialisierung generiert wurden.

Szenario: AMQSCLM ändert den Status einer Warteschlange nicht in ACTIVE oder INACTIVE

Mögliche Ursache: Die Warteschlange ist nicht in der Liste der Warteschlangen enthalten, die überwacht werden sollen.

Maßnahme: Überprüfen Sie die Werte der Parameter **-q** und **-f**.

Mögliche Ursache: Bei der Warteschlange handelt es sich nicht um eine lokale Warteschlange im richtigen Cluster.

Maßnahme: Überprüfen Sie, ob es sich bei der Warteschlange um eine lokale Warteschlange handelt und ob sie sich im richtigen Cluster befindet.

Mögliche Ursache: AMQSCLM ist für diesen Warteschlangenmanager und diesen Cluster nicht aktiv.

Maßnahme: Starten Sie AMQSCLM für den betreffenden Warteschlangenmanager und Cluster.

Mögliche Ursache: Die Warteschlange hat den Status INACTIVE (**CLWLPRTY**=0), da keine Konsumenten mit ihr verbunden sind, oder sie hat den Status ACTIVE (**CLWLPRTY** >=1, da mindestens ein Konsument mit ihr verbunden ist.

Maßnahme: Überprüfen Sie, ob konsumierende Anwendungen mit der Warteschlange verbunden sind.

Mögliche Ursache: Der Befehlsserver des Warteschlangenmanagers ist nicht aktiv.

Maßnahme: Überprüfen Sie die Berichtsdateien auf Fehler.

Szenario: Nachrichten werden nicht an Warteschlangen mit dem Status INACTIVE vorbeigeleitet.

Mögliche Ursache: Nachrichten werden direkt in den Warteschlangenmanager eingereiht, der Eigner der inaktiven Warteschlange ist; außerdem hat das Attribut **CLWLUSEQ** der Warteschlange nicht den Wert ANY und für AMQSCLM ist das Argument **-u** nicht angegeben.

Maßnahme: Überprüfen Sie den Wert der Eigenschaft **CLWLUSEQ** des betreffenden Warteschlangenmanagers oder stellen Sie sicher, dass für AMQSCLM das Argument **-u** verwendet wird.

Mögliche Ursache: Es sind keine aktiven Warteschlangen in den Warteschlangenmanagern verfügbar. Nachrichten werden gleichmäßig auf alle inaktiven Warteschlangen verteilt, bis eine Warteschlange wieder aktiv ist.

Maßnahme: Überprüfen Sie den Status der Warteschlangen auf allen Warteschlangenmanagern.

Mögliche Ursache: Die Nachrichten werden nicht in den Warteschlangenmanager eingereiht, der Eigner der inaktiven Warteschlange ist, sondern in einen anderen Warteschlangenmanager im Cluster, und die Eigenschaft **CLWLPRTY** mit dem aktualisierten Wert 0 wird nicht an den Warteschlangenmanager der Anwendung, die Nachrichten einreiht, weitergegeben.

Maßnahme: Überprüfen Sie, ob die Clusterkanäle zwischen dem überwachten Warteschlangenmanager und dem Warteschlangenmanager mit vollständigem Repository aktiv sind. Überprüfen Sie, ob die Kanäle zwischen dem Warteschlangenmanager, der Nachrichten einreiht, und dem Warteschlangenmanager mit vollständigem Repository aktiv sind. Überprüfen Sie die Fehlerprotokolle des überwachten Warteschlangenmanagers, des Warteschlangenmanagers, der Nachrichten einreiht und des Warteschlangenmanagers mit vollständigem Repository.

Mögliche Ursache: Die fernen Warteschlangeninstanzen sind aktiv (CLWLPRTY=1), es können jedoch keine Nachrichten an diese Warteschlangeninstanzen weitergeleitet werden, da der Clustersenderkanal vom lokalen Warteschlangenmanager nicht aktiv ist.

Maßnahme: Überprüfen Sie den Status der Clustersenderkanäle vom lokalen Warteschlangenmanager zum fernen Warteschlangenmanager (oder zu fernen Warteschlangenmanagern) mit einer aktiven Instanz der Warteschlange.

Szenario: AMQSCLM überträgt keine Nachrichten aus einer inaktiven Warteschlange

Mögliche Ursache: Die Nachrichtenübertragung ist nicht aktiviert (**-t**).

Maßnahme: Stellen Sie sicher, dass die Nachrichtenübertragung aktiviert ist (**-t**).

Mögliche Ursache: Die Warteschlange ist nicht in der Liste der Warteschlangen enthalten, die überwacht werden sollen.

Maßnahme: Überprüfen Sie die Werte der Parameter **-q** und **-f**.

Mögliche Ursache: AMQSCLM ist für diesen Warteschlangenmanager oder für andere Warteschlangenmanager im Cluster, die über Instanzen derselben Warteschlange verfügen, nicht aktiv.

Maßnahme: Starten Sie AMQSCLM.

Mögliche Ursache: Für die Warteschlange ist **CLWLUSEQ**=LOCAL oder **CLWLUSEQ**=QMGR angegeben und das Argument **-u** wurde nicht gesetzt.

Maßnahme: Setzen Sie den Parameter **-u** oder ändern Sie die Warteschlangenkonfiguration oder die Konfiguration des Warteschlangenmanagers in ANY.

Mögliche Ursache: Im Cluster sind keine aktiven Instanzen der Warteschlange vorhanden.

Maßnahme: Überprüfen Sie, ob Instanzen der Warteschlange vorhanden sind, für die die Eigenschaft **CLWLPRTY** den Wert 1 oder größer hat.

Mögliche Ursache: Ferne Warteschlangeninstanzen sind mit Konsumenten verbunden (**IPPROCS** =1), sind jedoch inaktiv auf diesen Warteschlangenmanagern (**CLWLPRTY**=0), da diese fernen Instanzen nicht von AMQSCLM überwacht werden.

Maßnahme: Überprüfen Sie die Werte der Parameter **-q** und **-f**, um sicherzustellen, dass AMQSCLM auf diesen Warteschlangenmanagern aktiv ist und/oder dass die Warteschlange in der Liste der Warteschlangen enthalten ist, die überwacht werden sollen.

Mögliche Ursache: Die fernen Warteschlangeninstanzen sind aktiv (**CLWLPRTY**=1), werden auf dem lokalen Warteschlangenmanager aber als inaktiv wahrgenommen (**CLWLPRTY**=0). Dies liegt daran, dass der aktualisierte Wert von **CLWLPRTY** nicht an diesen Warteschlangenmanager weitergegeben wird.

Maßnahme: Stellen Sie sicher, dass die fernen Warteschlangenmanager mit mindestens einem der Warteschlangenmanager mit vollständigem Repository im Cluster verbunden sind. Stellen Sie sicher, dass die Warteschlangenmanager mit vollständigem Repository ordnungsgemäß arbeiten. Überprüfen Sie, ob die Kanäle zwischen den Warteschlangenmanagern mit vollständigem Repository und den überwachten Warteschlangenmanagern aktiv sind.

Mögliche Ursache: Die Nachrichten wurden nicht festgeschrieben und können daher nicht abgerufen werden.

Maßnahme: Überprüfen Sie, ob die sendende Anwendung ordnungsgemäß arbeitet.

Mögliche Ursache: AMQSCLM hat keinen Zugriff auf die lokale Warteschlange, in der die Nachrichten eingereiht sind.

Maßnahme: Überprüfen Sie, ob AMQSCLM als Benutzer mit der entsprechenden Berechtigung für einen Zugriff auf die Warteschlange ausgeführt wird.

Mögliche Ursache: Der Befehlsserver des Warteschlangenmanagers ist nicht aktiv.

Maßnahme: Starten Sie den Befehlsserver des Warteschlangenmanagers.

Mögliche Ursache: Bei der Ausführung von AMQSCLM ist ein Fehler aufgetreten.

Maßnahme: Überprüfen Sie die Berichtsdateien auf Fehler.

Mögliche Ursache: Die fernen Warteschlangeninstanzen sind aktiv (CLWLPRTY=1), es können jedoch keine Nachrichten an diese Warteschlangeninstanzen übertragen werden, da der Clustersenderkanal vom lokalen Warteschlangenmanager nicht aktiv ist. In diesem Zusammenhang wird häufig eine CLM0030W-Warnung in das Berichtsprotokoll für AMQSCLM ausgegeben.

Maßnahme: Überprüfen Sie den Status der Clustersenderkanäle vom lokalen Warteschlangenmanager zum fernen Warteschlangenmanager (oder zu fernen Warteschlangenmanagern) mit einer aktiven Instanz der Warteschlange.

ALW *Das Beispielprogramm für die Suche nach Verbindungsendpunkten (CEPL)*

Das Beispielprogramm von IBM MQ für die Suche nach Verbindungsendpunkten ist ein einfaches, aber leistungsfähiges Exitmodul, das IBM MQ-Benutzern den Abruf von Verbindungsdefinitionen aus einem LDAP-Repository wie beispielsweise Tivoli Directory Server ermöglicht.

Damit CEPL verwendet werden kann, muss der Tivoli Directory Server V6.3-Client installiert sein.

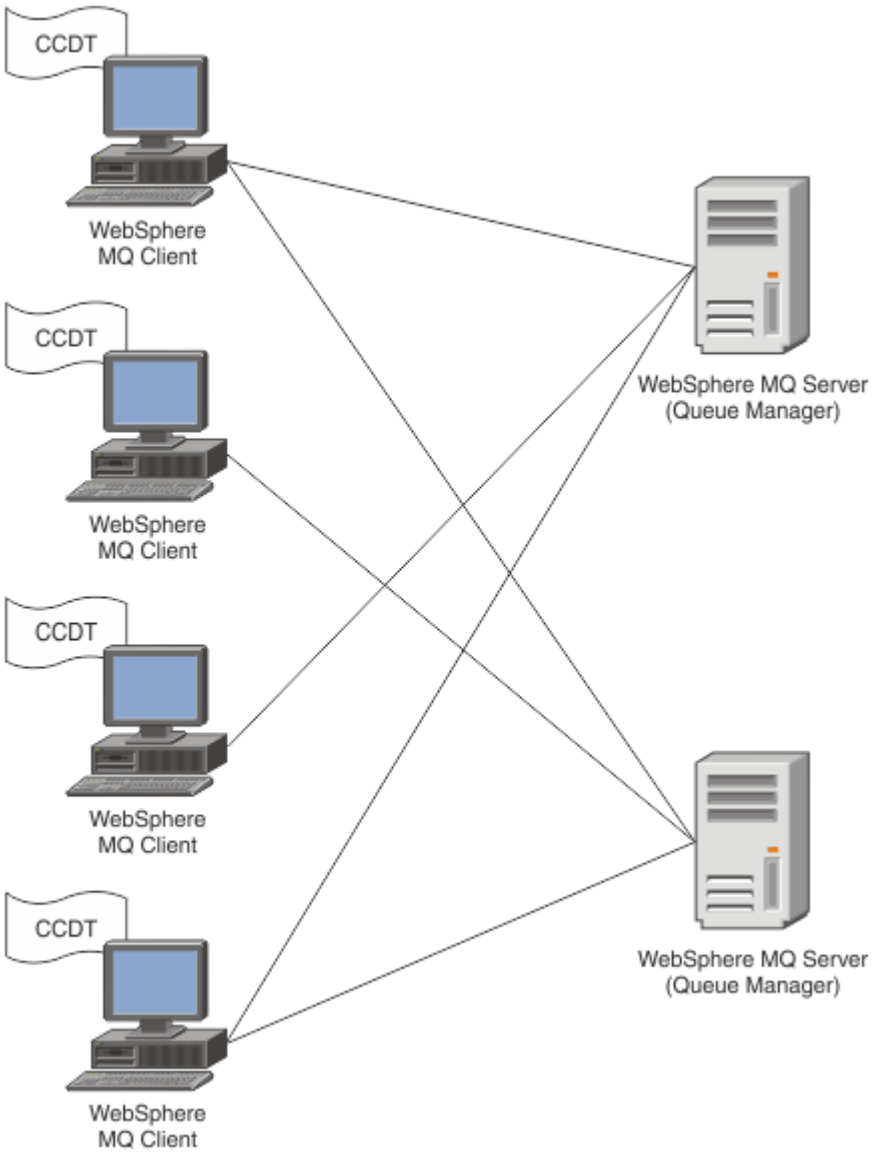
Für die Verwendung dieses Beispielprogramms sind praktische Erfahrungen mit der Verwaltung von IBM MQ auf den unterstützten Plattformen erforderlich.

Windows Linux AIX *Einführung*

Konfigurieren Sie ein globales Repository wie beispielsweise ein LDAP-Verzeichnis (Lightweight Directory Access Protocol), in dem die Clientverbindungsdefinitionen gespeichert werden; dies erleichtert die Wartung und Verwaltung.

Stellen Sie mithilfe einer IBM MQ-Clientanwendung über eine CCDT (Client Connection Definition Table; Tabelle mit Clientverbindungsdefinitionen) eine Verbindung zu einem Warteschlangenmanager her.

Die CCDT wird über die standardmäßige MQSC-Verwaltungsschnittstelle von IBM MQ erstellt. Der Benutzer muss mit einem Warteschlangenmanager verbunden sein, damit Clientverbindungsdefinitionen erstellt werden können, auch wenn die Daten in der Definition nicht auf diesen Warteschlangenmanager beschränkt sind. Die generierte CCDT-Datei muss manuell an die Clientmaschinen und Anwendungen



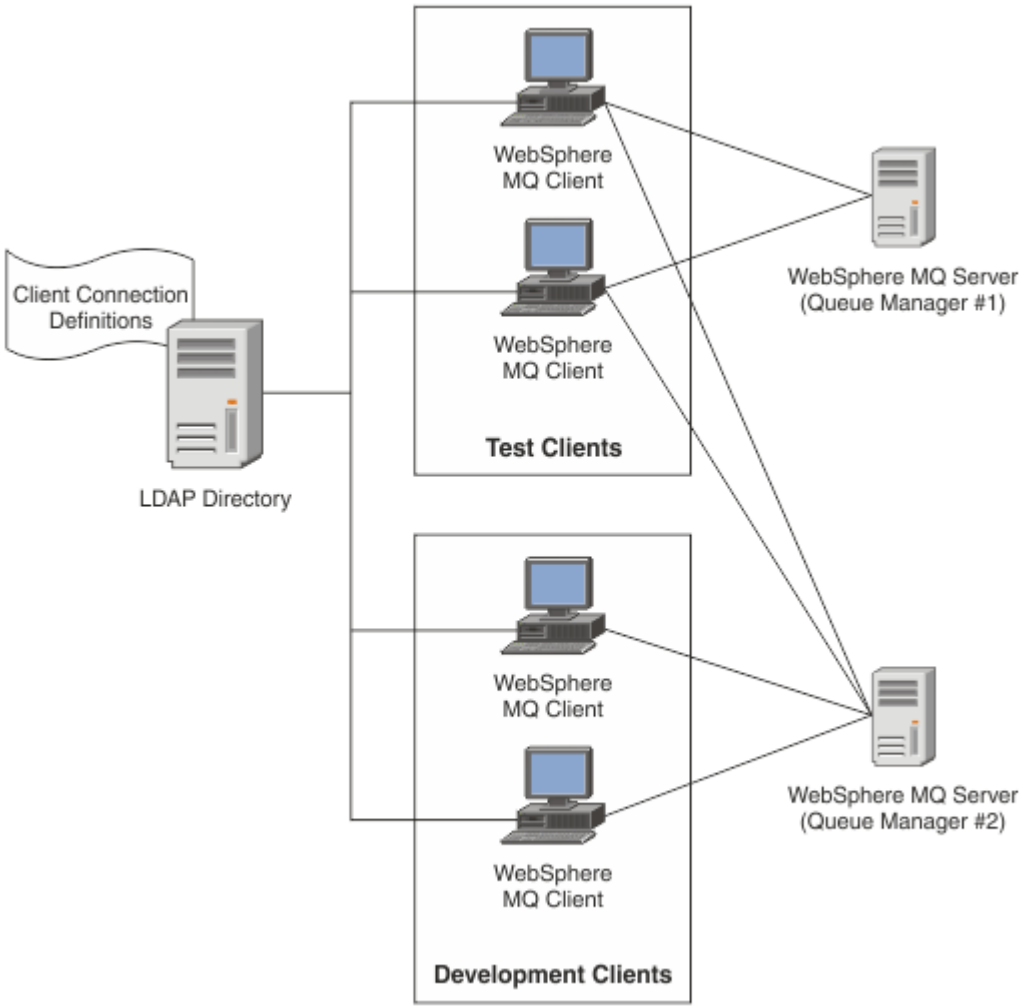
verteilt werden.

Die CCDT-Datei muss an jeden IBM MQ-Client verteilt werden. Bei Tausenden von lokalen oder globalen Clients würde die Wartung und Verwaltung schnell kompliziert werden. Daher ist ein flexiblerer Ansatz erforderlich, um sicherzustellen, dass für jeden Client die korrekten Clientdefinitionen verfügbar sind.

Eine Möglichkeit ist, die Clientverbindungsdefinitionen in einem globalen Repository wie beispielsweise einem LDAP-Verzeichnis (Lightweight Directory Access Protocol) zu speichern. Ein LDAP-Verzeichnis kann

außerdem noch zusätzliche Sicherheits-, Indexierungs- und Suchfunktionen bereitstellen, sodass jeder Client nur Zugriff auf die für ihn geltenden Verbindungsdefinitionen hat.

Das LDAP-Verzeichnis kann so konfiguriert werden, dass für bestimmte Benutzergruppen nur jeweils bestimmte Definitionen verfügbar sind. So können beispielsweise die Testclients auf Warteschlangenmanager 1 und 2 zugreifen, während die Entwicklungsclients nur Zugriff auf Warteschlangenmanager 2



haben.

Das Exitmodul kann ein LDAP-Repository (beispielsweise IBM Tivoli Directory Server) für den Abruf von Kanaldefinitionen durchsuchen. Mit diesen Verbindungsdefinitionen kann eine IBM MQ-Clientanwendung eine Verbindung zu einem Warteschlangenmanager herstellen.

Das Exitmodul ist ein Preconnect-Exitmodul, mit dem bei einem MQCONN-/MQCONNX-Aufruf Kanaldefinitionen aus einem LDAP-Repository abgerufen werden können.

Kunden, die das Exitmodul und -schema möglicherweise implementieren:

- Kunden, die bereits Erfahrung im Umgang mit CCDT-Dateien haben und den Verwaltungs- und Verteilungsaufwand reduzieren möchten.
- Bestandskunden, die bereits ihre eigene proprietäre Technologie für die Verteilung von Clientverbindungsdefinitionen verwenden.
- Neu- oder Bestandskunden, die momentan noch über keine Clientverbindungslösung verfügen und die von IBM MQ bereitgestellten Funktionen nutzen möchten.
- Neu- oder Bestandskunden, die ihr eigenes Messaging-Modell direkt innerhalb der bestehenden LDAP-Geschäftsarchitekturen verwenden oder optimieren möchten.

ALW *Unterstützte Umgebungen*

Überprüfen Sie vor der Ausführung des Beispielprogramms für die Suche nach Verbindungsendpunkten, ob Sie ein unterstütztes Betriebssystem verwenden und ob die erforderliche Software installiert ist.

Für das Beispielprogramm von IBM MQ für die Suche nach Verbindungsendpunkten ist die folgende Software erforderlich:

- IBM WebSphere MQ 7.0 oder höher
- Tivoli Directory Server V6.3-Client oder höher

Unterstützte Betriebssysteme:

1. Windows Windows (7/8/2008/2012)
2. AIX AIX
3. Linux Linux
   - RHEL V4 und V5 unter System p
   - SUSE V9 und V10 unter System p
   - RHEL V4 und V5 unter x86-64 32 Bit und 64 Bit
   - SUSE V9 und V10 unter x86-64 32 Bit und 64 Bit

Anmerkung: Das Beispielprogramm ist für die folgenden Plattformen nicht verfügbar:

- z/OS z/OS
- IBM i IBM i

ALW *Installation und Konfiguration*

Dieser Abschnitt enthält Informationen zum Installieren und Konfigurieren des Exitmoduls und des Schemas für Verbindungsendpunkte.

Exitmodul installieren

Das Exitmodul wird bei der Installation von IBM MQ im Verzeichnis `tools/samples/c/preconnexit/bin` installiert. Auf 32-Bit-Plattformen muss das Exitmodul in das Verzeichnis `exit/`*`installation_name`*`/` kopiert werden, damit es verwendet werden kann. Auf 64-Bit-Plattformen muss das Exitmodul in das Verzeichnis exit64/*Installationsname*/ kopiert werden, damit es verwendet werden kann.

Schema für Verbindungsendpunkte installieren

Der Exit verwendet das Verbindungsendpunktschema `ibm-amq.schema`. Die Schemadatei muss in einen LDAP-Server importiert werden, damit der Exit verwendet werden kann. Nach dem Import des Schemas müssen Werte für die Attribute hinzugefügt werden.

Das folgende Beispiel veranschaulicht den Import des Schemas für Verbindungsendpunkte. Bei diesem Beispiel wird davon ausgegangen, dass IBM Tivoli Directory Server (ITDS) verwendet wird.

- Stellen Sie sicher, dass IBM Tivoli Directory Server aktiv ist, und kopieren Sie die Datei `ibm-amq.schema` in den ITDS-Server.
- Geben Sie auf dem ITDS-Server folgenden Befehl ein, um das Schema im ITDS-Speicher zu installieren, wobei *LDAP-ID* und *LDAP-Kennwort* für den Root-DN und das Rootkennwort für den LDAP-Server stehen:

  ```
  ldapadd -D "LDAP ID" -w "LDAP password" -f ibm-amq.schema
  ```

- Geben Sie in einem Befehlsfenster den folgenden Befehl ein oder navigieren Sie mit dem Tool eines anderen Anbieters zu dem Schema, um es zu überprüfen:

```
ldapsearch objectclass=ibm-amqClientConnection
```

Weitere Informationen zum Import der Schemadatei finden Sie in der Dokumentation des LDAP-Servers.

Konfiguration

Zur Clientkonfigurationsdatei (z. B. `mqclient.ini`) muss ein neuer Abschnitt namens 'PreConnect' hinzugefügt werden. Der Abschnitt 'PreConnect' enthält die folgenden Schlüsselwörter:

Modul
Der Name des Moduls, das den API-Exit-Code enthält. Wenn dieses Feld den vollständigen Pfad des Moduls enthält, wird es unverändert übernommen. Andernfalls werden die Ordner `exit` oder `exit64` der IBM MQ-Installation durchsucht.

Funktion
Der Name des funktionalen Eingangspunkts in die Bibliothek, die den `LdapPreConnect` -Exit-Code enthält Die Funktionsdefinition entspricht dem Funktionsprototyp Ihres Unternehmens.

Achtung: Sie sollten die Anführungszeichen in der Funktionsanweisung entfernen, wenn Sie Ihren tatsächlichen Exiteingangspunkt angeben.

Data
URI des LDAP-Repositorys, das Kanaldefinitionen enthält.

Das folgende Snippet ist ein Beispiel für die erforderlichen Änderungen an der Datei `mqclient.ini`.

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW *Überblick über Exit und Schema*

Dieser Abschnitt enthält Informationen zur Syntax und zu den Parametern, mit denen eine Verbindung zu einem Warteschlangenmanager hergestellt wird.

In IBM MQ 9.3 ist für einen Eingangspunkt in einem Exitmodul die folgende Syntax definiert:

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

Bei der Ausführung des MQCONN/X-Aufrufs lädt der IBM MQ-Client für C das Exitmodul, das eine Implementierung der Funktionssyntax enthält. Anschließend ruft er eine Exitfunktion auf, um die Kanaldefinitionen abzurufen. Schließlich wird mithilfe der abgerufenen Kanaldefinitionen eine Verbindung zu einem Warteschlangenmanager hergestellt.

Parameter

pExitParms
Typ: PMQNXP - Ein-/Ausgabe

Die Parameterstruktur des Preconnection-Exits. Die Struktur wird von dem Programm zugeordnet und verwaltet, das den Exit aufruft.

```
struct tagMQNXP
{
  MQCHAR4    StrucId;            /* Structure identifier */
  MQLONG     Version;            /* Structure version number */
  MQLONG     ExitId;             /* Type of exit */
  MQLONG     ExitReason;         /* Reason for invoking exit */
  MQLONG     ExitResponse;       /* Response from exit */
```

```
  MQLONG     ExitResponse2;       /* Secondary response from exit */
  MQLONG     Feedback;            /* Feedback code (reserved) */
  MQLONG     ExitDataLength;      /* Exit data length */
  PMQCHAR    pExitDataPtr;        /* Exit data */
  MQPTR      pExitUserAreaPtr;    /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;      /* Array of pointers to MQCDs */
  MQLONG     MQCDArrayCount;      /* Number of entries found */
  MQLONG     MaxMQCDVersion;      /* Maximum MQCD version */
};
```

pQMgrName

Typ: PMQCHAR Ein-/Ausgabe

Name des Warteschlangenmanagers. Bei Eingabe ist dieser Parameter die Filterzeichenfolge, die über den Parameter **QMgrName** an den API-Aufruf MQCONN übergeben wird. Dieses Feld kann leer bleiben oder es kann einen definierten Namen oder bestimmte Platzhalterzeichen enthalten. Das Feld wird durch den Exit geändert. Wenn der Exit mit MQXR_TERM aufgerufen wird, hat der Parameter den Wert NULL.

ppConnectOpts

Typ: ppConnectOpts Ein-/Ausgabe

Optionen, mit denen die Aktion des MQCONNX-Aufrufs gesteuert wird. Hierbei handelt es sich um einen Verweis auf eine MQCNO-Struktur für Verbindungsoptionen, mit denen die Aktion des API-Aufrufs MQCONN gesteuert wird. Wenn der Exit mit MQXR_TERM aufgerufen wird, hat der Parameter den Wert NULL. Der MQI-Client stellt immer eine MQCNO-Struktur für den Exit bereit, auch wenn sie von der Anwendung ursprünglich nicht bereitgestellt wurde. Stellt eine Anwendung eine MQCNO-Struktur bereit, erstellt der Client ein Duplikat davon, das an den Exit übergeben und dort geändert wird. Der Client bleibt Eigner der MQCNO-Struktur. Eine MQCD-Struktur, auf die in der MQCNO-Struktur verwiesen wird, hat Vorrang vor allen Verbindungsdefinitionen, die über das Array bereitgestellt werden. Der Client stellt mithilfe der MQCNO-Struktur eine Verbindung zum Warteschlangenmanager her, während die anderen Strukturen ignoriert werden.

pCompCode

Typ: PMQLONG Ein-/Ausgabe

Beendigungscode. Verweis auf eine MQLONG-Struktur, die den Beendigungscode des Exits empfängt. Folgende Werte sind zulässig:

- MQCC_OK - Erfolgreiche Ausführung
- MQCC_WARNING - Warnung (teilweise Ausführung)
- MQCC_FAILED - Aufruf fehlgeschlagen

pReason

Typ: PMQLONG Ein-/Ausgabe

Ursachencode zur näheren Bestimmung von 'pCompCode'. Verweis auf eine MQLONG-Struktur, die den Ursachencode des Exits empfängt. Wenn der Beendigungscode MQCC_OK lautet, ist der einzige gültige Wert folgender: MQRC_NONE - (0, x'000') Keine Ursache zu melden.

Wenn der Beendigungscode MQCC_FAILED oder MQCC_WARNING lautet, kann das Feld für den Ursachencode von der Exitfunktion auf einen beliebigen MQRC_*-Wert gesetzt werden.

ALW *MQ-LDAP-Kontextinformationen*

Der Exit verwendet für Kontextinformationen die folgende Datenstruktur.

MQNLDAPCTX

Die MQNLDAPCTX-Struktur hat den folgenden C-Prototyp.

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
```

```
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Linux AIX *Beispielcode zum Erstellen des Exits für die Suche nach Verbindungsendpunkten*

Mit den Beispiel-Code-Snippets können Sie die Quelle unter AIX, Linux oder Windows kompilieren.

Quelle kompilieren

Sie können die Quelle mit beliebigen LDAP-Clientbibliotheken (beispielsweise den IBM Tivoli Directory Server V6.3-Clientbibliotheken) kompilieren. In dieser Dokumentation wird von einer Verwendung der Tivoli Directory Server V6.3-Clientbibliotheken ausgegangen.

Anmerkung: Die Preconnect-Exitbibliothek wird von den folgenden LDAP-Servern unterstützt:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

Die folgenden Code-Snippets veranschaulichen die Kompilierung der Exits:

Windows **Exit auf der Windows-Plattform kompilieren**

Mit dem folgenden Snippet können Sie die Exitquelle kompilieren:

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
              /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
$(CC) $(CCARGS) $*.c
```

Anmerkung: Wenn Sie die Clientbibliotheken von IBM Tivoli Directory Server V6.3 verwenden, die mit dem Microsoft Visual Studio 2003 -Compiler kompiliert wurden, erhalten Sie möglicherweise Warnungen, wenn Sie die Clientbibliotheken von IBM Tivoli Directory Server V6.3 mit dem Compiler von Microsoft Visual Studio 2012oder höher kompilieren.

Linux AIX **Exit unter AIX, Linux kompilieren**

Mit dem folgenden Snippet können Sie die Exitquelle unter Linux kompilieren. Einige Compileroptionen können sich unter AIX unterscheiden.

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32
```

```
TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server enthält Static Link Librarys und Dynamic Link Librarys, es kann aber nur ein Bibliothekstyp verwendet werden. Bei dem Script wird davon ausgegangen, dass die statischen Bibliotheken verwendet werden.

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl

amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW *Aufruf des PreConnect-Exitmoduls*

Das PreConnect-Exitmodul kann mit drei verschiedenen Ursachencodes aufgerufen werden: Ursachencode MQXR_INIT zum Initialisieren und Herstellen einer Verbindung zu einem LDAP-Server, Ursachencode MQXR_PRECONNECT zum Abrufen von Kanaldefinitionen von einem LDAP-Server oder Ursachencode MQXR_TERM, wenn der Exit bereinigt werden soll.

MQXR_INIT

Der Exit wird mit Ursachencode MQXR_INIT aufgerufen, um eine Verbindung zu einem LDAP-Server zu initialisieren und herzustellen.

Vor dem MQXR_INIT-Aufruf wird das Feld 'pExitDataPtr' der MQNXP-Struktur mit dem Attribut 'Data' aus der PreConnect-Zeilengruppe in der Datei `mqclient.ini` (also dem LDAP-Wert) gefüllt.

Eine LDAP-URL setzt sich mindestens aus dem Protokoll, dem Hostnamen, der Portnummer und dem Basis-DN für die Suche zusammen. Der Exit analysiert die im Feld 'pExitDataPtr' enthaltene LDAP-URL, ordnet eine MQNLDAPCTX-Struktur für die Kontextsuche zu und füllt diese entsprechend. Die Adresse dieser Struktur wird im Feld 'pExitUserAreaPtr' gespeichert. Wird die LDAP-URL nicht korrekt analysiert, führt dies zum Fehler MQCC_FAILED.

An diesem Punkt stellt der Exit mithilfe der Parameter **MQNLDAPCTX** eine Verbindung zum LDAP-Server her und stellt eine Bindung zum LDAP-Server her. Die daraus resultierenden LDAP-API-Kennungen werden ebenfalls in dieser Struktur gespeichert.

MQXR_PRECONNECT

Das Exitmodul wird mit Ursachencode MQXR_PRECONNECT aufgerufen, um Kanaldefinitionen von einem LDAP-Server abzurufen.

Der Exit durchsucht den LDAP-Server auf Kanaldefinitionen, die mit dem angegebenen Filter übereinstimmen. Wenn **QMgrNameparameter** einen bestimmten Warteschlangenmanagernamen enthält, gibt die Suche alle Kanaldefinitionen zurück, für die der Wert des LDAP-Attributs **ibm-amqQueueManagerName** mit dem angegebenen Warteschlangenmanagernamen übereinstimmt.

Wenn der Parameter **QMgrName** den Wert '*' oder ' ' (leer), gibt die Suche alle Kanaldefinitionen zurück, für die das Endpunktattribut **ibm-amqIsClientDefault Connection** auf WAHRgesetzt ist.

Nach einer erfolgreichen Suche bereitet der Exit ein oder mehrere Arrays mit MQCD-Definitionen vor und kehrt zum aufrufenden Programm zurück.

MQXR_TERM

Der Exit wird mit diesem Ursachencode aufgerufen, wenn er bereinigt werden soll. Bei dieser Reinigung trennt der Exit die Verbindung zum LDAP-Server und gibt den gesamten vom Exit zugeordneten und verwalteten Speicher frei, einschließlich der MQNLDAPCTX-Struktur, des Zeigerarrays und aller MQCD-Referenzen. Alle anderen Felder werden auf den jeweiligen Standardwert gesetzt. Die Exitparameter **pQMgrName** und **ppConnectOpts** werden während eines Exits mit dem Ursachencode MQXR_TERM nicht verwendet und können NULLlauten.

Zugehörige Verweise

Zeilengruppe für PreConnect der Clientkonfigurationsdatei

ALW *LDAP-Schemas*

Clientverbindungsdaten werden in einem globalen Repository gespeichert, dem LDAP-Verzeichnis (Lightweight Directory Access Protocol). Aus einem LDAP-Verzeichnis rufen IBM MQ-Clients die Verbindungsdefinitionen ab. Die Struktur der IBM MQ-Clientverbindungsdefinitionen im LDAP-Verzeichnis wird als LDAP-Schema bezeichnet. Bei einem LDAP-Schema handelt es sich um eine Reihe von Attributtypdefinitionen, Objektklassendefinitionen sowie weiteren Informationen, mit deren Hilfe ein Server feststellen kann, ob bei der Überprüfung anhand eines Filter- oder Attributwerts eine Übereinstimmung mit den Attributen eines Eintrags besteht und ob Operationen zugelassen, hinzugefügt und geändert werden sollen.

Daten im LDAP-Verzeichnis speichern

Die Clientverbindungsdefinitionen befinden sich unterhalb eines bestimmten Zweigs innerhalb der Verzeichnisstruktur, dem sogenannten Verbindungspunkt. Wie allen Knoten in einem LDAP-Verzeichnis ist dem Verbindungspunkt ein definierter Name (DN) zugeordnet. Dieser Knoten kann als Ausgangspunkt für Abfragen im Verzeichnis verwendet werden. Bei Verwendung eines Filters für Ihre Abfragen wird vom LDAP-Verzeichnis eine Gruppe von Clientverbindungsdefinitionen zurückgegeben. Sie können den Zugriff auf untergeordnete Verzeichnisstrukturen über Berechtigungen, die in anderen Teilen der Verzeichnisstruktur erteilt wurden, einschränken (beispielsweise auf Benutzer, Abteilungen oder Gruppen).

Eigene Attribute und Klassen definieren

Die Clientkanaldefinition wird gespeichert, indem das LDAP-Schema geändert wird. Für alle LDAP-Datendefinitionen sind Objekte und Attribute erforderlich. Diese Objekte und Attribute haben eine Objekt-ID (OID), über die das Objekt bzw. Attribut eindeutig gekennzeichnet ist. Alle Klassen in einem LDAP-Schema erben direkt oder indirekt vom übergeordneten Objekt. Das Clientkanaldefinitionsobjekt enthält die Attribute des übergeordneten Objekts. Für alle LDAP-Datendefinitionen sind Objekte und Attribute erforderlich:

- Objektdefinitionen bestehen aus einer Reihe von LDAP-Attributen.
- Attribute sind LDAP-Datentypen.

Eine Beschreibung der einzelnen Attribute und ihrer Zuordnung zu den normalen IBM MQ-Eigenschaften finden Sie unter LDAP-Attribute.

ALW *LDAP-Attribute*

Definierte LDAP-Attribute sind IBM MQ-spezifisch und entsprechen direkt den Clientverbindungseigenschaften.

Verzeichniszeichenfolgeattribute für IBM MQ-Clientkanäle

Die folgende Tabelle enthält eine Beschreibung der Zeichenfolgeattribute und ihre Zuordnung zu IBM MQ-Eigenschaften. Die Attribute können Werte der Syntax 'directoryString' enthalten (UTF-8-codierte Unicode-Zeichen, d. h., ein variables Bytecodierungssystem, das auch IA5/ASCII beinhaltet). Die Syntax wird durch die Objekt-ID (OID) angegeben.

Tabelle 166. Verzeichniszeichenfolgeattribute für IBM MQ-Clientkanäle

LDAP-Attribut	Beschreibung	IBM MQEigenschaft
KN	Der allgemeine Name, der sich aus dem Kanalnamen und dem Namen des Warteschlangenmanagers zusammensetzt.	
ibm-amqChannelName	Name der Kanaldefinition.	CHANNEL
ibm-amqConnectionName	Die ID der Kommunikationsverbindung.	CONNAME
ibm-amqDescription	Die Kanalbeschreibung.	DESCR
ibm-amqLocalAddress	Die lokale Kommunikationsadresse des Kanals.	LOCLADDR
ibm-amqModeName	Der Name des LU 6.2-Modus.	MODENAME

Tabelle 166. Verzeichniszeichenfolgeattribute für IBM MQ-Clientkanäle (Forts.)		
LDAP-Attribut	**Beschreibung**	**IBM MQEigenschaft**
ibm-amqPassword	Das Kennwort, das verwendet werden kann.	PASSWORD
ibm-amqQueueManagerName	Der Name des Warteschlangenmanagers bzw. der Gruppe von Warteschlangenmanagern, zu denen eine IBM MQ-Clientanwendung eine Verbindung anfordern kann.	QMNAME
ibm-amqSecurityExitUserData	Die Benutzerdaten, die an den Sicherheitsexit übergeben werden.	SCYDATA
ibm-amqSecurityExitName	Der Name des Exitprogramms, das vom Kanalsicherheitsexit ausgeführt werden soll.	SCYEXIT
ibm-amqSslCipherSpec	Eine einzelne CipherSpec für eine TLS-Verbindung.	SSLCIPH
ibm-amqSslPeerName	Überprüft den definierten Namen (DN) des Zertifikats vom Peer-Warteschlangenmanager oder Peer-Client am anderen Ende eines IBM MQ-Kanals.	SSLPEER
ibm-amqTransactionProgramName	Der Name des Transaktionsprogramms.	TPNAME
ibm-amqUserID	Die Benutzer-ID, die vom Nachrichtenkanalagenten verwendet werden soll, wenn eine sichere SNA-Sitzung zu einem fernen Nachrichtenkanalagenten hergestellt werden soll.	USERID

Ganzzahlattribute für IBM MQ-Clientverbindungen

Die Attribute mit vordefinierten Werten (beispielsweise einem aufgezählten Typ) werden als standardmäßige ganze Zahlen gespeichert. Diese Werte werden im LDAP-Verzeichnis als ganzzahlige Werte gespeichert, nicht unter Verwendung des zugeordneten Konstantennamens.

Tabelle 167. Verzeichnisganzzahlattribute für IBM MQ-Clientkanäle		
LDAP-Attribut	**Beschreibung**	**IBM MQEigenschaft**
ibm-amqConnectionAffinity	Gibt an, ob Clientanwendungen, die mehrmals Verbindungen über denselben Warteschlangenmanagernamen herstellen, denselben Clientkanal verwenden.	AFFINITY
ibm-amqClientChannelWeight	Eine Gewichtung, mit der Einfluss darauf genommen wird, welche Clientverbindungskanaldefinition verwendet werden soll.	CLNTWGHT
ibm-amqHeartBeatInterval	Der ungefähre zeitliche Abstand zwischen den Überwachungssignalen, die von einem sendenden Nachrichtenübertragungskanal übertragen werden sollen, wenn die Übertragungswarteschlange keine Nachrichten enthält.	HBINT
ibm-amqKeepAliveInterval	Ein Zeitlimitwert für einen Kanal.	KAINT
ibm-amqMaximumMessageLength	Gibt an, bis zu welcher Länge Nachrichten maximal übertragen werden können.	MAXMSGL
ibm-amqSharingConversations	Die maximale Anzahl an Dialogen, die eine TCP/IP-Kanalinstanz gemeinsam nutzen können.	SHARECNV

Tabelle 167. Verzeichnisganzzahlattribute für IBM MQ-Clientkanäle (Forts.)		
LDAP-Attribut	**Beschreibung**	**IBM MQEigenschaft**
ibm-amqTransportType	Der Transporttyp, der verwendet werden soll.	TRPTYPE

Boolesches Attribut für IBM MQ-Clientkanäle

Dieses boolesche Attribut wird keiner IBM MQ-Eigenschaft zugeordnet. Als Syntax für dieses Attribut wird ein boolescher Wert verwendet.

Tabelle 168. Boolesches Attribut für IBM MQ-Clientkanäle	
LDAP-Attribut	**Beschreibung**
ibm-amqIsClientDefault	Dieses boolesche Attribut ist für die Suche nach Einträgen gedacht, für die das Attribut 'ibm-amqQueueManagerName' nicht definiert ist.

Listenattribute für IBM MQ-Clientkanäle

IBM MQ-Eigenschaften werden im LDAP-Verzeichnis als Listenattribute mit durch Kommas getrennten Einzelwerten gespeichert. Die Attribute werden auf dieselbe Weise wie die anderen Verzeichniszeichenfolgeattribute definiert. Die folgende Tabelle enthält eine Beschreibung der Listenattribute und ihre Zuordnung zu IBM MQ-Eigenschaften.

Tabelle 169. Listenattribute für IBM MQ-Clientkanäle		
LDAP-Attribut	**Beschreibung**	**IBM MQEigenschaft**
ibm-amqHeaderCompression	Eine Liste mit Verfahren zur Headerdatenkomprimierung, die vom Kanal unterstützt werden.	COMPHDR
ibm-amqMessageCompression	Eine Liste mit Verfahren zur Nachrichtendatenkomprimierung, die vom Kanal unterstützt werden.	COMPMSG
ibm-amqSendExitUserData	Die Benutzerdaten, die an den Sendeexit übergeben werden.	SENDDATA
ibm-amqSendExitUserName, LDAP-Attribut	Der Name des Exitprogramms, das vom Kanalsendeexit ausgeführt werden soll.	SENDEXIT
ibm-amqReceiveExitUserData	Die Benutzerdaten, die an den Empfangsexit übergeben werden sollen.	RCVDATA
ibm-amqReceiveExitName	Der Name des Benutzerexitprogramms, das vom Kanalempfangsbenutzerexit ausgeführt werden soll.	RCVEXIT

ALW *Allgemeiner Name*

Der allgemeine Name (CN, Common Name) setzt sich aus dem Kanalnamen und dem Namen des Warteschlangenmanagers zusammen.

Dies ist ein vorab vorhandenes Attribut.

Der allgemeine Name hat das folgende Format:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

For example:

```
CN=TC1(QM_T1)
```

Sie können für dieses Attribut nur einen Wert angeben.

Dieses Attribut ist ein Zeichenfolgeattribut; bei der Angabe des Attributwerts muss die Groß-/Kleinschreibung nicht beachtet werden. Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in Unterschemas verwendet wird und mithilfe einer Teilzeichenfolge (beispielsweise CN=jim*, wobei CN ein Attribut ist), die einen oder mehrere Platzhalter enthält, das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqChannelName, LDAP-Attribut*
Dieses Attribut gibt den Namen der Kanaldefinition an.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 20 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in Unterschemas verwendet wird und mithilfe einer Teilzeichenfolge, die einen oder mehrere Platzhalter enthält, das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqDescription*
Dieses LDAP-Attribut gibt die Kanalbeschreibung an.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 64 Bytes angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqConnectionName, LDAP-Attribut*
Dieses LDAP-Attribut gibt die ID der Kommunikationsverbindung an. Es gibt die Kommunikationsverbindungen an, die von diesem Kanal verwendet werden sollen.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 264 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqLocalAddress, LDAP-Attribut*
Dieses Attribut gibt die lokale Kommunikationsadresse für den Kanal an.

Für dieses Attribut wird ein einziger Zeichenfolgewert mit einer Länge von maximal 48 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqModeName, LDAP-Attribut*
Dieses Attribut wird bei LU 6.2-Verbindungen verwendet. Es handelt sich um eine zusätzliche Angabe bei den Sitzungsmerkmalen der Verbindung, wenn die Zuordnung einer Kommunikationssitzung erfolgt.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von genau 8 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqPassword, LDAP-Attribut*
Dieses LDAP-Attribut gibt ein Kennwort an, das vom Nachrichtenkanalagenten zum Initialisieren einer sicheren LU 6.2-Sitzung mit einem fernen Nachrichtenkanalagenten verwendet werden kann.

Für dieses Attribut wird ein einzelner Ganzzahlwert mit einer Länge von maximal 12 Ziffern angegeben. Es ist kein vorab vorhandenes Attribut.

ALW *ibm-amqQueueManagerName, LDAP-Attribut*
Dieses Attribut gibt den Namen des Warteschlangenmanagers bzw. der Gruppe von Warteschlangenmanagern an, zu denen eine IBM MQ-Clientanwendung eine Verbindung anfordern kann.

Für dieses Attribut wird ein einziger Zeichenfolgewert mit einer Länge von maximal 48 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

Zugehörige Verweise
„ibm-amqIsClientDefault” auf Seite 1219
Dieses boolesche Attribut ist für die Suche nach Einträgen gedacht, für die das Attribut 'ibm-amqQueueManagerName' nicht definiert ist.

ALW *ibm-amqSecurityExitUserData*
Dieses LDAP-Attribut gibt die Benutzerdaten an, die an den Sicherheitsexit übergeben werden.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 999 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqSecurityExitName*
Dieses LDAP-Attribut gibt den Namen des Exitprogramms an, das vom Kanalsicherheitsexit ausgeführt werden soll.

Ist kein Kanalsicherheitsexit aktiv, wird das Attribut leer gelassen.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 999 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqSslCipherSpec, LDAP-Attribut*
Dieses LDAP-Attribut gibt eine einzelne CipherSpec für eine TLS-Verbindung an.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 32 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqSslPeerName*
Mit diesem LDAP-Attribut wird der definierte Name (DN) des Zertifikats vom Peer-Warteschlangenmanager oder Peer-Client am anderen Ende eines IBM MQ-Kanals überprüft.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 1024 Bytes angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqTransactionProgramName, LDAP-Attribut*
Dieses LDAP-Attribut gibt den Transaktionsprogrammnamen an. Es wird bei LU 6.2-Verbindungen verwendet.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 64 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqUserID, LDAP-Attribut*
Dieses LDAP-Attribut gibt die Benutzer-ID an, die vom Nachrichtenkanalagenten verwendet werden soll, wenn eine sichere SNA-Sitzung zu einem fernen Nachrichtenkanalagenten hergestellt werden soll.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von genau 12 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

ALW *ibm-amqConnectionAffinity, LDAP-Attribut*
Dieses LDAP-Attribut gibt an, ob Clientanwendungen, die mehrmals Verbindungen unter Angabe desselben Warteschlangenmanagernamens herstellen, denselben Clientkanal verwenden.

Für dieses Attribut wird ein einzelner Ganzzahlwert angegeben. Es ist kein vorab vorhandenes Attribut.

ALW *ibm-amqClientChannelWeight, LDAP-Attribut*
Dieses LDAP-Attribut gibt eine Gewichtung an, die Einfluss darauf hat, welche Clientverbindungskanaldefinition verwendet wird.

Mit dem Attribut zur Gewichtung von Clientkanälen kann die Auswahl der Clientkanaldefinitionen beeinflusst werden, wenn mehrere geeignete Definitionen zur Verfügung stehen.

Für dieses Attribut wird ein einzelner Ganzzahlwert angegeben. Es ist kein vorab vorhandenes Attribut.

ALW *ibm-amqHeartBeatInterval, LDAP-Attribut*
Dieses LDAP-Attribut gibt den ungefähren zeitlichen Abstand zwischen den Überwachungssignalen an, die von einem sendenden Nachrichtenkanalagenten übergeben werden sollen, wenn die Übertragungswarteschlange keine Nachrichten enthält.

Für dieses Attribut wird ein einzelner Ganzzahlwert angegeben. Es ist kein vorab vorhandenes Attribut. Der Standardwert ist 1. Er wird bei der aktuellen MQSERVER-Umgebungsvariablenoperation festgelegt.

ALW *ibm-amqKeepAliveInterval, LDAP-Attribut*
Über dieses LDAP-Attribut wird ein Zeitlimitwert für einen Kanal angegeben.

Der Wert dieses Attributs wird an den Kommunikationsstack übergeben und gibt das Keepalive-Zeitlimit für den Kanal an. Sie können über dieses Attribut für jeden Kanal einen eigenen Keepalive-Wert angeben.

Für dieses Attribut wird ein einzelner Ganzzahlwert angegeben. Es ist kein vorab vorhandenes Attribut.

ALW *ibm-amqMaximumMessageLength, LDAP-Attribut*
Dieses LDAP-Attribut gibt an, bis zu welcher Länge Nachrichten maximal übertragen werden können.

Der Standardwert dieses Attributs ist 104857600, wie durch die aktuelle MQSERVER-Umgebungsvariablenoperation vorgegeben. Für dieses Attribut wird ein einzelner Ganzzahlwert angegeben; es ist kein vorab vorhandenes Attribut.

ALW *ibm-amqSharingConversations, LDAP-Attribut*
Dieses LDAP-Attribut gibt die maximale Anzahl an Dialogen an, die eine TCP/IP-Kanalinstanz gemeinsam nutzen können.

Für dieses Attribut wird ein einzelner Ganzzahlwert angegeben. Es ist kein vorab vorhandenes Attribut.

ALW *ibm-amqTransportType, LDAP-Attribut*
Dieses LDAP-Attribut gibt den Transporttyp an, der verwendet werden soll.

Für dieses Attribut wird ein einzelner Ganzzahlwert angegeben. Es ist kein vorab vorhandenes Attribut.

ALW *ibm-amqIsClientDefault*
Dieses boolesche Attribut ist für die Suche nach Einträgen gedacht, für die das Attribut 'ibm-amqQueueManagerName' nicht definiert ist.

Für die Suche in den LDAP-Servern verwenden Preconnect-Exitmodule normalerweise den Wert des Attributs 'ibm-amqQueueManagerName' als Suchkriterium. Bei einer solchen Suche werden alle Einträge zurückgegeben, bei denen der Wert des Attributs 'ibm-amqQueueManagerName' mit dem Namen des im MQCONN/X-Aufruf angegebenen Warteschlangenmanagers übereinstimmt. Bei Verwendung der Definitionstabellen für Clientkanäle (Client Channel Definition Table, CCDT) kann der Name des Warteschlangenmanagers in einem MQCONN/X-Aufruf jedoch leer gelassen werden oder dem Namen kann ein Stern (*) vorangestellt werden. Wird der Name des Warteschlangenmanagers nicht angegeben, stellt der Client eine Verbindung zum Standardwarteschlangenmanager her. Ist dem Namen ein Stern (*) vorangestellt, stellt der Client eine Verbindung zu einem beliebigen Warteschlangenmanager her.

Ebenso kann auch das Attribut 'ibm-amqQueueManagerName' in einem Eintrag leer gelassen werden. In diesem Fall wird erwartet, dass der Client mit diesen Endpunktinformationen eine Verbindung zu einem beliebigen Warteschlangenmanager herstellen kann. Angenommen, ein Eintrag enthält die folgenden Zeilen:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

In diesem Beispiel versucht der Client, eine Verbindung zu dem angegebenen Warteschlangenmanager herzustellen, der auf `myhost` aktiv ist.

In LDAP-Servern wird jedoch eine Suche nicht anhand eines nicht definierten Attributwertes durchgeführt. Wenn beispielsweise ein Eintrag die Verbindungsinformationen mit Ausnahme von 'ibm-amqQueueManagerName' enthält, ist dieser Eintrag im Suchergebnis nicht enthalten. Dieses Problem kann behoben werden, indem Sie das Attribut 'ibm-amqIsClientDefault' setzen. Es ist ein boolesches Attribut und enthält den Wert FALSE, wenn es nicht definiert ist.

Setzen Sie das Attribut 'ibm-amqIsClientDefault' für Einträge, in denen 'ibm-amqQueueManagerName' nicht definiert ist, auf TRUE. Wird in einem MQCONN/X-Aufruf der Name des Warteschlangenmanagers nicht definiert oder wird als Name ein Stern (*) angegeben, durchsucht der Preconnect-Exit den LDAP-Server auf alle Einträge, in denen das Attribut 'ibm-amqIsClientDefault' auf TRUE gesetzt ist.

Anmerkung: Wenn das Attribut 'ibm-amqIsClientDefault' auf TRUE gesetzt ist, darf das Attribut 'ibm-amqQueueManagerName' nicht gesetzt oder definiert werden.

Zugehörige Verweise
„ibm-amqQueueManagerName, LDAP-Attribut” auf Seite 1217
Dieses Attribut gibt den Namen des Warteschlangenmanagers bzw. der Gruppe von Warteschlangenmanagern an, zu denen eine IBM MQ-Clientanwendung eine Verbindung anfordern kann.

ALW *ibm-amqHeaderCompression, LDAP-Attribut*
Bei diesem LDAP-Attribut handelt es sich um eine Liste mit Verfahren, die vom Kanal zur Komprimierung von Headerdaten unterstützt werden.

Dieses Attribut kann eine Länge von maximal 48 Zeichen haben. Es ist kein vorab vorhandenes Attribut.

Sie können für dieses Attribut nur einen Wert angeben.

Dieses Listenattribut wird in Form von Verzeichniszeichenfolgen angegeben, die durch Kommas getrennt sind. Der Wert 0 für **ibm-amqHeaderCompression** beispielsweise entspricht NONE. Alle Werte, die die maximal zulässige Länge überschreiten, werden vom Client ignoriert; so dürfen für 'ibm-amqHeaderCompression' beispielsweise nur maximal zwei Ganzzahlen in der Liste angegeben werden.

ALW *ibm-amqMessageCompression, LDAP-Attribut*
Bei diesem LDAP-Attribut handelt es sich um eine Liste mit Verfahren, die vom Kanal zur Komprimierung von Nachrichtendaten unterstützt werden.

Dieses Attribut kann eine Länge von maximal 48 Zeichen haben. Es ist kein vorab vorhandenes Attribut.

Für dieses Attribut wird die Angabe mehrerer Werte nicht unterstützt.

V 9.4.0 Dieses Listenattribut wird in Form von Verzeichniszeichenfolgen angegeben, die durch Kommas getrennt sind. Für dieses Attribut wird beispielsweise der Wert 1,2,4,16,32 angegeben, der der zugrunde liegenden Komprimierungsfolge RLE, ZLIBFAST, ZLIBHIGH, LZ4FASTund LZ4HIGHzugeordnet ist.

Alle Werte, die die maximal zulässige Länge überschreiten, werden vom Client ignoriert; so dürfen für 'ibm-amqMessageCompression' beispielsweise maximal 16 Ganzzahlen in der Liste angegeben werden.

ALW *ibm-amqSendExitUserData, LDAP-Attribut*
Dieses LDAP-Attribut gibt die Benutzerdaten an, die an den Sendeexit übergeben werden.

Für dieses LDAP-Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 999 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

Anmerkung: ibm-amqSendExitName und **ibm-amqSendExitUserData** müssen zusammen angegeben werden. Die Benutzerdaten sollten mit dem Exitnamen synchronisiert werden. Wird also eines dieser Attribute angegeben, muss das andere Attribut der Symmetrie wegen ebenfalls angegeben werden, auch wenn es keine Daten enthält.

ALW *ibm-amqSendExitName*
Dieses LDAP-Attribut gibt den Namen des Exitprogramms an, das vom Kanalsendeexit ausgeführt werden soll.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 999 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

Anmerkung: ibm-amqSendExitName und **ibm-amqSendExitUserData** müssen zusammen angegeben werden. Die Benutzerdaten müssen mit dem Exitnamen synchronisiert werden. Wird also eines dieser Attribute angegeben, muss das andere Attribut der Symmetrie wegen ebenfalls angegeben werden, auch wenn es keine Daten enthält.

ALW *ibm-amqReceiveExitUserData, LDAP-Attribut*
Dieses LDAP-Attribut gibt die Benutzerdaten an, die an den Empfangsexit übergeben werden.

Sie können eine Reihe von Empfangsexits ausführen. Die Zeichenfolge mit den Benutzerdaten wird durch Kommas und/oder Leerzeichen getrennt.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 999 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

Anmerkung: **ibm-amqReceiveExitName** und **ibm-amqReceiveExitUserData** müssen zusammen angegeben werden. Die Benutzerdaten müssen mit dem Exitnamen synchronisiert werden. Wird also eines dieser Attribute angegeben, muss das andere Attribut der Symmetrie wegen ebenfalls angegeben werden, auch wenn es keine Daten enthält.

ALW *ibm-amqReceiveExitName*
Dieses LDAP-Attribut gibt den Namen des Benutzerexitprogramms an, das vom Kanalempfangsbenutzerexit ausgeführt werden soll.

Dieses Attribut gibt eine Liste mit den Namen von Programmen an, die nacheinander ausgeführt werden sollen. Ist kein Kanalempfangsbenutzerexit aktiv, wird dieses Attribut leer gelassen.

Für dieses Attribut wird ein einzelner Zeichenfolgewert mit einer Länge von maximal 999 Zeichen angegeben; die Groß-/Kleinschreibung muss nicht beachtet werden. Es ist kein vorab vorhandenes Attribut.

Teilzeichenfolgenabgleiche werden ignoriert. Beim Teilzeichenfolgenabgleich handelt es sich um eine Abgleichsregel, die in einem Unterschema verwendet wird und das Verhalten des Attributs in einem Suchfilter angibt.

Anmerkung: **ibm-amqReceiveExitName** und **ibm-amqReceiveExitUserData** müssen zusammen angegeben werden. Die Benutzerdaten müssen mit dem Exitnamen synchronisiert werden. Wird also eines dieser Attribute angegeben, muss das andere Attribut der Symmetrie wegen ebenfalls angegeben werden, auch wenn es keine Daten enthält.

z/OS Using the sample programs for z/OS

The sample procedural applications that are delivered with IBM MQ for z/OS demonstrate typical uses of the Message Queue Interface (MQI).

About this task

IBM MQ for z/OS also provides sample data-conversion exits, described in "Datenkonvertierungsexits schreiben" on page 1035.

All the sample applications are supplied in source form; several are also supplied in executable form. The source modules include pseudocode that describes the program logic.

Note: Although some of the sample applications have basic panel-driven interfaces, they do not aim to demonstrate how to design the look and feel of your applications. For more information about how to design panel-driven interfaces for non-programmable terminals, see the *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) and its addendum (GG22-9508). These provide guidelines to help you to design applications that are consistent both within the application and across other applications.

Procedure

- Use the following links to find out more about the sample programs:
  - "Features demonstrated in the sample applications for z/OS" on page 1222
  - "Preparing and running sample applications for the batch environment on z/OS" on page 1228

- “Preparing sample applications for the TSO environment on z/OS” on page 1231
- “Preparing the sample applications for the CICS environment on z/OS” on page 1233
- “Preparing the sample application for the IMS environment on z/OS” on page 1236
- “The Put samples on z/OS” on page 1237
- “The Get samples on z/OS” on page 1239
- “The Browse sample on z/OS” on page 1242
- “The Print Message sample on z/OS” on page 1244
- “The Queue Attributes sample on z/OS” on page 1247
- “The Mail Manager sample on z/OS” on page 1248
- “The Credit Check sample on z/OS” on page 1255
- “The Message Handler sample on z/OS” on page 1266
- “The Asynchronous Put sample on z/OS” on page 1269
- “The Batch Asynchronous Consumption sample on z/OS” on page 1270
- “The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” on page 1272
- “The Publish/Subscribe sample on z/OS” on page 1274
- “The Set and Inquire message property sample on z/OS” on page 1277

Related tasks

“Beispielprogramme plattformübergreifend verwenden” on page 1112
Diese prozeduralen Beispielprogramme sind im Lieferumfang des Produkts enthalten. Sie sind in C und COBOL geschrieben und veranschaulichen die typische Verwendung der Message Queue Interface (MQI).

z/OS *Features demonstrated in the sample applications for z/OS*

This section summarizes the MQI features demonstrated in each of the sample applications, shows the programming languages in which each sample is written, and the environment in which each sample runs.

z/OS *Put samples on z/OS*

The Put samples demonstrate how to put messages on a queue using the MQPUT call.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

The program is delivered in COBOL and C, and runs in the batch and CICS environment. See Table 172 on page 1229 for the batch application and Table 179 on page 1234 for the CICS application.

z/OS *Get samples on z/OS*

The Get samples demonstrate how to get messages from a queue using the MQGET call.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

The program is delivered in COBOL and C, and runs in the batch and CICS environment. See Table 172 on page 1229 for the batch application and Table 179 on page 1234 for the CICS application.

z/OS *Browse sample on z/OS*

The Browse sample demonstrates how to use the Browse option to find a message, print it, then step through the messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET for browsing messages
- MQCLOSE
- MQDISC

The program is delivered in the COBOL, assembler, PL/I, and C languages. The application runs in the batch environment. See Table 173 on page 1230 for the batch application.

z/OS *Print Message sample on z/OS*

The Print Message sample demonstrates how to remove a message from a queue and print the data in the message, together with all the fields of its message descriptor. It can, optionally, display all of the message properties associated with each message.

By removing comment characters from two lines in the source module, you can change the program so that it browses, rather than removes, the messages on a queue. This program can usefully be used for diagnosing problems with an application that is putting messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET for removing messages from a queue (with an option to browse)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

The program is delivered in the C language. The application runs in the batch environment. See Table 174 on page 1230 for the batch application.

z/OS *Queue Attributes sample on z/OS*

The Queue Attributes sample demonstrates how to inquire about and set the values of IBM MQ for z/OS object attributes.

The application uses these MQI calls:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

The program is delivered in the COBOL, assembler, and C languages. The application runs in the CICS environment. See Table 180 on page 1234 for the CICS application.

z/OS *Mail Manager sample on z/OS*

Considerations to note when using Mail Manager sample.

The Mail Manager sample demonstrates these techniques:

- Using alias queues
- Using a model queue to create a temporary dynamic queue
- Using reply-to queues
- Using syncpoints in the CICS and batch environments
- Sending commands to the system-command input queue
- Testing return codes
- Sending messages to remote queue managers, both by using a local definition of a remote queue and by putting messages directly on a named queue at a remote queue manager

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

Three versions of the application are provided:

- A CICS application written in COBOL
- A TSO application written in COBOL
- A TSO application written in C

The TSO applications use the IBM MQ for z/OS batch adapter and include some ISPF panels.

See Table 177 on page 1232 for the TSO application, and Table 181 on page 1235 for the CICS application.

z/OS *Credit Check sample on z/OS*

This information contains points to consider when using Credit Check sample.

The Credit Check sample is a suite of programs that demonstrates these techniques:

- Developing an application that runs in more than one environment
- Using a model queue to create a temporary dynamic queue
- Using a correlation identifier
- Setting and passing context information
- Using message priority and persistence
- Starting programs by using triggering
- Using reply-to queues
- Using alias queues
- Using a dead-letter queue
- Using a namelist
- Testing return codes

The application uses these MQI calls:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET for browsing and getting messages, using the wait and signal options, and for getting a specific message
- MQINQ
- MQSET
- MQCLOSE

The sample can run as a stand-alone CICS application. However, to demonstrate how to design a message queuing application that uses the facilities provided by both the CICS and IMS environments, one module is also supplied as an IMS batch message processing program.

The CICS programs are delivered in C and COBOL. The single IMS program is delivered in C.

See Table 182 on page 1235 for the CICS application, and Table 184 on page 1237 for the IMS application.

z/OS *The Message Handler sample on z/OS*

The Message Handler sample allows you to browse, forward, and delete messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

The program is delivered in C and COBOL programming languages. The application runs under TSO. SeeTable 178 on page 1233for the TSO application.

z/OS *Distributed queuing exit samples on z/OS*

A table of source programs of Distributed queuing exit samples.

The names of the source programs of the distributed queuing exit samples are listed in the following table:

Table 170. Source for the distributed queuing exit samples

Member name	For language	Description	Supplied in library
CSQ4BAX0	Assembler	Source program	SCSQASMS
CSQ4BCX1	C	Source program	SCSQC37S
CSQ4BCX2	C	Source program	SCSQC37S
CSQ4BCX4	C	Source program	SCSQC37S

Note: The source programs are link-edited with CSQXSTUB.

z/OS *Data-conversion exit samples on z/OS*

A skeleton is provided for a data-conversion exit routine, and a sample is shipped with IBM MQ illustrating the MQXCNVC call.

The names of the source programs of the data-conversion exit samples are listed in the following table:

Table 171. Source for the data conversion exit samples (assembler language only)

Member name	Description	Supplied in library
CSQ4BAX8	Source program	SCSQASMS
CSQ4BAX9	Source program	SCSQASMS
CSQ4CAX9	Source program	SCSQASMS

Note: The source programs are link-edited with CSQASTUB.

See "Datenkonvertierungsexits schreiben" on page 1035 for more information.

z/OS *Publish/Subscribe samples on z/OS*

The Publish/Subscribe sample programs demonstrate the use of the publish and subscribe features in IBM MQ.

There are four C and two COBOL programming language sample programs demonstrating how to program to the IBM MQ Publish/Subscribe interface.

The applications use these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

The Public/Subscribe sample programs are delivered in the C and COBOL programming languages. The sample applications run in the batch environment. See Publish/Subscribe samples for the batch applications.

z/OS *Configuring a queue manager to accept client connections on z/OS*

Before you can run the sample applications, you must first create a queue manager. You can then configure the queue manager to securely accept incoming connection requests from applications that are running in client mode.

Before you begin

Ensure the queue manager already exists and has been started. Determine whether channel authentication records are already enabled by issuing the MQSC command:

```
DISPLAY QMGR CHLAUTH
```

Important: This task expects that channel authentication records are enabled. If this is a queue manager used by other users and applications, changing this setting will affect all other users and applications. If

your queue manager does not make use of channel authentication records then step 4 can be replaced with an alternate authentication method (for example a security exit) which sets the MCAUSER to the *non-privileged-user-id* you will obtain in step “1” on page 1227.

You must know which channel name your application expects to use so that the application can be permitted to use the channel. You must also know which objects, for example queues or topics, your application expects to use so that your application can be permitted to use them.

About this task

This task creates a non-privileged user ID to be used for a client application which connects to the queue manager. Access is granted for the client application only to be able to use the channel it needs and the queue it needs by use of this user ID.

Procedure

1. Obtain a user ID on the system your queue manager is running on.

   For this task this user ID must not be a privileged administrative user. This user ID is the authority under which the client connection will run on the queue manager.
2. Start a listener program.
   a) Ensure that your channel initiator is started. If not, start it by issuing the **START CHINIT** command.
   b) Start the listener program by issuing the following command:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   where *nnnn* is your chosen port number.
3. If your application uses the SYSTEM.DEF.SVRCONN then this channel is already defined. If your application uses another channel, create it by issuing the MQSC command:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name is the name of your channel.
4. Create a channel authentication rule allowing only the IP address of your client system to use the channel by issuing the MQSC command:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   where

   channel-name is the name of your channel.
   client-machine-IP-address is the IP address of your client system. If your sample client application is running on the same machine as the queue manager then use an IP address of '127.0.0.1' if your application is going to connect using 'localhost'. If several different client machines are going to connect in, you can use a pattern or a range instead of a single IP address. See Generic IP addresses for details.
   non-privileged-user-id is the user ID you obtained in step “1” on page 1227
5. If your application uses the SYSTEM.DEFAULT.LOCAL.QUEUE, then this queue is already defined. If your application uses another queue, create it by issuing the MQSC command:

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

   where *queue-name* is the name of your queue.
6. Grant access to connect to and inquire the queue manager:

a) Ensure that your channel initiator is started. If not, start the channel initiator by issuing the START CHINIT command.

b) Start a TCP listener, for example issue the following command:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

where *nnnn* is your chosen port number.

7. If your application is a point-to-point application, that is it makes use of queues, grant access to allow inquiring and the putting and getting messages using your queue by the user ID to be used, by issuing the MQSC commands:

   Issue the RACF commands:

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

   where

   qmgr-name is the name of your queue manager

   queue-name is the name of your queue.

   non-privileged-user-id is the user ID you obtained in step "1" on page 1227

8. If your application is a publish/subscribe application, that is it makes use of topics, grant access to allow publishing and subscribing using your topic by the user ID to be used, by issuing the following RACF commands:

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id) AC
CESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id) AC
CESS(UPDATE)
```

   where

   qmgr-name is the name of your queue manager

   non-privileged-user-id is the user ID you obtained in step "1" on page 1227

   This will give *non-privileged-user-id* access to any topic in the topic tree, alternatively, you can define a topic object using **DEFINE TOPIC** and grant accesses only to the part of the topic tree referenced by that topic object. For more information, see Controlling user access to topics.

What to do next

Your client application can now connect to the queue manager and put or get messages using the queue.

Related concepts

z/OS Authority to work with IBM MQ objects on z/OS

Related reference

SET CHLAUTH

DEFINE CHANNEL

DEFINE QLOCAL

SET AUTHREC

z/OS *Preparing and running sample applications for the batch environment on z/OS*

To prepare a sample application that runs in the batch environment, perform the same steps that you would when building any batch IBM MQ for z/OS application.

These steps are listed in “Building z/OS batch applications” on page 1077.

Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQLOAD load library.

Note: The assembler language version of the Browse sample uses data control blocks (DCBs), so you must link-edit it using RMODE(24).

The library members to use are listed in Table 172 on page 1229, Table 173 on page 1230, Table 174 on page 1230, and Table 175 on page 1230.

You must edit the run JCL supplied for the samples that you want to use (see Table 172 on page 1229, Table 173 on page 1230, Table 174 on page 1230, and Table 175 on page 1230).

The PARM statement in the supplied JCL contains a number of parameters that you need to modify. To run the C sample programs, separate the parameters by spaces; to run the assembler, COBOL, and PL/I sample programs, separate them by commas. For example, if the name of your queue manager is CSQ1 and you want to run the application with a queue named LOCALQ1, in the COBOL, PL/I, and assembler-language JCL, your PARM statement should look like this:

```
PARM=(CSQ1,LOCALQ1)
```

In the C language JCL, your PARM statement should look like this:

```
PARM=('CSQ1 LOCALQ1')
```

You are now ready to submit the jobs.

z/OS *Names of the sample batch applications on z/OS*

A summary of the programs that are supplied for sample batch applications.

The batch application programs are summarized in the following tables:

- Table 172 on page 1229 Put and Get samples
- Table 173 on page 1230 Browse sample
- Table 174 on page 1230 Print message sample
- Table 175 on page 1230 Publish/Subscribe samples
- Table 176 on page 1231 Other samples

Table 172. Batch Put and Get samples

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4BCJ1	C	Get source program	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	Put source program	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	Sample run JCL for CSQ4BCJ1 and CSQBCK1	SCSQPROC	None
CSQ4BVJ1	COBOL	Get source program	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Put source program	SCSQCOBS	SCSQLOAD

Table 172. Batch Put and Get samples (continued)				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4BVJR	COBOL	Sample run JCL for CSQBVJ1 and CSQBVK1	SCSQPROC	None

Table 173. Batch Browse sample				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4BVA1	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	Sample run JCL for CSQ4BVA1	SCSQPROC	None
CSQ4BAA1	Assembler	Source program	SCSQASMS	SCSQLOAD
CSQ4BAAR	Assembler	Sample run JCL for CSQ4BAA1	SCSQPROC	None
CSQ4BCA1	C	Source program	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	Sample run JCL for CSQ4BCA1	SCSQPROC	None
CSQ4BPA1	PL/I	Source program	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	Sample run JCL for CSQ4BPA1	SCSQPROC	None

Table 174. Batch Print Message sample (C language only)			
Member name	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4BCG1	Source program	SCSQC37S	SCSQLOAD
CSQ4BCGR	Sample run JCL for CSQ4BCG1	SCSQPROC	None
CSQ4BCL1	Browse source program	SCSQC37S	SCSQLOAD
CSQ4BCLR	Sample run JCL for CSQ4BCL1	SCSQPROC	None

Table 175. Publish/Subscribe samples					
Member name	**For language**	**Description**	**Source file supplied in library**	**JCL in SCSQPROC**	**Executable file supplied in library**
CSQ4BCP1	C	Publish to topic source program	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	Subscribe to topic and get messages source program	SCSQC37S	CSQ4BCPS	SCSQLOAD

Table 175. Publish/Subscribe samples (continued)					
Member name	**For language**	**Description**	**Source file supplied in library**	**JCL in SCSQPROC**	**Executable file supplied in library**
CSQ4BCP3	C	Subscribe to topic using a user provided destination and get messages source program	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	Subscribe to topic using extended options and get messages source program	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	Publish to topic source program	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	Subscribe to topic and get messages source program	SCSQCOBS	CSQ4BVPS	SCSQLOAD

Table 176. Other samples					
Member name	**For language**	**Description**	**Source file supplied in library**	**JCL in SCSQPROC**	**Executable file supplied in library**
CSQ4BCS1	C	Asynchronous consumption source program	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	Asynchronous Put, and Check status source program	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	Inquire message properties source program	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	Set message properties source program	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS *Preparing sample applications for the TSO environment on z/OS*

To prepare a sample application that runs in the TSO environment, perform the same steps that you would when building any batch IBM MQ for z/OS application.

These steps are listed in “Building z/OS batch applications” on page 1077. The library members to use are listed in Table 177 on page 1232.

Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQLOAD load library.

For the Mail Manager sample application, ensure that the queues that it uses are available on your system. They are defined in the member **thlqual**.SCSQPROC(CSQ4CVD). To ensure that these queues are always available, you could add these members to your CSQINP2 initialization input data set, or use the CSQUTIL program to load these queue definitions.

z/OS *Names of the sample TSO applications on z/OS*

Information about the names of the programs that are supplied for each of the sample TSO applications, and the libraries where the source, JCL, and, for the Message Handler sample only, the executable files reside.

The TSO application programs are summarized in the following tables:

- Table 177 on page 1232 Mail manager sample
- Table 178 on page 1233 Message handler sample

These samples use ISPF panels. You must therefore include the ISPF stub, ISPLINK, when you link-edit the programs.

Table 177. TSO Mail Manager sample

Member name	For language	Description	Source file supplied in library
CSQ4CVD	independent	IBM MQ for z/OS object definitions	SCSQPROC
CSQ40	independent	ISPF messages	SCSQMSGE
CSQ4RVD1	COBOL	CLIST to initiate CSQ4TVD1	SCSQCLST
CSQ4TVD1	COBOL	Source program for Menu program	SCSQCOBS
CSQ4TVD2	COBOL	Source program for Get Mail program	SCSQCOBS
CSQ4TVD4	COBOL	Source program for Send Mail program	SCSQCOBS
CSQ4TVD5	COBOL	Source program for Nickname program	SCSQCOBS
CSQ4VDP1-6	COBOL	Panel definitions	SCSQPNLA
CSQ4VD0	COBOL	Data definition	SCSQCOBC
CSQ4VD1	COBOL	Data definition	SCSQCOBC
CSQ4VD2	COBOL	Data definition	SCSQCOBC
CSQ4VD4	COBOL	Data definition	SCSQCOBC
CSQ4RCD1	C	CLIST to initiate CSQ4TCD1	SCSQCLST
CSQ4TCD1	C	Source program for Menu program	SCSQC37S
CSQ4TCD2	C	Source program for Get Mail program	SCSQC37S
CSQ4TCD4	C	Source program for Send Mail program	SCSQC37S
CSQ4TCD5	C	Source program for Nickname program	SCSQC37S
CSQ4CDP1-6	C	Panel definitions	SCSQPNLA
CSQ4TC0	C	Include file	SCSQC370

Table 178. TSO Message Handler sample				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4TCH0	C	Data definition	SCSQC370	None
CSQ4TCH1	C	Source program	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	Source program	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	Source program	SCSQC37S	SCSQLOAD
CSQ4RCH1	C and COBOL	CLIST to initiate CSQ4TCH1 or CSQ4TVH1	SCSQCLST	None
CSQ4CHP1	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP2	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP3	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP9	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4TVH0	COBOL	Data definition	SCSQCOBC	None
CSQ4TVH1	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	Source program	SCSQCOBS	SCSQLOAD

z/OS *Preparing the sample applications for the CICS environment on z/OS*

Before you run the CICS sample programs, log on to CICS using a LOGMODE of 32702. This is because the sample programs have been written to use a 3270 mode 2 screen.

To prepare a sample application that runs in the CICS environment, perform the following steps:

1. Create the symbolic description map and the physical screen map for the sample by assembling the BMS screen definition source (supplied in library **thlqual**.SCSQMAPS, where **thlqual** is the high-level qualifier used by your installation). When you name the maps, use the name of the BMS screen definition source (not available for Put and Get sample programs), but omit the last character of that name.
2. Perform the same steps that you would when building any CICS IBM MQ for z/OS application. These steps are listed in "Building CICS applications in z/OS" on page 1080. The library members to use are listed in Table 179 on page 1234, Table 180 on page 1234, Table 181 on page 1235, and Table 182 on page 1235.

   Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQCICS load library.
3. Identify the map set, programs, and transaction to CICS by updating the CICS system definition (CSD) data set. The definitions that you require are in the member **thlqual**.SCSQPROC(CSQ4S100). For guidance on how to do this, see *The CICS-IBM MQ Adapter* section in the CICS Transaction Server for z/OS 4.1 product documentation at: CICS Transaction Server for z/OS 4.1, The CICS-IBM MQ adapter.

   Note: For the Credit Check sample application, you get an error message at this stage if you have not already created the VSAM data set that the sample uses.
4. For the Credit Check and Mail Manager sample applications, ensure that the queues that they use are available on your system. For the Credit Check sample, they are defined in the member **thlqual**.SCSQPROC(CSQ4CVB) for COBOL, and **thlqual**.SCSQPROC(CSQ4CCB) for C. For the Mail Manager sample, they are defined in the member **thlqual**.SCSQPROC(CSQ4CVD). To ensure that these

queues are always available, you could add these members to your CSQINP2 initialization input data set, or use the CSQUTIL program to load these queue definitions.

For the Queue Attributes sample application, you could use one or more of the queues that are supplied for the other sample applications. Alternatively, you could use your own queues. However, in the form that it is supplied, this sample works only with queues that have the characters CSQ4SAMP in the first eight bytes of their name.

z/OS *Names of the sample CICS applications on z/OS*

This topic provides a summary of the programs supplied for sample CICS applications.

The CICS application programs are summarized in the following tables:

- Table 179 on page 1234 Put and Get samples
- Table 180 on page 1234 Queue Attributes sample
- Table 181 on page 1235 Mail Manager sample (COBOL only)
- Table 182 on page 1235 Credit Check sample
- Table 183 on page 1236 Asynchronous Consumption and Publish/Subscribe samples

Table 179. CICS Put and Get samples

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4CCK1	C	Put source program	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Get source program	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Get source program	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Put source program	SCSQCOBS	SCSQCICS
CSQ4S100	independent	CICS system definition data set	SCSQPROC	None

Table 180. CICS Queue Attributes sample

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4CVC1	COBOL	Source program	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Message definition	SCSQCOBC	None
CSQ4VCMS	COBOL	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4CAC1	Assembler	Source program	SCSQASMS	SCSQCICS
CSQ4AMSG	Assembler	Message definition	SCSQMACS	None
CSQ4ACMS	Assembler	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4CCC1	C	Source program	SCSQC37S	SCSQCICS
CSQ4CMSG	C	Message definition	SCSQC370	None
CSQ4CCMS	C	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)

Table 180. CICS Queue Attributes sample (continued)				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4S100	independent	CICS system definition data set	SCSQPROC	None

Table 181. CICS Mail Manager sample (COBOL only)		
Member name	**Description**	**Source file supplied in library**
CSQ4CVD	IBM MQ for z/OS object definitions	SCSQPROC
CSQ4CVD1	Source for Menu program	SCSQCOBS
CSQ4CVD2	Source for Get Mail program	SCSQCOBS
CSQ4CVD3	Source for Display Message program	SCSQCOBS
CSQ4CVD4	Source for Send Mail program	SCSQCOBS
CSQ4CVD5	Source for Nickname program	SCSQCOBS
CSQ4VDMS	BMS screen definition source	SCSQMAPS
CSQ4S100	CICS system definition data set	SCSQPROC
CSQ4VD0	Data definition	SCSQCOBC
CSQ4VD3	Data definition	SCSQCOBC
CSQ4VD4	Data definition	SCSQCOBC

Table 182. CICS Credit Check sample			
Member name	**For language**	**Description**	**Source file supplied in library**
CSQ4CVB	independent	IBM MQ object definitions	SCSQPROC
CSQ4CCB	independent	IBM MQ object definitions	SCSQPROC
CSQ4CVB1	COBOL	Source for user-interface program	SCSQCOBS
CSQ4CVB2	COBOL	Source for credit application manager	SCSQCOBS
CSQ4CVB3	COBOL	Source for checking-account program	SCSQCOBS
CSQ4CVB4	COBOL	Source for distribution program	SCSQCOBS
CSQ4CVB5	COBOL	Source for agency-query program	SCSQCOBS
CSQ4CCB1	C	Source for user-interface program	SCSQC37S
CSQ4CCB2	C	Source for credit application manager	SCSQC37S
CSQ4CCB3	C	Source for checking-account program	SCSQC37S
CSQ4CCB4	C	Source for distribution program	SCSQC37S
CSQ4CCB5	C	Source for agency-query program	SCSQC37S
CSQ4CB0	C	Include file	SCSQC370
CSQ4CBMS	C	BMS screen definition source	SCSQMAPS

Table 182. CICS Credit Check sample (continued)			
Member name	**For language**	**Description**	**Source file supplied in library**
CSQ4VBMS	COBOL	BMS screen definition source	SCSQMAPS
CSQ4VB0	COBOL	Data definition	SCSQCOBC
CSQ4VB1	COBOL	Data definition	SCSQCOBC
CSQ4VB2	COBOL	Data definition	SCSQCOBC
CSQ4VB3	COBOL	Data definition	SCSQCOBC
CSQ4VB4	COBOL	Data definition	SCSQCOBC
CSQ4VB5	COBOL	Data definition	SCSQCOBC
CSQ4VB6	COBOL	Data definition	SCSQCOBC
CSQ4VB7	COBOL	Data definition	SCSQCOBC
CSQ4VB8	COBOL	Data definition	SCSQCOBC
CSQ4BAQ	independent	Source for VSAM data set	SCSQPROC
CSQ4FILE	independent	JCL to build VSAM data set used by CSQ4CVB3	SCSQPROC
CSQ4S100	independent	CICS system definition data set	SCSQPROC

Table 183. CICS Asynchronous Consumption and Publish/Subscribe samples		
Member name	**Description**	**Source file supplied in library**
CSQ4CVCN	Source for Simple Message Consumption program	SCSQCOBS
CSQ4CVCT	Source for Control Message Consumption program	SCSQCOBS
CSQ4CVEV	Source for Event Handler program	SCSQCOBS
CSQ4CVPT	Source for Message Put Client program	SCSQCOBS
CSQ4CVRG	Source for Registration Client program	SCSQCOBS
CSQ4S100	CICS System Definition data set	SCSQPROC

z/OS *Preparing the sample application for the IMS environment on z/OS*

Part of the Credit Check sample application can run in the IMS environment.

To prepare this part of the application to run with the CICS sample, first perform the steps described in “Preparing the sample applications for the CICS environment on z/OS” on page 1233.

Then perform the following steps:

1. Perform the same steps that you would when building any IMS IBM MQ for z/OS application. These steps are listed in “Building IMS (BMP or MPP) applications” on page 1081. The library members to use are listed in Table 184 on page 1237.
2. Identify the application program and database to IMS. Samples are provided with PSBGEN, DBDGEN, ACB definition, IMSGEN, and IMSDALOC statements to enable this.

3. Load the database CSQ4CA by tailoring and running the sample JCL provided for this purpose (CSQ4ILDB). This JCL loads the database with data from the file CSQ4BAQ. Update the IMS control region with a DD statement for the database CSQ4CA.
4. Start the checking-account program as a batch message processing (BMP) program by tailoring and running the sample JCL provided for this purpose. This JCL starts a batch-oriented BMP program. To run the program as a message-oriented BMP program, remove the comment characters from the line in the JCL that contains the IN= statement.

z/OS *Names of the sample IMS application on z/OS*

This information provides a table with the list of the sources and JCLs that are supplied for the Credit Check sample IMS application.

Table 184. Source and JCL for the Credit Check IMS sample (C only)

Member name	Description	Supplied in library
CSQ4CVB	IBM MQ object definitions	SCSQPROC
CSQ4ICB3	Source for checking-account program	SCSQC37S
CSQ4ICBL	Source for loading the checking-account database	SCSQC37S
CSQ4CBI	Data definition	SCSQC370
CSQ4PSBL	PSBGEN JCL for database-load program	SCSQPROC
CSQ4PSB3	PSBGEN JCL for checking-account program	SCSQPROC
CSQ4DBDS	DBDGEN JCL for database CSQ4CA	SCSQPROC
CSQ4GIMS	IMSGEN macro definitions for CSQ4IVB3 and CSQ4CA	SCSQPROC
CSQ4ACBG	Application control block (ACB) definition for CSQ4IVB3	SCSQPROC
CSQ4BAQ	Source for database	SCSQPROC
CSQ4ILDB	Sample run JCL for database-load job	SCSQPROC
CSQ4ICBR	Sample run JCL for checking-account program	SCSQPROC
CSQ4DYNA	IMSDALOC macro definitions for database	SCSQPROC

z/OS ***The Put samples on z/OS***

The Put sample programs put messages on a queue using the MQPUT call.

The source programs are supplied in C and COBOL in the batch and CICS environments (see Table 172 on page 1229 and Table 179 on page 1234).

Design of the Put sample

The flow through the program logic is:

1. Connect to the queue manager using the MQCONN call. If this call fails, print the completion and reason codes and stop processing.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQCONN call; if you do, it returns DEF_HCONN. You can use the connection handle MQHC_DEF_HCONN for the MQI calls that follow.
2. Open the queue using the MQOPEN call with the MQOO_OUTPUT option. On input to this call, the program uses the connection handle that is returned in step “1” on page 1239. For the object descriptor structure (MQOD), it uses the default values for all fields except the queue name field, which is passed as a parameter to the program. If the MQOPEN call fails, print the completion and reason codes and stop processing.
3. Create a loop within the program issuing MQPUT calls until the required number of messages are put on the queue. If an MQPUT call fails, the loop is abandoned early, no further MQPUT calls are attempted, and the completion and reason codes are returned.
4. Close the queue using the MQCLOSE call with the object handle returned in step “2” on page 1240. If this call fails, print the completion and reason codes.
5. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “1” on page 1239. If this call fails, print the completion and reason codes.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQDISC call.

z/OS *The Put samples for the batch environment on z/OS*

Use this topic when considering Put samples for the batch environment.

To run the samples, edit and run the sample JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1228.

The programs take the following parameters in an EXEC PARM, separated by spaces in C and commas in COBOL:

1. The name of the queue manager (4 characters)
2. The name of the target queue (48 characters)
3. The number of messages (up to 4 digits)
4. The padding character to write in the message (1 character)
5. The number of characters to write in the message (up to 4 digits)
6. The persistence of the message (1 character: P for persistent or N for nonpersistent)

If you enter any of the these parameters wrongly, you receive appropriate error messages.

Any messages from the samples are written to the SYSPRINT data set.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. However, these differences are minimized if you use the layout of the parameters shown in the sample run JCL, CSQ4BCJR, and CSQ4BVJR. None of the differences relate to the MQI.
- CSQ4BCK1 allows you to enter more than four digits for the number of messages sent and the length of the messages.
- For the two numeric fields, enter any digit in the range 1 through 9999. The value that you enter should be a positive number. For example, to put a single message, you can enter 1, 01, 001, or 0001 as the value. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program sends a 1-byte message, but the C program receives an error.
- For both programs, CSQ4BCK1 and CSQ4BVK1, you must enter P in the persistence parameter, ++PER++, if you want the message to be persistent. If you fail to do so, the message will be nonpersistent.

z/OS *The Put samples for the CICS environment on z/OS*

Use this topic when considering Put samples for the CICS environment.

The transactions take the following parameters separated by commas:

1. The number of messages (up to 4 digits)
2. The padding character to write in the message (1 character)
3. The number of characters to write in the message (up to 4 digits)
4. The persistence of the message (1 character: P for persistent or N for nonpersistent)
5. The name of the target queue (48 characters)

If you enter any of these parameters wrongly, you receive appropriate error messages.

For the COBOL sample, invoke the Put sample in the CICS environment by entering:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

For the C sample, invoke the Put sample in the CICS environment by entering:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Any messages from the samples are displayed on the screen.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. None of the differences relate to the MQI.
- If you enter a queue name that is longer than 48 characters, its length is truncated to the maximum of 48 characters but no error message is returned.
- Before entering the transaction, press the CLEAR key.
- For the two numeric fields, enter any number in the range 1 through 9999. The value that you enter should be a positive number. For example, to put a single message, you can enter the value 1, 01, 001, or 0001. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program sends a 1-byte message, and the C program abends with an error from malloc().
- For both programs, CSQ4CCK1 and CSQ4CVK1, enter P in the persistence parameter if you want the message to be persistent. For non-persistent messages, enter N in the persistence parameter. If you enter any other value you receive an error message.
- The messages are put in syncpoint because default values are used for all parameters except those set during program invocation.

z/OS *The Get samples on z/OS*

The Get sample programs get messages from a queue using the MQGET call.

The source programs are supplied in C and COBOL in the batch and CICS environments (see Table 172 on page 1229 and Table 179 on page 1234).

z/OS *Design of the Get sample on z/OS*

Learn about the design of the Get sample, and some usage notes to consider.

The flow through the program logic is:

1. Connect to the queue manager using the MQCONN call. If this call fails, print the completion and reason codes and stop processing.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQCONN call; if you do, it returns DEF_HCONN. You can use the connection handle MQHC_DEF_HCONN for the MQI calls that follow.

2. Open the queue using the MQOPEN call with the MQOO_INPUT_SHARED and MQOO_BROWSE options. On input to this call, the program uses the connection handle that is returned in step “1” on page 1239. For the object descriptor structure (MQOD), it uses the default values for all fields except the queue name field, which is passed as a parameter to the program. If the MQOPEN call fails, print the completion and reason codes and stop processing.
3. Create a loop within the program issuing MQGET calls until the required number of messages are retrieved from the queue. If an MQGET call fails, the loop is abandoned early, no further MQGET calls are attempted, and the completion and reason codes are returned. The following options are specified on the MQGET call:
   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST and MQGMO_BROWSE_NEXT

   For a description of these options, see MQGET. For each message, the message number is printed followed by the length of the message and the message data.
4. Close the queue using the MQCLOSE call with the object handle returned in step “2” on page 1240. If this call fails, print the completion and reason codes.
5. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “1” on page 1239. If this call fails, print the completion and reason codes.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQDISC call.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. However, these differences are minimized if you use the layout of the parameters shown in the sample run JCL, CSQ4BCJR, and CSQ4BVJR,. None of the differences relate to the MQI.
- CSQ4BCJ1 allows you to enter more than four digits for the number of messages retrieved.
- Messages longer than 64 KB are truncated.
- CSQ4BCJ1 can only correctly display character messages because it only displays until the first NULL (\0) character is displayed.
- For the numeric number-of-messages field, enter any digit in the range 1 through 9999. The value that you enter should be a positive number. For example, to get a single message, you can enter 1, 01, 001, or 0001 as the value. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program retrieves one message, but the C program does not retrieve any messages.
- For both programs, CSQ4BCJ1 and CSQ4BVJ1, enter B in the get parameter, ++GET++, if you want to browse the messages.
- For both programs, CSQ4BCJ1 and CSQ4BVJ1, enter S in the syncpoint parameter, ++SYNC++, for messages to be retrieved in syncpoint.

z/OS *The Get samples for the batch environment on z/OS*

To run the samples, edit and run the sample JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1228.

The programs take the following parameters in an EXEC PARM, separated by spaces in C and commas in COBOL:

1. The name of the queue manager (4 characters)
2. The name of the target queue (48 characters)
3. The number of messages to get (up to 4 digits)
4. The browse/get message option (1 character: B to browse or D to destructively get the messages)

5. The syncpoint control (1 character: S for syncpoint or N for no syncpoint)

If you enter any of these parameters incorrectly, you receive appropriate error messages.

Output from the samples is written to the SYSPRINT data set:

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *The Get samples for the CICS environment on z/OS*

Special considerations for the Get samples for the CICS environment.

The transactions take the following parameters in an EXEC PARM, separated by commas:

1. The number of messages to get (up to four digits)
2. The browse/get message option (one character: B to browse or D to destructively get the messages)
3. The syncpoint control (one character: S for syncpoint or N for no syncpoint)
4. The name of the target queue (48 characters)

If you enter any of these parameters incorrectly, you receive appropriate error messages.

For the COBOL sample, invoke the Get sample in the CICS environment by entering:

```
MVGT,9999,B,S,QUEUE.NAME
```

For the C sample, invoke the Get sample in the CICS environment by entering:

```
MCGT,9999,B,S,QUEUE.NAME
```

When the messages are retrieved from the queue, they are put on a CICS temporary storage queue with the same name as the CICS transaction (for example, MCGT for the C sample).

Here is example output of the Get samples:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE *********************
```

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. None of the differences relate to the MQI.
- If you enter a queue name that is longer than 48 characters, its length is truncated to the maximum of 48 characters but no error message is returned.
- Before entering the transaction, press the CLEAR key.
- CSQ4CCJ1 can only correctly display character messages because it only displays until the first NULL (\0) character is displayed.

- For the numeric field, enter any number in the range 1 through 9999. The value that you enter should be a positive number. For example, to get a single message, you can enter the value 1, 01, 001, or 0001. If you enter a nonnumeric or negative value, you might receive an error.
- Messages longer than 24 526 bytes in C and 9 950 bytes in COBOL are truncated. This is due to the way that the CICS temporary storage queues are used.
- For both programs, CSQ4CCK1 and CSQ4CVK1, enter B in the get parameter if you want to browse the messages, otherwise enter D. This performs destructive MQGET calls. If you enter any other value you receive an error message.
- For both programs, CSQ4CCJ1 and CSQ4CVJ1, enter S in the syncpoint parameter to retrieve messages in syncpoint. If you enter N in the syncpoint parameter, the MQGET calls are issued out of syncpoint. If you enter any other value you receive an error message.

z/OS *The Browse sample on z/OS*

The Browse sample is a batch application that demonstrates how to browse messages on a queue using the MQGET call.

The application steps through all the messages in a queue, printing the first 80 bytes of each one. You could use this application to look at the messages on a queue without changing them.

Source programs and sample run JCL are supplied in the COBOL, assembler, PL/I, and C languages (see Table 173 on page 1230).

To start the application, edit and run the sample run JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1228. You can look at messages on one of your own queues by specifying the name of the queue in the run JCL.

When you run the application (and there are some messages on the queue), the output data set looks this:

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

If there are no messages on the queue, the data set contains the headings and the `End of report` message only. If an error occurs with any of the MQI calls, the completion and reason codes are added to the output data set.

z/OS *Design of the Browse sample on z/OS*

The Browse sample application uses a single program module; one is provided in each of the supported programming languages.

The flow through the program logic is:

1. Open a print data set and print the title line of the report. Check that the names of the queue manager and queue have been passed from the run JCL. If both names have been passed, print the lines of the report that contain the names. If they have not, print an error message, close the print data set, and stop processing.

   The way that the program tests the parameters it is passed from the JCL depends on the language in which the program is written; for more information, see “Language-dependent design considerations on z/OS” on page 1243.

2. Connect to the queue manager using the MQCONN call. If this call is not successful, print the completion and reason codes, close the print data set, and stop processing.
3. Open the queue using the MQOPEN call with the MQOO_BROWSE option. On input to this call, the program uses the connection handle returned in step “2” on page 1243. For the object descriptor structure (MQOD), it uses the default values for all the fields except the queue name (which was passed in step “1” on page 1242). If this call is not successful, print the completion and reason codes, close the print data set, and stop processing.
4. Browse the first message on the queue, using the MQGET call. On input to this call, the program specifies:
   - The connection and queue handles from steps “2” on page 1243 and “3” on page 1243
   - An MQMD structure with all fields set to their initial values
   - Two options:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - A buffer of size 80 bytes to hold the data copied from the message

   The MQGMO_ACCEPT_TRUNCATED_MSG option allows the call to complete even if the message is longer than the 80-byte buffer specified in the call. If the message is longer than the buffer, the message is truncated to fit the buffer, and the completion and reason codes are set to show this. The sample was designed so that messages are truncated to 80 characters to make the report easy to read. The buffer size is set by a `DEFINE` statement, so you can easily change it if you want to.
5. Perform the following loop until the MQGET call fails:
   a. Print a line of the report showing:
      - The sequence number of the message (this is a count of the browse operations).
      - The true length of the message (not the truncated length). This value is returned in the `DataLength` field of the MQGET call.
      - The first 80 bytes of the message data.

   b. Reset the `MsqId` and `CorrelId` fields of the MQMD structure to nulls

   c. Browse the next message, using the MQGET call with these two options:
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. If the MQGET call fails, test the reason code to see if the call has failed because the browse cursor has got to the end of the queue. In this case, print the `End of report` message and go to step “7” on page 1243 ; otherwise, print the completion and reason codes, close the print data set, and stop processing.
7. Close the queue using the MQCLOSE call with the object handle returned in step “3” on page 1243.
8. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “2” on page 1243.
9. Close the print data set and stop processing.

z/OS *Language-dependent design considerations on z/OS*

Source modules are provided for the Browse sample in four programming languages.

There are two main differences between the source modules:

- When testing the parameters passed from the run JCL, the COBOL, PL/I, and assembler-language modules search for the comma character (,). If the JCL passes `PARM=(,LOCALQ1)`, the application attempts to open queue LOCALQ1 on the default queue manager. If there is no name after the comma (or no comma), the application returns an error. The C module does not search for the comma character. If the JCL passes a single parameter (for example, `PARM=('LOCALQ1')`), the C module uses this as a queue name on the default queue manager.

- To keep the assembler-language module simple, it uses the date format *yy/ddd* (for example, 05/116) when it creates the print report. The other modules use the calendar date in *mm/dd/yy* format.

z/OS *The Print Message sample on z/OS*

The Print Message sample is a batch application that demonstrates how to remove all the messages from a queue using the MQGET call.

The Print Message sample uses three parameters:

1. The name of the queue manager
2. The name of the source queue
3. An optional parameter for properties

It also prints, for each message, the fields of the message descriptor, followed by the message data. The program prints the data both in hexadecimal and as characters (if they are printable). If a character is not printable, the program replaces it with a period character (.). You can use the program when diagnosing problems with an application that is putting messages on a queue.

Permissible values for the property parameter are:

Table 185. Zulässige Werte für den Eigenschaftenparameter

Wert	Verhalten
0	Standardverhalten. Die Eigenschaften, die der Anwendung bereitgestellt werden, hängen vom Warteschlangenattribut **PropertyControl** ab, von dem die Nachricht abgerufen wird.

1	Ein Nachrichtenhandle wird erstellt und mit MQGET verwendet. Eigenschaften der Nachricht, ausgenommen jene, die im Nachrichtendeskriptor (oder in der Erweiterung) enthalten sind, werden dem Nachrichtendeskriptor in einer ähnlichen Weise angezeigt. For example:
	`****Message properties****`
	`  property name: property value`
	Falls keine Eigenschaften verfügbar sind:
	`****Message properties****`
	`  None`
	Numerische Werte werden mittels `printf` angezeigt, Zeichenfolgewerte stehen in einfachen Anführungszeichen und Bytefolgen stehen zwischen einem X und einfachen Anführungszeichen, wie für den Nachrichtendeskriptor.

2	MQGMO_NO_PROPERTIES wird angegeben, sodass nur Nachrichtendeskriptoreigenschaften zurückgegeben werden.
3	MQGMO_PROPERTIES_FORCE_MQRFH2 ist angegeben, sodass alle Eigenschaften in den Nachrichtendaten zurückgegeben werden.
4	MQGMO_PROPERTIES_FORCE_MQRFH2 ist angegeben, sodass alle Eigenschaften zurückgegeben werden können, wenn eine IBM MQ-Eigenschaft enthalten ist. Andernfalls werden die Eigenschaften gelöscht.

You can change the application so that it browses the messages, rather than removing them from the queue. To do this, compile with the option of -DBROWSE, to define the BROWSE macro, as indicated in “Design of the Print Message sample on z/OS” on page 1245. Executable code is provided for you in the SCSQLOAD library. Module CSQ4BCG0 is built with -DBROWSE; module CSQ4BCG1 destructively reads the queue.

The application has a single source program, which is written in the C language. Sample run JCL code is also supplied (see Table 174 on page 1230).

To start the application, edit and run the sample run JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1228. When you run the application (and there are some messages on the queue), the output data set looks like that in Figure 139 on page 1245.

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR   '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                        '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6           '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '   '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

Figure 139. Example of a report from the Print Message sample application

z/OS *Design of the Print Message sample on z/OS*

The Print Message sample application uses a single program written in the C language.

The flow through the program logic is:

1. Check that the names of the queue manager and queue have been passed from the run JCL. If they have not, print an error message and stop processing.
2. Connect to the queue manager using the MQCONN call. If this call is not successful, print the completion and reason codes and stop processing; otherwise print the name of the queue manager.
3. Open the queue using the MQOPEN call with the MQOO_INPUT_SHARED option.

   Note: If you want the application to browse the messages rather than remove them from the queue, compile the sample with -DBROWSE, or, add `#define BROWSE` at the beginning of the source. When

you do this, the macro preprocessor adds the line in the program that selects the MQOO_BROWSE option in the compilation.

On input to this call, the program uses the connection handle returned in step “2” on page 1245. For the object descriptor structure (MQOD), it uses the default values for all the fields except the queue name (which was passed in step “1” on page 1245). If this call is not successful, print the completion and reason codes and stop processing; otherwise, print the name of the queue.

4. If you use a message handle to obtain the message properties use MQCRTMH to create such a handle for use with subsequent MQGET calls. If this call is not successful, print the completion and reason codes and stop processing.
5. Set the get message options to reflect the request action for any message properties.
6. Perform the following loop until the MQGET call fails:
   a. Initialize the buffer to blanks so that the message data does not get corrupted by any data already in the buffer.
   b. Set the `MsgId` and `CorrelId` fields of the MQMD structure to nulls so that the MQGET call selects the first message from the queue.
   c. Get a message from the queue, using the MQGET call. On input to this call, the program specifies:
      - The connection and object handles from steps “2” on page 1245 and “3” on page 1245.
      - An MQMD structure with all fields set to their initial values. (`MsgId` and `CorrelId` are reset to nulls for each MQGET call.)
      - The option MQGMO_NO_WAIT.

        Note: If you want the application to browse the messages rather than remove them from the queue, compile the sample with -DBROWSE, or, add `#define BROWSE` at the beginning of the source. When you do this, the macro preprocessor adds the line in the program that selects the MQGMO_BROWSE_NEXT option to the compilation. When this option is used on a call against a queue for which no browse cursor has previously been used with the current object handle, the browse cursor is positioned logically before the first message.
      - A buffer of size 64KB to hold the data copied from the message.
   d. Call the printMD subroutine. This prints the name of each field in the message descriptor, followed by its contents.
   e. If you created a message handle in step “4” on page 1246 call the printProperties subroutine to display any message properties.
   f. Print the length of the message, followed by the message data. Each line of message data is in this format:
      - Relative position (in hexadecimal) of this part of the data
      - 16 bytes of hexadecimal data
      - The same 16 bytes of data in character format, if it is printable (nonprintable characters are replaced by periods)
7. If the MQGET call fails, test the reason code to see if the call failed because there are no more messages on the queue. In this case, print the message: No more messages; otherwise, print the completion and reason codes. In both cases, go to step “9” on page 1246.

   Note: The MQGET call fails if it finds a message that has more than 64KB of data. To change the program to handle larger messages, you could do one of the following:

   - Add the MQGMO_ACCEPT_TRUNCATED_MSG option to the MQGET call, so that the call gets the first 64KB of data and discards the remainder
   - Make the program leave the message on the queue when it finds one with this amount of data
   - Increase the size of the buffer
8. If you created a message handle in step “4” on page 1246 call MQDLTMH to delete it.
9. Close the queue using the MQCLOSE call with the object handle returned in step “3” on page 1245.

10. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “2” on page 1245.

z/OS *The Queue Attributes sample on z/OS*

The Queue Attributes sample is a conversational-mode CICS application that demonstrates the use of the MQINQ and MQSET calls.

It shows how to inquire about the values of the **InhibitPut** and **InhibitGet** attributes of queues, and how to change them so that programs cannot put messages on, or get messages from, a queue. You might want to *lock* a queue in this way when you are testing a program.

To prevent accidental interference with your own queues, this sample works only on a queue object that has the characters CSQ4SAMP in the first eight bytes of its name. However, the source code includes comments to show you how to remove this restriction.

Source programs are supplied in the COBOL, assembler, and C languages (see Table 180 on page 1234).

The assembler-language version of the sample uses reenterable code. To do this, you will notice that the code for each MQI call in that version of the sample includes the MF keyword; for example:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(The VL keyword means that you can use the CICS Execution Diagnostic Facility (CEDF) supplied transaction for debugging the program.) For more information about writing reenterable programs, see Coding in System/390 assembler language.

To start the application, start your CICS system and use the following CICS transactions:

- For COBOL, MVC1
- For assembler language, MAC1
- For C, MCC1

You can change the name of any of these transactions by changing the CSD data set mentioned in step 3.

Design of the sample

When you start the sample, it displays a screen map that has fields for:

- Name of the queue
- User request (valid actions are: inquire, allow, or inhibit)
- Current status of put operations for the queue
- Current status of get operations for the queue

The first two fields are for user input. The last two fields are filled by the application: they show the word `INHIBITED` or the word `ALLOWED`.

The application validates the values that you enter in the first two fields. It checks that the queue name starts with the characters CSQ4SAMP and that you entered one of the three valid requests in the Action field. The application converts all your input to uppercase, so you cannot use any queues with names that contain lowercase characters.

If you enter `inquire` in the **Action** field, the flow through the program logic is:

1. Open the queue using the MQOPEN call with the MQOO_INQUIRE option
2. Call MQINQ using the selectors MQIA_INHIBIT_GET and MQIA_INHIBIT_PUT
3. Close the queue using the MQCLOSE call
4. Analyze the attributes that are returned in the **IntAttrs** parameter of the MQINQ call and move the words INHIBITED or ALLOWED, as appropriate, to the relevant screen fields

If you enter `inhibit` in the **Action** field, the flow through the program logic is:

1. Open the queue using the MQOPEN call with the MQOO_SET option
2. Call MQSET using the selectors MQIA_INHIBIT_GET and MQIA_INHIBIT_PUT, and with the values MQQA_GET_INHIBITED and MQQA_PUT_INHIBITED in the **IntAttrs** parameter
3. Close the queue using the MQCLOSE call
4. Move the word INHIBITED to the relevant screen fields

If you enter `allow` in the **Action** field, the application performs similar processing to that for an inhibit request. The only differences are the settings of the attributes and the words displayed on the screen.

When the application opens the queue, it uses the default connection handle to the queue manager. (CICS establishes a connection to the queue manager when you start your CICS system.) The application can trap the following errors at this stage:

- The application is not connected to the queue manager
- The queue does not exist
- The user is not authorized to access the queue
- The application is not authorized to open the queue

For other MQI errors, the application displays the completion and reason codes.

z/OS *The Mail Manager sample on z/OS*

The Mail Manager sample application is a suite of programs that demonstrates sending and receiving messages, both within a single environment and across different environments. The application is a simple electronic mailing system that allows users to exchange messages, even if they use different queue managers.

The application demonstrates how to create queues using the MQOPEN call and by putting IBM MQ for z/OS commands on the system-command input queue.

Three versions of the application are provided:

- A CICS application written in COBOL
- A TSO application written in COBOL
- A TSO application written in C

z/OS *Preparing the Mail Manager sample on z/OS*

The Mail Manager is provided in versions that run in two environments. The preparation that you must carry out before you run the application depends on the environment that you want to use.

Users can access mail queues and nickname queues from both TSO and CICS so long as their sign-on user IDs are the same on each system.

Before you can send messages to another queue manager, you must set up a message channel to that queue manager. To do this, use the channel control function of IBM MQ, described in Channel control function.

Preparing the sample for the TSO environment

Follow these steps:

1. Prepare the sample as described in “Preparing sample applications for the TSO environment on z/OS” on page 1231.
2. Tailor the CLIST provided for the sample to define:
   - The location of the panels
   - The location of the message file
   - The location of the load modules
   - The name of the queue manager that you want to use with the application

A separate CLIST is provided for each language version of the sample:

- For the COBOL version: CSQ4RVD1
- For the C version: CSQ4RCD1

3. Ensure that the queues used by the application are available on the queue manager. (The queues are defined in CSQ4CVD.)

Note: VS COBOL II does not support multitasking with ISPF. This means that you cannot use the Mail Manager sample application on both sides of a split screen. If you do, the results are unpredictable.

z/OS *Running the Mail Manager sample on z/OS*

To start the sample in the CICS Transaction Server for z/OS environment, run transaction MAIL. If you have not already signed on to CICS, the application prompts you to enter a user ID to which it can send your mail.

When you start the application, it opens your mail queue. If this queue does not exist, the application creates one for you. Mail queues have names of the form CSQ4SAMP.MAILMGR. *userid*, where *userid* depends on the environment:

In TSO
The user's TSO ID

In CICS
The user's CICS sign-on or the user ID entered by the user when prompted when the Mail Manager started

All parts of the queue names that the Mail Manager uses must be uppercase.

The application then presents a menu panel that has options for:

- Read incoming mail
- Send mail
- Create nickname

The menu panel also shows you how many messages are waiting on your mail queue. Each of the menu options displays a further panel:

Read incoming mail
The Mail Manager displays a list of the messages that are on your mail queue. (Only the first 99 messages on the queue are displayed.) For an example of this panel, see Figure 142 on page 1253. When you select a message from this list, the contents of the message are displayed (see Figure 143 on page 1254).

Send mail
A panel prompts you to enter:

- The name of the user to whom you want to send a message
- The name of the queue manager that owns their mail queue
- The text of your message

In the user name field, you can enter either a user ID or a nickname that you created using the Mail Manager. You can leave the queue manager name field blank if the user's mail queue is owned by the same queue manager that you are using, and you must leave it blank if you entered a nickname in the user name field:

- If you specify only a user name, the program first assumes that the name is a nickname, and sends the message to the object defined by that name. If there is no such nickname, the program attempts to send the message to a local queue of that name.
- If you specify both a user name and a queue manager name, the program sends the message to the mail queue that is defined by those two names.

For example, if you want to send a message to user JONESM on remote queue manager QM12, you could send them a message in either of two ways:

- Use both fields to specify user JONESM at queue manager QM12.
- Define a nickname (for example, MARY) for that user and send them a message by putting MARY in the user name field and nothing in the queue manager name field.

Create nickname

You can define an easy-to-remember name that you can use when you send a message to another user who you contact frequently. You are prompted to enter the user ID of the other user and the name of the queue manager that owns their mail queue.

Nicknames are queues that have names of the form CSQ4SAMP.MAILMGR. *userid.nickname*, where *userid* is your own user ID and *nickname* is the nickname that you want to use. With names structured in this way, users can each have their own set of nicknames.

The type of queue that the program creates depends on how you complete the fields of the Create Nickname panel:

- If you specify only a user name, or the queue manager name is the same as that of the queue manager to which the Mail Manager is connected, the program creates an alias queue.
- If you specify both a user name and a queue manager name (and the queue manager is not the one to which the Mail Manager is connected), the program creates a local definition of a remote queue. The program does not check the existence of the queue to which this definition resolves, or even that the remote queue manager exists.

For example, if your own user ID is SMITHK and you create a nickname called MARY for user JONESM (who uses the remote queue manager QM12), the nickname program creates a local definition of a remote queue named CSQ4SAMP.MAILMGR.SMITHK.MARY. This definition resolves to Mary's mail queue, which is CSQ4SAMP.MAILMGR.JONESM at queue manager QM12. If you are using queue manager QM12 yourself, the program instead creates an alias queue of the same name (CSQ4SAMP.MAILMGR.SMITHK.MARY).

The C version of the TSO application makes greater use of ISPF's message-handling capabilities than does the COBOL version. You might notice that different error messages are displayed by the C and COBOL versions.

z/OS *Design of the Mail Manager sample on z/OS*

The following sections describe each of the programs that make up the Mail Manager sample application.

The relationships between the programs and the panels that the application uses is shown in Figure 140 on page 1251 for the TSO version, and Figure 141 on page 1252 for the CICS Transaction Server for z/OS version.

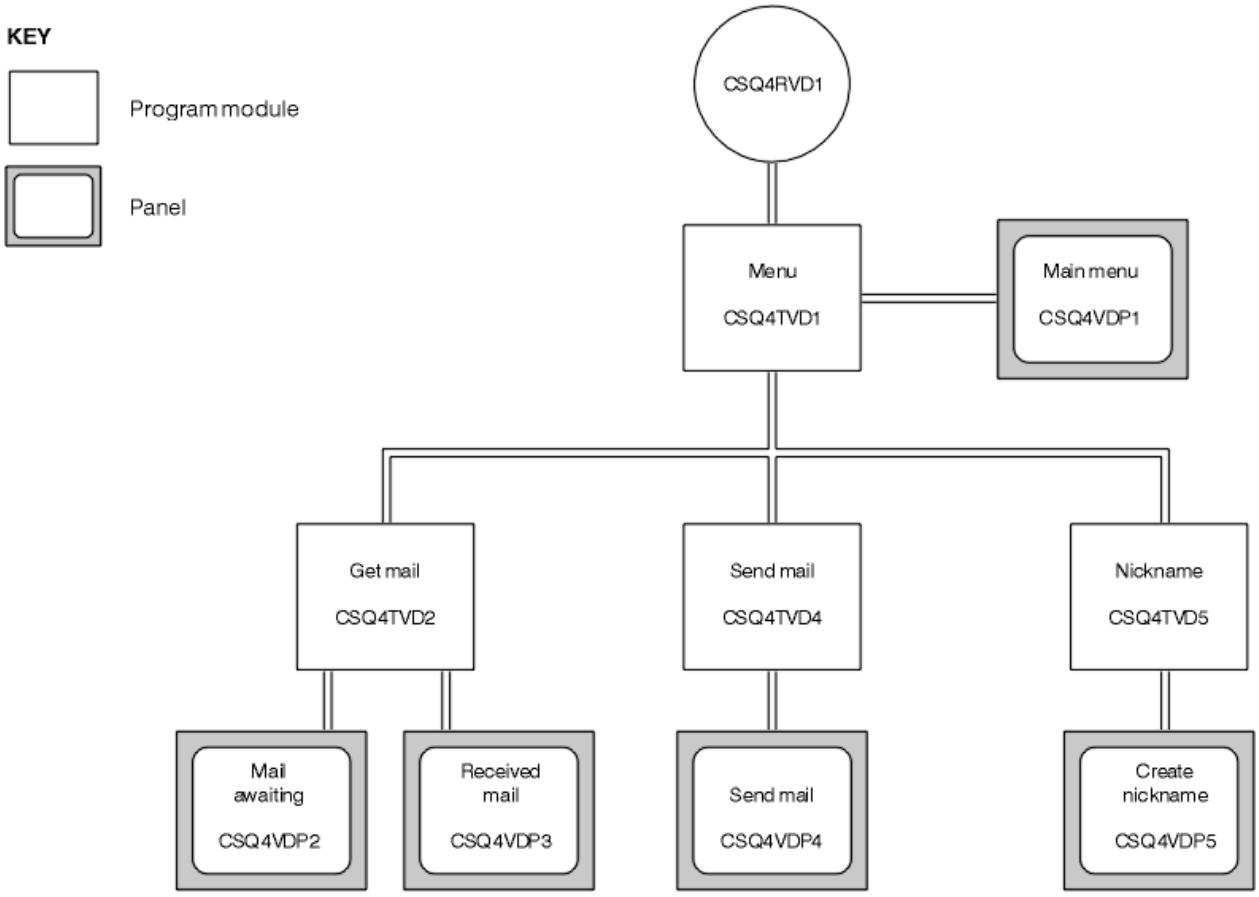


Figure 140. Programs and panels for the TSO versions of the Mail Manager

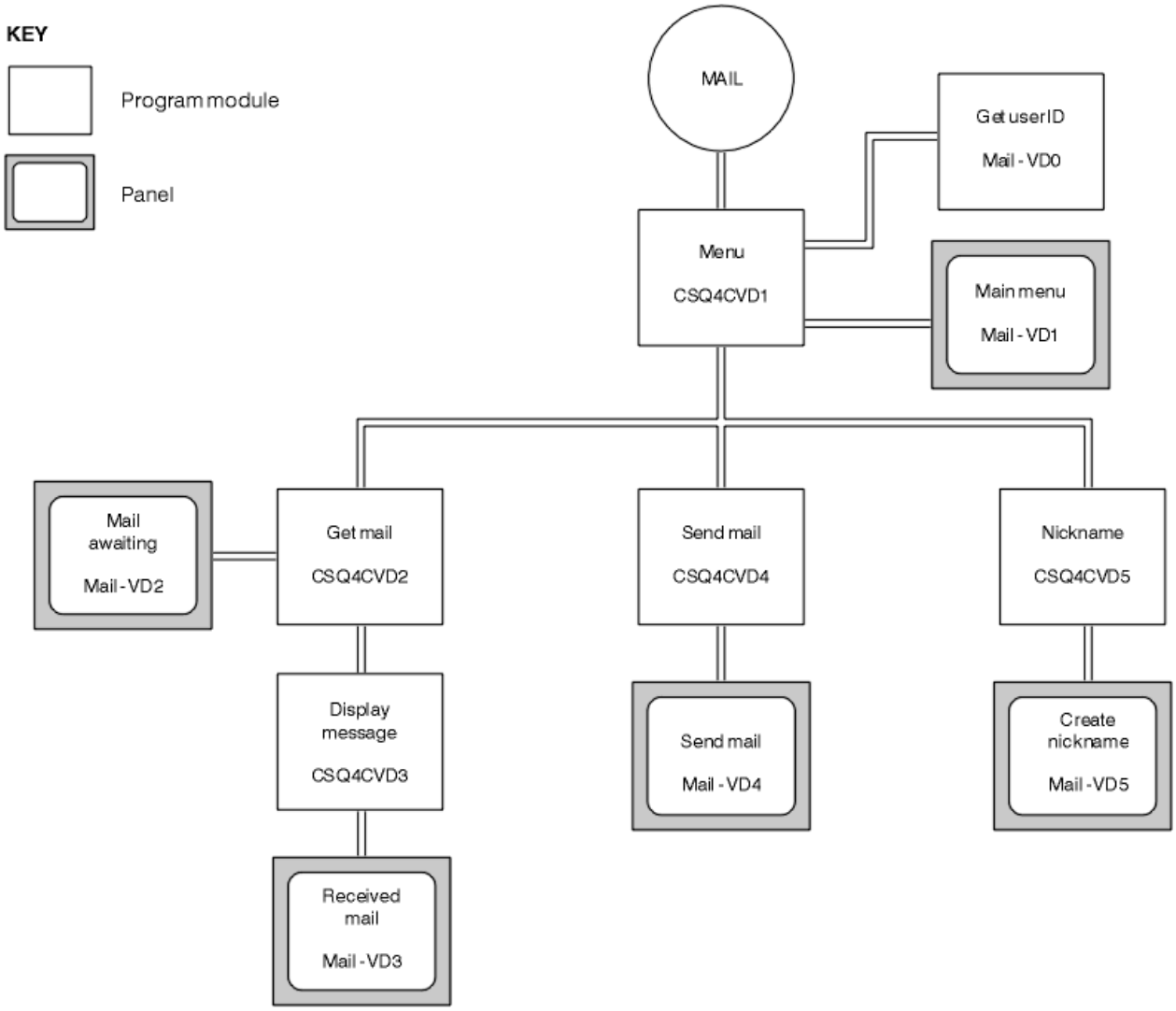


Figure 141. Programs and panels for the CICS version of the Mail Manager

z/OS *Menu program on z/OS*

In the TSO environment, the menu program is invoked by the CLIST. In the CICS environment, the program is invoked by transaction MAIL.

The menu program (CSQ4TVD1 for TSO, CSQ4CVD1 for CICS) is the initial program in the suite. It displays the menu (CSQ4VDP1 for TSO, VD1 for CICS) and invokes the other programs when they are selected from the menu.

The program first obtains the user's ID:

- In the CICS version of the program, if the user has signed on to CICS, the user ID is obtained by using the CICS command ASSIGN USERID. If the user has not signed on, the program displays the sign on panel (CSQ4VD0) to prompt the user to enter a user ID. There is no security processing within this program; the user can give any user ID.
- In the TSO version, the user's ID is obtained from TSO in the CLIST. It is passed to the menu program as a variable in the ISPF shared pool.

After the program has obtained the user ID, it checks to ensure that the user has a mail queue (CSQ4SAMP.MAILMGR. *userid*). If a mail queue does not exist, the program creates one by putting a message on the system-command input queue. The message contains the IBM MQ for z/OS command DEFINE QLOCAL. The object definition that this command uses sets the maximum depth of the queue to 9999 messages.

The program also creates a temporary dynamic queue to handle replies from the system-command input queue. To do this, the program uses the MQOPEN call, specifying the SYSTEM.DEFAULT.MODEL.QUEUE as the template for the dynamic queue. The queue manager creates the temporary dynamic queue with a name that has the prefix CSQ4SAMP; the remainder of the name is generated by the queue manager.

The program then opens the user's mail queue and finds the number of messages on the queue by inquiring about the current depth of the queue. To do this, the program uses the MQINQ call, specifying the MQIA_CURRENT_Q_DEPTH selector.

The program then performs a loop that displays the menu and processes the selection that the user makes. The loop is stopped when the user presses the PF3 key. When a valid selection is made, the appropriate program is started; otherwise an error message is displayed.

z/OS *Get-mail and display-message programs on z/OS*

In the TSO versions of the application, the get-mail and display-message functions are performed by the same program (CSQ4TVD2). In the CICS version of the application, these functions are performed by separate programs (CSQ4CVD2 and CSQ4CVD3).

The Mail Awaiting panel (CSQ4VDP2 for TSO, VD2 for CICS ; see Figure 142 on page 1253 for an example) shows all the messages that are on the user's mail queue. To create this list, the program uses the MQGET call to browse all the messages on the queue, saving information about each one. In addition to the information displayed, the program records the `MsgId` and `CorrelId` of each message.

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

Figure 142. Example of a panel showing a list of waiting messages

From the Mail Awaiting panel the user can select one message and display the contents of the message (see Figure 143 on page 1254 for an example). The program uses the MQGET call to remove this message from the queue, using the `MsgId` and `CorrelId` that the program noted when it browsed all the messages. This MQGET call is performed using the MQGMO_SYNCPOINT option. The program displays the contents of the message, then declares a syncpoint: this commits the MQGET call, so the message now no longer exists.

```
--------------------- IBM MQ for z/OS Sample Programs ---------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
----------------------------------- Message ------------------------------
| HELLO FROM JOHNJ                                |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
'-------------------------------------------------------------------------'
```

Figure 143. Example of a panel showing the contents of a message

An obvious extension to the function provided by the Mail Manager is to give the user the option to leave the message on the queue after viewing its contents. To do this, you would have to back out the MQGET call that removes the message from the queue, after displaying the message.

z/OS *Send-mail program on z/OS*

When the user has completed the Send Mail panel (CSQ4VDP4 for TSO, VD4 for CICS), the send-mail program (CSQ4TVD4 for TSO, CSQ4CVD4 for CICS) puts the message on the receiver's mail queue.

To do this, the program uses the MQPUT1 call. The destination of the message depends on how the user has filled the fields in the Send Mail panel:

- If the user has specified only a user name, the program first assumes that the name is a nickname, and sends the message to the object defined by that name. If there is no such nickname, the program attempts to send the message to a local queue of that name.
- If the user has specified both a user name and a queue manager name, the program sends the message to the mail queue that is defined by those two names.

The program does not accept blank messages, and it removes leading blanks from each line of the message text.

If the MQPUT1 call is successful, the program displays a message that shows that the user name and queue manager name to which the message was put. If the call is unsuccessful, the program checks specifically for the reason codes that indicate the queue or the queue manager do not exist; these are MQRC_UNKNOWN_OBJECT_NAME and MQRC_UNKNOWN_OBJECT_Q_MGR. The program displays its own error message for each of these errors; for other errors, the program displays the completion and reason codes returned by the call.

z/OS *Nickname program on z/OS*

When the user defines a nickname, the nickname program (CSQ4TVD5 for TSO, CSQ4CVD5 for CICS) creates a queue that has the nickname as part of its name.

The program does this by putting a message on the system-command input queue. The message contains the IBM MQ for z/OS command DEFINE QALIAS or DEFINE QREMOTE. The type of queue that the program creates depends on how the user has filled the fields of the Create Nickname panel (CSQ4VDP5 for TSO, VD5 for CICS):

- If the user has specified only a user name, or the queue manager name is the same as that of the queue manager to which the Mail Manager is connected, the program creates an alias queue.

- If the user has specified both a user name and a queue manager name (and the queue manager is not the one to which the Mail Manager is connected), the program creates a local definition of a remote queue. The program does not check the existence of the queue to which this definition resolves, or even that the remote queue manager exists.

The program also creates a temporary dynamic queue to handle replies from the system-command input queue.

If the queue manager cannot create the nickname queue for a reason that the program expects (for example, the queue already exists), the program displays its own error message. If the queue manager cannot create the queue for a reason that the program does not expect, the program displays up to two of the error messages that are returned to the program by the command server.

Note: For each nickname, the nickname program creates only an alias queue or a local definition of a remote queue. The local queues to which these queue names resolve are created only when the user ID that is contained in the nickname is used to start the Mail Manager application.

z/OS *The Credit Check sample on z/OS*

The Credit Check sample application is a suite of programs that demonstrates how to use many of the features provided by IBM MQ for z/OS. It shows how the many component programs of an application can pass messages to each other using message queuing techniques.

The sample can run as a stand-alone CICS application. However, to demonstrate how to design a message queuing application that uses the facilities provided by both the CICS and IMS environments, one module is also supplied as an IMS batch message processing program. This extension to the sample is described in “The IMS extension to the Credit Check sample on z/OS” on page 1265.

You can also run the sample on more than one queue manager, and send messages between each instance of the application. To do so, see “The Credit Check sample with multiple queue managers on z/OS” on page 1264.

The CICS programs are delivered in C and COBOL. The single IMS program is delivered only in C. The supplied data sets are shown in Table 182 on page 1235 and Table 184 on page 1237.

The application demonstrates a method of assessing the risk when bank customers ask for loans. The application shows how a bank could work in two ways to process loan requests:

- When dealing directly with a customer, bank staff want immediate access to account and credit-risk information.
- When dealing with written applications, bank staff can submit a series of requests for account and credit-risk information, and deal with the replies at a later time.

The financial and security details in the application have been kept simple so that the message queuing techniques are clear.

z/OS *Preparing and running the Credit Check sample on z/OS*

To prepare and run the Credit Check sample, perform the following steps:

1. Create the VSAM data set that holds information about some example accounts. Do this by editing and running the JCL supplied in data set CSQ4FILE.
2. Perform the steps in “Preparing the sample applications for the CICS environment on z/OS” on page 1233. (The additional steps that you must perform if you want to use the IMS extension to the sample are described in “The IMS extension to the Credit Check sample on z/OS” on page 1265.)
3. Start the CKTI trigger monitor (supplied with IBM MQ for z/OS) against queue CSQ4SAMP.INITIATION.QUEUE, using the CICS transaction CKQC.
4. To start the application, start your CICS system and use the transaction MVB1.
5. Select **Immediate** or **Batch** inquiry from the first panel.

   The immediate and batch inquiry panels are similar; Figure 144 on page 1256 shows the Immediate Inquiry panel.

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . _____________________
Social security number ___ __ ____
Bank account name . . ______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : _____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

Figure 144. Immediate Inquiry panel for the Credit Check sample application

6. Enter an account number and loan amount in the appropriate fields. See “Entering information in the inquiry panels” on page 1256 for guidance on what information to enter in these fields.

Entering information in the inquiry panels

The Credit Check sample application checks that the data you enter in the **Amount requested** field of the inquiry panels is in the form of integers.

If you enter one of the following account numbers, the application finds the appropriate account name, average account balance, and credit worthiness index in the VSAM data set CSQ4BAQ:

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

You can enter any, or no, information in the other fields. The application retains any information that you enter and returns the same information in the reports that it generates.

z/OS *Design of the Credit Check sample on z/OS*

This section describes the design of each of the programs that make up the Credit Check sample application.

For more information about of some of the techniques that were considered during the design of the application, see “Design considerations for the Credit check sample on z/OS” on page 1262.

Figure 145 on page 1257 shows the programs that make up the application, and also the queues that these programs serve. In this figure, the prefix CSQ4SAMP has been omitted from all the queue names to make the figure easier to understand.

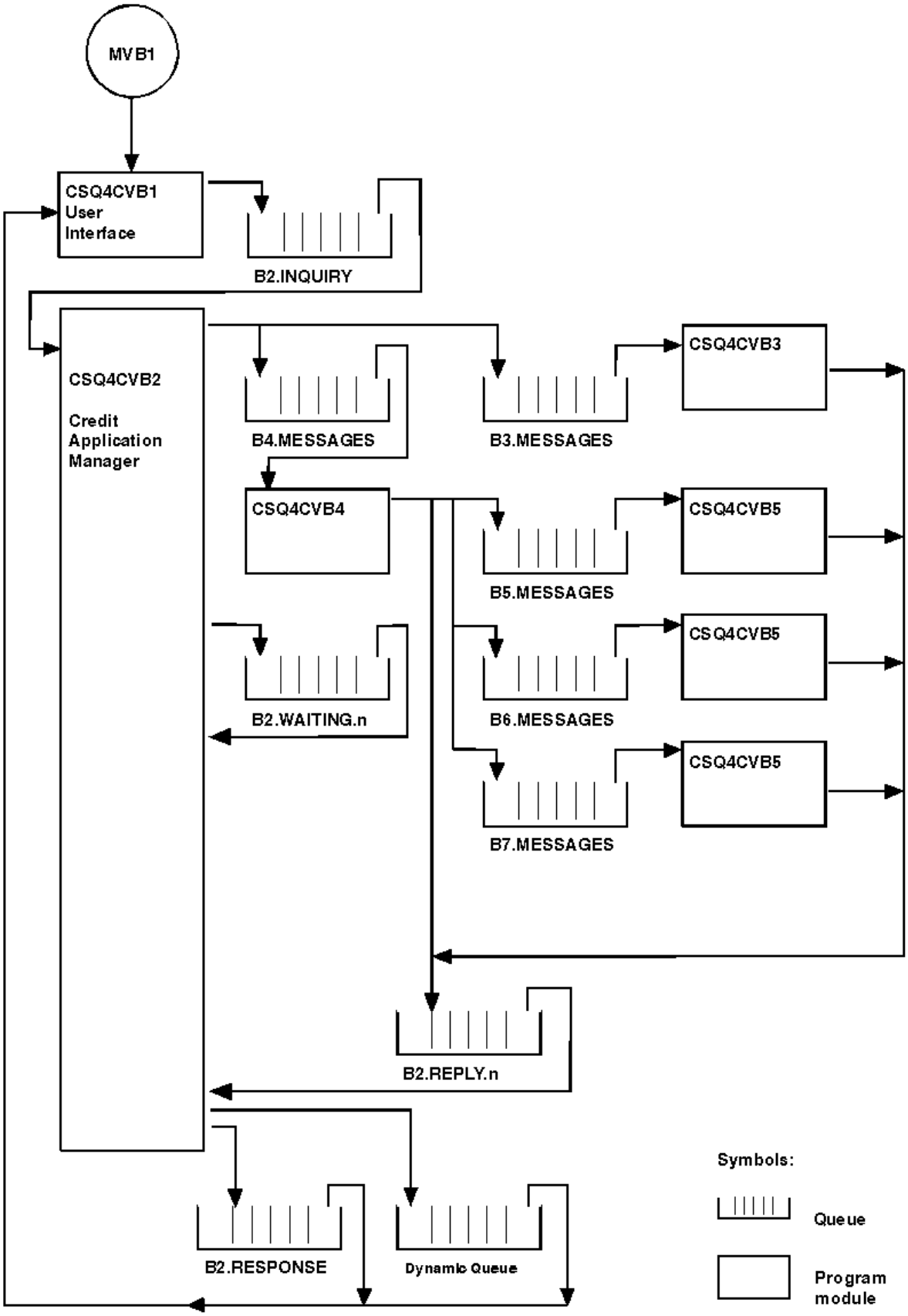


Figure 145. Programs and queues for the Credit Check sample application (COBOL programs only)

z/OS *User interface program (CSQ4CVB1) on z/OS*

When you start the conversational-mode CICS transaction MVB1, this starts the user interface program for the application.

This program puts inquiry messages on queue CSQ4SAMP.B2.INQUIRY and gets replies to those inquiries from a reply-to queue that it specifies when it makes the inquiry. From the user interface you can submit either immediate or batch inquiries:

- For immediate inquiries, the program creates a temporary dynamic queue that it uses as a reply-to queue. This means that each inquiry has its own reply-to queue.
- For batch inquiries, the user-interface program gets replies from the queue CSQ4SAMP.B2.RESPONSE. For simplicity, the program gets replies for all its inquiries from this one reply-to queue. It is easy to see that a bank might want to use a separate reply-to queue for each user of MVB1, so that they could each see replies to only those inquiries that they had initiated.

Important differences between the properties of messages used in the application when in batch and immediate mode are:

- For batch working, the messages have a low priority, so they are processed after any loan requests that are entered in immediate mode. Also, the messages are persistent, so they are recovered if the application or the queue manager has to restart.
- For immediate working, the messages have a high priority, so they are processed before any loan requests that are entered in batch mode. Also, messages are not persistent so they are discarded if the application or the queue manager has to restart.

However, in all cases, the properties of loan request messages are propagated throughout the application. So, for example, all messages that result from a high-priority request will also have a high priority.

z/OS *Credit application manager (CSQ4CVB2) on z/OS*

The Credit Application Manager (CAM) program performs most of the processing for the Credit Check application.

The CAM is started by the CKTI trigger monitor (supplied with IBM MQ for z/OS) when a trigger event occurs on either queue CSQ4SAMP.B2.INQUIRY or queue CSQ4SAMP.B2.REPLY. *n*, where *n* is an integer that identifies one of a set of reply queues. The trigger message contains data that includes the name of the queue on which the trigger event occurred.

The CAM uses queues with names of the form CSQ4SAMP.B2.WAITING.n to store information about inquiries that it is processing. The queues are named so that they are each paired with a reply-to queue; for example, queue CSQ4SAMP.B2.WAITING.3 contains the input data for a particular inquiry, and queue CSQ4SAMP.B2.REPLY.3 contains a set of reply messages (from programs that query databases) all relating to that same inquiry. To understand the reasons behind this design, see “Separate inquiry and reply queues in the CAM” on page 1262.

Startup logic

If the trigger event occurs on queue CSQ4SAMP.B2.INQUIRY, the CAM opens the queue for shared access. It then tries to open each reply queue until a free one is found. If it cannot find a free reply queue, the CAM logs the fact and terminates normally.

If the trigger event occurs on queue CSQ4SAMP.B2.REPLY.n, the CAM opens the queue for exclusive access. If the return code reports that the object is already in use, the CAM terminates normally. If any other error occurs, the CAM logs the error and terminates. The CAM opens the corresponding waiting queue and the inquiry queue, then starts getting and processing messages. From the waiting queue, the CAM recovers details of partially-completed inquiries.

For the sake of simplicity in this sample, the names of the queues used are held in the program. In a business environment, the queue names would probably be held in a file accessed by the program.

Getting a message from the enquiry queue

The CAM first attempts to get a message from the inquiry queue using the MQGET call with the MQGMO_SET_SIGNAL option. If a message is available immediately, the message is processed; if no message is available, a signal is set.

The CAM then attempts to get a message from the reply queue, again using the MQGET call with the same option. If a message is available immediately, the message is processed; otherwise a signal is set.

When both signals are set, the program waits until one of the signals is posted. If a signal is posted to indicate that a message is available, the message is retrieved and processed. If the signal expires or the queue manager is terminating, the program terminates.

Processing the message retrieved by the CAM

A message retrieved by the CAM can be one of four types:

- An inquiry message
- A reply message
- A propagation message
- An unexpected or unwanted message

The CAM processes these messages as described in "Processing the message retrieved by the CAM on z/OS" on page 1259.

Sending an answer

When the CAM has received all the replies it is expecting for an inquiry, it processes the replies and creates a single response message. It consolidates into one message all the data from all reply messages that have the same `CorrelId`. This response is put on the reply-to queue specified in the original loan request. The response message is put within the same unit of work that contains the retrieval of the final reply message. This is to simplify recovery by ensuring that there is never a completed message on queue CSQ4SAMP.B2.WAITING.n.

Recovery of partially-completed inquiries

The CAM copies onto queue CSQ4SAMP.B2.WAITING.n all the messages that it receives. It sets the fields of the message descriptor like this:

- *`Priority`* is determined by the type of message:
  - For request messages, priority = 3
  - For datagrams, priority = 2
  - For reply messages, priority = 1
- *`CorrelId`* is set to the *`MsgId`* of the loan request message
- Other MQMD fields are copied from those of the received message

When an inquiry has been completed, the messages for a specific inquiry are removed from the waiting queue during answer processing. Therefore, at any time, the waiting queue contains all messages relevant to in-progress inquiries. These messages are used to recover details of in-progress inquiries if the program has to restart. The different priorities are set so that inquiry messages are recovered before propagations or reply messages.

z/OS *Processing the message retrieved by the CAM on z/OS*

A message retrieved by the Credit Application Manager (CAM) can be one of four types. The way in which the CAM processes a message depends on its type.

A message retrieved by the CAM can be one of four types:

- An inquiry message

- A reply message
- A propagation message
- An unexpected or unwanted message

The CAM processes these messages as follows:

Inquiry message
Inquiry messages come from the user interface program. It creates an inquiry message for each loan request.

For all loan requests, the CAM requests the average balance of the customer's checking account. It does this by putting a request message on alias queue CSQ4SAMP.B2.OUTPUT.ALIAS. This queue name resolves to queue CSQ4SAMP.B3.MESSAGES, which is processed by the checking-account program, CSQ4CVB3. When the CAM puts a message on this alias queue, it specifies the appropriate CSQ4SAMP.B2.REPLY.n queue for the reply-to queue. An alias queue is used here so that program CSQ4CVB3 can easily be replaced by another program that processes a base queue of a different name. To do this, you redefine the alias queue so that its name resolves to the new queue. Also, you could assign differing access authorities to the alias queue and to the base queue.

If a user requests a loan that is larger than 10000 units, the CAM initiates checks on other databases as well. It does this by putting a request message on queue CSQ4SAMP.B4.MESSAGES, which is processed by the distribution program, CSQ4CVB4. The process serving this queue propagates the message to queues served by programs that have access to other records such as credit card history, savings accounts, and mortgage payments. The data from these programs is returned to the reply-to queue specified in the put operation. Additionally, a propagation message is sent to the reply-to queue by this program to specify how many propagation messages have been sent.

In a business environment, the distribution program would probably reformat the data provided to match the format required by each of the other types of bank account.

Any of the queues referred to can be on a remote system.

For each inquiry message, the CAM initiates an entry in the memory-resident Inquiry Record Table (IRT). This record contains:

- The `MsgId` of the inquiry message
- In the ReplyExp field, the number of responses expected (equal to the number of messages sent)
- In the ReplyRec field, the number of replies received (zero at this stage)
- In the PropsOut field, an indication of whether a propagation message is expected

The CAM copies the inquiry message onto the waiting queue with:

- `Priority` set to 3
- `CorrelId` set to the `MsgId` of the inquiry message
- The other message-descriptor fields set to those of the inquiry message

Propagation message
A propagation message contains the number of queues to which the distribution program has forwarded the inquiry. The message is processed as follows:

1. Add to the ReplyExp field of the appropriate record in the IRT the number of messages sent. This information is in the message.
2. Increment by 1 the ReplyRec field of the record in the IRT.
3. Decrement by 1 the PropsOut field of the record in the IRT.
4. Copy the message onto the waiting queue. The CAM sets the `Priority` to 2 and the other fields of the message descriptor to those of the propagation message.

Reply message
A reply message contains the response to one of the requests to the checking-account program or to one of the agency-query programs. Reply messages are processed as follows:

1. Increment by 1 the ReplyRec field of the record in the IRT.
2. Copy the message onto the waiting queue with `Priority` set to 1 and the other fields of the message descriptor set to those of the reply message.
3. If ReplyRec = ReplyExp, and PropsOut = 0, set the MsgComplete flag.

Other messages

The application does not expect other messages. However, the application might receive messages broadcast by the system, or reply messages with a unknown `CorrelIds`.

The CAM puts these messages on queue CSQ4SAMP.DEAD.QUEUE, where they can be examined. If this put operation fails, the message is lost and the program continues. For more information about the design of this part of the program, see "How the sample handles unexpected messages" on page 1263.

z/OS *Checking-account program (CSQ4CVB3) on z/OS*

The checking-account program is started by a trigger event on queue CSQ4SAMP.B3.MESSAGES. After it has opened the queue, this program gets a message from the queue using the MQGET call with the wait option, and with the wait interval set to 30 seconds.

The program searches VSAM data set CSQ4BAQ for the account number in the loan request message. It retrieves the corresponding account name, average balance, and credit worthiness index, or notes that the account number is not in the data set.

The program then puts a reply message (using the MQPUT1 call) on the reply-to queue named in the loan request message. For this reply message, the program:

- Copies the `CorrelId` of the loan request message
- Uses the MQPMO_PASS_IDENTITY_CONTEXT option

The program continues to get messages from the queue until the wait interval expires.

z/OS *Distribution program (CSQ4CVB4) on z/OS*

The distribution program is started by a trigger event on queue CSQ4SAMP.B4.MESSAGES.

To simulate the distribution of the loan request to other agencies that have access to records such as credit card history, savings accounts, and mortgage payments, the program puts a copy of the same message on all the queues in the namelist CSQ4SAMP.B4.NAMELIST. There are three of these queues, with names of the form CSQ4SAMP.B *n*.MESSAGES, where *n* is 5, 6, or 7. In a business application, the agencies could be at separate locations, so these queues could be remote queues. If you want to modify the sample application to show this, see "The Credit Check sample with multiple queue managers on z/OS" on page 1264.

The distribution program performs the following steps:

1. From the namelist, gets the names of the queues that the program is to use. The program does this by using the MQINQ call to inquire about the attributes of the namelist object.
2. Opens these queues and also CSQ4SAMP.B4.MESSAGES.
3. Performs the following loop until there are no more messages on queue CSQ4SAMP.B4.MESSAGES:
   a. Get a message using the MQGET call with the wait option, and with the wait interval set to 30 seconds.
   b. Put a message on each queue listed in the namelist, specifying the name of the appropriate CSQ4SAMP.B2.REPLY.n queue for the reply-to queue. The program copies the *`CorrelId`* of the loan request message to these copy messages, and it uses the MQPMO_PASS_IDENTITY_CONTEXT option on the MQPUT call.
   c. Send a datagram message to queue CSQ4SAMP.B2.REPLY.n to show how many messages it has successfully put.
   d. Declare a syncpoint.

z/OS *Agency-query program (CSQ4CVB5/CSQ4CCB5) on z/OS*

The agency-query program is supplied as both a COBOL program and a C program. Both programs have the same design. This shows that programs of different types can easily coexist within an IBM MQ application, and that the program modules that make up such an application can easily be replaced.

An instance of the program is started by a trigger event on any of these queues:

- For the COBOL program (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- For the C program (CSQ4CCB5), queue CSQ4SAMP.B8.MESSAGES

Note: If you want to use the C program, you must alter the definition of the namelist CSQ4SAMP.B4.NAMELIST to replace the queue CSQ4SAMP.B7.MESSAGES with CSQ4SAMP.B8.MESSAGES. To do this, you can use any one of:

- The IBM MQ for z/OS operations and control panels
- The ALTER NAMELIST command
- The CSQUTIL utility

After it has opened the appropriate queue, this program gets a message from the queue using the MQGET call with the wait option, and with the wait interval set to 30 seconds.

The program simulates the search of an agency's database by searching the VSAM data set CSQ4BAQ for the account number that was passed in the loan request message. It then builds a reply that includes the name of the queue that it is serving and a creditworthiness index. To simplify the processing, the creditworthiness index is selected at random.

When putting the reply message, the program uses the MQPUT1 call and:

- Copies the `CorrelId` of the loan request message
- Uses the MQPMO_PASS_IDENTITY_CONTEXT option

The program sends the reply message to the reply-to queue named in the loan request message. (The name of the queue manager that owns the reply-to queue is also specified in the loan request message.)

z/OS *Design considerations for the Credit check sample on z/OS*

Design considerations for the Credit Check sample.

This topic contains information about:

Separate inquiry and reply queues in the CAM

The application could use a single queue for both inquiries and replies, but it was designed to use separate queues for the following reasons:

- When the program is handling the maximum number of inquiries, further inquiries can be left on the queue. If a single queue is being used, this would have to be taken off the queue and stored elsewhere.
- Other instances of the CAM could be started automatically to service the same inquiry queue if message traffic was high enough to warrant it. But the program must track in-progress inquiries, and to do this,

it must get back all replies to inquiries it has initiated. If only one queue is used, the program would have to browse the messages to see if they were for this program or for another. This would make the operation much less efficient.

The application can support multiple CAMs and can recover in-progress inquiries effectively by using paired reply-to and waiting queues.

- The program can wait on multiple queues effectively by using signaling.

How the sample handles errors

The user interface program handles errors by reporting them directly to the user.

The other programs do not have user interfaces, so they have to handle errors in other ways. Also, in many situations (for example, if an MQGET call fails) these other programs do not know the identity of the user of the application.

The other programs put error messages on a CICS temporary storage queue called CSQ4SAMP. You can browse this queue using the CICS-supplied transaction CEBR. The programs also write error messages to the CICS CSML log.

How the sample handles unexpected messages

When you design a message-queuing application, you must decide how to handle messages that arrive on a queue unexpectedly.

The two basic choices are:

- The application does no more work until it has processed the unexpected message. This probably means that the application notifies an operator, terminates itself, and ensures that it is not restarted automatically (it can do this by setting triggering off). This choice means that all processing for the application can be halted by a single unexpected message, and the intervention of an operator is required to restart the application.
- The application removes the message from the queue it is serving, puts the message in another location, and continues processing. The best place to put this message is on the system dead-letter queue.

If you choose the second option:

- An operator, or another program, should examine the messages that are put on the dead-letter queue to find out where the messages are coming from.
- An unexpected message is lost if it cannot be put on the dead-letter queue.
- A long unexpected message is truncated if it is longer than the limit for messages on the dead-letter queue, or longer than the buffer size in the program.

To ensure that the application smoothly handles all inquiries with minimal effect from outside activities, the Credit Check sample application uses the second option. To allow you to keep the sample separate from other applications that use the same queue manager, the Credit Check sample does not use the system dead-letter queue; instead, it uses its own dead-letter queue. This queue is named CSQ4SAMP.DEAD.QUEUE. The sample truncates any messages that are longer than the buffer area provided for the sample programs. You can use the Browse sample application to browse messages on this queue, or use the Print Message sample application to print the messages together with their message descriptors.

However, if you extend the sample to run across more than one queue manager, unexpected messages, or messages that cannot be delivered, could be put on the system dead-letter queue by the queue manager.

How the sample uses syncpoints

The programs in the Credit Check sample application declare syncpoints to ensure that:

- Only one reply message is sent in response to each expected message

- Multiple copies of unexpected messages are never put on the sample's dead-letter queue
- The CAM can recover the state of all partially completed inquiries by getting persistent messages from its waiting queue

To achieve this, a single unit of work is used to cover the getting of a message, the processing of that message, and any subsequent put operations.

How the sample uses message context information

When the user interface program (CSQ4CVB1) sends messages, it uses the MQPMO_DEFAULT_CONTEXT option. This means that the queue manager generates both identity and origin context information. The queue manager gets this information from the transaction that started the program (MVB1) and from the user ID that started the transaction.

When the CAM sends inquiry messages, it uses the MQPMO_PASS_IDENTITY_CONTEXT option. This means that the identity context information of the message being put is copied from the identity context of the original inquiry message. With this option, origin context information is generated by the queue manager.

When the CAM sends reply messages, it uses the MQPMO_ALTERNATE_USER_AUTHORITY option. This causes the queue manager to use an alternate user ID for its security check when the CAM opens a reply-to queue. The CAM uses the user ID of the submitter of the original inquiry message. This means that users are allowed to see replies to only those inquiries that they have originated. The alternate user ID is obtained from the identity context information in the message descriptor of the original inquiry message.

When the query programs (CSQ4CVB3/4/5) send reply messages, they use the MQPMO_PASS_IDENTITY_CONTEXT option. This means that the identity context information of the message being put is copied from the identity context of the original inquiry message. With this option, origin context information is generated by the queue manager.

Note: The user ID associated with the MVB3/4/5 transactions requires access to the B2.REPLY.n queues. These user IDs might not be the same as those associated with the request being processed.
To get around this possible security exposure, the query programs could use the MQPMO_ALTERNATE_USER_AUTHORITY option when putting their replies. This would mean that each individual user of MVB1 needs authority to open the B2.REPLY.n queues.

Use of message and correlation identifiers in the CAM

The application has to monitor the progress of all the live inquiries it is processing at any one time. To do this it uses the unique message identifier of each loan request message to associate all the information that it has about each inquiry.

The CAM copies the `MsgId` of the inquiry message into the `CorrelId` of all the request messages it sends for that inquiry. The other programs in the sample (CSQ4CVB3 - 5) copy the `CorrelId` of each message that they receive into the `CorrelId` of their reply message.

z/OS *The Credit Check sample with multiple queue managers on z/OS*

You can use the Credit Check sample application to demonstrate distributed queuing by installing the sample on two queue managers and CICS systems (with each queue manager connected to a different CICS system).

When the sample program is installed, and the trigger monitor (CKTI) is running on each system, you need to:

1. Set up the communication link between the two queue managers. For information on how to do this, see Configuring distributed queuing.
2. On one queue manager, create a local definition for each of the remote queues (on the other queue manager) that you want to use. These queues can be any of CSQ4SAMP.B *n*.MESSAGES, where *n* is 3, 5, 6, or 7. (These are the queues that are served by the checking-account program and the agency-query program.) For information on how to do this, see DEFINE QREMOTE and DEFINE queues.

3. Change the definition of the namelist (CSQ4SAMP.B4.NAMELIST) so that it contains the names of the remote queues that you want to use. For information on how to do this, see DEFINE NAMELIST.

z/OS *The IMS extension to the Credit Check sample on z/OS*

A version of the checking-account program is supplied as an IMS batch message processing (BMP) program. It is written in the C language.

The program performs the same function as the CICS version, except that to obtain the account information, the program reads an IMS database instead of a VSAM file. If you replace the CICS version of the checking-account program with the IMS version, you see no difference in the method of using the application.

To prepare and run the IMS version you must:

1. Follow the steps in "Preparing and running the Credit Check sample on z/OS" on page 1255.
2. Follow the steps in "Preparing the sample application for the IMS environment on z/OS" on page 1236.
3. Alter the definition of the alias queue CSQ4SAMP.B2.OUTPUT.ALIAS to resolve to queue CSQ4SAMP.B3.IMS.MESSAGES (instead of CSQ4SAMP.B3.MESSAGES). To do this, you can use one of:
   - The IBM MQ for z/OS operations and control panels
   - The ALTER QALIAS command .

Another way of using the IMS checking-account program is to make it serve one of the queues that receives messages from the distribution program. In the delivered form of the Credit Check sample application, there are three of these queues (B5/6/7.MESSAGES), all served by the agency-query program. This program searches a VSAM data set. To compare the use of the VSAM data set and the IMS database, you could make the IMS checking-account program serve one of these queues instead. To do this, you must alter the definition of the namelist CSQ4SAMP.B4.NAMELIST to replace one of the CSQ4SAMP.B *n*.MESSAGES queues with the CSQ4SAMP.B3.IMS.MESSAGES queue. You can use one of:

- The IBM MQ for z/OS operations and control panels
- The ALTER NAMELIST command.

You can then run the sample from CICS transaction MVB1. The user sees no difference in operation or response. The IMS BMP stops either after receiving a stop message or after being inactive for five minutes.

Design of the IMS checking-account program (CSQ4ICB3)

This program runs as a BMP. Start the program using its JCL before any IBM MQ messages are sent to it.

The program searches an IMS database for the account number in the loan request messages. It retrieves the corresponding account name, average balance, and credit worthiness index.

The program sends the results of the database search to the reply-to queue named in the IBM MQ message being processed. The message returned appends the account type and the results of the search to the message received so that the transaction building the response can confirm that the correct query is being processed. The message is in the form of three 79-character groups, as follows:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

When running as a message-oriented BMP, the program drains the IMS message queue, then reads messages from the IBM MQ for z/OS queue and processes them. No information is received from the IMS message queue. The program reconnects to the queue manager after each checkpoint because the handles have been closed.

When running in a batch-oriented BMP, the program continues to be connected to the queue manager after each checkpoint because the handles are not closed.

z/OS *The Message Handler sample on z/OS*

The Message Handler sample TSO application allows you to browse, forward, and delete messages on a queue. The sample is available in C and COBOL.

Preparing and running the sample

Follow these steps:

1. Prepare the sample as described in "Preparing sample applications for the TSO environment on z/OS" on page 1231.
2. Tailor the CLIST (CSQ4RCH1) provided for the sample to define the location of the panels, the location of the message file, and the location of the load modules.

You can use CLIST CSQ4RCH1 to run both the C and the COBOL version of the sample. The supplied version of CSQ4RCH1 runs the C version, and contains instructions on the tailoring necessary for the COBOL version.

Note:

1. There are no sample queue definitions provided with the sample.
2. VS COBOL II does not support multitasking with ISPF, so do not use the Message Handler sample application on both sides of a split screen. If you do, the results are unpredictable.

z/OS *Using the Message Handler sample on z/OS*

Having installed the sample and invoked it from the tailored CLIST CSQ4RCH1, the screen shown in Figure 146 on page 1266 is displayed.

```
----------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :

Queue Manager Name  : ________________________________________________ :

Queue Name      : ________________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP     F8=DOWN    F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

Figure 146. Initial screen for Message Handler sample

Enter the queue manager and queue name to be viewed (case sensitive) and the message list screen is displayed (see Figure 147 on page 1267).

```
---------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

Figure 147. Message list screen for Message Handler sample

This screen shows the first 99 messages on the queue and, for each, shows the following fields:

Msg No
Message number

Put Date MM/DD/YYYY
Date that the message was put on the queue (GMT)

Put Time HH:MM:SS
Time that the message was put on the queue (GMT)

Format Name
MQMD.Format field

User Identifier
MQMD.UserIdentifier field

Put Application Type
MQMD.PutApplType field

Put Application Name
MQMD.PutApplName field

The total number of messages on the queue is also displayed.

From this screen a message can be chosen, by number not by cursor position, and then displayed. For an example, see Figure 148 on page 1268.

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE               :
Forward to Q Mgr : VM03                       :
Forward to Queue : QL.TEST.ISCRES1                 :

Action : _ :  (D)elete (F)orward

Message Content :
------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence    : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `00000000000000000000000000000000000000000000000000000000`X
BackoutCount    : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                `
ReplyToQMgr    : `VM03                      `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                        `
PutApplType    : 000000002
PutApplName    : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .............`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

Figure 148. Chosen message is displayed

Once the message has been displayed it can be deleted, left on the queue, or forwarded to another queue. The `Forward to Q Mgr` and `Forward to Queue` fields are initialized with values from the MQMD, these can be changed before forwarding the message.

The sample design allows only messages with unique `MsgId / CorrelId` combinations to be selected and displayed, because the message is retrieved using the `MsgId` and `CorrelId` as the key. If the key is not unique the sample cannot retrieve the chosen message with certainty.

Note: When you use the SCSQCLST(CSQ4RCH1) sample to browse messages, each invocation causes the backout count of the message to increase. If you want to change the behavior of this sample, copy the sample and modify the contents as necessary. You should be aware that other applications that rely on this backout count can be influenced by this increasing count.

z/OS *Design of the sample Message Handler sample on z/OS*

This topic describes the design of each of the programs that make up the Message Handler sample application.

Object validation program

This requests a valid queue and queue manager name.

If you do not specify a queue manager name, the default queue manager is used, if available. Only local queues can be used; an MQINQ is issued to check that the queue type and an error is reported if the queue is not local. If the queue is not opened successfully, or the MQGET call is inhibited on the queue, error messages are returned indicating the CompCode and Reason return code.

Message list program

This displays a list of messages on a queue with information about them such as the putdate, puttime, and the message format.

The maximum number of messages stored in the list is 99. If there are more messages on the queue than this, the current queue depth is also displayed. To choose a message for display, type the message number into the entry field (the default is 01). If your entry is not valid, you receive an appropriate error message.

Message content program

This displays message content.

The content is formatted and split into two parts:

1. Message descriptor
2. Message buffer

The message descriptor shows the contents of each field on a separate line.

The message buffer is formatted depending on its contents. If the buffer holds a dead letter header (MQDLH) or a transmission queue header (MQXQH), these are formatted and displayed before the buffer itself.

Before the buffer data is formatted, a title line shows the buffer length of the message in bytes. The maximum buffer size is 32768 bytes, and any message longer than this is truncated. The full size of the buffer is displayed along with a message indicating that only the first 32768 bytes of the message are displayed.

The buffer data is formatted in two ways:

1. After the offset into the buffer is printed, the buffer data is displayed in hexadecimal.
2. The buffer data is then displayed again as EBCDIC values. If any EBCDIC value cannot be printed, it prints a period (.) instead.

You can enter D for delete, or F for forward into the action field. If you choose to forward the message, the `forward-to queue` and `queue manager name` must be set correctly. The defaults for these fields are read from the message descriptor ReplyToQ and ReplyToQMgr fields.

If you forward a message, any header block stored in the buffer is stripped. If the message is forwarded successfully, it is removed from the original queue. If you enter invalid actions, error messages are displayed.

An example help panel called CSQ4CHP9 is also available.

z/OS *The Asynchronous Put sample on z/OS*

The Asynchronous Put sample program puts messages on a queue using the asynchronous MQPUT call. The sample also retrieves status information using the MQSTAT call.

The Asynchronous Put applications use these MQI calls:

- MQCONN
- MQOPEN

- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

The sample programs are delivered in the C programming language.

The Asynchronous Put applications run in the batch environment. See Other samples for the batch applications.

This topic also provides information about the design of the Asynchronous Consumption program, and running the CSQ4BCS2 sample.

- "Running the CSQ4BCS2 sample" on page 1270
- "Design of the Asynchronous Put sample program" on page 1270

Running the CSQ4BCS2 sample

This sample program takes up to six parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).
5. The name of the target queue manager (optional).
6. The name of the dynamic queue (optional).

If a queue manager is not specified, CSQ4BCS2 connects to the default queue manager. Message content is provided through standard input (**SYSIN DD**).

There is a sample JCL to run the program, it resides in CSQ4BCSP.

Design of the Asynchronous Put sample program

The program uses the MQOPEN call with either the output options supplied, or with the MQOO_OUTPUT and MQOO_FAIL_IF_QUIESCING options, to open the target queue for putting messages.

If the program cannot open the queue, the program outputs an error message containing the reason code returned by the MQOPEN call. To keep the program simple on this and subsequent MQI calls, default values are used for many of the options.

For each line of input, the program reads the text into a buffer and uses the MQPUT call with MQPMO_ASYNC_RESPONSE to create a datagram message containing the text of that line and asynchronously puts the message on the target queue. The program continues until it reaches the end of the input, or until the MQPUT call fails. If the program reaches the end of the input, it closes the queue using the MQCLOSE call.

The program then issues the MQSTAT call which returns an MQSTS structure, and displays messages containing the number of messages put successfully, the number of messages put with a warning, and the number of failures.

Note: To observe what happens when an MQPUT error is detected by the MQSTAT call, set MAXDEPTH on the target queue to a low value.

z/OS *The Batch Asynchronous Consumption sample on z/OS*

The CSQ4BCS1 sample program is delivered in C, it demonstrates the use of MQCB and MQCTL to consume messages from multiple queues asynchronously.

The Asynchronous Consumption samples run in the batch environment. See Other samples for the batch applications.

There is also a COBOL sample which runs in the CICS environment, see “The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” on page 1272.

The applications use these MQI calls:

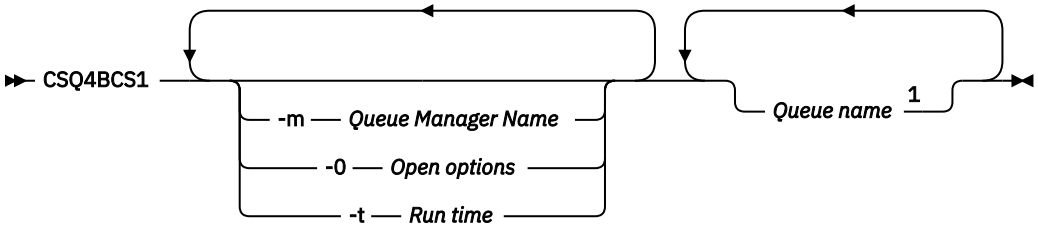
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

This topic also provided information about the following headings:

- “Running the CSQ4BCS1 sample” on page 1271
- “Design of the Batch Asynchronous Consumption sample program” on page 1271

Running the CSQ4BCS1 sample

This sample program follows the following syntax:



Notes:

[1] A maximum of 10 queues are supported by this sample. Provide more than one queue name in order to read messages from multiple queues.

There is a sample JCL to run this program, it resides in CSQ4BCSC.

Design of the Batch Asynchronous Consumption sample program

The sample shows how to read messages from multiple queues in the order of their arrival. This would require more code using synchronous MQGET. With asynchronous consumption, no polling is required, and thread and storage management is performed by IBM MQ. In the sample program, errors are written to the console.

The sample code has the following steps:

1. Define the single message consumption callback function.

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Connect to the queue manager.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Open the input queues, and associate each queue with the `MessageConsumer` callback function.

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   `cbd.CallbackFunction` does not need to be set for each queue; it is an input-only field. You can associate a different callback function with each queue.

4. Start consumption of the messages.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Wait for the user to press Enter, then stop consumption of messages.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Finally, disconnect from the queue manager.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS*

The Asynchronous Consumption and Publish/Subscribe sample programs demonstrate the use of asynchronous consumption, and publish and subscribe features within CICS.

A *Registration client* program registers three Callback handlers (an event handler, and two message consumers), and starts Asynchronous Consumption. A *Messaging client* program puts messages to a queue, or publishes suitable messages from a CICS console for consumption by the two Message Consumers (CSQ4CVCN and CSQ4CVCT).

To provide runtime control over the behavior of the sample, one of the message consumers can be instructed using the messages it receives, to SUSPEND, RESUME, or DEREGISTER any of the Callback handlers. It can also be used to issue an MQCTL STOP to end Asynchronous Consumption under control. The other message consumer is registered to subscribe to a topic.

Each program issues COBOL DISPLAY statements at appropriate points to display the behavior of the sample.

The applications use these MQI calls:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

The programs are delivered in the COBOL language. See CICS Asynchronous Consumption and Publish/Subscribe samples for the CICS applications.

This topic also provides the following information:

- “Setup” on page 1273
- “Registration Client CSQ4CVRG” on page 1273
- “Event handler CSQ4CVEV” on page 1273
- “Simple Message Consumer CSQ4CVCN” on page 1273

Setup

The names of the Queue and Topic used by the Message Consumers are hardcoded in the Registration and Messaging Client programs.

The Queue, **SAMPLE.CONTROL.QUEUE**, should be defined to the Queue Manager associated with the CICS region before running the sample. The Topic, **News/Media/Movies**, can be defined if required, or it is created at runtime under the default Administrative Object if it does not exist.

CICS programs and transaction definitions can be installed by installing a group: CSQ4SAMP.

Registration Client CSQ4CVRG

The Registration Client program must be started under the CICS transaction MVRG. It takes no input.

When started, the Registration Client registers the following Callback handlers using MQCB:

- CSQ4CVEV as an Event Handler.
- CSQ4CVCN as a Message Consumer on a topic, **News/Media/Movies**.
- CSQ4CVCT as a Message Consumer on a Queue, **SAMPLE.CONTROL.QUEUE**.

The Registration Client passes a data structure containing the names of all three registered Callback handlers to CSQ4CVCT, together with the object handles associated with the two message consumers.

Having registered the Callback handlers, the Registration Client issues an MQCTL START_WAIT to start Asynchronous Consumption, and suspend until control is returned to it (for example, by one of the Callback handlers issuing an MQCTL STOP).

Event handler CSQ4CVEV

When driven, the Event Handler displays a message indicating the call type (for example, START). When driven for IBM MQ reason code CONNECTION_QUIESCING, the Event Handler issues an MQCTL STOP to end Asynchronous Consumption and return control to the Registration Client.

Simple Message Consumer CSQ4CVCN

When driven, this Message Consumer displays a message indicating the call type (for example, REGISTER). When driven for the MSG_REMOVED call type, the Message Consumer retrieves the inbound message and outputs it to the CICS job log.

Control Message Consumer CSQ4CVCT

When driven, this Message Consumer displays a message indicating the call type (for example, START). When driven for the MSG_REMOVED call type, the Message Consumer retrieves the inbound message and the data structure passed by the Registration Client. Based on the message content, it issues appropriate MQCB or MQCTL commands to one of the following:

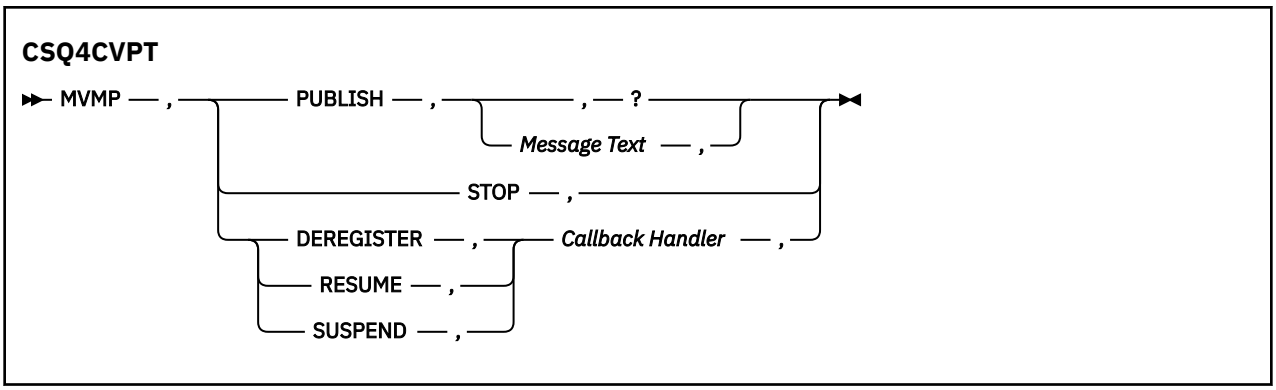
- STOP Asynchronous Consumption (returning control to the Registration Client).
- SUSPEND, RESUME, or DEREGISTER a named Callback handler (including itself).

Messaging Client CSQ4CVPT

The Messaging Client has two functions:

- It publishes a message to a topic for consumption by the Message Consumer CSQ4CVCN.
- It puts a control message to a queue for consumption by the Control Message Consumer CSQ4CVCT, resulting in a potential change in behavior of the sample.

The Messaging Client program must be started from a CICS console under a CICS transaction, and it takes command line input with the following syntax:



PUBLISH
Publish the Message Text (or a default message) as a Retained Message for consumption by the Simple Message Consumer.

STOP
Stop Asynchronous Consumption.

DEREGISTER
Deregister the named Callback handler.

RESUME
Resume the named Callback handler.

SUSPEND
Suspend the named Callback handler.

Input fields are positional, and comma-separated. Keywords and Callback Handler names are not case-sensitive.

Examples:

Table 186. Input examples

Example	Description
MVMP,PUBLISH,,	Publish a default message
MVMP,publish, `A short message`,	Publish the given text
MVMP,STOP,	Stop Asynchronous Consumption
MVMP,DEREGISTER,CSQ4CVEV,	Deregister the Event Handler
MVMP,resume,csq4cvcn,	Resume the Simple Message Consumer
MVMP,SUSPEND,CSQ4CVEV,	Suspend the Event Handler

Where MVMP is the CICS transaction associated with the Messaging Client program CSQ4CVPT.

Note:

- Suspending or deregistering all Callback handlers terminates the START_WAIT issued by the Registration Client, returning control to it, and ending the task.
- Suspending or deregistering the Control Callback Handler has deliberately not been prevented, but it removes the ability to further control the behavior of the sample.

z/OS *The Publish/Subscribe sample on z/OS*

The Publish/Subscribe sample programs demonstrate the use of the publish and subscribe features in IBM MQ.

There are four C and two COBOL programming language sample programs demonstrating how to program to the IBM MQ Publish/Subscribe interface. The programs are delivered in the C and COBOL language. The applications run in the batch environment; see Publish/Subscribe samples for the batch applications.

There are also COBOL samples that run in the CICS environment; see "The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS" on page 1272.

This topic also provides information about how to run Publish/Subscribe sample programs. These sample programs include:

Running the CSQ4BCP1 sample

This program is written in C; it publishes messages to a topic. Start one of the subscriber samples before running this program.

This program takes up to four parameters:

1. The name of the target topic string (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).

If a queue manager is not specified, CSQ4BCP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPP.

Message content is provided through standard input (**SYSIN DD**).

Running the CSQ4BCP2 sample

This program is written in C; it subscribes to a topic and prints the messages received.

This program takes up to three parameters:

1. The name of the target topic string (required).
2. The name of the queue manager (optional).
3. MQSD subscription options (optional).

If a queue manager is not specified, CSQ4BCP2 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPS.

Running the CSQ4BCP3 sample

This program is written in C; it subscribes to a topic using a user-specified destination queue and prints the messages received.

This program takes up to four parameters:

1. The name of the target topic string (required).
2. The name of the destination (required).
3. The name of the queue manager (optional).
4. MQSD subscription options (optional).

If a queue manager is not specified, CSQ4BCP3 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPD.

Running the CSQ4BCP4 sample

This program is written in C; it subscribes and gets messages from a topic allowing the use of extended options on the MQSUB call, extending those available on the simpler MQSUB sample: CSQ4BCP2. In addition to the message payload, message properties for each message are received and displayed.

This program takes a variable set of parameters:

- **-t** *Topic string*.
- **-o** *Topic object name*.

  Important: One of **-t** or **-o**, or both, is required
- **-m** *Queue manager name* (optional).
- **-b** *Connection binding type* (optional), where *type* can have any of the following values:
  - *standard* : MQCNO_STANDARD_BINDING , which is the default value
  - *shared*: MQCNO_SHARED_BINDING
  - *fastpath*: MQCNO_FASTPATH_BINDING
  - *isolated*: MQCNO_ISOLATED_BINDING
- **-q** *Destination queue name* (optional).
- **-w** *Wait interval on MQGET in seconds* (optional), where *seconds* can have any of the following values:
  - unlimited: MQWI_UNLIMITED
  - none: No wait
  - *n*: Wait interval in seconds
  - No value specified: When no value is specified, the default is 30 seconds
- **-d** *Subscription name* (optional). Creates or resumes named durable subscription.
- **-k** (optional). Keeps durable subscription on MQCLOSE.

If a queue manager is not specified, CSQ4BCP4 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPE.

Running the CSQ4BVP1 sample

This program is written in COBOL, it publishes messages to a topic. Start one of the subscriber samples before running this program.

This program takes no parameters. **SYSIN DD** provides the input topic name, queue manager name, and message content.

If a queue manager is not specified, CSQ4BVP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BVPP.

Running the CSQ4BVP2 sample

This program is written in COBOL, it subscribes to a topic and prints the messages received.

This program takes no parameters. **SYSIN DD** provides the input for topic name and queue manager name.

If a queue manager is not specified, CSQ4BVP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BVPP.

z/OS *The Set and Inquire message property sample on z/OS*

The message property sample programs demonstrate the addition of user-defined properties to a message handle, and the inquisition of the properties associated with that message.

The applications use these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

The programs are delivered in the C language. The applications run in the batch environment. See Other samples for the batch applications.

The CSQ4BCM1 program is used to inquire the properties of a message handle from a message queue, and it is an example of the use of the MQINQMP API call. The sample gets one message from a queue and then prints all the message handle properties.

The CSQ4BCM2 program is used to set the properties of a message handle on a message queue, and it is an example of the use of the MQSETMP API call. The sample creates a message handle and puts it into the `MsgHandle` field of the MQGMO structure. It then puts the message to a queue.

Other examples of inquiring and printing message properties are included in the CSQ4BCG1 and CSQ4BCP4 sample programs.

This topic also provides information on running the Set and Inquire message property samples under the following headings:

- "Running the CSQ4BCM1 sample" on page 1277
- "Running the CSQ4BCM2 sample" on page 1277

Running the CSQ4BCM1 sample

This program takes up to four parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).

Running the CSQ4BCM2 sample

This program takes up to six parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).
5. The name of the target queue manager (optional).
6. The name of the dynamic queue (optional).

The property names, values, and message content are provided through the standard input (**SYSIN DD**). There is a sample JCL to run the program, it resides in CSQ4BCMP.

Anwendungen für Managed File Transfer entwickeln

Geben Sie Programme an, die mit Managed File Transfer ausgeführt werden sollen, verwenden Sie Apache Ant mit Managed File Transfer, passen Sie Managed File Transfer mit Benutzerexits an und steuern Sie Managed File Transfer, indem Sie Nachrichten in die Befehlswarteschlange des Agenten einreihen.

Programme angeben, die mit MFT ausgeführt werden sollen

Auf einem System, auf dem ein Managed File Transfer Agent aktiv ist, können Programme ausgeführt werden. Bei einer Dateiübertragungsanforderung können Sie auch ein Programm angeben, das vor dem Beginn der Übertragung oder im Anschluss an die Übertragung ausgeführt werden soll. Darüber hinaus können Sie durch Übergabe einer Anforderung eines verwalteten Aufrufs auch Programme starten, die nicht Teil einer Dateiübertragungsanforderung sind.

Informationen zu diesem Vorgang

Es gibt fünf Möglichkeiten für die Ausführung von Programmen:

- Als Teil einer Übertragungsanforderung beim Quellenagenten, bevor die Übertragung gestartet wird.
- Als Teil einer Übertragungsanforderung beim Zielagenten, bevor die Übertragung gestartet wird.
- Als Teil einer Übertragungsanforderung auf dem Quellenagenten nach Abschluss der Übertragung.
- Als Teil einer Übertragungsanforderung auf dem Zielagenten nach Abschluss der Übertragung.
- Unabhängig von einer Übertragungsanforderung. Sie können an den Agenten eine Anforderung zur Ausführung eines Programms übergeben. Dieses Szenario wird auch als verwalteter Aufruf bezeichnet.

Benutzerexits und Programmaufrufe werden in der folgenden Reihenfolge aufgerufen:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Anmerkungen:

1. Die **DestinationTransferEndExits** wird nur ausgeführt, wenn die Übertragung erfolgreich oder teilweise erfolgreich abgeschlossen wird.
2. Die **postDestinationCall** wird nur ausgeführt, wenn die Übertragung erfolgreich oder teilweise erfolgreich abgeschlossen wird.
3. **SourceTransferEndExits** wird für erfolgreiche, teilweise erfolgreiche oder fehlgeschlagene Übertragungen ausgeführt.
4. **postSourceCall** wird nur in folgenden Fällen aufgerufen:
   - Die Übertragung wurde nicht abgebrochen.
   - Es gibt ein erfolgreiches oder teilweise erfolgreiches Ergebnis.
   - Alle 'postdestination'-Übertragungsprogramme wurden erfolgreich ausgeführt.

Prozedur

- Geben Sie das Programm, das ausgeführt werden soll, mit einer der folgenden Optionen an:

Verwendung einer Apache Ant-Task

Verwenden Sie eine der Tasks `fte:filecopy`, `fte:filemove` und `fte:call` Ant, um ein Programm zu starten. Mit einer Ant-Task können Sie ein Programm in einem der fünf Szenarios angeben, indem Sie die verschachtelten Elemente `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrc` und `fte:command` verwenden. Weitere Informationen finden Sie im Abschnitt Verschachtelte Elemente des Programmaufrufs.

Durch Bearbeiten der Nachricht mit der Dateiübertragungsanforderung

Sie können die durch eine Übertragungsanforderung generierte XML-Datei bearbeiten. Mit dieser Methode können Sie ein Programm in einem der fünf Szenarios ausführen, indem Sie der XML-Datei die Elemente **preSourceCall**, **postSourceCall**, **preDestinationCall**, **postDestinationCall** und **managedCall** hinzufügen. Anschließend kann diese modifizierte XML-Datei als Übertragungsdefinition für eine neue Übertragungsanforderung verwendet werden, beispielsweise unter Angabe des **fteCreateTransfer**-Parameters **-td**. Weitere Informationen finden Sie im Abschnitt Beispiele für Anrufanforderungsnachrichten des MFT-Agenten.

Mithilfe des Befehls `fteCreateTransfer`

Mit dem Befehl **fteCreateTransfer** können Sie Programme angeben, die gestartet werden sollen. Allerdings können mit diesem Befehl nur die ersten vier der oben aufgeführten fünf Szenarios abgedeckt werden; der Start eines verwalteten Aufrufs ist mit diesem Befehl nicht möglich. Informationen zu den zu verwendenden Parametern finden Sie im Abschnitt **fteCreateTransfer**: Neue Dateiübertragung starten. Beispiele für die Verwendung dieses Befehls finden Sie im Abschnitt Beispiele für Verwendung von fteCreateTransfer zum Starten von Programmen.

Zugehörige Verweise

MFT-Eigenschaft 'commandPath'

Verwaltete Aufrufe

Managed File Transfer-Agenten (MFT) werden normalerweise zum Übertragen von Dateien oder Nachrichten verwendet. Diese werden als *Verwaltete Übertragungen* bezeichnet. Agenten können auch verwendet werden, um Befehle, Scripts oder JCL auszuführen, ohne dass Dateien oder Nachrichten übertragen werden müssen. Diese Funktion wird als *Verwaltete Aufrufe* bezeichnet.

Anforderungen für verwaltete Aufrufe können auf verschiedene Arten an einen Agenten übergeben werden:

- Die Ant-Task 'fte:call'verwenden.
- Konfigurieren einer Ressourcenüberwachung mit einer Task-XML, die einen Befehl oder ein Script ausführt. Weitere Informationen finden Sie in Überwachungstasks zum Starten von Befehlen und Scripts konfigurieren .
- Direkte Einreihung einer XML-Nachricht in die Befehlswarteschlange des Agenten. Weitere Details zum XML-Schema für verwaltete Aufrufe finden Sie unter Nachrichtenformat für Dateiübertragungsanforderung .

Bei verwalteten Aufrufen muss das Verzeichnis mit dem Befehl oder Script, der bzw. das ausgeführt wird, in der Agenteneigenschaft **commandPath** angegeben werden.

Verwaltete Aufrufe können keine Befehle oder Scripts ausführen, die sich in Verzeichnissen befinden, die nicht im **commandPath**des Agenten angegeben sind. Dadurch wird sichergestellt, dass der Agent keinen zerstörerischen Programmcode ausführt.

Wichtig: Gehen Sie wie folgt vor, um sicherzustellen, dass dies der Fall ist, wenn Sie **commandPath**angeben:

- Alle vorhandenen Agentensandboxes werden vom Agenten beim Start konfiguriert, sodass alle **commandPath** -Verzeichnisse automatisch zur Liste der Verzeichnisse hinzugefügt werden, die den Zugriff auf eine Übertragung verweigert haben.

- Alle vorhandenen Benutzersandboxes werden beim Start des Agenten aktualisiert, sodass alle **commandPath** -Verzeichnisse (und ihre Unterverzeichnisse) als <exclude> -Elemente zu den Elementen <read> und <write> hinzugefügt werden.
- Wenn der Agent nicht für die Verwendung einer Agentensandbox oder einer Benutzersandbox konfiguriert ist, wird beim Start des Agenten eine neue Agentensandbox erstellt, in der die Verzeichnisse **commandPath** als verweigerte Verzeichnisse angegeben sind.

Darüber hinaus können Sie die Berechtigungsprüfung für einen Agenten aktivieren, um sicherzustellen, dass nur berechtigte Benutzer verwaltete Aufrufanforderungen übergeben dürfen. Weitere Informationen hierzu finden Sie in Benutzerberechtigungen für MFT-Agentenaktionen beschränken.

Der Befehl, das Script oder die JCL, die als Teil eines verwalteten Aufrufs aufgerufen wird, wird als externer Prozess ausgeführt, der vom Agenten überwacht wird. Wenn der Prozess beendet wird, wird der verwaltete Aufruf abgeschlossen und der Rückkehrcode des Prozesses wird entweder dem Agenten oder dem Ant -Script zur Verfügung gestellt, das die **fte:call** Ant -Task aufgerufen hat.

Wenn der verwaltete Aufruf von der Aufgabe **fte:call** Ant gestartet wurde, kann Ihr Script Ant den Wert des Rückgabecodes überprüfen, um festzustellen, ob der verwaltete Aufruf erfolgreich war oder nicht.

Für alle anderen Typen verwalteter Aufrufe können Sie angeben, welche Rückkehrcodewerte verwendet werden sollen, um anzuzeigen, dass der verwaltete Aufruf erfolgreich ausgeführt wurde. Der Agent vergleicht den Rückkehrcode des Prozesses mit diesen Rückkehrcodes, wenn der externe Prozess beendet ist.

Anmerkung: Da verwaltete Aufrufe als externe Prozesse ausgeführt werden, können sie nach dem Start nicht abgebrochen werden.

Verwaltete Aufrufe und Quellenübertragungsslots

Ein Agent enthält eine Reihe von Quellenübertragungsslots, wie durch die Agenteneigenschaft **maxSourceTransfers** angegeben, die in Erweiterte Agenteneigenschaften: Übertragungslimitbeschrieben werden.

Wenn ein verwalteter Aufruf oder eine verwaltete Übertragung ausgeführt wird, belegen sie ein Quellenübertragungszeitfenster. Der Steckplatz wird freigegeben, sobald der verwaltete Aufruf oder die verwaltete Übertragung abgeschlossen ist.

Wenn alle Quellenübertragungsslots im Gebrauch sind, wenn ein Agent entweder einen neuen verwalteten Aufruf oder eine verwaltete Übertragungsanforderung empfängt, wird die Anforderung vom Agenten in die Warteschlange gestellt, bis ein Slot verfügbar ist.

Wenn ein verwalteter Aufruf eine verwaltete Übertragung startet (z. B. wenn ein verwalteter Aufruf ein Ant-Script ausführt und dieses Ant-Script die Task fte: filecopy oder fte: filemove zum Übertragen einer Datei verwendet), sind zwei Quellenübertragungssegmente erforderlich:

- Eine für die verwaltete Übertragung
- Eine für den verwalteten Aufruf

In dieser Situation ist es wichtig zu beachten, dass die beiden Quellenübertragungszeitfenster belegt sind, bis die verwaltete Übertragung abgeschlossen ist, abgebrochen wird oder das Zeitlimit aufgrund einer **transferRecoveryTimeout**überschritten wird, wenn die verwaltete Übertragung entweder lange dauert oder eine Wiederherstellung in Anspruch nimmt. Ausführliche Informationen zu **transferRecoveryTimeout**finden Sie unter Zeitlimitkonzepte für die Übertragungswiederherstellung . Dies kann die Anzahl anderer verwalteter Übertragungen oder Aufrufe, die der Agent verarbeiten kann, potenziell begrenzen.

Aus diesem Grund sollten Sie das Design eines verwalteten Aufrufs in Betracht ziehen, um sicherzustellen, dass er Quellenübertragungszeitfenster über einen langen Zeitraum nicht belegt.

REST API mit verwalteten Aufrufen verwenden

Die HTTP-Verben GET und HTTP POST werden für die Aktivierung verwalteter Aufrufe unterstützt und funktionieren nur in Version 3 von REST API.

Andere Verben, z. B. HTTP DELETE und HTTP UPDATE, werden nicht unterstützt und geben den HTTP-Fehlercode 405 zurück, wenn Sie versuchen, sie zu verwenden.

Achtung: Nach der Übergabe kann ein verwalteter Aufruf nicht über die REST APIabgebrochen werden.

Apache Ant mit MFT verwenden

In Managed File Transfer stehen Tasks bereit, mit denen Dateiübertragungsfunktionen in das Apache Ant-Tool integriert werden können.

Mithilfe des Befehls **`fteAnt`** können Sie Ant-Tasks in einer Managed File Transfer-Umgebung ausführen, die Sie bereits konfiguriert haben. Sie können Ant-Dateiübertragungstasks aus Ihren Ant-Scripts verwenden, um komplexe Dateiübertragungsoperationen über eine interpretierte Scripting-Sprache zu koordinieren.

Weitere Informationen zu Apache Ant finden Sie auf der Webseite des Apache Ant-Projekts unter https://ant.apache.org/.

Zugehörige Konzepte

„Einführung in die Verwendung von Ant-Scripts mit MFT” auf Seite 1281
Durch die Verwendung von Ant-Scripts mit Managed File Transfer können Sie komplexe Dateiübertragungen über eine interpretierte Scriptsprache koordinieren.

`fteAnt`: Ant-Tasks in MFT ausführen

Zugehörige Verweise

„Ant-Beispieltasks für MFT” auf Seite 1282
Es gibt eine Reihe von Ant -Beispielscripts, die mit Ihrer Installation von Managed File Transferbereitgestellt werden. Diese Beispiele befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*`/mqft/samples/fteant`. Jedes Beispielscript enthält ein `init`-Ziel. Bearbeiten Sie die Eigenschaftengruppe im `init`-Ziel, um diese Scripts mit Ihrer Konfiguration auszuführen.

Einführung in die Verwendung von Ant-Scripts mit MFT

Durch die Verwendung von Ant-Scripts mit Managed File Transfer können Sie komplexe Dateiübertragungen über eine interpretierte Scriptsprache koordinieren.

Ant-Scripts

Ant-Scripts (oder Builddateien) sind XML-Dokumente, die ein oder mehrere Ziele definieren. Diese Ziele enthalten auszuführende Taskelemente. Managed File Transfer stellt Tasks bereit, mit denen Sie die Dateiübertragungsfunktion in Apache Ant integrieren können. Weitere Informationen zu Ant-Scripts finden Sie auf folgender Website des Apache Ant-Projekts: https://ant.apache.org/.

Beispiele für Ant-Scripts, die Managed File Transfer-Tasks verwenden, finden Sie in Ihrer Produktinstallation im Verzeichnis *MQ_INSTALLATION_PATH*`/mqft/samples/fteant`.

Auf Protokollbridgeagenten werden Ant-Scripts auf dem System des Protokollbridgeagenten ausgeführt. Diese Ant-Scripts haben keinen direkten Zugriff auf die Dateien auf dem FTP- oder SFTP-Server.

Namensbereich

Ein Namensbereich dient dazu, die Ant-Tasks für Dateiübertragungen von anderen Ant-Tasks zu unterscheiden, die möglicherweise denselben Namen nutzen. Der Namensbereich wird mit dem Tag 'project' im Ant-Script festgelegt.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

Das Attribut `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` weist Ant an, in der Bibliothek `com.ibm.wmqfte.ant.taskdefs` nach Definitionen von Tasks mit dem Präfix `fte` zu suchen.

`fte` muss nicht als Namensbereichspräfix verwendet werden; Sie können jeden beliebigen Wert verwenden. Das Namensbereichspräfix `fte` wird in allen Beispielen und Ant-Beispielscripts verwendet.

Ant-Scripts ausführen

Verwenden Sie zur Ausführung von Ant-Scripts, die Ant-Tasks für Dateiübertragungen enthalten, den Befehl **fteAnt**. For example:

```
fteAnt -file ant_script_location/ant_script_name
```

Weitere Informationen finden Sie unter **fteAnt**: Ausführen von Ant-Tasks in MFT.

Rückkehrcodes

Von den Ant-Tasks für Dateiübertragungen werden dieselben Rückkehrcodes wie von den Managed File Transfer-Befehlen zurückgegeben. Weitere Informationen finden Sie im Abschnitt Rückgabecodes für MFT.

Zugehörige Verweise

fteAnt: Ant-Tasks in MFT ausführen

„Ant-Beispieltasks für MFT" auf Seite 1282

Es gibt eine Reihe von Ant -Beispielscripts, die mit Ihrer Installation von Managed File Transferbereitgestellt werden. Diese Beispiele befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Jedes Beispielscript enthält ein `init`-Ziel. Bearbeiten Sie die Eigenschaftengruppe im `init`-Ziel, um diese Scripts mit Ihrer Konfiguration auszuführen.

Ant-Beispieltasks für MFT

Es gibt eine Reihe von Ant -Beispielscripts, die mit Ihrer Installation von Managed File Transferbereitgestellt werden. Diese Beispiele befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Jedes Beispielscript enthält ein `init`-Ziel. Bearbeiten Sie die Eigenschaftengruppe im `init`-Ziel, um diese Scripts mit Ihrer Konfiguration auszuführen.

email

Im Beispiel email ist dargestellt, wie unter Verwendung von Ant-Tasks eine Datei übertragen und im Falle einer fehlgeschlagenen Übertragung eine E-Mail an eine angegebene E-Mail-Adresse gesendet werden kann. Das Script überprüft mithilfe der Managed File Transfer-Task 'ping', ob der Quellen- und der Zielagent aktiv sind und Übertragungen verarbeiten können. Wenn beide Agenten aktiv sind, überträgt das Script mithilfe der Managed File Transfer-Task 'fte:filecopy' eine Datei zwischen dem Quellen- und dem Zielagenten, ohne die Originaldatei zu löschen. Bei einer gescheiterten Übertragung sendet das Script mithilfe der Ant-Standardtask email eine E-Mail mit Fehlerinformationen.

hub

Das Beispiel hub besteht aus den beiden Scripts `hubcopy.xml` und `hubprocess.xml`.. Das Script `hubcopy.xml` zeigt, wie mithilfe von Ant-Scripts Hub-and-Spoke-Topologien aufgebaut werden können. In diesem Beispiel werden zwei Dateien von Agenten auf Peripheriesystemen (Spoke-Systeme) an einen Agenten auf einem zentralen System (Hub) übertragen. Beide Dateien werden gleichzeitig übertragen; nach Abschluss der Übertragungen wird das Ant-Script `hubprocess.xml` auf der Hub-Maschine ausgeführt, um die Dateien zu verarbeiten. Wenn beide Dateien fehlerfrei übertragen werden, verknüpft das Ant-Script die Inhalte der Dateien. Werden die Dateien nicht korrekt übertragen, führt das Ant-Script eine Bereinigung durch und löscht zu diesem Zweck alle übertragenen Dateidaten. Damit dieses Beispiel richtig funktioniert, muss das Script `hubprocess.xml` in den Befehlspfad des Hub-Agenten gestellt werden. Weitere Informationen zur Festlegung des Befehlspfads eines Agenten finden Sie im Abschnitt commandPath MFT-Eigenschaft.

librarytransfer (nur IBM i-Plattform)

IBM i

IBM i Das Beispiel 'librarytransfer' zeigt, wie mithilfe von Ant-Tasks eine IBM i-Bibliothek von einem IBM i-System auf ein zweites IBM i-System übertragen werden kann.

IBM i Im Beispiel 'librarytransfer' wird die Unterstützung nativer Sicherungsdateien unter IBM i in Verbindung mit vordefinierten Ant-Tasks in Managed File Transfer für die Übertragung nativer Bibliotheksobjekte zwischen zwei IBM i-Systemen eingesetzt. Das Beispiel verwendet ein verschachteltes < presrc> -Element in einer Managed File Transfer -Task 'filecopy' zum Aufrufen eines ausführbaren Scripts `librarysave.sh` , das die angeforderte Bibliothek auf dem Quellenagentensystem in einer temporären Sicherungsdatei speichert. Die Sicherungsdatei wird von der Ant-Task filecopy auf das Zielagentensystem verschoben, auf dem ein verschachteltes < postdst> -Element zum Aufrufen des ausführbaren Scripts `libraryrestore.sh` verwendet wird, um die in der Sicherungsdatei gespeicherte Bibliothek auf dem Zielsystem wiederherzustellen.

IBM i Vor der Ausführung dieses Beispiels müssen Sie die in der Datei `librarytransfer.xml` beschriebenen Konfigurationstasks ausführen. Sie müssen außerdem über eine funktionsfähige Managed File Transfer-Umgebung auf zwei IBM i-Maschinen verfügen. Die Konfiguration muss einen Quellenagenten auf der ersten IBM i-Maschine und einen Zielagenten auf der zweiten IBM i-Maschine beinhalten. Die beiden Agenten müssen miteinander kommunizieren können.

IBM i Das Beispiel "librarytransfer" besteht aus den folgenden drei Dateien:

- `librarytransfer.xml`
- `librarysave.sh` (ausführbares Script für <presrc>)
- `libraryrestore.sh` (ausführbares Script für <postdst>)

Die Beispieldateien befinden sich im folgenden Verzeichnis: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer`.

IBM i Zur Ausführung dieses Beispiels muss der Benutzer die folgenden Schritte ausführen:

1. Starten Sie eine Qshell-Sitzung. Geben Sie in einem IBM i-Befehlsfenster Folgendes ein: STRQSH
2. Wechseln Sie wie folgt in das Verzeichnis `bin`:

   ```
   cd  /QIBM/ProdData/WMQFTE/V7/bin
   ```

3. Führen Sie das Beispiel nach Abschluss der erforderlichen Konfiguration mit dem folgenden Befehl aus:

   ```
   fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
   ```

physicalfiletransfer (nur IBM i-Plattform)

IBM i Das Beispiel 'physicalfiletransfer' veranschaulicht, wie mithilfe von Ant-Tasks eine Datei vom Typ 'Source Physical' (physische Quelle) oder 'Database' (Datenbank) aus einer Bibliothek in einem IBM i-System in eine Bibliothek in einem zweiten IBM i-System übertragen werden kann.

IBM i Im Beispiel 'physicalfiletransfer' wird die Unterstützung nativer Sicherungsdateien unter IBM i in Verbindung mit vordefinierten Ant-Tasks in Managed File Transfer für die Übertragung kompletter 'Source Physical'- und 'Database'-Dateien zwischen zwei IBM i-Systemen eingesetzt. Das Beispiel verwendet ein verschachteltes < presrc> -Element in einer Managed File Transfer -Task 'filecopy' zum Aufrufen eines ausführbaren Scripts `physicalfilesave.sh` , um die angeforderte physische Quellendatei oder Datenbankdatei aus einer Bibliothek auf dem Quellenagentensystem in einer temporären Sicherungsdatei zu speichern. Die Sicherungsdatei wird von der Ant-Task 'filecopy' auf das Zielagentensystem verschoben, wo ein verschachteltes < postdst> -Element verwendet wird, um das ausführbare Script `physicalfilerestore.sh` aufzurufen. Anschließend wird das Dateiobjekt in der Sicherungsdatei in einer angegebenen Bibliothek auf dem Zielsystem wiederhergestellt.

IBM i Vor der Ausführung dieses Beispiels müssen Sie einige in der Datei `physicalfiletransfer.xml` beschriebene Konfigurationstasks ausführen. Sie müssen außerdem über eine funktionsfähige Managed File Transfer-Umgebung auf zwei IBM i-Systemen verfügen. Die Konfiguration muss einen Quellenagenten auf dem ersten IBM i-System und einen Zielagenten auf dem zweiten IBM i-System beinhalten. Die beiden Agenten müssen miteinander kommunizieren können.

IBM i Das Beispiel "physicalfiletransfer" besteht aus den folgenden drei Dateien:

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (ausführbares Script für <presrc>)
- `physicalfilerestore.sh` (ausführbares Script für <postdst>)

Die Beispieldateien befinden sich im folgenden Verzeichnis: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`.

IBM i Zur Ausführung dieses Beispiels muss der Benutzer die folgenden Schritte ausführen:

1. Starten Sie eine Qshell-Sitzung. Geben Sie in einem IBM i-Befehlsfenster Folgendes ein: STRQSH
2. Wechseln Sie wie folgt in das Verzeichnis `bin`:

   ```
   cd  /QIBM/ProdData/WMQFTE/V7/bin
   ```

3. Führen Sie das Beispiel nach Abschluss der erforderlichen Konfiguration mit dem folgenden Befehl aus:

   ```
   fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/physicalfile
   transfer.xml
   ```

Zeitlimit

Das Beispiel timeout zeigt, wie mithilfe von Ant-Tasks eine Dateiübertragung versucht und bei Überschreiten eines angegebenen Zeitlimitwertes abgebrochen werden kann. Das Script startet mit der Managed File Transfer-Task 'fte:filecopy' eine Dateiübertragung. Bei dem Ergebnis dieser Übertragung kommt es zu einer Verzögerung. Das Script verwendet Managed File Transfer Die Ant-Task 'fte:awaitoutcome', um eine bestimmte Anzahl Sekunden auf den Abschluss der Übertragung zu warten. Wenn die Übertragung nicht innerhalb der angegebenen Zeit abgeschlossen wird, wird die Managed File Transfer Die Task 'fte:cancel Ant ' zum Abbrechen der Dateiübertragung verwendet.

vsamtransfer

z/OS

z/OS Das Beispiel vsamtransfer zeigt, wie mithilfe von Ant-Tasks unter Verwendung von Managed File Transfer eine Übertragung aus einem VSAM-Dataset in ein anderes VSAM-Dataset erfolgen kann. Managed File Transfer unterstützt derzeit keine Übertragung von VSAM-Datasets. Das Beispielscript entlädt die VSAM-Datensätze in eine sequenzielle Datei, indem es die `presrc` Verschachtelte Elemente des Programmaufrufs zum Aufrufen der ausführbaren Datei `datasetcopy.sh` verwendet. Das Script überträgt die sequenzielle Datei mithilfe der Managed File Transfer-Task 'fte:filemove' vom Quellenagenten an den Zielagenten. Das Script verwendet dann `postdst` Verschachtelte Elemente des Programmaufrufs, um das Script `loadvsam.jcl` aufzurufen. Dieses JCL-Script übernimmt die übertragenen Datensätze in einen VSAM-Zieldatensatz. In diesem Beispiel wird JCL für den Zielaufruf verwendet, um diese Sprachoption zu veranschaulichen. Dasselbe Ergebnis lässt sich auch bei Verwendung eines zweiten Shell-Scripts erzielen.

z/OS Bei den in diesem Beispiel verwendeten Quellen- und Zieldatensätzen muss es sich nicht VSAM-Datensätze handeln. Das Beispiel funktioniert für alle Dateien, wenn die Quellen- und Zieldateien demselben Typ angehören.

z/OS Damit dieses Beispiel richtig funktioniert, muss das Script `datasetcopy.sh` in den Befehlspfad des Quellenagenten und das Script `loadvsam.jcl` in den Befehlspfad des Zielagenten gestellt werden. Weitere Informationen zur Festlegung des Befehlspfads eines Agenten finden Sie im Abschnitt commandPath MFT-Eigenschaft.

zip

Das Beispiel zip besteht aus den beiden Scripts `zip.xml` und `zipfiles.xml`. Das Beispiel veranschaulicht, wie die `presrc` verschachteltes Element in der Task Managed File Transfer fte: filemove verwendet wird, um ein Script Ant auszuführen, bevor eine Verschiebeoperation für Dateiübertragungen ausgeführt wird. Das Script `zipfiles.xml`, das vom verschachtelten Element `presrc` im Script `zip.xml` aufgerufen wird, komprimiert den Inhalt eines Verzeichnisses. Das Script `zip.xml` überträgt die komprimierte Datei. Für dieses Beispiel muss das Ant-Script `zipfiles.xml` im Befehlspfad des Quellenagenten enthalten sein. Dies ist erforderlich, weil das Ant-Script `zipfiles.xml` das Ziel für die Komprimierung des Inhalts des Verzeichnisses auf dem Quellenagenten enthält. Weitere Informationen zur Festlegung des Befehlspfads eines Agenten finden Sie im Abschnitt commandPath MFT-Eigenschaft.

Zugehörige Konzepte

„Einführung in die Verwendung von Ant-Scripts mit MFT” auf Seite 1281
Durch die Verwendung von Ant-Scripts mit Managed File Transfer können Sie komplexe Dateiübertragungen über eine interpretierte Scriptsprache koordinieren.

Zugehörige Verweise

`fteAnt`: Ant-Tasks in MFT ausführen

MFT mit Benutzerexits anpassen

Sie können die Funktionen von Managed File Transfer mit Ihren eigenen Programmen anpassen. Diese werden Benutzerexitroutinen genannt.

Wichtig: Jeder Code in einem Benutzerexit wird von IBMnicht unterstützt und alle Probleme mit diesem Code müssen zunächst entweder von Ihrem Unternehmen oder vom Anbieter untersucht werden, der den Exit bereitgestellt hat.

Managed File Transfer bietet Punkte im Code, an denen Managed File Transfer die Steuerung auf ein von Ihnen geschriebenes Programm (eine Benutzerexitroutine) übertragen kann. Diese Punkte werden Exitpunkte genannt. Managed File Transfer kann die Steuerung anschließend wieder aufnehmen, sobald Ihr Programm seine Arbeit fertig gestellt hat. Sie müssen keine Benutzerexits verwenden, jedoch sind sie hilfreich, wenn Sie die Funktionen Ihres Managed File Transfer-Systems Ihren Anforderungen entsprechend erweitern und anpassen möchten.

Während der Verarbeitung der Dateiübertragung gibt es zwei Punkte, an denen Sie einen Benutzerexit am Quellensystem aufrufen können. Außerdem gibt es während Verarbeitung der Dateiübertragung zwei Punkte, an denen Sie einen Benutzerexit am Zielsystem aufrufen können. In der folgenden Tabelle sind

die einzelnen Benutzerexitpunkte aufgeführt sowie die Java-Schnittstellen, die installiert werden müssen, damit die Exitpunkte verwendet werde können.

Tabelle 187. Übersicht über die Exitpunkte der Quellen- und der Zielseite und die Java-Schnittstellen

Exitpunkt	**Zu implementierende Java**
Exitpunkte auf der Quellenseite:	
Vor dem Starten der gesamten Dateiübertragung	SourceTransferStartExit.java-Schnittstelle
Nach Abschluss der gesamten Dateiübertragung	SourceTransferEndExit.java-Schnittstelle
Exitpunkte auf der Zielseite:	
Vor dem Starten der gesamten Dateiübertragung	DestinationTransferStartExit.java-Schnittstelle
Nach Abschluss der gesamten Dateiübertragung	DestinationTransferEndExit.java-Schnittstelle

Die Benutzerexits werden in der folgenden Reihenfolge aufgerufen:

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

Von den Exits SourceTransferStartExit und DestinationTransferStartExit durchgeführte Änderungen werden als Eingabe an folgende Exits weitergegeben. Wenn beispielsweise der Exit SourceTransferStartExit die Übertragungsmetadaten ändert, werden die Änderungen in den eingegebenen Übertragungsmetadaten der anderen Exits reflektiert.

Benutzerexits und Programmaufrufe werden in der folgenden Reihenfolge aufgerufen:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Anmerkungen:

1. Die **DestinationTransferEndExits** wird nur ausgeführt, wenn die Übertragung erfolgreich oder teilweise erfolgreich abgeschlossen wird.
2. Die **postDestinationCall** wird nur ausgeführt, wenn die Übertragung erfolgreich oder teilweise erfolgreich abgeschlossen wird.
3. **SourceTransferEndExits** wird für erfolgreiche, teilweise erfolgreiche oder fehlgeschlagene Übertragungen ausgeführt.
4. **postSourceCall** wird nur in folgenden Fällen aufgerufen:
   - Die Übertragung wurde nicht abgebrochen.
   - Es gibt ein erfolgreiches oder teilweise erfolgreiches Ergebnis.
   - Alle 'postdestination'-Übertragungsprogramme wurden erfolgreich ausgeführt.

Eigenen Benutzerexit erstellen

Die Schnittstellen, mit denen Benutzerexits erstellt werden, sind in der Datei *MQ_INSTALL_DIRECTORY*`/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar` enthalten. Diese JAR-Datei muss bei der Erstellung des Exits in den Klassenpfad eingefügt werden. Um den Exit auszuführen, müssen Sie ihn

wie im folgenden Abschnitt beschrieben als JAR-Datei extrahieren und diese Datei in ein Verzeichnis einfügen.

Benutzerexit-Speicherpositionen

Sie können Ihre Benutzerexitroutinen an zwei möglichen Positionen speichern:

- Im Verzeichnis `exits`. Unter jedem Verzeichnis "agents" gibt es ein Verzeichnis "exits". Beispiel: `var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits`
- Sie können die Eigenschaft exitClassPath so einrichten, dass eine alternative Position angegeben wird. Wenn sich sowohl im Verzeichnis `exits` als auch in dem von exitClassPath festgelegten Klassenpfad Exitklassen befinden, erhalten die Klassen im Verzeichnis `exits` Priorität. Das bedeutet, wenn sich an beiden Positionen Klassen mit demselben Namen befinden, erhalten die Klassen im Verzeichnis `exits` Priorität.

Agenten für die Verwendung von Benutzerexits konfigurieren

Die Benutzerexits, die ein Agent aufruft, können über vier Agenteneigenschaften angegeben werden. Diese Agenteneigenschaften sind `sourceTransferStartExitClasses`, `sourceTransferEndExitClasses`, `destinationTransferStartExitClassesund` `destinationTransferEndExitClasses`. Informationen zur Verwendung dieser Eigenschaften finden Sie im Abschnitt MFTAgenteneigenschaften für Benutzerexits.

Benutzerexits auf Protokoll-Bridge-Agenten ausführen

Wenn der Quellenagent den Exit aufruft, übergibt er dem Exit eine Liste der Quellenelemente für die Übertragung. Bei normalen Agenten ist dies eine Liste mit vollständig qualifizierten Dateinamen. Da die Dateien lokal (oder über eine Mountfunktion zugänglich) sein sollten, kann der Exit auf die Liste zugreifen und sie verschlüsseln.

Für einen Protokollbridgeagenten haben die Einträge in der Liste jedoch folgendes Format:

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

Für jeden Eintrag in der Liste muss der Exit zuerst eine Verbindung zum Dateiserver herstellen (über das FTP-, FTPS- oder SFTP-Protokoll), die Datei herunterladen und sie lokal verschlüsseln und die verschlüsselte Datei dann wieder auf den Dateiserver hochladen.

Benutzerexits auf Connect:Direct-Bridgeagenten ausführen

Auf Connect:Direct-Bridgeagenten können keine Benutzerexits ausgeführt werden.

Zugehörige Konzepte

„Quellen- und Zielbenutzerexits in MFT” auf Seite 1287

Metadaten für MFT-Benutzerexits

Java-Schnittstellen für MFT-Benutzerexits

Zugehörige Verweise

„Remote-Debugging für MFT-Benutzerexits aktivieren” auf Seite 1292

Bei der Entwicklung Ihrer Benutzerexits können Sie Probleme im Code mithilfe eines Debuggers lokalisieren.

„Beispiel für Benutzerexits am MFT-Quellenübertragungsende” auf Seite 1293

„Musterbenutzerexit für Protokoll-Bridge-Berechtigungsnachweis” auf Seite 1294

Benutzerexits für MFT-Ressourcenüberwachungen

MFT-Agenteneigenschaften für Benutzerexits

Quellen- und Zielbenutzerexits in MFT

Verzeichnistrennzeichen

Für Verzeichnistrennzeichen in Quellendateispezifikationen wird immer der Schrägstrich (/) dargestellt, unabhängig davon, wie Sie Verzeichnisseparatoren im Befehl **fteCreateTransfer** oder im IBM MQ Explorer angegeben haben. Dies müssen Sie berücksichtigen, wenn Sie einen Exit schreiben. Wenn beispielsweise festgestellt werden soll, ob die Quellendatei `c:\a\b.txt` vorhanden ist, und Sie den Namen dieser Quellendatei im Befehl **fteCreateTransfer** oder in IBM MQ Explorer angegeben haben, wird dieser Dateiname als `c:/a/b.txt` gespeichert. Wenn Sie also nach der ursprünglichen Zeichenfolge `c:\a\b.txt` suchen, werden Sie diese Datei nicht finden.

Exitpunkte der Quellenseite

Vor dem Starten der gesamten Dateiübertragung

Dieser Exit wird vom Quellenagenten aufgerufen, wenn als nächstes in der Liste der anstehenden Übertragungen eine Übetragungsanforderung ansteht und die Übertragung gerade gestartet werden soll.

Ein Beispiel für die Verwendung dieses Exitpunkts ist das gestaffelte Senden von Dateien mithilfe eines externen Befehls an ein Verzeichnis, auf das der Agent Lese- und Schreibzugriff hat, oder das Umbenennen der Dateien auf dem Zielsystem.

Reichen Sie die folgenden Argumente an den Exit weiter:

- Name des Quellenagenten
- Name des Zielagenten
- Umgebungsmetadaten
- Übertragungsmetadaten
- Dateispezifikationen (einschließlich Dateimetadaten)

Die von diesem Exit zurückgegebenen Daten sind wie folgt:

- Aktualisierte Übertragungsmetadaten. Einträge können hinzugefügt, geändert und gelöscht werden.
- Aktualisierte Liste von Dateispezifikationen mit Quellendateinamens- und Zieldateinamenspaaren. Einträge können hinzugefügt, geändert und gelöscht werden.
- Anzeige, die angibt, ob die Übertragung fortgesetzt wird
- Die in das Übertragungsprotokoll einzufügende Zeichenfolge.

Implementieren Sie die Schnittstelle SourceTransferStartExit.java, um Benutzerexitcode an diesem Exitpunkt aufzurufen.

Nach Abschluss der gesamten Dateiübertragung

Dieser Exit wird vom Quellenagenten nach Abschluss der gesamten Dateiübertragung aufgerufen.

Ein Beispiel für die Verwendung dieses Exitpunkts ist das Ausführen einiger Abschlussaufgaben wie z. B. dem Senden einer E-Mail oder einer IBM MQ-Nachricht, um die Übertragung als abgeschlossen anzuzeigen.

Reichen Sie die folgenden Argumente an den Exit weiter:

- Übertragungsexitergebnis
- Name des Quellenagenten
- Name des Zielagenten
- Umgebungsmetadaten
- Übertragungsmetadaten
- Dateiergebnisse

Die von diesem Exit zurückgegebenen Daten sind wie folgt:

- Die in das Übertragungsprotokoll einzufügende aktualisierte Zeichenfolge.

Implementieren Sie die Schnittstelle SourceTransferEndExit.java, um Benutzerexitcode an diesem Exitpunkt aufzurufen.

Exitpunkte der Zielseite

Vor dem Starten der gesamten Dateiübertragung

Ein Beispiel für diesen Exitpunkt ist das Validieren der Berechtigungen am Ziel.

Reichen Sie die folgenden Argumente an den Exit weiter:

- Name des Quellenagenten
- Name des Zielagenten
- Umgebungsmetadaten
- Übertragungsmetadaten
- Dateispezifikationen

Die von diesem Exit zurückgegebenen Daten sind wie folgt:

- Aktualisierter Satz an Zieldateinamen. Einträge können geändert, aber nicht hinzugefügt und gelöscht werden.
- Anzeige, die angibt, ob die Übertragung fortgesetzt wird
- Die in das Übertragungsprotokoll einzufügende Zeichenfolge.

Implementieren Sie die Schnittstelle DestinationTransferStartExit.java, um Benutzerexitcode an diesem Exitpunkt aufzurufen.

Nach Abschluss der gesamten Dateiübertragung

Ein Beispiel der Verwendung dieses Benutzerexits ist das Starten eines Stapelprozesses, der die übertragenen Dateien verwendet, oder das Senden einer E-Mail beim Fehlschlagen der Übertragung.

Reichen Sie die folgenden Argumente an den Exit weiter:

- Übertragungsexitergebnis
- Name des Quellenagenten
- Name des Zielagenten
- Umgebungsmetadaten
- Übertragungsmetadaten
- Dateiergebnisse

Die von diesem Exit zurückgegebenen Daten sind wie folgt:

- Die in das Übertragungsprotokoll einzufügende aktualisierte Zeichenfolge.

Implementieren Sie die Schnittstelle DestinationTransferEndExit.java, um Benutzerexitcode an diesem Exitpunkt aufzurufen.

Zugehörige Konzepte

Java-Schnittstellen für MFT-Benutzerexits

Zugehörige Verweise

„Remote-Debugging für MFT-Benutzerexits aktivieren” auf Seite 1292

Bei der Entwicklung Ihrer Benutzerexits können Sie Probleme im Code mithilfe eines Debuggers lokalisieren.

„Beispiel für Benutzerexits am MFT-Quellenübertragungsende” auf Seite 1293

Benutzerexits für MFT-Ressourcenüberwachungen

Ein-/Ausgabebenutzerexits für MFT-Übertragungen verwenden

Mit den Ein-/Ausgabebenutzerexits für die Managed File Transfer-Übertragung können Sie angepassten Code für die zugrunde liegenden Ein-/Ausgabeprozesse des Dateisystems bei Managed File Transfer-Übertragungen konfigurieren.

Für MFT-Übertragungen wählt ein Agent in der Regel einen der integrierten Ein-/Ausgabeprovider für die Interaktion mit den Dateisystemen während der Übertragung aus. Integrierte Ein-/Ausgabeprovider unterstützen die folgenden Dateisystemtypen:

- Reguläre UNIX- und Windows-Dateisysteme
- z/OS Sequenzielle und partitionierte z/OS-Datasets (nur unter z/OS)
- IBM i Native IBM i-Sicherungsdateien (nur unter IBM i)
- IBM MQ-Warteschlangen
- Ferne FTP- und SFTP-Protokollserver (nur für Protokollbridgeagenten)
- Ferne Connect:Direct-Knoten (nur für Connect:Direct-Bridgeagenten)

Für nicht unterstützte Dateisysteme oder bei einer erforderlichen Anpassung des Ein-/Ausgabeverhaltens können Sie einen Ein-/Ausgabebenutzerexit für die Übertragung schreiben.

Ein-/Ausgabebenutzerexits für Übertragungen nutzen die vorhandene Infrastruktur für Benutzerexits. Diese Ein-/Ausgabebenutzerexits für Übertragungen unterscheiden sich jedoch von anderen Benutzerexits, da während der Übertragung der einzelnen Dateien mehrmals auf ihre Funktionalität zugegriffen wird.

Geben Sie mit der Agenteneigenschaft 'IOExitClasses' (in der Datei `agent.properties`) an, welche Exitklassen der Ein-/Ausgabe geladen werden sollen. Trennen Sie die einzelnen Exitklassen durch Kommas. Beispiel:

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

Die Java-Schnittstellen der Ein-/Ausgabebenutzerexits für Übertragungen lauten wie folgt:

IOExit
Der Haupteinstiegspunkt, mit dem bestimmt wird, ob der Ein-/Ausgabeexit verwendet wird. Diese Instanz ist für die Erstellung von IOExitPath-Instanzen verantwortlich.

Sie müssen für die Agenteneigenschaft nur die Ein-/Ausgabeexitschnittstelle 'IOExit' angeben.

IOExitPath
Stellt eine abstrakte Schnittstelle dar, beispielsweise einen Datencontainer oder ein Platzhalterzeichen, das eine Gruppe von Datencontainern darstellt. Sie können keine Klasseninstanz erstellen, die diese Schnittstelle implementiert. Die Schnittstelle ermöglicht eine Prüfung des Pfades und die Auflistung abgeleiteter Pfade. Die Schnittstellen 'IOExitResourcePath' und 'IOExitWildcardPath' sind eine Erweiterung von 'IOExitPath'.

IOExitChannel
Ermöglicht das Auslesen von Daten aus einer IOExitPath-Ressource und das Schreiben von Daten in diese Ressource.

IOExitRecordChannel
Erweitert die Schnittstelle 'IOExitChannel' für satzorientierte IOExitPath-Ressourcen, was das Auslesen von Daten aus einer IOExitPath-Ressource und das Schreiben von Daten in diese Ressource in einem Vielfachen der Datensätze ermöglicht.

IOExitLock
Stellt eine Sperre für eine IOExitPath-Ressource bei einem gemeinsamen oder exklusiven Zugriff dar.

z/OS **IOExitRecordResourcePath**
Erweitert die Schnittstelle 'IOExitResourcePath' zur Darstellung eines Datencontainers für eine satzorientierte Datei (beispielsweise ein z/OS-Dataset). Mit der Schnittstelle können Daten gesucht und IOExitRecordChannel-Instanzen für Lese- und Schreiboperationen erstellt werden.

IOExitResourcePath
Erweitert die Schnittstelle 'IOExitPath' zur Darstellung eines Datencontainers (beispielsweise eine Datei oder ein Verzeichnis). Mit der Schnittstelle können Daten gesucht werden. Wenn die Schnittstelle ein Verzeichnis darstellt, können Sie mit der Methode 'listPaths' eine Liste der Pfade zurückgeben.

IOExitWildcardPath
Erweitert die Schnittstelle 'IOExitPath' zur Darstellung eines Pfads mit einem Platzhalterzeichen. Mit dieser Schnittstelle können Sie auf mehrere IOExitResourcePath-Instanzen verweisen.

IOExitProperties
Gibt Eigenschaften an, die bestimmen, wie Managed File Transfer bei bestimmten Aspekten der Ein-/Ausgabe den IOExitPath handhabt. Hiermit wird beispielsweise festgelegt, ob temporäre Dateien verwendet werden oder ob eine Ressource bei einem Neustart der Übertragung nochmals von vorne gelesen werden soll.

Zugehörige Konzepte

„MFT mit Benutzerexits anpassen” auf Seite 1285
Sie können die Funktionen von Managed File Transfer mit Ihren eigenen Programmen anpassen. Diese werden Benutzerexitroutinen genannt.

Zugehörige Verweise

IOExit.java (Schnittstelle)
IOExitChannel.java (Schnittstelle)
IOExitLock.java (Schnittstelle)
IOExitPath.java (Schnittstelle)
IOExitProperties.java (Schnittstelle)
IOExitRecordChannel.java (Schnittstelle)
z/OS IOExitRecordResourcePath.java (Schnittstelle)
IOExitResourcePath.java (Schnittstelle)
IOExitWildcardPath.java (Schnittstelle)
Datei MFT `agent.properties`

IBM i Beispiele für MFT on IBM i-Benutzerexits

In der Installation von Managed File Transfer werden einige für IBM i spezifische Benutzerexits bereitgestellt. Die Beispiele befinden sich in den Verzeichnissen *MQMFT_install_dir*`/samples/ioexit-IBMi` und *MQMFT_install_dir*`/samples/userexit-IBMi`.

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit
Der Beispiel-Benutzerexit 'com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit' überträgt Dateien in das QDLS-Dateisystem unter IBM i. Nach der Installation des Exits wird bei Übertragungen in Dateien, die mit `/QDLS` beginnen, automatisch dieser Exit verwendet.

Führen Sie zum Installieren dieses Exits die folgenden Schritte aus:

1. Kopieren Sie die Datei `com.ibm.wmqfte.samples.ibmi.ioexits.jar` aus dem Verzeichnis *WMQFTE_install_dir*`/samples/ioexit-IBMi` in das Verzeichnis `exits` des Agenten.
2. Fügen Sie der Eigenschaft 'IOExitClasses' den Eintrag 'com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit' hinzu.
3. Starten Sie den Agenten erneut.

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit

Der Beispiel-Benutzerexit 'com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit' verhält sich wie ein MFT-Dateimonitor und überträgt automatisch physische Teildateien aus einer IBM i-Bibliothek.

Geben Sie zum Ausführen dieses Exits einen Wert für das Metadatenfeld "library.qsys.monitor" an (z. B. mithilfe des Parameters **-md**). Gültige Werte für diesen Parameter sind IFS-kompatible (Integrated File System) Pfadangaben zu einer Teildatei, die auch Platzhalter für Dateien und Teildateien enthalten können. Beispiele: /QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR, /QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR, /QSYS.LIB/FOO.LIB/*.FILE/*.MBR.

Dieser Beispiel-Exit enthält darüber hinaus das optionale Metadatenfeld "naming.scheme.qsys.monitor", mit dem Sie das Benennungsschema angeben können, das während der Übertragung verwendet werden soll. Da dieses Feld standardmäßig auf 'unix' eingestellt ist, erhält die Zieldatei den Namen F00.MBR. Sie können auch den Wert "ibmi" angeben, um die IBM i FTP FILE.MEMBER , z. B. /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR wird als BAR.BAZ.

Führen Sie zum Installieren dieses Exits die folgenden Schritte aus:

1. Kopieren Sie die Datei `com.ibm.wmqfte.samples.ibmi.userexits.jar` aus dem Verzeichnis *WMQFTE_install_dir*`/samples/userexit-IBMi` in das Verzeichnis `exits` des Agenten.
2. Fügen Sie der Eigenschaft 'sourceTransferStartExitClasses' in der Datei `agent.properties` den Eintrag 'com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit' hinzu.
3. Starten Sie den Agenten erneut.

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

Mit dem Beispiel-Benutzerexit 'com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit' wird ein leeres Dateiobjekt gelöscht, wenn dessen Quellenteildatei als Teil einer Dateiübertragung gelöscht wird. Da IBM i-Dateiobjekte potenziell viele Teildateien enthalten können, werden Dateiobjekte von MFT wie Verzeichnisse behandelt. Deshalb können Sie mit MFT keine Verschiebeoperation für ein Dateiobjekt ausführen, denn Verschiebeoperationen werden nur auf Teildateiebene unterstützt. Daher entsteht beim Ausführen einer Verschiebeoperation für eine Teildatei eine anschließend leere Datei. Mit diesem Beispielexit können Sie diese leeren Dateien als Teil einer Übertragungsanforderung löschen.

Wenn Sie für das Metadatenfeld "empty.file.delete" den Wert "true" eingeben und ein Objekt des Typs "FTEFileMember" übertragen, löscht der Beispielexit die übergeordnete Datei, sofern die Datei leer ist.

Führen Sie zum Installieren dieses Exits die folgenden Schritte aus:

1. Kopieren Sie die Datei 'com.ibm.wmqfte.samples.ibmi.userexits.jar' aus dem Verzeichnis *WMQFTE_install_dir*`/samples/userexit-IBMi` in das Verzeichnis `exits` des Agenten.
2. Fügen Sie der Eigenschaft 'sourceTransferStartExitClasses' in der Datei `agent.properties` den Eintrag 'ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit' hinzu.
3. Starten Sie den Agenten erneut.

Zugehörige Verweise

„Ein-/Ausgabebenutzerexits für MFT-Übertragungen verwenden” auf Seite 1290
Mit den Ein-/Ausgabebenutzerexits für die Managed File Transfer-Übertragung können Sie angepassten Code für die zugrunde liegenden Ein-/Ausgabeprozesse des Dateisystems bei Managed File Transfer-Übertragungen konfigurieren.

MFT-Agenteneigenschaften für Benutzerexits

Remote-Debugging für MFT-Benutzerexits aktivieren

Bei der Entwicklung Ihrer Benutzerexits können Sie Probleme im Code mithilfe eines Debuggers lokalisieren.

Da Exits in der JVM (Java Virtual Machine) ausgeführt werden, in der der Agent aktiv ist, können Sie nicht die direkte Debugging-Unterstützung verwenden, die standardmäßig in einer integrierten Entwick-

lungsumgebung bereitgestellt ist. Sie können jedoch die Remote-Debug-Funktion der JVM aktivieren und anschließend eine Verbindung zu einem geeigneten Remote Debugger herstellen.

Um das remote Debugging zu aktivieren, verwenden Sie die JVM-Standardparameter **-Xdebug** und **-Xrunjdwp**. Diese Eigenschaften werden über die Umgebungsvariable **BFG_JVM_PROPERTIES** an die JVM übergeben, die den Agenten ausführt. Unter AIX and Linux beispielsweise wird mit den folgenden Befehlen der Agent gestartet und die JVM angewiesen, den TCP-Port 8765 auf Debugger-Verbindungen zu überwachen.

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

Der Agent wird erst gestartet, wenn die Verbindung zum Debugger hergestellt ist. Verwenden Sie den Befehl **set** unter Windows anstelle des Befehls **export** .

Sie können zwischen Debugger und JVM auch andere Kommunikationsverfahren verwenden. So kann die JVM zum Beispiel die Verbindung mit dem Debugger eröffnen (statt umgekehrt) oder Sie können statt TCP gemeinsam genutzten Speicher verwenden. Weitere Informationen finden Sie in der Dokumentation für Java Platform Debugger Architecture.

Sie müssen den Parameter **-F** (Vordergrund) verwenden, wenn Sie den Agenten im fernen Debugmodus starten.

Eclipse-Debugger verwenden

Die folgenden Schritte gelten für die Remote-Debugging-Funktion der Entwicklungsumgebung Eclipse. Sie können aber auch andere JPDA-kompatible Remote-Debugger verwenden.

1. Klicken Sie je nach Eclipse-Version auf **Run** > **Open Debug Dialog** (Ausführen, Debug-Dialog öffnen) oder auf **Run** > **Debug Configurations** (Ausführen, Debug-Konfigurationen) oder auf **Run** > **Debug Dialog** (Ausführen, Debug-Dialog).
2. Klicken Sie in der Liste der Konfigurationstypen doppelt auf **Remote Java Application** (Ferne Java-Anwendung), um eine Debugkonfiguration zu erstellen.
3. Füllen Sie die Konfigurationsfelder aus und speichern Sie die Debug-Konfiguration. Wenn Sie die JVM des Agenten bereits im Debug-Modus gestartet haben, können Sie jetzt eine Verbindung zur JVM herstellen.

Beispiel für Benutzerexits am MFT-Quellenübertragungsende

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */
```

```
import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
                destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

Musterbenutzerexit für Protokoll-Bridge-Berechtigungsnachweis

Informationen zur Verwendung dieses Beispielbenutzerexits finden Sie im Abschnitt Berechtigungsnachweise für einen Dateiserver mithilfe von Exitklassen zuordnen.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
```

```
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been spe
cified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();

             // Open and load the properties from the properties file
             final File propertiesFile = new File (propertiesFilePath);
             FileInputStream inputStream = null;
             try {
                 // Create a file input stream to the file
                 inputStream = new FileInputStream(propertiesFile);

                 // Load the properties from the file
                 mappingProperties.load(inputStream);
             }
             catch (FileNotFoundException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Unable to find the mqUserId mapping properties file: " + proper
tiesFilePath);
                 initialisationResult = false;
             }
             catch (IOException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                 initialisationResult = false;
             }
             finally {
                 // Close the inputStream
                 if (inputStream != null) {
                     try {
                         inputStream.close();
                     }
                     catch (IOException ex) {
                         System.err.println("Error initializing SampleCredentialExit.");
```

```
                         System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of mqUserId to server credentials from the properties
                final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                 while ( propertyNames.hasMoreElements()) {
                    final Object name = propertyNames.nextElement();
                    if (name instanceof String ) {
                        final String mqUserId = ((String)name).trim();
                         // Get the value and split into serverUserId and serverPassword
                        final String value = mappingProperties.getProperty(mqUserId);
                        final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                         String serverUserId = "";
                         String serverPassword = "";
                         if (valueTokenizer.hasMoreTokens()) {
                             serverUserId = valueTokenizer.nextToken().trim();
                         }
                         if (valueTokenizer.hasMoreTokens()) {
                             serverPassword = valueTokenizer.nextToken().trim();
                         }
                         // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new CredentialPass
word(serverPassword));
                         // Insert the credentials into the map
                         credentialsMap.put(mqUserId, credentials);
                    }
                }

           }
        }

        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED, creden
tials);
        }
        return result;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

Benutzerexit für Eigenschaften der Protokollbrückeneigenschaften

Informationen zur Verwendung dieses Beispielbenutzerexits finden Sie im Abschnitt ProtocolBridgePropertiesExit2: Protokolldateiservereigenschaften durchsuchen

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
```

```
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
```

```
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(ja□
va.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when avai□
lable.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
        if (info != null) {
            result = new Properties();
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1) result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VA□
LUE, ""+info.getPort());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
            System.err.println("The location of the protocol server properties file has not been speci□
fied in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " + propertiesFi□
lePath);
                initialisationResult = false;
            } catch (final IOException ex) {
```

```
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " + pro
pertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
                    servers.put(serverName, info);
                }
            }
        }

        return initialisationResult;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
        return envLocationPath;
    }

}
```

MFT durch Einreihen von Nachrichten in die Befehlswarteschlange des Agenten steuern

Sie können eine Anwendung erstellen, mit deren Hilfe Managed File Transfer durch Einreihen von Nachrichten in die Befehlswarteschlangen von Agenten gesteuert werden kann.

Sie können eine Nachricht in die Befehlswarteschlange eines Agenten einreihen und auf diese Weise eine der folgenden Aktionen des Agenten anfordern:

- Erstellen einer Dateiübertragung
- Erstellen einer geplanten Dateiübertragung
- Abbrechen einer Dateiübertragung
- Abbrechen einer geplanten Dateiübertragung
- Aufrufen eines Befehls
- Erstellen eines Überwachungsprogramms
- Löschen eines Überwachungsprogramms

- Rückgabe eines Pignsignals, um anzugeben, dass der Agent aktiv ist

Damit eine dieser Agentenaktionen angefordert werden kann, muss die Nachricht in einem XML-Format vorliegen, welches einem der folgenden Schemas entspricht:

`FileTransfer.xsd`
Mit Nachrichten dieses Formats können Dateiübertragungen bzw. geplante Dateiübertragungen erstellt oder abgebrochen und Befehle aufgerufen werden. Weitere Informationen finden Sie im Abschnitt Format für Dateiübertragungsanforderungsnachricht.

`Monitor.xsd`
Mit Nachrichten in diesem Format kann eine Ressourcenüberwachung erstellt oder gelöscht werden. Weitere Informationen finden Sie im Abschnitt Formate für MFT-Überwachungsanforderungsnachrichten.

`PingAgent.xsd`
Mit Nachrichten in diesem Format kann ein Pingsignal an einen Agenten abgesetzt und auf diese Weise geprüft werden, ob der Agent aktiv ist. Weitere Informationen finden Sie im Abschnitt Format für MFT-Agentenanforderungsnachricht.

Der Agent gibt eine Antwort auf die Anforderungsnachrichten zurück. Die Antwortnachricht wird in eine in der Anforderungsnachricht festgelegte Warteschlange für Antwortnachrichten eingereiht. Die Antwortnachricht weist ein durch das folgende Schema definiertes XML-Format auf:

`Reply.xsd`
Weitere Informationen finden Sie im Abschnitt Format für MFT-Agentenantwortnachricht.

Anwendungen für MQ Telemetry entwickeln

Telemetrieanwendungen integrieren Sensor- und Steuerungsgeräte mit anderen Informationsquellen, die im Internet oder im Unternehmen zur Verfügung stehen.

Es können Anwendungen für MQ Telemetry mithilfe von Designmustern, Arbeitsbeispielen, Beispielprogrammen, Programmierungskonzepten und Referenzinformationen entwickelt werden.

Zugehörige Konzepte

MQ Telemetry

Telemetrieanwendungsfälle

Zugehörige Tasks

Installieren von MQ Telemetry

MQ Telemetry verwalten

Fehlerbehebung bei Problemen mit MQ Telemetry

Zugehörige Verweise

MQ Telemetry Referenz

IBM MQ Telemetry Transport-Beispielprogramme

Es werden Beispielscripts bereitgestellt, die mit einem Beispiel für eine Clientanwendung von IBM MQ Telemetry Transport der Version 3 3 arbeitet (`mqttv3app.jar`). Für IBM MQ 8.0.0 und höher ist die Beispielclientanwendung nicht mehr in MQ Telemetry enthalten. Sie war Teil von IBM Messaging Telemetry Clients SupportPac (nicht mehr verfügbar). Ähnliche Beispielanwendungen sind bei Eclipse Paho und MQTT.org weiterhin kostenlos erhältlich.

Die aktuellsten Informationen und Downloads finden Sie in den folgenden Ressourcen:

- Das Eclipse Paho-Projekt und MQTT.org verfügen über kostenlose Downloads der neuesten Telemetrieclients und Beispiele für eine Reihe von Programmiersprachen. Nutzen Sie diese Sites, um Beispielprogramme zum Veröffentlichen und Subskribieren von IBM MQ Telemetry Transport zu entwickeln und Sicherheitsfunktionen hinzuzufügen.
- IBM Messaging Telemetry Clients SupportPac kann nicht mehr heruntergeladen werden. Wenn Sie eine zuvor heruntergeladene Kopie haben, hat diese die folgenden Inhalte:

– Die MA9B-Version von IBM Messaging Telemetry Clients SupportPac enthielt eine kompilierte Beispielanwendung (`mqttv3app.jar`) und eine zugehörige Clientbibliothek (`mqttv3.jar`). Sie wurden in den folgenden Verzeichnissen bereitgestellt:
  - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
  - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
– In der MA9C-Version dieses SupportPac wurden das Verzeichnis `/SDK/` und die Inhalte entfernt:
  - Nur die Quelle für die Beispielanwendung (`mqttv3app.jar`) wurde bereitgestellt. Sie befand sich in diesem Verzeichnis:

    ```
    ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
    ```

  - Die kompilierte Clientbibliothek wurde weiterhin bereitgestellt. Sie befand sich in diesem Verzeichnis:

    ```
    ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
    ```

Wenn Sie noch über eine Kopie von IBM Messaging Telemetry Clients SupportPac (nicht mehr verfügbar) verfügen, finden Sie Informationen zur Installation und Ausführung der Beispielanwendung unter Installation von MQ Telemetry über die Befehlszeile bestätigen.

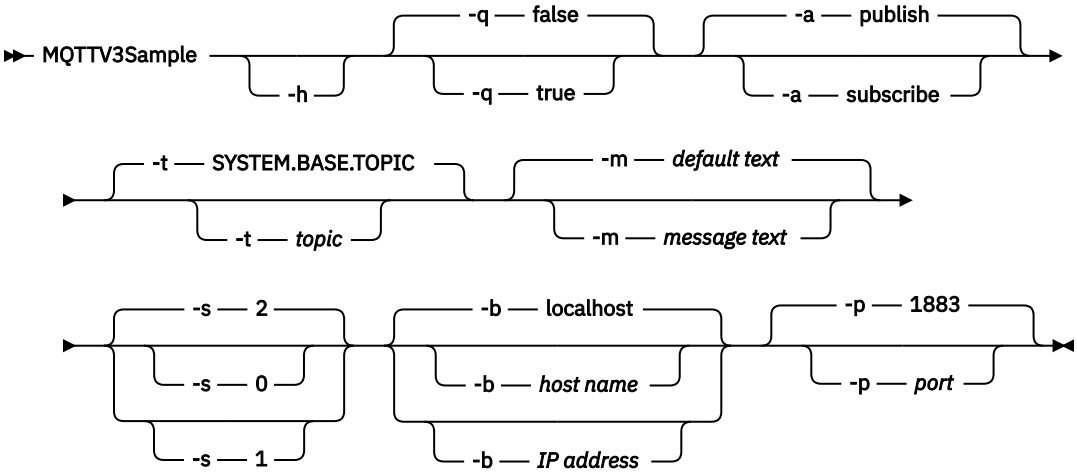
Programm MQTTV3Sample

Referenzinformationen zur Beispielsyntax und zu den Parametern für das Programm `MQTTV3Sample`.

Zweck

Mit dem Programm `MQTTV3Sample` kann eine Nachricht für ein Thema veröffentlicht und subskribiert werden. Informationen zum Abrufen dieses Beispielprogramms finden Sie unter „IBM MQ Telemetry Transport-Beispielprogramme” auf Seite 1300.

MQTTV3Sample syntax



Parameter

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

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

-a
Angabe von 'publish' oder 'subscribe' statt Übernahme der Veröffentlichung als Standardaktion.

-t
Veröffentlichen oder subskribieren eines Themas, statt das Standardthema zu veröffentlichen oder zu subskribieren.

-m
Veröffentlichen von Nachrichtentext anstelle des Sendens des Standardveröffentlichungstexts "Hello from an MQTT v3 application".

-s
Festlegen der Servicequalität (QoS) statt Verwendung der standardmäßigen Servicequalität QoS=2.

-b
Verbindung mit dem hier angegebenen Hostnamen oder der hier angegebenen IP-Adresse statt mit dem Standardhost 'localhost' herstellen.

-p
Verwendung des hier angegebenen Ports anstelle des Standardports 1883.

Programm 'MQTTV3Sample' ausführen

Subskribieren Sie ein Thema unter Windows mit folgendem Befehl:

```
run MQTTV3Sample -a subscribe
```

Veröffentlichen Sie eine Nachricht unter Windows mit folgendem Befehl:

```
run MQTTV3Sample
```

Konzepte zur Programmierung von MQTT-Clients

Die in diesem Abschnitt beschriebenen Konzepte helfen Ihnen, die Clientbibliotheken für das MQTT protocol besser zu verstehen. Die Konzepte ergänzen die API-Dokumentation, die zu den Clientbibliotheken gehört.

Die aktuellsten Informationen und Downloads finden Sie in den folgenden Ressourcen:

- Das Eclipse Paho-Projekt und MQTT.org verfügen über kostenlose Downloads der neuesten Telemetrieclients und Beispiele für eine Reihe von Programmiersprachen. Nutzen Sie diese Sites, um Beispielprogramme zum Veröffentlichen und Subskribieren von IBM MQ Telemetry Transport zu entwickeln und Sicherheitsfunktionen hinzuzufügen.
- IBM Messaging Telemetry Clients SupportPac kann nicht mehr heruntergeladen werden. Wenn Sie eine zuvor heruntergeladene Kopie haben, hat diese die folgenden Inhalte:
  - Die MA9B-Version von IBM Messaging Telemetry Clients SupportPac enthielt eine kompilierte Beispielanwendung (`mqttv3app.jar`) und eine zugehörige Clientbibliothek (`mqttv3.jar`). Sie wurden in den folgenden Verzeichnissen bereitgestellt:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - In der MA9C-Version dieses SupportPac wurden das Verzeichnis `/SDK/` und die Inhalte entfernt:
    - Nur die Quelle für die Beispielanwendung (`mqttv3app.jar`) wurde bereitgestellt. Sie befand sich in diesem Verzeichnis:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - Die kompilierte Clientbibliothek wurde weiterhin bereitgestellt. Sie befand sich in diesem Verzeichnis:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Zum Entwickeln und Ausführen eines MQTT-Clients müssen Sie diese Ressourcen auf die Clienteinheit kopieren oder dort installieren. Das Installieren einer separaten Clientlaufzeit ist nicht erforderlich.

Die Lizenzbedingungen für Clients sind dem Server zugeordnet, mit dem die Clients verbunden werden.

Bei den MQTT-Clientbibliotheken handelt es sich um Referenzimplementierungen des MQTT protocols. Sie können Ihre eigenen Clients in unterschiedlichen Sprachen implementieren, die für unterschiedliche Einheitenplattformen geeignet sind. Weitere Informationen finden Sie im Abschnitt IBM MQ Telemetry Transport-Format und -Protokoll.

In der API-Dokumentation wird kein bestimmter MQTT-Server vorausgesetzt, mit dem der Client verbunden ist. Das Verhalten des Clients kann bei der Verbindung zu anderen Servern möglicherweise abweichen. In den folgenden Beschreibungen wird das Verhalten des Clients dargestellt, wenn er mit dem IBM MQ Telemetry Service verbunden ist.

Callbacks und Synchronisierung in MQTT-Clientanwendungen

Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Callbacks

Anmerkung: Die neuesten Änderungen an `MqttCallback`finden Sie auf der Website Eclipse Paho . `MqttCallback` ist beispielsweise als Schnittstelle in der Paho-Version des Clients definiert; asynchrone Methoden werden von der Paho-Klasse `MqttAsyncClient` bereitgestellt.

Die `MqttCallback`-Schnittstelle verfügt über drei Callback-Methoden:

`connectionLost(java.lang.Throwable cause)`

`connectionLost` wird aufgerufen, wenn ein Übertragungsfehler zum Löschen der Verbindung führt. Diese Methode wird auch aufgerufen, wenn der Server die Verbindung als Ergebnis eines Fehlers auf dem Server löscht, nachdem die Verbindung hergestellt wurde. Serverfehler werden im Fehlerprotokoll des Warteschlangenmanagers protokolliert. Der Server löscht die Verbindung zum Client und der Client ruft `MqttCallback.connectionLost` auf.

Bei den einzigen fernen Fehlern, die als Ausnahmen im gleichen Thread wie die Clientanwendung ausgelöst werden, handelt es sich um Ausnahmen aus `MqttClient.connect`. Fehler, die vom Server nach der Verbindungsherstellung ermittelt werden, werden der Callback-Methode `MqttCallback.connectionLost` als `throwables` gemeldet.

Typische Serverfehler, die zu `connectionLost` führen, sind Berechtigungsfehler. Diese treten beispielsweise auf, wenn der Telemetrieserver ein Thema auf Anweisung eines Clients veröffentlichen möchte, der nicht zur Veröffentlichung dieses Themas berechtigt ist. Alles, was dazu führt, dass der Bedingungscode `MQCC_FAIL` an den Telemetrieserver zurückgegeben wird, kann zur Folge haben, dass die Verbindung unterbrochen wird.

`deliveryComplete(IMqttDeliveryToken token)`

`deliveryComplete` wird vom MQTT-Client aufgerufen, um ein Zustellungstoken wieder an die Clientanwendung zu übergeben; siehe „Zustellungstoken” auf Seite 1310. Mithilfe des Zustellungstokens kann der Callback mit der Methode `token.getMessage` auf die veröffentlichte Nachricht zugreifen.

Wenn der Anwendungscallback die Steuerung an den MQTT -Client zurückgibt, nachdem er von der Methode `deliveryComplete` aufgerufen wurde, ist die Zustellung abgeschlossen. Bis zum Abschluss der Übergabe werden Nachrichten mit der Servicequalität QoS von 1 oder 2 von der Persistenzklasse beibehalten.

Der Anruf von `deliveryComplete` stellt einen Synchronisationspunkt zwischen der Anwendung und der Persistenzklasse dar. Die Methode `deliveryComplete` wird für dieselbe Nachricht nie zweimal aufgerufen.

Wenn der Anwendungscallback die Steuerung nach seinem Aufruf durch die Methode `deliveryComplete` an den MQTT-Client zurückgibt, ruft der Client `MqttClientPersistence.remove` für Nachrichten mit der Servicequalität (QoS) 1 oder 2 auf. `MqttClientPersistence.remove` löscht die lokal gespeicherte Kopie der veröffentlichten Nachricht.

Aus Sicht der Transaktionsverarbeitung ist der Aufruf von `deliveryComplete` eine einphasige Transaktion, mit der die Übergabe festgeschrieben wird. Falls während des Callbacks ein Verarbeitungsfehler auftritt, wird `MqttClientPersistence.remove` beim Neustart des Clients erneut aufgerufen, um die lokale Kopie der veröffentlichten Nachricht zu löschen. Der Callback wird nicht erneut aufgerufen. Wenn Sie den Callback zum Speichern eines Protokolls der zugestellten Nachrichten verwenden, können Sie das Protokoll nicht mit dem MQTT-Client synchronisieren. Um ein Protokoll auf zuverlässige Weise zu speichern, müssen Sie das Protokoll in der Klasse `MqttClientPersistence` aktualisieren.

Das Zustellungstoken und die Nachricht werden vom Hauptanwendungsthread und vom MQTT-Client referenziert. Der MQTT-Client nimmt die Referenz zum `MqttMessage`-Objekt zurück, sobald die Übergabe abgeschlossen ist, und das Zustellungstokenobjekt, wenn der Client die Verbindung trennt. Das `MqttMessage`-Objekt kann fehlerhafte Daten, die nach Abschluss der Übergabe erfasst werden, enthalten, wenn die Clientanwendung die Referenz löscht. Das Zustellungstoken kann fehlerhafte Daten enthalten, die nach der Unterbrechung der Sitzung erfasst werden.

Sie können die Attribute `IMqttDeliveryToken` und `MqttMessage` abrufen, nachdem eine Nachricht veröffentlicht wurde. Wenn Sie versuchen, beliebige `MqttMessage`-Attribute nach dem Veröffentlichen einer Nachricht festzulegen, führt dies zu einem nicht definierten Ergebnis.

Der MQTT-Client fährt mit der Verarbeitung von Empfangsbestätigungen fort, wenn der Client die Verbindung zur vorherigen Sitzung mit der gleichen `Client-ID` wiederherstellt; siehe „Sitzungen bereinigen" auf Seite 1307. Die MQTT-Clientanwendung muss `MqttClient.CleanSession` für die vorherige Sitzung auf `false` und in einer neuen Sitzung ebenfalls auf `false` setzen. Der MQTT-Client erstellt neue Zustellungstokens und Nachrichtenobjekte in der neuen Sitzung für anstehende Übergaben. Er stellt die Objekte mithilfe der Klasse `MqttClientPersistence` wieder her. Falls der Anwendungsclient noch Referenzen auf die alten Zustellungstokens und Nachrichten enthält, löschen Sie diese Referenzen. Der Anwendungscallback wird in der neuen Sitzung für alle Übergaben aufgerufen, die in der früheren Sitzung eingeleitet wurden und in dieser Sitzung abgeschlossen werden.

Der Anwendungscallback wird nach Herstellung der Anwendungsclientverbindung aufgerufen, wenn eine anstehende Übergabe abgeschlossen ist. Bevor der Anwendungsclient die Verbindung herstellt, kann er anstehende Übergaben mit der Methode `MqttClient.getPendingDeliveryTokens` abrufen.

Beachten Sie, dass das Nachrichtenobjekt, das veröffentlicht wird, und dessen Bytefeldgruppe mit Nutzdaten ursprünglich von der Clientanwendung erstellt wurden. Der MQTT-Client verweist auf diese Objekte. Das Nachrichtenobjekt, das vom Zustellungstoken in der Methode `token.getMessage` zurückgegeben wird, ist nicht notwendigerweise dasselbe Nachrichtenobjekt, das vom Client erstellt wurde. Wenn eine neue MQTT-Clientinstanz das Zustellungstoken erneut erstellt, erstellt die Klasse `MqttClientPersistence` das Objekt `MqttMessage` erneut. Aus Konsistenzgründen gibt `token.getMessage` den Wert `null` zurück, wenn `token.isCompleted` auf `true` gesetzt ist, unabhängig davon, ob das Nachrichtenobjekt vom Anwendungsclient oder von der Klasse `MqttClientPersistence` erstellt wurde.

`messageArrived(String topic, MqttMessage message)`

`messageArrived` wird aufgerufen, wenn eine Veröffentlichung für den Client eintrifft, die einem Subskriptionsthema entspricht. `topic` gibt das Veröffentlichungsthema an, nicht den Subskriptionsfilter. Die beiden können unterschiedlich sein, wenn der Filter Platzhalterzeichen enthält.

Wenn das Thema mit mehreren vom Client erstellten Subskriptionen übereinstimmt, empfängt der Client mehrere Kopien der Veröffentlichung. Wenn ein Client eine Veröffentlichung für ein Thema durchführt, das er selbst subskribiert hat, empfängt er eine Kopie seiner eigenen Veröffentlichung.

Wenn eine Nachricht mit der Servicequalität QoS von 1 oder 2 gesendet wird, wird sie von der Klasse `MqttClientPersistence` gespeichert, bevor der MQTT-Client `messageArrived` aufruft. `messageArrived` verhält sich wie `deliveryComplete`: Dieses Element wird nur einmal für eine Veröffentlichung aufgerufen und die lokale Kopie der Veröffentlichung wird von `MqttClientPersistence.remove` entfernt, wenn `messageArrived` an den MQTT-Client zurückgegeben wird.

Der MQTT-Client löscht seine Verweise auf das Thema und die Nachricht, wenn `messageArrived` an den MQTT-Client zurückgegeben wird. Die Themen- und Nachrichtenobjekte enthalten fehlerhafte Daten, wenn der Anwendungsclient keine Referenzen auf die Objekte beibehalten hat.

Synchronisation von Callbacks, Threading und Clientanwendungen

Der MQTT-Client ruft eine Callback-Methode nicht im Hauptanwendungsthread, sondern in einem separaten Thread auf. Die Clientanwendung erstellt keinen Thread für den Callback, sondern dieser wird vom MQTT-Client erstellt.

Der MQTT-Client synchronisiert Callback-Methoden. Es wird immer nur eine Instanz der Callback-Methode ausgeführt. Dank der Synchronisation ist es einfach, ein Objekt zu aktualisieren, das mitzählt, welche Veröffentlichungen zugestellt wurden. Da immer nur eine Instanz von `MqttCallback.deliveryComplete` ausgeführt wird, ist die Aktualisierung des Veröffentlichungszählers auch ohne weitere Synchronisation ein zuverlässiger Vorgang. Es trifft außerdem immer nur eine Veröffentlichung ein. Ihr Code in der Methode `messageArrived` kann ein Objekt aktualisieren, ohne es zu synchronisieren. Wenn Sie sich in einem anderen Thread auf den Zähler oder das Objekt, das aktualisiert wird, beziehen, synchronisieren Sie den Zähler oder das Objekt.

Das Zustellungstoken stellt einen Synchronisationsmechanismus zwischen dem Hauptanwendungsthread und der Übergabe einer Veröffentlichung bereit. Die Methode `token.waitForCompletion` wartet, bis die Übergabe einer bestimmten Veröffentlichung abgeschlossen ist oder bis ein optionales Zeitlimit überschritten wird. Sie können `token.waitForCompletion` dazu verwenden, um eine Veröffentlichung nach der anderen zu verarbeiten:

Führen Sie eine Synchronisation mit der Methode `MqttCallback.deliveryComplete` durch. Nur wenn `MqttCallback.deliveryComplete` zum MQTT Client zurückkehrt, wird `token.waitForCompletion` fortgesetzt. Mithilfe dieses Mechanismus können Sie aktiven Code in `MqttCallback.deliveryComplete` synchronisieren, bevor Code im Hauptanwendungsthread ausgeführt wird.

Was ist, wenn Sie Veröffentlichungen durchführen wollen, ohne auf die Übergabe der einzelnen Veröffentlichungen zu warten, sondern erst dann eine Bestätigung möchten, wenn alle Veröffentlichungen übergeben wurden? Wenn Sie die Veröffentlichungen in einem Einzelthread durchführen, ist die letzte Veröffentlichung, die gesendet wird, auch die letzte, die übergeben wird.

Synchronisation von an den Server gesendete Anforderungen

In Tabelle 188 auf Seite 1305 werden die Methoden im MQTT Java-Client beschrieben, die eine Anforderung an den Server senden. Wenn der Anwendungsclient keinen unendlichen Zeitlimitwert festgelegt hat, wartet der Client nie unendlich auf den Server. Wenn der Client blockiert ist, liegt ein Problem in der Anwendungsprogrammierung oder ein Fehler im MQTT-Client vor.

Tabelle 188. Synchronisationsverhalten von Methoden, die zu Anforderungen an den Server führen

Methode	Synchronisation	Zeitlimitintervall
`MqttClient.Connect`	Wartet auf die Herstellung einer Verbindung mit dem Server.	30 Sekunden (Standardwert) oder ein mit einem Parameter festgelegter Wert, bevor eine Ausnahme ausgelöst wird.
`MqttClient.Disconnect`	Wartet, bis der MQTT-Client eventuell erforderliche Arbeitsschritte abgeschlossen hat und bis die TCP/IP-Sitzung getrennt wurde.	
`MqttClient.Subscribe`	Wartet bis zum Abschluss der Methode zur Subskription oder zur Aufhebung der Subskription.	
`MqttClient.UnSubscribe`		

Tabelle 188. Synchronisationsverhalten von Methoden, die zu Anforderungen an den Server führen (Forts.)

Methode	Synchronisation	Zeitlimitintervall
`MqttClient.Publish`	Kehrt nach der Übergabe der Anforderung an den MQTT-Client unverzüglich zum Anwendungsthread zurück.	Keine.
`IMqttDeliveryToken.waitForCompletion`	Wartet auf die Rückgabe des Zustellungstokens.	Unendlich oder ein mit einem Parameter festgelegter Wert.

Zugehörige Konzepte

Sitzungen bereinigen

Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"

Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients

Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen

Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität

Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients

Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Sitzungen bereinigen

Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Beim Verbinden einer MQTT-Clientanwendung mithilfe der Methode `MqttClient.connect` ermittelt der Client die Verbindung mithilfe der Client-ID und der Adresse des Servers. Der Server prüft, ob Sitzungsdaten aus einer vorherigen Verbindung zum Server gespeichert wurden. Wenn noch Informationen zu einer früheren Sitzung vorhanden sind und `cleanSession=true` festgelegt ist, werden die vorherigen Sitzungsdaten im Client und Server gelöscht. Wenn `cleanSession=false` festgelegt ist, wird die vorherige Sitzung fortgesetzt. Wenn keine Informationen zu einer vorherigen Sitzung vorhanden sind, wird eine neue Sitzung gestartet.

Anmerkung: Der IBM MQ-Administrator kann das Schließen einer geöffneten Sitzung und das Löschen aller Sitzungsdaten erzwingen. Wenn der Client eine Sitzung mit dem Wert `cleanSession=false` erneut öffnet, wird eine neue Sitzung gestartet.

Veröffentlichungen

Wenn Sie vor der Herstellung einer Verbindung zum Client den Standardwert `MqttConnectOptions` verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, werden alle Übergaben von anstehenden Veröffentlichungen für den Client entfernt, wenn der Client eine Verbindung herstellt.

Die Einstellung zum Bereinigen einer Sitzung hat keine Auswirkung auf Veröffentlichungen, die mit `QoS=0` gesendet werden. Die Verwendung von `cleanSession=true` kann für `QoS=1` und `QoS=2` zum Verlust einer Veröffentlichung führen.

Subskriptionen

Wenn Sie den Standardwert `MqttConnectOptions`verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor der Client verbunden wird, werden alle alten Subskriptionen für den Client entfernt, wenn der Client eine Verbindung herstellt. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung auf `false` setzen, werden alle Subskriptionen, die der Client erstellt, zu allen Subskriptionen hinzugefügt, die für den Client vor dem Herstellen der Verbindung vorhanden waren. Alle Subskriptionen bleiben aktiv, wenn die Clientverbindung getrennt wird.

Eine andere Möglichkeit, um zu verstehen, wie sich das Attribut `cleanSession` auf Subskriptionen auswirkt, besteht darin, es sich als modales Attribut vorzustellen. Der Standardmodus (`cleanSession=true`) bedeutet, dass der Client nur im Rahmen der Sitzung Subskriptionen erstellt und Veröffentlichungen empfängt. Im alternativen Modus (`cleanSession=false`) sind Subskriptionen permanent. Der Client kann Verbindungen herstellen und trennen, seine Subskriptionen bleiben aktiv. Bei der Wiederher-

stellung einer Verbindung empfängt der Client alle nicht zugestellten Veröffentlichungen. Solange die Verbindung besteht, kann er die Gruppe der Subskriptionen, die in seinem Auftrag aktiv sind, ändern.

Sie müssen den Modus `cleanSession` festlegen, bevor Sie eine Verbindung herstellen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie die Modi von `cleanSession=false` in `cleanSession=true`ändern, werden alle vorherigen Subskriptionen für den Client und alle Veröffentlichungen, die nicht empfangen wurden, verworfen.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Client-ID
Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet.

Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Client-ID

Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Die Client-ID wird bei der Verwaltung eines MQTT-Systems verwendet. Da unter Umständen hunderttausende Clients verwaltet werden, muss es möglich sein, einen bestimmten Client schnell zu erkennen. Nehmen wir beispielsweise an, bei einem Gerät liegt eine Störung vor und Sie werden von einem Kunden, der bei einem Help-Desk anruft, davon benachrichtigt. Wie wird das Gerät vom Kunden identifiziert und wie korrelieren Sie diese Identifikation mit dem Server, der für gewöhnlich mit dem Client verbunden ist?

In der Anzeige der MQTT-Clientverbindungen sehen Sie zu jeder Verbindung die Client-ID. Eine Entscheidung über die optimale Zuordnung dieser ID zur Einheit und zum Server kann durch die Beantwortung der folgenden Fragen vereinfacht werden:

- Ist es sinnvoll, eine Datenbank zu pflegen und zu verwenden, die jedes Gerät einer Client-ID und einem Server zuordnet?
- Gibt der Name des Geräts Aufschluss darüber, mit welchem Server es verbunden ist?
- Müssen Sie in einer Tabelle nachschlagen, um eine Client-ID einem physischen Gerät zuzuordnen?
- Gibt die Client-ID ein bestimmtes Gerät, einen Benutzer oder eine Anwendung an, die auf dem Client ausgeführt wird?
- Wenn ein Kunde ein fehlerhaftes Gerät durch ein neues ersetzt, hat das neue Gerät dieselbe ID wie das alte Gerät? (Wenn Sie ein physisches Gerät ändern, aber dieselbe ID beibehalten, werden ausstehende Veröffentlichungen und aktive Subskriptionen automatisch auf das neue Gerät übertragen.)

Neben einem System für die Generierung eindeutiger IDs müssen Sie über einen zuverlässigen Prozess zur Festlegung der ID auf dem Client verfügen. Möglicherweise ist der Client eine Funktionseinheit ohne Benutzerschnittstelle. Fertigen Sie das Gerät mit einer Client-ID? Oder verfügen Sie über einen Softwareinstallations- und Konfigurationsprozess, mit dem das Gerät vor seiner Aktivierung konfiguriert wird?

Eine Client-ID können Sie beispielsweise auf Basis der aus 48 Bit bestehenden MAC-Adresse des Geräts erstellen, damit die ID kurz und eindeutig ist. Falls die Übertragungsgröße eine untergeordnete Rolle spielt, können Sie die verbleibenden 17 Bytes verwenden, um die Verwaltung der Adresse zu vereinfachen.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlustereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen
Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Spei-

chern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Zustellungstoken

Wenn ein Client ein Thema veröffentlicht, wird ein neues Zustellungstoken erstellt. Verwenden Sie das Zustellungstoken, um die Zustellung einer Veröffentlichung zu überwachen oder die Clientanwendung zu blockieren, bis die Zustellung abgeschlossen ist.

Beim Token handelt es sich um ein `MqttDeliveryToken`-Objekt. Es wird durch den Aufruf der Methode 'MqttTopic.publish()' erstellt und im MQTT-Client aufbewahrt, bis die Clientsitzung getrennt wird und die Übergabe abgeschlossen ist.

Die normale Verwendung des Tokens ist die Überprüfung, ob die Übergabe abgeschlossen wurde. Sperren Sie die Clientanwendung bis zum Abschluss der Übergabe, indem Sie das zurückgegebene Token zum Aufrufen von `token.waitForCompletion` verwenden. Als Alternative stellen Sie einen `MqttCallBack`-Handler bereit. Wenn der MQTT-Client alle Bestätigung empfangen hat, die er als Teil der

Übergabe der Veröffentlichung erwartet, wird `MqttCallBack.deliveryComplete` aufgerufen und das Zustellungstoken wird als Parameter übergeben.

Bis zum Abschluss der Übergabe können Sie die Veröffentlichung mithilfe des zurückgegebenen Zustellungstokens überprüfen, indem Sie `token.getMessage` aufrufen.

Abgeschlossene Übergaben

Der Abschluss von Übergaben ist asynchron und hängt von der Qualität des Service ab, der der Veröffentlichung zugeordnet ist.

At most once (Höchstens einmal)

`QoS=0`
Die Übergabe ist sofort nach der Rückgabe von `MqttTopic.publish` abgeschlossen. `MqttCallback.deliveryComplete` wird unverzüglich aufgerufen.

At least once (Mindestens einmal)

`QoS=1`
Die Übergabe ist abgeschlossen, wenn eine Bestätigung der Veröffentlichung vom Warteschlangenmanager empfangen wurde. `MqttCallback.deliveryComplete` wird aufgerufen, wenn die Bestätigung empfangen wurde. Die Nachricht wird möglicherweise mehrfach vor dem Aufrufen von `MqttCallback.deliveryComplete` übergeben, wenn die Datenübertragung langsam oder störanfällig ist.

Exactly once (Exakt einmal)

`QoS=2`
Die Übergabe ist abgeschlossen, wenn der Client eine Beendigungsnachricht darüber empfängt, dass die Veröffentlichung für Subskribenten veröffentlicht wurde. `MqttCallback.deliveryComplete` wird aufgerufen, sobald die Veröffentlichungsnachricht empfangen wurde. Es wird nicht auf die Beendigungsnachricht gewartet.

In seltenen Fällen kehrt Ihre Clientanwendung von `MqttCallback.deliveryComplete` nicht normal zum MQTT -Client zurück. Sie wissen, dass die Übergabe abgeschlossen ist, da `MqttCallback.deliveryComplete` aufgerufen wurde. Wenn der Client die gleiche Sitzung erneut startet, wird `MqttCallback.deliveryComplete` nicht erneut aufgerufen.

Unvollständige Übergaben

Wenn die Übergabe nach dem Trennen der Clientsitzung nicht abgeschlossen ist, können Sie den Client erneut verbinden und die Übergabe abschließen. Sie können die Übergabe einer Nachricht nur abschließen, wenn die Nachricht in einer Sitzung veröffentlicht wurde, in der das Attribut `MqttConnectionOptions` auf `false` gesetzt ist.

Erstellen Sie den Client mit der gleichen Client-ID und Serveradresse und stellen Sie dann die Verbindung her, wobei das `cleanSession`-Attribut `MqttConnectionOptions` erneut auf `false` gesetzt ist. Wenn Sie `cleanSession` auf `true` setzen, werden anstehende Zustellungstoken verworfen.

Durch das Aufrufen von `MqttClient.getPendingDeliveryTokens` können Sie prüfen, ob anstehende Übergaben vorhanden sind. Sie können `MqttClient.getPendingDeliveryTokens` vor dem Herstellen einer Verbindung zum Client aufrufen.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen
Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung

und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID
Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Veröffentlichung "Last Will and Testament"

Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and

Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Erstellen Sie ein Thema für "Last Will and Testament". Sie können ein Thema wie `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`erstellen.

Installieren Sie "Last Will and Testament" mithilfe der Methode `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Sie können in der Nachricht `lastWillPayload` auch eine Zeitmarke erstellen. Integrieren Sie weitere Clientinformationen, die das Ermitteln des Clients und die Bedingungen der Verbindung unterstützen. Übergeben Sie das Objekt `MqttConnectionOptions` an den `MqttClient`-Konstruktor.

Setzen Sie `lastWillQos` auf 1 oder 2, damit die Nachricht in IBM MQ persistent ist und die Übergabe garantiert wird. Um die Informationen der letzten Verbindung aufzubewahren, setzen Sie `lastWillRetained` auf `true`.

Die Veröffentlichung "Last Will and Testament" wird an Subskribenten gesendet, wenn die Verbindung unerwartet beendet wird. Sie wird gesendet, wenn der Client beim Beenden der Verbindung nicht die Methode `MqttClient.disconnect` aufruft.

Um Verbindungen zu überwachen, ergänzen Sie die Veröffentlichung "Last Will and Testament" mit anderen Veröffentlichungen, damit Verbindungen und programmierte Unterbrechungen aufgezeichnet werden.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen
Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID
Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und

"exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Nachrichtenpersistenz in MQTT-Clients

Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Die Nachrichtenpersistenz in MQTT umfasst zwei Aspekte: Die Funktionsweise der Nachrichtenübertragung und die Angabe darüber, ob die Nachricht in IBM MQ als eine persistente Nachricht in die Warteschlange gestellt wird.

1. Der MQTT-Client verbindet die Nachrichtenpersistenz mit der Servicequalität. Abhängig von der von Ihnen für eine Nachricht ausgewählten Servicequalität wird die Nachricht zu einer persistenten Nachricht. Die Nachrichtenpersistenz ist für die Implementierung der erforderlichen Servicequalität notwendig.

   Wenn Sie "at most once", `QoS=0` angeben, löscht der Client die Nachricht, sobald diese veröffentlicht ist. Wenn es bei der vorgelagerten Verarbeitung der Nachricht zu Fehlern kommt, wird die Nachricht nicht erneut gesendet. Selbst wenn der Client weiterhin aktiv ist, wird die Nachricht nicht erneut gesendet. Das Verhalten von `QoS=0`-Nachrichten entspricht dem Verhalten von schnellen, nicht persistenten IBM MQ-Nachrichten.
   Wenn eine Nachricht von einem Client veröffentlicht wird, für den `QoS` auf 1 oder 2 gesetzt ist, wird die Nachricht zu einer persistenten Nachricht. Die Nachricht wird lokal gespeichert und nur dann aus dem Client gelöscht, wenn die Übergabe vom Typ "at least once", `QoS=1` oder "exactly once", `QoS=2` nicht mehr garantiert werden muss.
2. Wenn eine Nachricht als `QoS` 1 oder 2 markiert ist, wird sie als persistente Nachricht in IBM MQ eingereiht. Wenn sie als `QoS=0` markiert ist, wird sie als nicht persistente Nachricht in IBM MQ eingereiht. In IBM MQ werden nicht persistente Nachrichten "exactly once" (exakt einmal) zwischen Warteschlangenmanagern übertragen, es sei denn, das Attribut `NPMSPEED` ist im Nachrichtenkanal auf `FAST` gesetzt.

Eine persistente Veröffentlichung wird im Client gespeichert, bis sie von einer Clientanwendung empfangen wird. Wenn `QoS=2` festgelegt ist, wird die Veröffentlichung aus dem Client gelöscht, wenn der Callback der Anwendung die Steuerung zurückgibt. Wenn `QoS=1` festgelegt ist, empfängt die Anwendung die Veröffentlichung im Falle eines Fehlers möglicherweise erneut. Wenn `QoS=0` festgelegt ist, empfängt der Callback die Veröffentlichung höchstens einmal. Die Veröffentlichung wird möglicherweise nicht empfangen, wenn ein Fehler auftritt oder die Verbindung zum Client zum Zeitpunkt der Veröffentlichung getrennt wird.

Wenn Sie ein Thema subskribieren, können Sie die Servicequalität QoS, mit der der Subskribent Nachrichten empfängt, reduzieren, damit sie mit den Funktionen der zugehörigen Persistenz übereinstimmt. Die in einer höheren `Servicequalität` erstellten Veröffentlichungen werden mit der höchsten vom Subskribenten angeforderten `Servicequalität` gesendet.

Nachrichten speichern

Bei der Implementierung von Datenspeichern auf kleinen Einheiten gibt es große Unterschiede. Das Modell zum temporären Speichern persistenter Nachrichten in einem vom MQTT-Client verwalteten Speicher ist möglicherweise zu langsam oder beansprucht einen zu großen Speicherbereich. Das Betriebssystem für mobile Geräte stellt möglicherweise einen Speicherservice bereit, der für MQTT-Nachrichten optimal geeignet ist.

Um eine Flexibilität bei der Einhaltung der Einschränkungen von kompakten Endgeräten bereitzustellen, verfügt der MQTT-Client über zwei Schnittstellen für die Persistenz. Die Schnittstellen definieren die Operationen, die beim Speichern persistenter Nachrichten beteiligt sind. Die Schnittstellen werden in der API-Dokumentation für den MQTT client for Java beschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference. Sie können die Schnittstellen auf ein Gerät angepasst implementieren. Der MQTT-Client, der auf Java SE ausgeführt wird, verfügt über eine Standardimplementierung der Schnittstellen, die persistente Nachrichten im Dateisystem speichern. Dabei wird das Paket `java.io` verwendet.

Persistenzklassen

`MqttClientPersistence`
: Übergibt eine Instanz Ihrer Implementierung von `MqttClientPersistence` an den MQTT-Client als Parameter des `MqttClient`-Konstruktors. Wenn Sie den Parameter `MqttClientPersistence` vom `MqttClient`-Konstruktor übergehen, speichert der MQTT-Client persistente Nachrichten mithilfe der Klasse `MqttDefaultFilePersistence`.

`MqttPersistable`
: `MqttClientPersistence` ruft `MqttPersistable`-Objekte mithilfe eines Speicherschlüssels ab und reiht sie ein. Sie müssen eine Implementierung von `MqttPersistable` sowie die Implementierung von `MqttClientPersistence` bereitstellen, wenn Sie die Klasse `MqttDefaultFilePersistence` nicht verwenden.

`MqttDefaultFilePersistence`

: Der MQTT-Client stellt die Klasse `MqttDefaultFilePersistence` bereit. Wenn Sie eine Instanz von `MqttDefaultFilePersistence` in Ihrer Clientanwendung erstellen, können Sie das Verzeichnis bereitstellen, in dem persistente Nachrichten als ein Parameter des `MqttDefaultFilePersistence`-Konstruktors gespeichert werden.

  Alternativ dazu kann der MQTT-Client eine Instanz von `MqttDefaultFilePersistence` erstellen und Dateien im folgenden Standardverzeichnis speichern:

  `client identifier` `-tcp` *`hostname portnumber`*

  Die folgenden Zeichen werden aus der Zeichenfolge für den Verzeichnisnamen entfernt:

  `"\", "\\", "/", ":"` und `" "`

  Der Pfad zum Verzeichnis entspricht dem Wert der Systemeigenschaft `rcp.data`; falls `rcp.data` nicht festgelegt ist, entspricht der Pfad dem Wert der Systemeigenschaft `usr.data`. Dabei gilt Folgendes:

  - `rcp.data` ist eine Eigenschaft, die der Installation einer OSGi- oder Eclipse-Rich-Client-Plattform (RCP) zugeordnet ist.
  - `usr.data` ist das Verzeichnis, in dem der Java-Befehl zum Starten der Anwendung gestartet wurde.

Zugehörige Konzepte
Callbacks und Synchronisierung in MQTT-Clientanwendungen

Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen

Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"

Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Veröffentlichungen

Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität

Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients

Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen

Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients

Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Veröffentlichungen

Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Die Nutzdaten von `MqttMessage` umfassen eine Bytefeldgruppe. Nachrichten sollten so klein wie möglich sein. Das MQTT protocol erlaubt eine maximale Nachrichtenlänge von 250 MB.

Ein MQTT-Clientprogramm verwendet normalerweise `java.lang.String` oder `java.lang.StringBuffer` zum Bearbeiten von Nachrichteninhalten. Für eine einfachere Verarbeitung enthält die Klasse `MqttMessage` eine `toString`-Methode, mit der die zugehörigen Nutzdaten in eine Zeichenfolge umgewandelt werden können. Verwenden Sie die Methode `getBytes`, um die Nutzdaten aus der Bytefeldgruppe aus `java.lang.String` oder `java.lang.StringBuffer` zu erstellen.

Die Methode `getBytes` wandelt eine Zeichenfolge in den Standardzeichensatz für die Plattform um. Als Standardzeichensatz wird im Allgemeinen `UTF-8` verwendet. MQTT-Veröffentlichungen, die nur Text enthalten, werden normalerweise in `UTF-8` codiert. Überschreiben Sie den Standardzeichensatz mit der Methode `getBytes("UTF8")`.

In IBM MQ wird eine MQTT-Veröffentlichung als `jms-bytes`-Nachricht empfangen. Die Nachricht enthält einen Ordner `MQRFH2`, in den die Ordner `<mqtt>` und `<mqps>` integriert sind. Der Ordner `<mqtt>` enthält die Angaben für `clientId` (Client-ID), `msgId` (Nachrichten-ID) und `qos` (Servicequalität), doch deren Inhalte können sich in der Zukunft ändern.

`MqttMessage` verfügt über drei zusätzliche Attribute: Servicequalität, Angabe darüber, ob die Methode aufbewahrt wird und ob es sich um eine Kopie handelt. Das Flag für eine Kopie wird nur festgelegt, wenn die Servicequalität auf "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesetzt ist. Wenn die Nachricht zuvor gesendet und nicht schnell genug vom MQTT-Client bestätigt wurde, wird die Nachricht erneut gesendet, wobei das doppelte Attribut auf `true` gesetzt wird.

Veröffentlichen

Wenn Sie eine Veröffentlichung in einer MQTT-Clientanwendung erstellen möchten, erstellen Sie ein `MqttMessage`-Objekt. Legen Sie die Nutzdaten und die Servicequalität fest, geben Sie an, ob das Objekt gespeichert werden soll, und rufen Sie die Methode `MqttTopic.publish(MqttMessage message)` auf;`MqttDeliveryToken` wird zurückgegeben und der Abschluss der Veröffentlichung ist asynchron.

Alternativ dazu kann der MQTT-Client beim Erstellen einer Veröffentlichung ein temporäres Nachrichtenobjekt aus den Parametern in der Methode `MqttTopic.publish(byte [] payload, int qos, boolean retained)` erstellen.

Wenn für die Veröffentlichung die Servicequalität "at least once" oder "exactly once" festgelegt ist (`QoS=1` oder `QoS=2`), ruft der MQTT-Client die Schnittstelle `MqttClientPersistence` auf. Die Nachricht wird in der Schnittstelle `MqttClientPersistence` gespeichert, bevor ein Zustellungstoken an die Anwendung zurückgegeben wird.

Die Anwendung kann mithilfe der Methode `MqttDeliveryToken.waitForCompletion` gesperrt werden, bis die Nachricht an den Server übergeben wird. Sie kann aber auch ohne Sperren fortgesetzt werden. Wenn Sie überprüfen möchten, ob Veröffentlichungen ohne Sperren übergeben wurden, registrieren Sie eine Instanz einer Callback-Klasse, die `MqttCallback` im MQTT-Client implementiert. Der MQTT-Client ruft die Methode `MqttCallback.deliveryComplete` auf, sobald die Veröffentlichung übergeben wurde. Je nach Servicequalität findet die Übergabe bei `QoS=0` direkt statt oder beansprucht bei `QoS=2` einige Zeit.

Fragen Sie mit der Methode `MqttDeliveryToken.isComplete` ab, ob die Übergabe abgeschlossen ist. Wenn der Wert von `MqttDeliveryToken.isComplete` auf `false` gesetzt ist, können Sie die Methode `MqttDeliveryToken.getMessage` zum Abrufen der Nachrichteninhalte aufrufen. Wenn nach dem Aufrufen von `MqttDeliveryToken.isComplete` das Ergebnis `true` lautet, wurde die Nachricht gelöscht und beim Aufrufen von `MqttDeliveryToken.getMessage` würde eine Nullzeigerausnahme ausgelöst. Es gibt keine integrierte Synchronisierung zwischen `MqttDeliveryToken.getMessage` und `MqttDeliveryToken.isComplete`.

Wenn die Verbindung zum Client vor dem Empfang aller anstehender Zustellungstoken getrennt wird, kann eine neue Instanz des Clients anstehende Zustellungstoken vor dem Verbinden abfragen. Bis zum Herstellen einer Verbindung zum Client werden keine neuen Übergaben abgeschlossen und die Methode `MqttDeliveryToken.getMessage` kann aufgerufen werden. Finden Sie mithilfe der Methode `MqttDeliveryToken.getMessage` heraus, welche Veröffentlichungen nicht übergeben wurden. Anstehende Zustellungstoken werden gelöscht, wenn `MqttConnectOptions.cleanSession` beim Herstellen einer Verbindung auf den Standardwert `true` gesetzt ist.

Subskribieren

Ein Warteschlangenmanager ist für das Erstellen von Veröffentlichungen verantwortlich, die an einen MQTT-Subskribenten gesendet werden. Der Warteschlangenmanager überprüft, ob der von einem MQTT-Client erstellte Themenfilter in einer Subskription mit der Themenzeichenfolge in einer Veröffentlichung übereinstimmt. Es kann sich dabei um eine exakte Übereinstimmung oder um eine Übereinstimmung mit Platzhalterzeichen handeln. Vor dem Weiterleiten der Veröffentlichung an den Subskribenten durch den Warteschlangenmanager überprüft der Warteschlangenmanager die Themenattribute, die der Veröffentlichung zugeordnet sind. Er folgt dabei der im Abschnitt Subskription mithilfe einer Themenzeichenfolge mit Platzhalterzeichen beschriebenen Suchprozedur, um zu ermitteln, ob ein administratives Themenobjekt dem Benutzer die Berechtigung zur Subskription erteilt.

Wenn der MQTT-Client eine Veröffentlichung mit der Servicequalität "at least once" empfängt, ruft er zur Verarbeitung der Veröffentlichung die Methode `MqttCallback.messageArrived` auf. Wenn die Servicequalität der Veröffentlichung "exactly once" (QoS=2) ist, ruft der MQTT-Client die Schnittstelle `MqttClientPersistence` auf, um die Nachricht nach dem Empfang zu speichern. Anschließend wird die Methode `MqttCallback.messageArrived` aufgerufen.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen
Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID
Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Per-

sistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Von einem MQTT-Client bereitgestellte Servicequalität

Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Bei der Servicequalität einer Veröffentlichung handelt es sich um ein Attribut von `MqttMessage`. Es wird von der Methode `MqttMessage.setQos` festgelegt.

Die Methode `MqttClient.subscribe` kann die für Veröffentlichungen angewendete Servicequalität verringern, die einem Client zu einem Thema gesendet werden. Die Servicequalität einer Veröffentlichung, die an einen Subskribenten weitergeleitet wird, kann von der Servicequalität der Veröffentlichung abweichen. Zur Weiterleitung einer Veröffentlichung wird der niedrigere der beiden Werte verwendet.

At most once (Höchstens einmal)
`QoS=0`

Die Nachricht wird höchstens einmal oder gar nicht übermittelt. Die Übergabe im Netz wird nicht bestätigt.

Die Nachricht wird nicht gespeichert. Die Nachricht kann verloren gehen, wenn die Verbindung zum Client getrennt wird oder der Server fehlschlägt.

`QoS=0` ist der schnellste Übertragungsmodus. Er wird auch als "Fire-and-Forget" (Abfeuern und vergessen) bezeichnet.

Für das MQTT protocol sind keine Server zum Weiterleiten von Veröffentlichungen mit der Servicequalität `QoS=0` an einen Client erforderlich. Wenn die Verbindung zum Client zu dem Zeitpunkt getrennt wird, an dem der Server die Veröffentlichung empfängt, wird die Veröffentlichung je nach Server möglicherweise gelöscht. Der Telemetrieservice (MQXR) löscht keine Nachrichten, die mit der Servicequalität `QoS=0` gesendet wurden. Sie werden als nicht persistente Nachrichten gespeichert und nur gelöscht, wenn der Warteschlangenmanager angehalten wird.

At least once (Mindestens einmal)
`QoS=1`

QoS=1 ist der Standardmodus für die Übertragung.

Die Nachricht wird immer mindestens einmal übertragen. Wenn der Absender keine Bestätigung empfängt, wird die Nachricht erneut gesendet und das Flag DUP wird festgelegt, bis eine Bestätigung empfangen wird. Dadurch kann die gleiche Nachricht mehrfach an einen Empfänger gesendet und möglicherweise mehrfach von ihm verarbeitet werden.

Die Nachricht muss bis zur Verarbeitung lokal im Absender und im Empfänger gespeichert sein.

Die Nachricht wird aus dem Empfänger gelöscht, nachdem dieser die Nachricht verarbeitet hat. Wenn es sich beim Empfänger um einen Broker handelt, wird die Nachricht auf den zugehörigen Subskribenten veröffentlicht. Wenn es sich beim Empfänger um einen Client handelt, wird die Nachricht an die Subskribentenanwendung übergeben. Nach dem Löschen der Nachricht sendet der Empfänger eine Bestätigung an den Absender.

Die Nachricht wird nach dem Empfang der Bestätigung vom Empfänger aus dem Absender gelöscht.

Exactly once (Exakt einmal)

QoS=2

Die Nachricht wird immer exakt einmal übergeben.

Die Nachricht muss bis zur Verarbeitung lokal im Absender und im Empfänger gespeichert sein.

Bei QoS=2 handelt es sich um die sicherste Servicequalität, aber auch um den langsamsten Übertragungsmodus. Es müssen mindestens zwei Übertragungen zwischen dem Absender und dem Empfänger stattfinden, bevor die Nachricht aus dem Absender gelöscht wird. Die Nachricht kann nach der ersten Übertragung im Empfänger verarbeitet werden.

Bei der ersten Übertragung überträgt der Absender die Nachricht und erhält eine Bestätigung vom Empfänger, dass dieser die Nachricht gespeichert hat. Wenn der Absender keine Bestätigung empfängt, wird die Nachricht erneut gesendet und das Flag DUP wird festgelegt, bis eine Bestätigung empfangen wird.

Bei der zweiten Übertragung teilt der Absender dem Empfänger mit, dass dieser die Verarbeitung der Nachricht "PUBREL" durch führen kann. Wenn der Absender keine Bestätigung der Nachricht "PUBREL" empfängt, wird die Nachricht "PUBREL" erneut gesendet, bis eine Bestätigung empfangen wird. Der Absender löscht die gespeicherte Nachricht, wenn er die Bestätigung für die Nachricht "PUBREL" empfängt.

Der Empfänger kann die Nachricht in der ersten oder zweiten Phase verarbeiten, unter der Voraussetzung, dass die Nachricht nicht erneut verarbeitet wird. Wenn es sich beim Empfänger um einen Broker handelt, wird die Nachricht für Subskribenten veröffentlicht. Wenn es sich beim Empfänger um einen Client handelt, wird die Nachricht für die Subskribentenanwendung übergeben. Der Empfänger sendet dem Absender eine Beendigungsnachricht mit der Information, dass die Verarbeitung der Nachricht abgeschlossen wurde.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen

Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlustereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen

Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist

wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Ständige Veröffentlichungen und MQTT-Clients

Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Geben Sie mithilfe der Methode `MqttMessage.setRetained` an, ob eine Veröffentlichung für ein Thema aufbewahrt wird.

Wenn Sie eine ständige Veröffentlichung erstellen oder aktualisieren, senden Sie die Veröffentlichung mit der Servicequalität (QoS) 1 oder 2. Wenn Sie es mit der QoS 0 senden, erstellt IBM MQ eine nicht persistente ständige Veröffentlichung. Die Veröffentlichung wird nicht beibehalten, wenn der Warteschlangenmanager gestoppt wird.

Wenn Sie zu einem Thema, dem eine ständige Veröffentlichung zugeordnet ist, eine nicht ständige Veröffentlichung erstellen, wirkt sich dies nicht auf die ständige Veröffentlichung aus. Bereits bestehende Subskribenten erhalten die neue Veröffentlichung. Neue Subskribenten erhalten zunächst die ständige Veröffentlichung und dann die neue Veröffentlichung.

Sie können eine ständige Veröffentlichung verwenden, um den neuesten Wert einer Messung aufzuzeichnen. Neue Subskribenten für ein Thema erhalten sofort den neuesten Wert der Messung. Falls keine neuen Messungen durchgeführt wurden, seitdem der Subskribent das Veröffentlichungsthema zuletzt

subskribiert hat, erhält der Subskribent bei einer weiteren Subskription dieses Themas erneut die aktuellste ständige Veröffentlichung zu diesem Thema.

Sie haben zwei Möglichkeiten zum Löschen einer ständigen Veröffentlichung:

- Führen Sie den MQSC-Befehl **CLEAR TOPICSTR** aus.
- Erstellen einer ständigen Veröffentlichung der Länge Null. Wie in der MQTT 3.1.1-Spezifikation angegeben, werden alle aufbewahrten Nachrichten für ein Thema gelöscht, wenn eine aufbewahrte Nachricht der Länge Null für das Thema veröffentlicht wird.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen
Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID
Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Subskriptionen

Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Subskriptionen

Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Erstellen Sie mithilfe der `MqttClient.subscribe`-Methoden Subskriptionen und übergeben Sie dabei mindestens einen Themenfilter und Parameter für die Servicequalität. Der Parameter für die Servicequalität legt die maximale Servicequalität fest, die der Subskribent zum Empfang einer Nachricht verwenden möchte. Nachrichten, die an diesen Client gesendet werden, können nicht mit einer höheren Servicequalität zugestellt werden. Die Servicequalität wird auf den kleineren der ursprünglichen Werte gesetzt, die galten, als die Nachricht veröffentlicht und die Stufe für die Subskription angegeben wurde. Die standardmäßige Servicequalität für den Nachrichtenempfang lautet `QoS=1` (mindestens einmal).

Die Subskriptionsanforderung selbst wird mit der Einstellung `QoS=1` gesendet.

Veröffentlichungen werden von einem Subskribenten empfangen, wenn der MQTT-Client die Methode `MqttCallback.messageArrived` aufruft. Die Methode `messageArrived` übergibt außerdem die Themenzeichenfolge, mit der die Nachricht an den Subskribenten übergeben wurde.

Sie können eine Subskription oder Subskriptionsgruppe mithilfe der `MqttClient.unsubscribe`-Methoden entfernen.

Ein IBM MQ-Befehl kann eine Subskription entfernen. 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*

Wenn Sie den Standardwert `MqttConnectOptions`verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor der Client verbunden wird, werden alle alten Subskriptionen für den Client entfernt, wenn der Client eine Verbindung herstellt. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung auf `false` setzen, werden alle Subskriptionen, die der Client erstellt, zu allen Subskriptionen hinzugefügt, die für den Client vor dem Herstellen der Verbindung vorhanden waren. Alle Subskriptionen bleiben aktiv, wenn die Clientverbindung getrennt wird.

Eine andere Möglichkeit, um zu verstehen, wie sich das Attribut `cleanSession` auf Subskriptionen auswirkt, besteht darin, es sich als modales Attribut vorzustellen. Der Standardmodus (`cleanSession=true`) bedeutet, dass der Client nur im Rahmen der Sitzung Subskriptionen erstellt und Veröffentlichungen empfängt. Im alternativen Modus (`cleanSession=false`) sind Subskriptionen permanent. Der Client kann Verbindungen herstellen und trennen, seine Subskriptionen bleiben aktiv. Bei der Wiederherstellung einer Verbindung empfängt der Client alle nicht zugestellten Veröffentlichungen. Solange die Verbindung besteht, kann er die Gruppe der Subskriptionen, die in seinem Auftrag aktiv sind, ändern.

Sie müssen den Modus `cleanSession` festlegen, bevor Sie eine Verbindung herstellen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie die Modi von `cleanSession=false` in `cleanSession=true`ändern, werden

alle vorherigen Subskriptionen für den Client und alle Veröffentlichungen, die nicht empfangen wurden, verworfen.

Veröffentlichungen, die aktiven Subskriptionen entsprechen, werden unmittelbar bei ihrer Veröffentlichung an den Client gesendet. Veröffentlichungen, die aktiven Subskriptionen entsprechen, werden unmittelbar bei ihrer Veröffentlichung an den Client gesendet. Wenn der Client nicht verbunden ist, werden sie an den Client gesendet, sobald dieser die Verbindung zu demselben Server wiederherstellt. Dabei muss die Client-ID identisch sein und `MqttConnectOptions.cleanSession` muss auf `false` gesetzt sein.

Subskriptionen für einen bestimmten Client werden über die Client-ID ermittelt. Sie können den Client von einem anderen Clientgerät aus erneut mit demselben Server verbinden und weiterhin dieselben Subskriptionen nutzen und nicht zugestellte Veröffentlichungen empfangen.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlustereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen
Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID
Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription

an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Themenzeichenfolgen und Themenfilter in MQTT-Clients
Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Themenzeichenfolgen und Themenfilter in MQTT-Clients

Themenzeichenfolgen und Themenfilter werden beim Veröffentlichen und Subskribieren verwendet. Die Syntax von Themenzeichenfolgen und -filtern in MQTT-Clients ist weitgehend mit derjenigen von Themenzeichenfolgen in IBM MQ identisch.

Mit Themenzeichenfolgen werden Veröffentlichungen an Subskribenten gesendet. Erstellen Sie eine Themenzeichenfolge mithilfe der Methode `MqttClient.getTopic(java.lang.String topicString)`.

Mit Themenfiltern werden Themen subskribiert und Veröffentlichungen empfangen. Themenfilter können Platzhalterzeichen enthalten. Mithilfe von Platzhalterzeichen können mehrere Themen subskribiert werden. Erstellen Sie einen Themenfilter mithilfe einer Subskriptionsmethode, z. B. `MqttClient.subscribe(java.lang.String topicFilter)`.

Themenzeichenfolgen

Die Syntax einer IBM MQ-Themenzeichenfolge ist im Abschnitt Themenzeichenfolgen beschrieben. Die Syntax von MQTT-Themenzeichenfolgen ist in der Klasse `MqttClient` in der API-Dokumentation für den MQTT client for Java beschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

Die Syntax ist bei allen Themenzeichenfolgen nahezu identisch. Es gibt vier geringfügige Unterschiede:

1. Themenzeichenfolgen, die von MQTT -Clients an IBM MQ gesendet werden, müssen der Konvention für Warteschlangenmanagernamen entsprechen.
2. Die maximale Länge ist unterschiedlich. IBM MQ-Themenzeichenfolgen sind auf 10.240 Zeichen begrenzt. Ein MQTT-Client kann Themenzeichenfolgen mit bis zu 65.535 Bytes erstellen.
3. Eine Themenzeichenfolge, die von einem MQTT-Client erstellt wurde, kann kein Nullzeichen enthalten.
4. In IBM Integration Bus ist ein Thema der Ebene null (`'...//...'`) ungültig. IBM MQ unterstützt Themen der Ebene null.

Anders als die Publish/Subscribe-Funktion von IBM MQ kennt das Protokoll `mqttv3` kein Konzept eines verwalteten Themenobjekts. Es kann keine Themenzeichenfolge aus einem Themenobjekt und einer Themenzeichenfolge erstellt werden. Eine Themenzeichenfolge wird jedoch einem Verwaltungsthema in IBM MQ zugeordnet. Die Zugriffssteuerung, die dem Verwaltungsthema zugeordnet ist, bestimmt, ob eine Veröffentlichung für ein Thema veröffentlicht oder verworfen wird. Die Attribute, die auf eine Veröffentlichung angewendet werden, wenn sie an Subskribenten weitergeleitet wird, werden durch die Attribute des Verwaltungsthemas beeinflusst.

Themenfilter

Die Syntax eines IBM MQ-Themenfilters ist im Abschnitt Themenbasiertes Platzhalterschema beschrieben. Die Syntax der Themenfilter, die Sie mit einem MQTT-Client erstellen können, ist in der Klasse `MqttClient` in der API-Dokumentation für den MQTT client for Java beschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

Zugehörige Konzepte
Callbacks und Synchronisierung in MQTT-Clientanwendungen

Im Programmiermodell für den MQTT-Client werden Threads intensiv genutzt. Durch die Threads wird eine MQTT-Clientanwendung beim Übertragen von Nachrichten zum Server und vom Server so weit wie möglich von Verzögerungen entkoppelt. Veröffentlichungen, Zustellungstoken und Verbindungsverlusterereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Sitzungen bereinigen
Im MQTT-Client und Telemetrieservice (MQXR) werden Informationen zum Sitzungsstatus gespeichert. Mit den Statusinformationen wird sichergestellt, dass "mindestens eine" und "exakt eine" Übermittlung und "exakt ein" Empfang von Veröffentlichungen stattfindet. Der Sitzungsstatus enthält auch Subskriptionen, die von einem MQTT-Client erstellt wurden. Sie können einen MQTT-Client mit oder ohne das Speichern von Statusinformation zwischen Sitzungen ausführen. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID
Bei der Client-ID handelt es sich um eine Zeichenfolge mit 23 Byte, die zur Identifikation eines MQTT-Clients dient. Jede ID muss eindeutig sein, damit jeweils nur eine Verbindung zu einem Client hergestellt wird. Die ID darf nur Zeichen enthalten, die in einem Warteschlangenmanagernamen gültig sind. Innerhalb dieser Beschränkungen können Sie beliebige Zeichenfolgen für die Identifikation verwenden. Es ist wichtig, eine Vorgehensweise für die Zuordnung von Client-IDs und für die Konfiguration eines Clients mit der ausgewählten ID festzulegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Wenn eine MQTT-Clientverbindung unerwartet beendet wird, können Sie MQ Telemetry zum Senden einer Veröffentlichung des Typs "Last Will and Testament" konfigurieren. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Nachrichtenpersistenz in MQTT-Clients
Veröffentlichungsnachrichten sind persistent, wenn Sie mit der Servicequalität "at least once" (mindestens einmal) oder "exactly once" (exakt einmal) gesendet werden. Sie können Ihren eigenen Persistenzmechanismus im Client implementieren oder den mit dem Client bereitgestellten standardmäßigen Persistenzmechanismus verwenden. Die Persistenz gilt in beiden Richtungen, also für Veröffentlichungen, die an den Client oder vom Client gesendet werden.

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Clients können Veröffentlichungen erstellen, die an IBM MQ gesendet werden, und Themen auf IBM MQ subskribieren, damit sie Veröffentlichungen empfangen können.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt zur Übergabe von Veröffentlichungen an IBM MQ und an den MQTT-Client drei Servicequalitäten bereit: "at most once" (höchstens einmal), "at least once" (mindestens einmal) und "exactly once" (exakt einmal). Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an IBM MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Ein Thema kann nur genau eine ständige Veröffentlichung enthalten. Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die Veröffentlichung sofort an Sie weitergeleitet.

Subskriptionen
Durch das Erstellen von Subskriptionen können Sie mithilfe eines Themenfilters Bedarf an bestimmten Veröffentlichungsthemen anmelden. Ein Client kann zur Anmeldung des Bedarfs an mehreren Themen mehrere Subskriptionen erstellen oder eine Subskription, die einen Themenfilter mit Platzhalterzeichen enthält. Veröffentlichungen zu Themen, die den Filtern entsprechen, werden an den Client gesendet. Subskriptionen können auch dann aktiv bleiben, wenn die Verbindung zu einem Client getrennt ist. Die Veröffentlichungen werden an den Client gesendet, sobald die Verbindung wiederhergestellt ist.

Microsoft Windows Communication Foundation -Anwendungen mit IBM MQ entwickeln

Der angepasste WCF-Kanal (Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

Zugehörige Konzepte

„Einführung in den angepassten IBM MQ-Kanal für WCF mit .NET" auf Seite 1327
Beim angepassten Kanal für IBM MQ handelt es sich um einen Transportkanal, der das einheitliche Programmiermodell von Microsoft Windows Communication Foundation (WCF) verwendet.

„Angepasste IBM MQ-Kanäle für WCF verwenden" auf Seite 1331
Übersicht über die Informationen, die für Programmierer verfügbar sind, die die angepassten IBM MQ-Kanäle für Windows Communication Foundation (WCF) verwenden.

„WCF-Beispiele verwenden" auf Seite 1352
Das Beispiel für Windows Communication Foundation (WCF) stellt einige einfache Beispiele zur Verwendung des angepassten IBM MQ-Kanals bereit.

FFST: WCF XMS First Failure Support Technology

Zugehörige Tasks

Traceerstellung für den angepassten WCF-Kanal für IBM MQ

Fehlerbehebung für angepasste WCF-Kanäle für IBM MQ -Probleme

Einführung in den angepassten IBM MQ-Kanal für WCF mit .NET

Beim angepassten Kanal für IBM MQ handelt es sich um einen Transportkanal, der das einheitliche Programmiermodell von Microsoft Windows Communication Foundation (WCF) verwendet.

Das in Microsoft.NET 3 eingeführte Framework 'Microsoft Windows Communication Foundation' ermöglicht die Entwicklung von .NET-Anwendungen und -Services unabhängig vom Transport und von den Protokollen, die zu ihrer Verbindung verwendet werden, weshalb je nach Umgebung, in der der Service oder die Anwendung implementiert ist, alternative Transporte oder Konfigurationen genutzt werden können.

Verbindungen werden während Laufzeit von WCF durch das Erstellen eines Kanal-Stacks mit der erforderlichen Kombination der folgenden Komponenten hergestellt:

- Protokollelemente: Eine optional Gruppe von Elementen, in der kein Element, ein oder mehrere Elemente hinzugefügt werden können, um Protokolle wie beispielsweise die WS-*-Standards zu unterstützen.
- Nachrichtenencoder: Ein verbindliches Element im Stack, mit dem die Serialisierung der Nachricht in das zugehörige Sendeformat gesteuert wird.
- Transportkanal: Ein verbindliches Element im Stack, das für den Transport der serialisierten Nachricht auf den zugehörigen Endpunkt verantwortlich ist.

Beim angepassten Kanal für IBM MQ handelt es sich um einen Transportkanal, der als solcher mit einem Nachrichtenencoder und optionalen Protokollen paarweise verbunden werden muss, wie dies für die Anwendung, die eine angepasste WCF-Bindung verwendet, erforderlich ist. Dadurch können Anwendungen, die für die Verwendung von WCF entwickelt wurden, den angepassten Kanal für IBM MQ verwenden, um Daten auf die gleiche Weise zu senden und zu empfangen, wie dies bei der Verwendung von integrierten Transporten geschieht, die von Microsoft bereitgestellt werden. Dies ermöglicht die einfache Integration in die asynchronen, skalierbaren und zuverlässigen Funktionen zur Nachrichtenübermittlung von IBM MQ. Eine vollständige Liste der unterstützten Funktionen finden Sie unter „Funktionen und Leistungsmerkmale des angepassten WCF-Kanals" auf Seite 1332.

Wann und warum wird der angepasste IBM MQ-Kanal für WCF verwendet?

Über den angepassten IBM MQ-Kanal können Sie Nachrichten zwischen WCF-Clients und -Services auf dieselbe Weise senden und empfangen wie über die von Microsoft bereitgestellten integrierten Transportprotokolle. Anwendungen erhalten somit Zugriff auf die Funktionen von IBM MQ innerhalb des einheitlichen WCF-Programmiermodells.

Ein typisches Verwendungsmusterszenario für den angepassten IBM MQ-Kanal für WCF ist eine Nicht-SOAP-Schnittstelle für die Übertragung nativer IBM MQ-Nachrichten.

Mit dem Nicht-SOAP-/Nicht-JMS-Nachrichtenformat (reinen MQMessage-Format) übertragene Nachrichten

Wenn Sie den angepassten IBM MQ-Kanal für WCF als Nicht-SOAP-Schnittstelle für die Übertragung nativer IBM MQ-Nachrichten verwenden, werden die Nachrichten mit dem Nicht-SOAP-/Nicht-JMS-Nachrichtenformat (reines MQMessage-Format) von IBM MQ übertragen.

WCF-Benutzer können den Service starten, mit anderen Worten, die Servicebenutzer können über MQMessages eine Nachricht an eine IBM MQ-Warteschlange senden. Die MQMD-Felder und -Nutzdaten können von Anwendungen abgerufen und festgelegt werden. Wenn die Nachricht in IBM MQ -Warteschlangen verfügbar ist, kann sie von jedem WCF-Service oder von Nicht-WCF-Anwendungen wie C-oder Java -Anwendungen verarbeitet werden, die unter AIX, Linux, Windowsoder z/OSausgeführt werden.

Softwarevoraussetzungen für den angepassten IBM MQ-Kanal für WCF

In diesem Abschnitt werden die Softwarevoraussetzungen für den angepassten IBM MQ-Kanal für WCF zusammengefasst. Der angepasste IBM MQ-Kanal für WCF kann nur Verbindungen zu Warteschlangenmanagern von IBM WebSphere MQ 7.0 oder höher herstellen.

Laufzeitumgebungsanforderungen

- Microsoft.NET Framework v4.7.2 oder höher muss auf der Hostmaschine installiert sein.
- *Java and .NET Messaging and Web Services* wird standardmäßig als Teil des IBM MQ-Installationsprogramms installiert. Diese Komponente installiert die für den angepassten Kanal erforderlichen .NET-Assemblys im Global Assembly Cache.

Anmerkung: Wenn Microsoft .NET Framework V4.7.2 oder höher nicht vor der Installation von IBM MQinstalliert wird, wird die IBM MQ -Produktinstallation ohne Fehler fortgesetzt, aber IBM MQ classes for .NET ist nicht verfügbar. Wenn das .NET Framework installiert wird, nachdem Sie IBM MQ installiert haben, müssen die IBM MQ.NET-Assemblys durch Ausführung des Scripts *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` registriert werden, wobei *WMQ-Installationsverzeichnis* für das Verzeichnis steht, in dem IBM MQ installiert ist. Mit diesem Script werden die erforderlichen Assemblys im Global Assembly Cache (GAC) installiert. Im Verzeichnis %TEMP% wird eine Gruppe von Dateien `amqi*.log` erstellt, in denen die ausgeführten Aktionen dokumentiert werden. Es ist nicht erforderlich, das Script `amqiRegisterdotNet.cmd` erneut auszuführen, wenn für .NET ein Upgrade auf V4.7.2 oder höher von einer früheren Version durchgeführt wird, beispielsweise von .NET V3.5.

Entwicklungsumgebungsanforderungen

- Microsoft Visual Studio 2015 oder Windows Software Development Kit für .NET 4.7.2 oder höher.
- Microsoft.NET Framework V4.7.2 oder höher muss auf der Hostmaschine installiert sein, damit die Beispiellösungsdateien erstellt werden.

Angepasster IBM MQ-Kanal für WCF: Welche Komponenten sind installiert?

Beim angepassten Kanal für IBM MQ handelt es sich um einen Transportkanal, der das einheitliche Programmiermodell von Microsoft Windows Communication Foundation (WCF) verwendet. Der angepasste Kanal wird standardmäßig im Rahmen der Installation installiert.

Angepasster IBM MQ-Kanal für WCF

Der angepasste Kanal und die zugehörigen Abhängigkeiten sind in der Komponente `Java and .NET Messaging and Web Services` enthalten, die standardmäßig installiert ist. Bei einem von einer früheren IBM MQ-Version als IBM MQ 8.0 ausgehenden Upgrade wird der angepasste IBM MQ-Kanal für WCF

standardmäßig installiert, wenn die Komponente `Java and .NET Messaging and Web Services` zuvor auch in einer früheren Installation installiert war.

Die Komponente `.NET Messaging and Web Services` enthält die Datei `IBM.XMS.WCF.dll` und die Datei `IBM.WMQ.WCF.dll`. Bei diesen Dateien handelt es sich um die wichtigste Assembly des angepassten Kanals, in der die Klassen der WCF-Schnittstelle enthalten sind. Diese Dateien sind im Global Assembly Cache (GAC) installiert und außerdem im Verzeichnis *MQ_INSTALLATION_PATH*\`bin` verfügbar. Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem IBM MQ installiert ist.

In der folgenden Tabelle werden die wichtigsten Klassen zusammengefasst, die zur Verwendung des angepassten Kanals erforderlich sind.

Tabelle 189. Wichtige Klassen, die zur Verwendung des angepassten Kanals erforderlich sind

	SOAP/JMS-Schnittstelle (Vorhanden)	**Andere Schnittstellen als SOAP/JMS-Schnittstelle (ab IBM MQ 8.0)**
Assembly für den angepassten Kanal	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
Name der Transportbindung	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Importkomponente der Transportbindung	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Konfiguration der Transportbindung	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Samples(Oneway)	SimpleOneWay_Client, SimpleOneWay_Service	MQMessaging_OneWay_Client,MQMessaging_OneWay_Service
Samples(RequestReply)	SimpleRequestReply_Client, SimpleRequestReply_Service	MQMessaging_RequestReply_Client,MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll unterstützt sowohl SOAP/JMS-Schnittstellen als auch andere Schnittstellen als SOAP/JMS. Wenn neue Anwendungen entwickelt werden, sollten sie die Assembly IBM.WMQ.WCF-Assembly verwenden, da sie beide Schnittstellen unterstützt.

MQSTR-formatierte Nachrichten senden

Wenn die Anforderungsnachricht den Typ MQSTR hat, können Sie auswählen, dass die Antwortnachricht im MQSTR-Format gesendet werden soll.

Um das Format der Antwortnachricht zu ändern, müssen Sie den zusätzlichen URI-Parameter **`replyMessageFormat`** verwenden. Folgende Werte werden unterstützt:

""

" " ist der Standardwert.

Die Antwortnachricht hat das Byte-Format (MQMFT_NONE). For example:

```
"jms:/queue?destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueMana
ger(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageFor
mat= "
```

MQSTR

Die Antwortnachricht hat das MQSTR-Format (MQMFT_STRING). For example:

```
"jms:/queue?destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueMana
ger(QM1)
```

```
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageFor
mat=MQSTR"
```

Anmerkungen:

1. Der Wert für **`replyMessageFormat`** ist von der Groß-/Kleinschreibung unabhängig.
2. Wird ein anderer Wert als " " oder *MQSTR* verwendet, verursacht dies eine Ausnahmebedingung wegen eines ungültigen Parameterwerts.

Beispiele für den angepassten IBM MQ-Kanal

In den Beispielen wird eine einfache Verwendung des angepassten IBM MQ-Kanals für WCF gezeigt. Die Beispiele und die zugehörigen Dateien befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf`. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ. Weitere Informationen zum Beispiel für den angepassten IBM MQ-Kanal finden Sie unter „WCF-Beispiele verwenden” auf Seite 1352.

svcutil.exe.config

Die Datei `svcutil.exe.config` ist ein Beispiel für die Konfigurationseinstellungen, die zur Aktivierung des Microsoft-WCF-Generierungstools `svcutil` für den Client-Proxy zur Erkennung des angepassten Kanals erforderlich sind. Die Datei `svcutil.exe.config` befindet sich im Verzeichnis *MQ_INSTALLATION_PATH*`\tools\wcf\docs\examples\`. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ. Weitere Informationen zur Verwendung der Datei `svcutil.exe.config` finden Sie unter „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit Metadaten aus einem aktiven Service generieren” auf Seite 1349.

WCF-Architektur

Der angepasste IBM MQ-Kanal wird am Anfang der API `IBM Message Service Client for .NET (XMS .NET)` eingeschlossen.

SOAP/JMS-Schnittstelle

Die WCF-Architektur wird im folgenden Diagramm gezeigt:

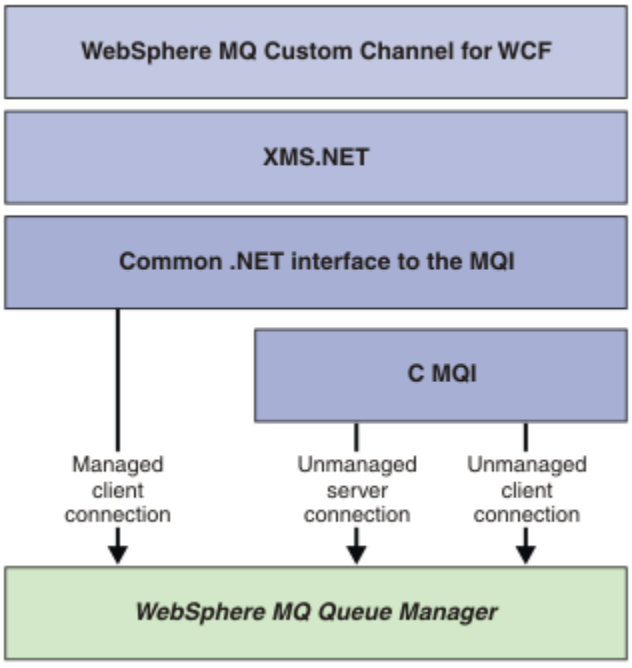


Abbildung 149. WCF-Architektur für die SOAP-/JMS-Schnittstelle

Bei der Produktinstallation werden standardmäßig alle erforderlichen Komponenten installiert.

Es sind drei Verbindungstypen verfügbar:

- Verwaltete Clientverbindungen
- Nicht verwaltete Serververbindungen
- Nicht verwaltete Clientverbindungen

Weitere Informationen zu diesen Verbindungen finden Sie unter „Optionen für WCF-Verbindungen” auf Seite 1338.

Andere Schnittstelle als SOAP-/JMS-Schnittstellen

Der angepasste IBM MQ-Kanal für WCF unterstützt die SOAP-/JMS-Schnittstelle (verfügbar ab IBM WebSphere MQ 7.0.1) und andere Schnittstellen als die SOAP-/JMS-Schnittstelle.

Die WCF-Architektur wird im folgenden Diagramm gezeigt:

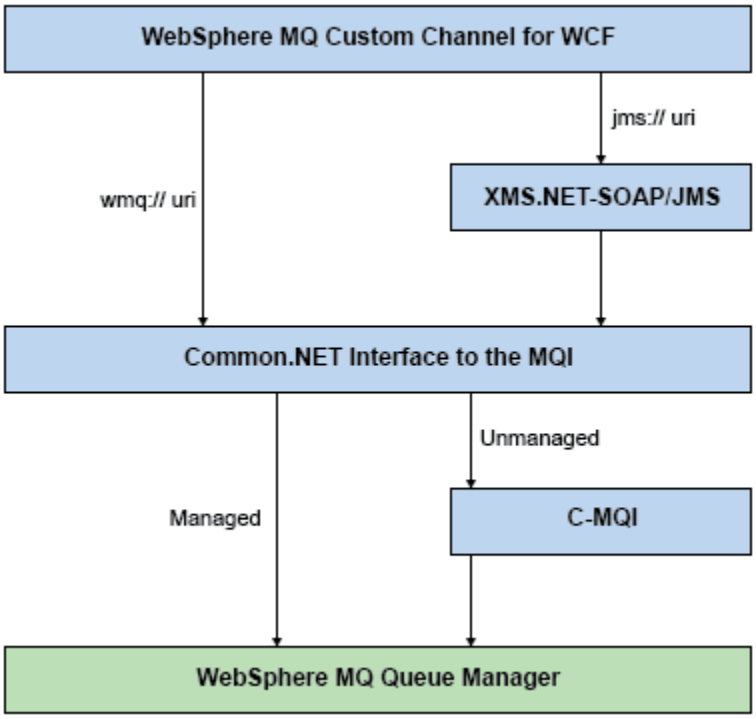


Abbildung 150. WCF-Architektur für eine andere Schnittstelle als eine SOAP-/JMS-Schnittstelle

Angepasste IBM MQ-Kanäle für WCF verwenden

Übersicht über die Informationen, die für Programmierer verfügbar sind, die die angepassten IBM MQ-Kanäle für Windows Communication Foundation (WCF) verwenden.

Die Microsoft Windows Communication Foundation unterstützt die Web-Services und die Messaging-Unterstützung in Microsoft.NET Framework 3. IBM MQ kann als angepasster Kanal innerhalb von WCF in .NET Framework 3 auf dieselbe Weise wie die integrierten Kanäle von Microsoftverwendet werden.

Nachrichten, die über den angepassten Kanal transportiert werden, werden gemäß der SOAP over JMS -Implementierung von IBM MQformatiert. Anwendungen können anschließend mit Services kommunizieren, die von WCF oder von der Serviceinfrastruktur von WebSphere SOAP over JMS gehostet werden.

Funktionen und Leistungsmerkmale des angepassten WCF-Kanals

In den folgenden Abschnitten finden Sie Informationen zu den Funktionen und Leistungsmerkmalen des angepassten WCF-Kanals.

Formen des angepassten WCF-Kanals

Übersicht über die Formen des angepassten Kanals, die von IBM MQ wie die angepassten Kanäle von Microsoft Windows Communication Foundation (WCF) verwendet werden können.

Der angepasste IBM MQ-Kanal für WCF unterstützt zwei Kanalformen:

- Unidirektional
- Anforderung/Antwort

Die Kanalform wird von WCF automatisch gemäß dem gehosteten Servicevertrag ausgewählt.

Verträge mit Methoden, die nur den Service **IsOneWay** verwenden, werden von einem Einwegkanal bedient; Beispiel:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

Verträge, die eine Kombination aus Einweg- und Anforderungs-/Antwortmethoden einschließen, werden von der Kanalform für Anforderung/Antwort bedient. For example:

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

Anmerkung: Bei einer Kombination aus Einweg- und Anforderungs-/Antwortmethoden im gleichen Vertrag müssen Sie sicherstellen, dass das Verhalten wie beabsichtigt ist, insbesondere bei der Arbeit in einer heterogenen Umgebung, da Einwegmethoden warten, bis sie eine leere Antwort vom Service empfangen.

Einwegkanal

Der angepasste IBM MQ-Einwegkanal für WCF wird beispielsweise zum Senden von Nachrichten aus einem WCF-Client mithilfe eines Kanals in Form eines Einwegkanals verwendet. Der Kanal kann Nachrichten nur in eine Richtung senden, beispielsweise aus einem Warteschlangenmanager des Clients an eine Warteschlange in einem WCF-Service.

Anforderungs-/Antwortkanal

Der angepasste IBM MQ-Anforderungs-/Antwortkanal für WCF wird beispielsweise zum asynchronen Senden von Nachrichten in zwei Richtungen verwendet. Für die asynchrone Nachrichtenübermittlung muss die gleiche Clientinstanz verwendet werden. Der Kanal kann Nachrichten in eine Richtung senden, beispielsweise aus dem Warteschlangenmanager eines Clients an eine Warteschlange in einem WCF-Service, und anschließend eine Antwortnachricht aus WCF an eine Warteschlange im Warteschlangenmanager des Clients senden.

Parameternamen und Werte der WCF-URI

Parameternamen und Werte für die URI der SOAP-/JMS-Schnittstelle und für andere Schnittstellen als die SOAP-/JMS-Schnittstelle.

SOAP/JMS-Schnittstelle

connectionFactory
Der Parameter 'connectionFactory' ist erforderlich.

initialContextFactory
Der Parameter 'initialContextFactory' ist erforderlich und muss auf 'com.ibm.mq.jms.Nojndi' gesetzt sein, damit die Kompatibilität mit WebSphere Application Server und anderen Produkten gewährleistet ist.

Andere Schnittstellen als die SOAP-/JMS-Schnittstelle

Das URI-Format entspricht dem Format für die MA93-Spezifikationen. Weitere Einzelheiten zu den IBM MQ IRI-Spezifikationen finden Sie im SupportPac - MA93.

Syntax der IBM MQ-URI:

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

Beispiel für die IBM MQ-IRI:
Die folgende Beispiel-IRI informiert einen Serviceanforderer darüber, dass eine Verbindung von einem IBM MQ TCP-Client zu einem System mit der Bezeichnung 'example.com' auf Port 1414 verwendet werden kann und persistente Anforderungsnachrichten in eine Warteschlange mit der Bezeichnung SampleQ auf Warteschlangenmanager QM1 eingereiht werden. Die IRI gibt an, dass der Service-Provider Antworten in eine Warteschlange mit der Bezeichnung SampleReplyQ einreiht.

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

Für TLS-fähige Verbindungen:
Für gesicherte Verbindungen (TLS) über den WCF-Client/-Service legen Sie die folgenden Eigenschaften mit den geeigneten Werte in der URI fest. Alle Eigenschaften, die mit dem Präfix "*" versehen sind, sind für eine sichere Verbindung obligatorisch.

- **sslKeyRepository**: *SYSTEM oder *USER
- * **sslCipherSpec**: eine gültig CipherSpec, z. B. TLS_RSA_WITH_AES_128_CBC_SHA256.
- **sslCertRevocationCheck**: 'true' oder 'false'.
- **sslKeyResetCount**: ein Wert größer als 32 KB.
- **sslPeerName:** der definierte Name des Serverzertifikats

For example:

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

Zuverlässige Nachrichtenübermittlung mit dem angepassten WCF-Kanal

Die zuverlässige Nachrichtenübermittlung garantiert, dass eine Serviceanforderung oder -antwort ausgeführt wird und nicht verloren geht.

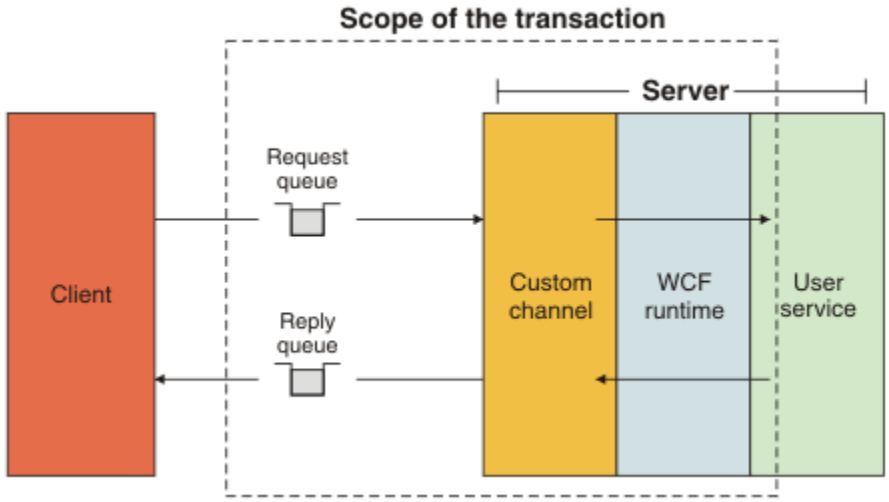
Eine Anforderungsnachricht kann unter einem Synchronisationspunkt für eine lokale Transaktion empfangen werden und von dort können alle Antwortnachrichten gesendet werden. Diese Nachrichten können im Falle eines Laufzeitfehlers rückgängig gemacht werden. Diese Fehler sind beispielsweise eine nicht

behandelte Ausnahme, die vom Service ausgelöst wird, ein Fehler beim Senden der Nachricht an den Service oder ein Fehler bei der Übergabe der Antwortnachricht.

`AssuredDelivery` ist das Attribut für die zuverlässige Nachrichtenübermittlung, das in einem Servicevertrag angegeben werden kann, um zu garantieren, dass alle Anforderungsnachrichten, die von einem Service empfangen werden, und alle Antwortnachrichten, die von einem Service gesendet werden, im Falle eines Fehlers während der Laufzeit nicht verloren gehen.

Um sicherzustellen, dass Nachrichten auch im Falle eines Systemausfalls oder Stromausfalls beibehalten werden, müssen sie als persistente Nachrichten gesendet werden. Zur Verwendung von persistenten Nachrichten muss diese Option in der Endpunkt-URI der Clientanwendung angegeben sein.

Die dezentrale Transaktionsverarbeitung wird nicht unterstützt und der Bereich der Transaktion reicht nicht über die Verarbeitung von Anforderungs- und Antwortnachrichten hinaus, die von IBM MQ ausgeführt werden. Alle innerhalb des Service ausgeführten Operationen werden als Ergebnis des Fehlers möglicherweise erneut ausgeführt, wodurch die Nachricht erneut empfangen wird. Im folgenden Diagramm wird der Bereich der Transaktion gezeigt:



Die zuverlässige Nachrichtenübermittlung wird aktiviert, indem das Attribut `AssuredDelivery` wie im folgenden Beispiel gezeigt für die Serviceklasse angewendet wird:

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

Bei der Verwendung des Attributs `AssuredDelivery` müssen Sie die folgenden Punkte beachten:

- Wenn ein Kanal ermittelt, dass ein Fehler wahrscheinlich erneut auftritt, falls eine Nachricht rückgängig gemacht und erneut empfangen wurde, wird die Nachricht als falsch formatierte Nachricht behandelt und nicht zur erneuten Verarbeitung an die Anforderungswarteschlange zurückgegeben. Beispiel: Die empfangene Nachricht ist nicht ordnungsgemäß formatiert oder kann einem Service nicht zugeteilt werden. Nicht behandelte Ausnahmen, die von einer Serviceoperation ausgelöst wurden, werden immer erneut gesendet, bis die Nachricht mit der maximalen Häufigkeit erneut übergeben wurde, die in der Eigenschaft für den Grenzwert für die Zurücksetzung der Anforderungswarteschlange angegeben wurde. Weitere Informationen finden Sie in „Falsch formatierte Nachrichten im angepassten WCF-Kanal” auf Seite 1335.
- Der Kanal führt das Lesen, Verarbeiten und Beantworten jeder Anforderungsnachricht als atomare Operation aus, in der die Transaktionsintegrität mithilfe eines Einzelthreads für die Ausführung erzwungen wird. Damit Serviceoperationen gleichzeitig ausgeführt werden können, aktiviert der Kanal den WCF-

Service, mit dem mehrere Instanzen des Kanals erstellt werden. Die Anzahl der Kanalinstanzen, die für die Verarbeitung von Anforderungen verfügbar ist, wird durch die Bindungseigenschaft `MaxConcurrentCalls` gesteuert. Weitere Informationen finden Sie in „Optionen zur WCF-Bindungskonfiguration" auf Seite 1344.

- Bei der Funktion für die zuverlässige Nachrichtenübermittlung werden die WCF-Erweiterbarkeitspunkte 'IOperationInvoker' und 'IErrorHandler' verwendet. Wenn diese Erweiterbarkeitspunkte extern von einer Anwendung verwendet werden, muss die Anwendung sicherstellen, dass alle zuvor registrierten Erweiterbarkeitspunkte aufgerufen werden. Wenn dies für 'IErrorHandler' nicht geschieht, werden Fehler möglicherweise nicht gemeldet. Wenn dies für 'IOperationInvoker' nicht geschieht, antwortet der WCF-Service möglicherweise nicht mehr.

Sicherheit für den angepassten WCF-Kanal

Der angepasste IBM MQ-Kanal für WCF unterstützt die Verwendung von TLS nur für nicht verwaltete Clientverbindungen zum Warteschlangenmanager.

Geben Sie TLS über einen Eintrag in der Clientkanaldefinitionstabelle (CCDT) an. Weitere Informationen zu Definitionstabellen für Clientkanäle finden Sie unter Definitionstabelle für den Clientkanal.

Definitionstabellen für den Clientkanal (CCDT) in WCF

Der angepasste IBM MQ-Kanal für WCF unterstützt die Verwendung von Definitionstabellen für den Clientkanal (Client Channel Definition Tables, CCDT), um die Verbindungsinformationen für Clientverbindungen zu konfigurieren.

CCDTs werden über diese beiden Umgebungsvariablen gesteuert:

- *MQCHLLIB* gibt das Verzeichnis an, in dem sich die Tabelle befindet.
- *MQCHLTAB* gibt den Dateinamen der Tabelle an.

Wenn diese Umgebungsvariablen definiert sind, haben Sie Priorität vor allen Clientverbindungsdetails, die in der URI angegeben sind.

Weitere Informationen zu den Definitionstabellen für den Clientkanal finden Sie unter Definitionstabelle für den Clientkanal.

Falsch formatierte Nachrichten im angepassten WCF-Kanal

Wenn ein Service eine Anforderungsnachricht nicht verarbeiten kann oder die Übergabe einer Antwortnachricht an eine Antwortwarteschlange fehlschlägt, wird die Nachricht als falsch formatierte Nachricht behandelt.

Falsch formatierte Anforderungsnachrichten

Wenn eine Anforderungsnachricht nicht verarbeitet werden kann, wird sie als falsch formatierte Nachricht behandelt. Diese Aktion hindert den Service daran, die gleiche nicht verarbeitbare Nachricht erneut zu empfangen. Damit eine nicht verarbeitbare Anforderungsnachricht als falsch formatierte Nachricht behandelt werden kann, muss eine der folgenden Situationen zutreffen:

- Der Zurücksetzungszähler für Nachrichten hat den in der Anforderungswarteschlange angegebenen Rücksetzschwellenwert überschritten. Dies geschieht nur, wenn für den Service die zuverlässige Nachrichtenübermittlung festgelegt wurde. Weitere Informationen zur zuverlässigen Nachrichtenübermittlung finden Sie unter „Zuverlässige Nachrichtenübermittlung mit dem angepassten WCF-Kanal" auf Seite 1333
- Die Nachricht wurde nicht ordnungsgemäß formatiert und konnte nicht als SOAP over JMS-Nachricht interpretiert werden.

Falsch formatierte Antwortnachrichten

Wenn ein Service eine Antwortnachricht nicht an die Antwortwarteschlange übergeben kann, wird die Antwortnachricht als falsch formatierte Nachricht behandelt. Durch diese Aktion können Antwortnachrichten zur Unterstützung der Problembestimmung später abgerufen werden.

Verarbeitung von falsch formatierten Nachrichten

Die Aktion, die für falsch formatierte Nachrichten ausgeführt werden muss, ist von der Warteschlangenmanagerkonfiguration und den Werten abhängig, die in den Berichtsoptionen der Nachricht festgelegt sind. Die folgenden Berichtsoptionen sind für SOAP over JMS in Anforderungsnachrichten standardmäßig festgelegt und können nicht konfiguriert werden:

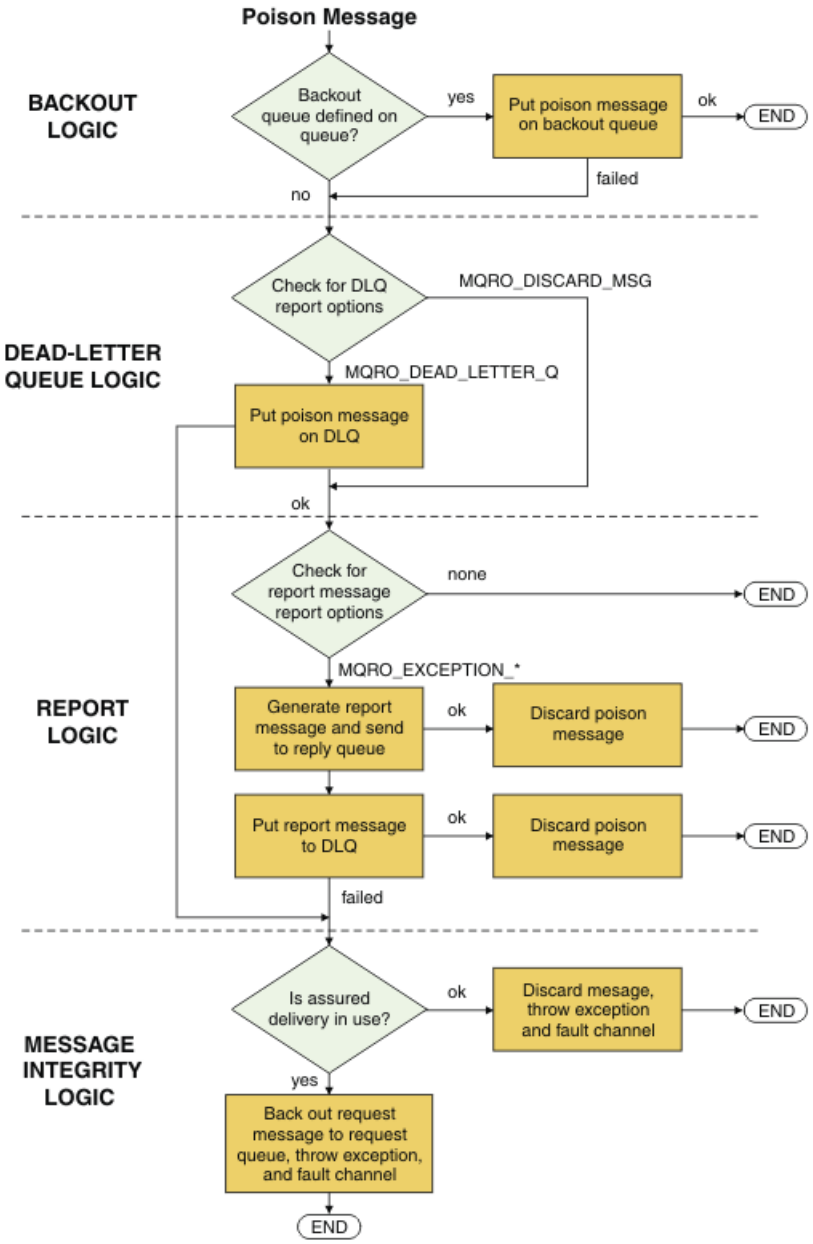
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

Die folgende Berichtsoption ist für SOAP over JMS in den Antwortnachrichten standardmäßig festgelegt und kann nicht konfiguriert werden:

- MQRO_DEAD_LETTER_Q

Wenn Nachrichten aus einer anderen Quelle als der WCF-Quelle empfangen werden, lesen Sie die Dokumentation zu dieser Quelle.

Im folgenden Diagramm werden die möglichen Aktionen und Schritte gezeigt, die beim Fehlschlagen der Behandlung nicht verarbeitbarer Nachrichten ausgeführt werden:



IBM MQ-Nachrichtenfunktionen für WCF-Anwendungen

Nicht-SOAP/Nicht-JMS-Nachrichtenfunktionen (d. h. IBM MQ-Nachrichtenfunktionen) für WCF-Anwendungen.

Für die Nicht-SOAP-/Nicht-JMS-Schnittstelle gibt es folgende IBM MQ-Nachrichtenfunktionen für WCF-Anwendungen:

- WCF-Anwendungen können die IBM MQ-Basisnachrichten senden und empfangen, die von jeder IBM MQ-Anwendung verarbeitet werden können.
- WCF-Anwendungen haben uneingeschränkten Zugriff zur Aktualisierung des MQMD und der Nutzdaten.
- Der WCF-Client kann IBM MQ-Nachrichten senden, die von allen IBM MQ-Clients gelesen werden können, also beispielsweise von C-, Java-, JMS- und .NET-Clients.

Die WCF für Nicht-SOAP/Nicht-JMS-Schnittstelle muss folgende Klassen verwenden, um die Nachrichtennutzdaten und den MQMD für die Nachricht festzulegen:

- WmqStringMessage für Nutzdaten vom Typ Zeichenfolge (String)
- WmqBytesMessage für Nutzdaten vom Type Bytes

- WmqXmlMessage für Nutzdaten vom Typ XML

Verwenden Sie zur Festlegung der Nachrichtennutzdaten die Eigenschaft **Data** je nach Nutzdatentyp für die WmqStringMessage-, WmqBytesMessage- bzw. WmqXmlMessage-Klasse. Verwenden Sie zum Beispiel den folgenden Code, um Nutzdaten vom Typ Zeichenfolge festzulegen:

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

Optionen für WCF-Verbindungen

Ein angepasster IBM MQ-Kanal für WCF kann auf drei Arten mit einem Warteschlangenmanager verbunden werden. Prüfen Sie, welcher Verbindungstyp für Ihre Anforderungen am besten geeignet ist.

Weitere Informationen zu Verbindungsoptionen finden Sie unter „Unterschiede bei der Verbindung" auf Seite 608

Weitere Informationen zur WCF-Architektur finden Sie unter „WCF-Architektur" auf Seite 1330

Nicht verwaltete Clientverbindung

Wenn in diesem Modus eine Verbindung hergestellt wird, wird ein IBM MQ-Client mit einem IBM MQ-Server verbunden, der auf einem lokalen System oder einem fernen System ausgeführt wird.

Wenn Sie den angepassten IBM MQ-Kanal für WCF als IBM MQ-Client verwenden möchten, können Sie ihn mit dem IBM MQ MQI client auf dem IBM MQ-Server oder auf einem separaten System installieren.

Nicht verwaltete Serververbindung

Im Serverbindungsmodus kommuniziert der angepasste IBM MQ-Kanal für WCF nicht über ein Netz, sondern verwendet die Warteschlangenmanager-API. Durch die Verwendung von Verbindungen über Bindungen wird eine bessere Leistung für IBM MQ-Anwendungen bereitgestellt als bei der Verwendung von Netzverbindungen.

Zum Verwenden von Verbindungen über Bindungen müssen Sie den angepassten IBM MQ-Kanal für WCF auf dem IBM MQ-Server installieren.

Verwaltete Clientverbindung

Wenn in diesem Modus eine Verbindung hergestellt wird, wird ein IBM MQ-Client mit einem IBM MQ-Server verbunden, der auf einem lokalen System oder einem fernen System ausgeführt wird.

Die Klassen des angepassten IBM MQ-Kanals für .NET 3, für die in diesem Modus eine Verbindung hergestellt wird, verbleiben im verwalteten .NET-Code und geben keine Aufrufe an native Services aus. Weitere Informationen zu verwaltetem Code finden Sie in der Microsoft-Dokumentation.

Die Nutzung des verwalteten Clients unterliegt einer Reihe von Einschränkungen. Weitere Informationen zu diesen Einschränkungen finden Sie unter „Verwaltete Clientverbindungen" auf Seite 608.

Angepassten IBM MQ-Kanal für WCF erstellen und konfigurieren

Die angepassten IBM MQ-Kanäle für WCF funktionieren auf die gleiche Weise wie Transport-WCF-Kanäle, die von Microsoft angeboten werden. Der angepasste IBM MQ-Kanal für WCF kann auf zwei Arten erstellt werden.

Informationen zu diesem Vorgang

Der angepasste IBM MQ-Kanal wird als WCF-Transportkanal in WCF integriert und muss als solcher mit einem Nachrichtenencoder und optionalen Protokollkanälen paarweise verbunden werden, damit ein voll-

ständiger Kanal-Stack zur Verwendung durch eine Anwendung erstellt werden kann. Für die erfolgreiche Erstellung eines vollständigen Kanal-Stacks sind zwei Elemente erforderlich:

1. Eine Bindungsdefinition: Sie gibt an, welche Elemente zum Erstellen des Kanal-Stacks für die Anwendung erforderlich sind, einschließlich Transportkanal, Nachrichtenencoder und allen Protokollen und zusätzlich allen allgemeinen Konfigurationseinstellungen. Die Bindungsdefinition muss für den angepassten Kanal in Form einer angepassten WCF-Bindung erstellt werden.
2. Eine Endpunktdefinition: Sie verknüpft den Servicevertrag mit der Bindungsdefinition und stellt außerdem die tatsächliche Verbindungs-URI bereit, mit der angegeben wird, an welcher Stelle die Anwendung eine Verbindung herstellen kann. Die URI liegt für den angepassten Kanal in Form einer SOAP over JMS-URI vor.

Diese Definitionen können auf zwei unterschiedliche Arten erstellt werden:

- Administrativ: Die Definitionen werden erstellt, indem die Details in einer Anwendungskonfigurationsdatei (z. B. `app.config`) bereitgestellt werden.
- Programmgesteuert: Die Definitionen werden direkt aus dem Anwendungscode erstellt.

Die Auswahl der Methode, die zum Erstellen der Definitionen verwendet werden soll, muss auf den Anforderungen der Anwendung basieren. Berücksichtigen Sie dabei Folgendes:

- Die administrative Konfigurationsmethode bietet Ihnen die Flexibilität, die Details des Service und des Clients nach der Bereitstellung ändern zu können, ohne die Anwendung erneut erstellen zu müssen.
- Die programmgesteuerte Konfigurationsmethode bietet Ihnen einen größeren Schutz vor Konfigurationsfehlern und die Möglichkeit, eine Konfiguration während der Ausführung dynamisch zu generieren.

Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen in einer Anwendungskonfigurationsdatei administrativ erstellen

Beim angepassten IBM MQ-Kanal für WCF handelt es sich um einen WCF-Kanal auf Transportebene. Zur Verwendung des angepassten Kanals müssen ein Endpunkt und eine Bindung definiert werden. Diese Definitionen können durch die Bereitstellung der Bindungs- und Endpunktinformationen in einer Anwendungskonfigurationsdatei bereitgestellt werden.

Zur Konfiguration und Verwendung des angepassten IBM MQ-Kanals für WCF, bei dem es sich um einen WCF-Kanal auf Transportebene handelt, müssen eine Bindung und ein Endpunkt definiert sein. Die Bindung enthält die Konfigurationsinformationen für den Kanal und die Endpunktdefinition enthält die Verbindungsdetails. Diese Definitionen können auf zwei Arten erstellt werden:

- Programmgesteuert, also direkt aus dem Anwendungscode (siehe „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen programmgesteuert erstellen” auf Seite 1341).
- Administrativ durch das Bereitstellen der Details in einer Anwendungskonfigurationsdatei, wie in der folgenden Prozedur beschrieben.

Die Anwendungskonfigurationsdatei für den Client oder Service hat allgemein die Bezeichnung *yourappname*`.exe.config`. Dabei steht *Anwendungsname* für den Namen Ihrer Anwendung. Die Anwendungskonfigurationsdatei wird am einfachsten durch Verwendung des Microsoft-Tools `SvcConfigEditor.exe` für den Servicekonfigurationseditor geändert. Gehen Sie dazu folgendermaßen vor:

- Starten Sie das Tool `SvcConfigEditor.exe` für den Konfigurationseditor. Die Standardinstallationsposition des Tools lautet *`Drive:`*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe`. Hierbei steht *Laufwerk:* für den Namen des Installationslaufwerks.

Schritt 1: Hinzufügen einer Erweiterung des Bindungselements, um die Suche des angepassten Kanals durch WCF zu aktivieren

1. Klicken Sie zum Öffnen des Menüs mit der rechten Maustaste auf **Advanced** > **Extension** > **binding element** (Erweitert > Erweiterung > Bindungselement) und wählen Sie **New** (Neu) aus.
2. Füllen Sie die Felder wie in dieser Tabelle gezeigt aus:

Tabelle 190. Felder für neue Bindungselemente	
Feld	**Wert**
Name	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`
Typ	Navigieren Sie im Global Assembly Cache (GAC) zu `IBM.XMS.WCF.dll` und wählen Sie `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig` aus.

Schritt 2: Erstellen einer angepassten Bindungsdefinition, mit der der angepasste Kanal mit einem WCF-Nachrichten-Encoder kombiniert wird

1. Klicken Sie zum Öffnen des Menüs mit der rechten Maustaste auf **Bindings** (Bindungen) und wählen Sie **New Binding Configuration** (Neue Bindungskonfiguration) aus.
2. Füllen Sie die Felder wie in dieser Tabelle gezeigt aus:

Tabelle 191. Felder für neue Bindungskonfigurationen	
Feld	**Wert**
Name	`CustomBinding_WMQ`
Bindungselement 1	`textMessageEncoding (MessageVersion: Soap11)`
Bindungselement 2	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

Schritt 3: Angabe der Bindungseigenschaften

1. Wählen Sie die Transportbindung *IBM.XMS.WCF.SoapJmsIbmTransportChannel* aus der Bindung aus, die Sie in „Schritt 2: Erstellen einer angepassten Bindungsdefinition, mit der der angepasste Kanal mit einem WCF-Nachrichten-Encoder kombiniert wird" auf Seite 1340 erstellt haben.
2. Nehmen Sie alle erforderlichen Änderungen an den Standardwerten der Eigenschaften wie in „Optionen zur WCF-Bindungskonfiguration" auf Seite 1344 beschrieben vor.

Schritt 4: Erstellen einer Endpunktdefinition

Erstellen Sie eine Endpunktdefinition, die auf die in „Schritt 2: Erstellen einer angepassten Bindungsdefinition, mit der der angepasste Kanal mit einem WCF-Nachrichten-Encoder kombiniert wird" auf Seite 1340 erstellte angepasste Bindung verweist, und stellen Sie die Verbindungsdetails des Service bereit. Wie diese Informationen angegeben werden, hängt davon ab, ob die Definition für eine Clientanwendung oder eine Serviceanwendung gilt.

Fügen Sie für eine Clientanwendung dem Client-Abschnitt folgendermaßen eine Endpunktdefinition hinzu:

1. Klicken Sie zum Öffnen des Menüs mit der rechten Maustaste auf **Client** > **Endpoints** (Client > Endpunkte) und wählen Sie **New Client Endpoint** (Neuer Clientendpunkt) aus.
2. Füllen Sie die Felder wie in dieser Tabelle gezeigt aus:

Tabelle 192. Felder für neue Clientendpunkte	
Feld	**Wert**
Name	`Endpoint_WMQ`
Adresse	*Die SOAP/JMS-URI, mit der die WMQ-Verbindungsdetails beschrieben werden, die für den Zugriff auf den Service erforderlich sind.* Weitere Einzelheiten finden Sie unter „Angepasste IBM MQ-Kanäle für das Adressformat der URI von WCF-Endpunkten" auf Seite 1343

Tabelle 192. Felder für neue Clientendpunkte (Forts.)	
Feld	**Wert**
Bindung	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Vertrag	*Name der Servicevertragsschnittstelle*

Fügen Sie für eine Serviceanwendung dem Services-Abschnitt folgendermaßen eine Servicedefinition hinzu:

1. Klicken Sie zum Öffnen des Menüs mit der rechten Maustaste auf **Services** und wählen Sie **New Service** (Neuer Service) und anschließend die Serviceklasse, die gehostet werden soll, aus.
2. Fügen Sie für Ihren neuen Service dem Abschnitt **Endpunkte** eine Endpunktdefinition hinzu und füllen Sie die Felder wie in dieser Tabelle gezeigt aus:

Tabelle 193. Felder für neue Serviceendpunkte	
Feld	**Wert**
Name	`Endpoint_WMQ`
Adresse	*Die SOAP/JMS-URI, mit der die WMQ-Verbindungsdetails beschrieben werden, die für den Zugriff auf den Service erforderlich sind.* Weitere Einzelheiten finden Sie unter „Angepasste IBM MQ-Kanäle für das Adressformat der URI von WCF-Endpunkten” auf Seite 1343
Bindung	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Vertrag	*Name der Serviceimplementierungsklasse*

Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen programmgesteuert erstellen

Beim angepassten IBM MQ-Kanal für WCF handelt es sich um einen WCF-Kanal auf Transportebene. Zur Verwendung des angepassten Kanals müssen ein Endpunkt und eine Bindung definiert werden und diese Definitionen können programmgesteuert direkt aus dem Anwendungscode vorgenommen werden.

Zur Konfiguration und Verwendung des angepassten IBM MQ-Kanals für WCF, bei dem es sich um einen WCF-Kanal auf Transportebene handelt, müssen eine Bindung und ein Endpunkt definiert sein. Die Bindung enthält die Konfigurationsinformationen für den Kanal und die Endpunktdefinition enthält die Verbindungsdetails. Weitere Informationen finden Sie unter „WCF-Beispiele verwenden” auf Seite 1352.

Diese Definitionen können auf zwei Arten erstellt werden:

- Administrativ durch das Bereitstellen der Details in einer Anwendungskonfigurationsdatei (siehe „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen in einer Anwendungskonfigurationsdatei administrativ erstellen” auf Seite 1339).
- Programmgesteuert, also direkt aus dem Anwendungscode. Dies wird in den folgenden Unterabschnitten beschrieben.

Bindungs- und Endpunktinformationen programmgesteuert definieren: SOAP-/JMS-Schnittstelle
Für die SOAP-/JMS-Schnittstelle können Sie direkt aus dem Anwendungscode programmgesteuert einen Endpunkt und eine Bindung definieren.

Informationen zu diesem Vorgang

Um Bindungs- und Endpunktinformationen programmgesteuert anzugeben, fügen Sie den erforderlichen Code zu Ihrer Anwendung hinzu und führen Sie dazu die folgenden Schritte aus.

Vorgehensweise

1. Erstellen Sie eine Instanz des Transportbindungselements des Kanals, indem Sie den folgenden Code zu Ihrer Anwendung hinzufügen:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new SoapJmsIbmTransportBinding
Element();
```

2. Definieren Sie alle erforderlichen Bindungseigenschaften. Fügen Sie beispielsweise folgenden Code zu Ihrer Anwendung hinzu, um den Clientverbindungsmodus (`ClientConnectionMode`) festzulegen:

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. Erstellen Sie eine benutzerdefinierte Bindung, die den Transportkanal mit einem Nachrichtenencoder koppelt, und fügen Sie dazu Ihrer Anwendung den folgenden Code hinzu:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(), transportBin
dingElement);
```

4. Erstellen Sie den SOAP-/JMS-URI.

   Der SOAP-/JMS-URI, der die für den Zugriff auf den Service erforderlichen IBM MQ-Verbindungsdetails beschreibt, muss als Endpunktadresse angegeben werden. Welche Adresse Sie angeben, hängt davon ab, ob der Kanal für eine Serviceanwendung oder eine Clientanwendung verwendet wird.

   - Bei Clientanwendungen muss der SOAP-/JMS-URI wie folgt als Endpunktadresse (`EndpointAddress`) erstellt werden:

     ```
     EndpointAddress address = new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connecti
     onFactory
     =connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

   - Bei Serviceanwendungen muss der SOAP-/JMS-URI wie folgt als URI erstellt werden:

     ```
     Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

     Weitere Informationen zu Endpunktadressen finden Sie im Abschnitt „Angepasste IBM MQ-Kanäle für das Adressformat der URI von WCF-Endpunkten” auf Seite 1343.

Bindungs- und Endpunktinformationen programmgesteuert definieren: Nicht-SOAP/Nicht-JMS-Schnittstelle
Für die Nicht-SOAP/Nicht-JMS-Schnittstelle können Sie direkt aus dem Anwendungscode programmgesteuert einen Endpunkt und eine Bindung definieren.

Informationen zu diesem Vorgang

Um Bindungs- und Endpunktinformationen programmgesteuert anzugeben, fügen Sie den erforderlichen Code zu Ihrer Anwendung hinzu und führen Sie dazu die folgenden Schritte aus.

Vorgehensweise

1. Erstellen Sie eine WMQ-Bindung (WmqBinding), indem Sie Ihrer Anwendung den folgenden Code hinzufügen:

```
WmqBinding binding = new WmqBinding();
```

Mit diesem Code wird eine Bindung erstellt, die die für die Nicht-SOAP/Nicht-JMS-Schnittstelle erforderlichen WmqMsgEncodingElement und WmqIbmTransportBindingElement koppelt.

2. Geben Sie den URI wmq:// an, der die für den Zugriff auf den Service erforderlichen IBM MQ-Verbindungsdetails beschreibt.

   Wie Sie den URI wmq:// angeben, hängt davon ab, ob der Kanal für eine Serviceanwendung oder eine Clientanwendung verwendet wird.

   - Bei Clientanwendungen muss der URI wmq:// wie folgt als Endpunktadresse (EndpointAddress) erstellt werden:

     ```
     EndpointAddress address = new EndpointAddress
     ("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

   - Bei Serviceanwendungen muss der URI wmq:// wie folgt als URI erstellt werden:

     ```
     Uri sampleAddress = new Uri(
     "wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

Angepasste IBM MQ-Kanäle für das Adressformat der URI von WCF-Endpunkten

Ein Web-Service wird mithilfe einer URI (Universal Resource Identifier) angegebenen, in der Einzelheiten zur Position und zur Verbindung bereitgestellt werden. Das URI-Format ist davon abhängig, ob Sie die SOAP/JMS-Schnittstelle oder eine andere Schnittstelle als die SOAP/JMS-Schnittstelle verwenden.

SOAP/JMS-Schnittstelle

Das in IBM MQ Transport for SOAP unterstützte URI-Format ermöglicht beim Zugriff auf Zielservices eine umfassende Steuerung von SOAP/ IBM MQ-spezifischen Parametern und Optionen. Dieses Format ist kompatibel mit WebSphere Application Server und mit CICS und erleichtert die Integration von IBM MQ in diese beiden Produkte.

Die URI-Syntax lautet folgendermaßen:

```
jms:/queue? name=value&name=value...
```

Dabei steht Name für einen Parameternamen und *Wert* für den entsprechenden Wert. Das Element Name=*Wert* kann beliebig oft wiederholt werden, wobei dem zweiten und allen nachfolgenden Elementen ein Et-Zeichen (&) vorangestellt werden muss.

Bei Parameternamen wird die Groß-/Kleinschreibung beachtet. Dasselbe gilt für Namen von IBM MQ-Objekten. Wenn ein beliebiger Parameter mehrmals angegeben wird, wird das letzte Auftreten des Parameters wirksam, d. h., dass Clientanwendungen Parameterwerte durch das Anhängen an die URI überschreiben können. Wenn zusätzliche nicht erkannte Parameter enthalten sind, werden diese ignoriert.

Wenn Sie eine URI in einer XML-Zeichenfolge speichern, müssen Sie das Et-Zeichen als "&" darstellen. Wenn eine URI in einem Script codiert ist, müssen Sie entsprechend sorgfältig vorgehen und Zeichen wie **&**, die andernfalls von der Shell interpretiert würden, in Escapezeichen setzen.

Dies ist ein Beispiel einer einfachen URI für einen Axis-Service:

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Hier ist ein Beispiel einer einfachen URI für einen .NET-Service:

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Es werden nur die erforderlichen Parameter bereitgestellt (`targetService` ist nur für .NET-Services erforderlich und `connectionFactory` verfügt über keine Optionen).

In diesem Axis-Beispiel enthält `connectionFactory` eine Reihe von Optionen:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

In diesem Axis-Beispiel wurde auch die `connectionFactory`-Option `sslPeerName` angegeben. Der Wert von 'sslPeerName' selbst enthält Wertepaare und wichtige eingebettete Leerzeichen:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Andere Schnittstelle als SOAP-/JMS-Schnittstellen

Das URI-Format für andere Schnittstellen als die SOAP/JMS-Schnittstelle ermöglicht beim Zugriff auf Zielservices eine umfangreiche Steuerung über IBM MQ-spezifische Parameter und Optionen.

Die URI-Syntax lautet folgendermaßen:

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/INS.QUOTE.REPLY@BRANCH452&per
sistence=MQPER_NOT_PERSISTENT
```

Diese IRI informiert einen Serviceanforderer darüber, dass eine Verbindung von einem IBM MQ TCP-Client zu einem System mit der Bezeichnung 'example.com' auf Port 1415 verwendet werden kann und persistente Anforderungsnachrichten in eine Warteschlange mit der Bezeichnung INS.QUOTE.REQUEST auf Warteschlangenmanager MOTOR.INS eingereiht werden. Die IRI gibt an, dass der Service-Provider Antworten in eine Warteschlange mit der Bezeichnung INS.QUOTE.REPLY auf Warteschlangenmanager BRANCH452 einreiht. Das URI-Format ist das für SupportPac MA93 angegebene Format. Weitere Informationen zu den IBM MQ IRI-Spezifikationen finden Sie im SupportPac MA93: IBM MQ -Servicedefinition .

Optionen zur WCF-Bindungskonfiguration

Konfigurationsoptionen können auf zwei Arten für die Bindungsinformationen von angepassten Kanäle angewendet werden. Sie können die Eigenschaften administrativ oder programmgesteuert festlegen.

Die Optionen zur Bindungskonfiguration können auf zwei Arten festgelegt werden:

1. Administrativ: Die Einstellungen der Bindungseigenschaft müssen im Abschnitt zum Transport der angepassten Bindungsdefinition in der Anwendungskonfigurationsdatei angegeben werden, z. B. `app.config`.
2. Programmgesteuert: Der Anwendungscode muss so geändert werden, dass die Eigenschaft während der Initialisierung der angepassten Bindung angegeben wird.

Bindungseigenschaften administrativ festlegen

Die Einstellungen der Bindungseigenschaft können in der Anwendungskonfigurationsdatei angegeben werden, z. B. `app.config`. Die Konfigurationsdatei wird wie in den folgenden Beispielen gezeigt vom Tool **`svcutil`** generiert.

SOAP/JMS-Schnittstelle

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

Andere Schnittstelle als SOAP-/JMS-Schnittstellen

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

Bindungseigenschaften programmgesteuert festlegen

Zum Hinzufügen einer WCF-Bindungseigenschaft zur Angabe des Clientverbindungsmodus müssen Sie den Service-Code so ändern, dass die Eigenschaft während der Initialisierung der angepassten Bindung angegeben wird.

Geben Sie den nicht verwalteten Client-Verbindungsmodus mithilfe des folgenden Beispiels an:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                        transportBindingElement);
```

WCF-Bindungseigenschaften

Tabelle 194. Werte von Bindungseigenschaften beim administrativen oder programmgesteuerten Festlegen

Eigenschaftsname	Client- oder Service-anwen-dung	Administ-rativer Wert	Programmgesteuerter Wert	Beschreibung
maxBufferPoolSize	Beide	Ganze Zahl mit Vorzei-chen mit 0 bis 64 Bit	Ganze Zahl mit Vorzei-chen mit 0 bis 64 Bit	Gibt die maximale Größe des Speichers an, der zum Speichern von WCF-Nachrichtenpuffern für eine Instanz des Kanals verwen-det werden kann.
maxMessageSize	Beide	Ganze Zahl mit Vorzei-chen mit 1 bis 32 Bit	Ganze Zahl mit Vorzei-chen mit 1 bis 32 Bit	Gibt die maximale Speicherkapa-zität an, die für einzelne WCF-Nachrichten verwendet werden kann

Tabelle 194. Werte von Bindungseigenschaften beim administrativen oder programmgesteuerten Festlegen (Forts.)

Eigenschaftsname	Client- oder Service-anwendung	Administrativer Wert	Programmgesteuerter Wert	Beschreibung
clientConnectionMode	Beide	0 (Standardwert) 1	AS_URI (Standardwert) CLIENT_UNMANAGED	Gibt den Clientverbindungsmodus des Transportkanals an. 0 zeigt an, dass der Clientverbindungsmodus dem in der URI angegebenen Modus entspricht. Wird nur bei Verwendung der Clientverbindung verwendet. Gibt an, dass der Clientverbindungsmodus dem in der URI angegebenen Modus entspricht. 0 ist der Standardwert, wenn kein Clientverbindungsmodus festgelegt ist. 1 zeigt an, dass der Clientverbindungsmodus ein nicht verwalteter Client ist. Wird nur bei Verwendung der Clientverbindung verwendet.
MaxConcurrentCalls	Client	Der Bereich liegt bei 0 - 2 147 483 647 16 ist der Standardwert	Der Bereich liegt bei 0 - 2 147 483 647 16 ist der Standardwert	Diese Eigenschaft definiert die maximale Anzahl gleichzeitig stattfindender Operationen, die in einem individuellen Client-Proxy zu einem beliebigen Zeitpunkt ausgeführt werden können. Wenn mehr Operationen gestartet werden, werden diese in eine Warteschlange eingereiht, bis eine derzeit ausgeführte Operation abgeschlossen wird oder das Zeitlimit überschreitet. Mit dieser Einstellung kann die maximale Anzahl der Threads und Ressourcen gesteuert werden, die von einem einzelnen Proxy verbraucht werden. Mit 0 wird dieser Grenzwert entfernt und es wird ermöglicht, dass das gleichzeitige Ausführen aller Operationen versucht wird.

Tabelle 194. Werte von Bindungseigenschaften beim administrativen oder programmgesteuerten Festlegen (Forts.)

Eigenschaftsname	Client- oder Service-anwendung	Administrativer Wert	Programmgesteuerter Wert	Beschreibung
MaxConcurrentCalls	Service	Der Bereich liegt bei 1 - 2 147 483 647 16 ist der Standardwert	Der Bereich liegt bei 1 - 2 147 483 647 16 ist der Standardwert	Diese Eigenschaft wird nur verwendet, wenn die Funktion zur gesicherten Zustellung aktiviert ist (weiter Informationen zur gesicherten Zustellung finden Sie unter „Zuverlässige Nachrichtenübermittlung mit dem angepassten WCF-Kanal” auf Seite 1333). Damit wird die maximale Anzahl der gleichzeitig stattfindenden Operationen angegeben, die für den angegebenen Endpunkt gleichzeitig ausgeführt werden können. Gehen Sie beim Ändern dieser Einstellung vorsichtig vor. Für jede gleichzeitig stattfindende Operation sind zusätzliche Ressourcen erforderlich, insbesondere eine neue Instanz des angepassten Kanals und der zugehörigen Threads aus dem Thread-Pool zur Verarbeitung der Anforderungen. Eine übermäßige Zuordnung kann kontraproduktiv sein und die Leistung schwerwiegenden beeinträchtigen. Zur Unterstützung dieser Eigenschaft muss eine entsprechende Konfiguration des Thread-Pools vorgenommen werden.

Services für WCF erstellen und bereitstellen

Übersicht über Services für Microsoft Windows Communication Foundation (WCF), in der das Erstellen und Konfigurieren von WCF-Services erläutert wird.

Der angepasste IBM MQ-Kanal für WCF und die WCF-Services, die diesen verwenden, können von den folgenden Methoden bereitgestellt werden:

- Self-Hosting
- Windows-Service

Der angepasste IBM MQ-Kanal für WCF kann nicht im Windows-Prozessaktivierungsdienst bereitgestellt werden.

In den folgenden Abschnitten finden Sie einige einfache Beispiele zum Self-Hosting, in denen die zugehörigen Schritte gezeigt werden. Die Onlinedokumentation zu Microsoft-WCF mit weiteren Informationen und den aktuellsten Einzelheiten finden Sie auf der Website von Microsoft-MSDN unter https://msdn.microsoft.com.

WCF-Serviceanwendungen mit der Methode 1 erstellen: Administratives Self-Hosting mithilfe einer Anwendungskonfigurationsdatei

Nach dem Erstellen einer Anwendungskonfigurationsdatei öffnen Sie eine Instanz des Service und fügen Ihrer Anwendung den angegebenen Code hinzu.

Vorbereitende Schritte

Erstellen oder bearbeiten Sie wie unter „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen in einer Anwendungskonfigurationsdatei administrativ erstellen" auf Seite 1339 beschrieben eine Anwendungskonfigurationsdatei für den Service:

Informationen zu diesem Vorgang

1. Instanziieren und öffnen Sie eine Instanz des Service im Service-Host. Der Servicetyp muss dem in der Servicekonfigurationsdatei angegebenen Servicetyp entsprechen.
2. Fügen Sie Ihrer Anwendung den folgenden Code hinzu:

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

WCF-Serviceanwendung mit der Methode 2 erstellen: Direktes programmgesteuertes Self-Hosting aus der Anwendung

Fügen Sie die Bindungseigenschaften hinzu, erstellen Sie den Service-Host mit einer Instanz der erforderlichen Serviceklasse und öffnen Sie den Service.

Vorbereitende Schritte

1. Fügen Sie der Datei `IBM.XMS.WCF.dll` für den angepassten Kanal einen Verweis zum Projekt hinzu. Die Datei `IBM.XMS.WCF.dll` befindet sich im Verzeichnis *`WMQInstallDir`*`\bin`. Dabei ist *WMQ-Installationsverzeichnis* das Verzeichnis, in dem IBM MQ installiert ist.
2. Fügen Sie eine Anweisung *using* zum Namensbereich `IBM.XMS.WCF` hinzu. Beispiel: `using IBM.XMS.WCF`
3. Erstellen Sie eine Instanz der Bindungseigenschaften für den Kanal und des Endpunkts, wie unter „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen programmgesteuert erstellen" auf Seite 1341 beschrieben.

Informationen zu diesem Vorgang

Wenn Änderungen an den Bindungseigenschaften des Kanals erforderlich sind, führen Sie die folgenden Schritte aus:

1. Fügen Sie `transportBindingElement` die Bindungseigenschaften hinzu, wie im folgenden Beispiel gezeigt wird:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new SoapJmsIbmTransportBinding
Element();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(), transportBin
dingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
                       connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojn
di");
```

2. Erstellen Sie den Service-Host mit einer Instanz der erforderlichen Serviceklasse:

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. Öffnen Sie den Service:

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

Metadaten mithilfe eines HTTP-Endpunkts bereitstellen

Anweisungen zum Bereitstellen der Metadaten eines Service, der für die Verwendung des angepassten IBM MQ-Kanals für WCF konfiguriert ist.

Informationen zu diesem Vorgang

Wenn die Servicemetadaten verfügbar sein müssen (damit Tools wie z. B. `svcutil` statt beispielsweise aus einer WSDL-Datei im Offline-Modus direkt aus dem aktiven Service darauf zugreifen können), müssen dazu die Servicemetadaten mit einem HTTP-Endpunkt bereitgestellt werden. Dieser zusätzliche Endpunkt kann folgendermaßen hinzugefügt werden.

1. Fügen Sie 'ServiceHost' die Basisadresse hinzu, unter der die Metadaten verfügbar sein müssen, wie beispielsweise:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. Fügen Sie 'ServiceHost' vor dem Öffnen des Service folgenden Code hinzu:

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

Ergebnisse

Die Metadaten sind jetzt unter der Adresse `http://localhost:8000/MyService` verfügbar.

Clientanwendungen für WCF erstellen

Übersicht über das Generieren und Erstellen von Clientanwendungen für Microsoft Windows Communication Foundation (WCF).

Eine Clientanwendung kann für einen WCF-Service erstellt werden. Clientanwendungen werden normalerweise mithilfe des Microsoft-Tools für das Dienstprogramm für Metadaten des Servicemodells (svcutil.exe) generiert, um die erforderlichen Konfigurations- und Proxy-Dateien zur direkten Verwendung durch die Anwendung zu erstellen.

WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit Metadaten aus einem aktiven Service generieren

In diesem Abschnitt finden Sie Anweisungen zur Verwendung des Microsoft-Tools 'svcutil.exe' zum Generieren eines Clients für einen Service, der zur Verwendung des angepassten IBM MQ-Kanals für WCF konfiguriert ist.

Vorbereitende Schritte

Es gibt drei Voraussetzungen für die Verwendung des Tools 'svcutil' zum Erstellen der erforderlichen Konfigurations- und Proxy-Dateien, die direkt von der Anwendung verwendet werden können:

- Der WCF-Service muss vor dem Start des Tools 'svcutil' aktiv sein.
- Zum Generieren eines Clients direkt aus einem aktiven Service muss der WCF-Service neben den Verweisen auf den Endpunkt des angepassten IBM MQ-Kanals die zugehörigen Metadaten mithilfe eines HTTP-Ports verfügbar machen.
- Der angepasste Kanal muss in den Konfigurationsdaten für 'svcutil' registriert sein.

Informationen zu diesem Vorgang

In den folgenden Schritten wird das Generieren eines Clients für einen Service beschrieben, der zur Verwendung des angepassten IBM MQ-Kanals konfiguriert ist, aber auch die zugehörigen Metadaten während der Ausführung über einen separaten HTTP-Port bereitstellt:

1. Starten Sie den WCF-Service. (Der Service muss vor dem Start des Tools 'svcutil' aktiv sein.)
2. Fügen Sie die Einzelheiten aus der Konfigurationsdatei `svcutil.exe` im Stammverzeichnis der Installation zu der aktiven Konfigurationsdatei für das Tool 'svcutil' (normalerweise `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config`) hinzu, damit 'svcutil' den angepassten IBM MQ-Kanal erkennt.
3. Führen Sie das Tool 'svcutil' aus einer Eingabeaufforderung auf; Beispiel:

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. Kopieren Sie die generierten Dateien `app.config` und `YourService.cs` in das Projekt des Microsoft Visual Studio-Clients.

Nächste Schritte

Wenn die Servicemetadaten nicht direkt abgerufen werden können, kann das Tool 'svcutil' stattdessen zum Generieren der Clientdateien aus WSDL verwendet werden. Weitere Informationen finden Sie unter „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit WSDL generieren” auf Seite 1350

WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit WSDL generieren

In diesem Abschnitt finden Sie Anweisungen zum Generieren von WCF-Clients aus WSDL, wenn die Metadaten des Service nicht verfügbar sind.

Wenn die Metadaten des Service zum Generieren eines Clients aus den Metadaten eines aktiven Service nicht direkt abgerufen werden können, können die Clientdateien stattdessen mithilfe des Tools 'svcutil' aus WSDL generiert werden. Die folgenden Änderungen an der WSDL müssen vorgenommen werden, um anzugeben, dass der angepasste IBM MQ-Kanal verwendet werden soll:

1. Fügen Sie die folgenden Namensbereichsdefinitionen und Richtlinieninformationen hinzu:

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utili
ty-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExtactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. Ändern Sie den Abschnitt zu den Bindungen, damit auf den Abschnitt mit der neuen Richtlinie verwiesen wird, und entfernen Sie alle `transport`-Definitionen aus dem zugrundeliegenden Bindungselement:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
```

```
        </wsdl:binding>
</wsdl:definitions>
```

3. Führen Sie das Tool 'svcutil' aus einer Eingabeaufforderung auf; Beispiel:

```
svcutil /language:C# /r:  MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config  MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

WCF-Clientanwendungen mithilfe eines Client-Proxys mit einer Anwendungskonfigurationsdatei erstellen

Vorbereitende Schritte

Erstellen oder bearbeiten Sie wie unter „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen in einer Anwendungskonfigurationsdatei administrativ erstellen” auf Seite 1339 beschrieben eine Anwendungskonfigurationsdatei für den Client:

Informationen zu diesem Vorgang

Instanziieren und öffnen Sie eine Instanz des Client-Proxys. Der an das generierte Proxy übergebene Parameter muss dem in der Clientkonfigurationsdatei angegebenen Endpunktnamen entsprechen, beispielsweise `Endpoint_WMQ`:

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

WCF-Clientanwendungen mithilfe eines Client-Proxys mit programmgesteuerter Konfiguration erstellen

Vorbereitende Schritte

1. Fügen Sie der Datei `IBM.XMS.WCF.dll` für den angepassten Kanal einen Verweis zum Projekt hinzu. Die Datei `IBM.XMS.WCF.dll` befindet sich im Verzeichnis *WMQInstallDir*`\bin`. Dabei ist *WMQ-Installationsverzeichnis* das Verzeichnis, in dem IBM MQ installiert ist.
2. Fügen Sie eine Anweisung *using* zum Namensbereich `IBM.XMS.WCF` hinzu. Beispiel: `using IBM.XMS.WCF`
3. Erstellen Sie eine Instanz des Bindungselements und des Endpunkt für den Kanal, wie unter „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen programmgesteuert erstellen” auf Seite 1341 beschrieben.

Informationen zu diesem Vorgang

Wenn Änderungen an den Bindungseigenschaften des Kanals erforderlich sind, führen Sie die folgenden Schritte aus.

1. Fügen Sie `transportBindingElement` die Bindungseigenschaften hinzu, wie in der folgenden Abbildung gezeigt wird:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new SoapJmsIbmTransportBinding
Element();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(), transportBin
dingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Erstellen Sie den Client-Proxy, wie in der folgenden Abbildung gezeigt wird. Dabei sind *Bindung* und *Endpunktadresse* die in Schritt 1 konfigurierte und übergebene Bindung und Endpunktadresse:

```
MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
           try {
                  myClient.myMethod("HelloWorld!");
                  myClient.Close();
           }
           catch (TimeoutException e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
           catch (CommunicationException e) {
               Console.Out.WriteLine(e);
                                myClient.Abort();
           }
           catch (Exception e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
```

WCF-Beispiele verwenden

Das Beispiel für Windows Communication Foundation (WCF) stellt einige einfache Beispiele zur Verwendung des angepassten IBM MQ-Kanals bereit.

Zum Erstellen der Beispielprojekte ist Microsoft.NET 3.5 SDK oder Microsoft Visual Studio 2008 erforderlich.

Beispiel für eine einfache unidirektionale Verbindung zwischen Client und Server in WCF

In diesem Beispiel wird gezeigt, wie der angepasste IBM MQ-Kanal zum Starten eines WCF-Services(Windows Communication Foundation) aus einem WCF-Client in Form eines Einwegkanals verwendet wird.

Informationen zu diesem Vorgang

Der Service implementiert eine einzelne Methode, die eine Zeichenfolge an die Konsole ausgibt. Der Client wurde vom Tool `svcutil` zum Abrufen der Service-Metadaten aus einem separat verfügbaren HTTP-Endpunkt generiert (siehe „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit Metadaten aus einem aktiven Service generieren” auf Seite 1349).

Das Beispiel wurde wie in in der folgenden Prozedur beschrieben mit bestimmten Ressourcennamen konfiguriert. Wenn Sie die Ressourcennamen ändern müssen, müssen Sie auch den zugehörigen Wert in der Clientanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` und in der Serviceanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` ändern. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ. Weitere Informationen zum Formatieren der JMS-Endpunkt-URI finden Sie unter *IBM MQ Transport for SOAP* in der IBM MQ-Produktdokumentation. Wenn Sie die Beispiellösung und die Beispielquelle ändern müssen, benötigen Sie eine integrierte Entwicklungsumgebung (IDE) wie beispielsweise Microsoft Visual Studio 8 oder höher.

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager mit der Bezeichnung *QM1*.
2. Erstellen Sie ein Warteschlangenziel mit der Bezeichnung *SampleQ*.
3. Starten Sie den Service, damit der Listener auf Nachrichten warten kann: Führen Sie die Datei *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.
4. Führen Sie den Client einmal aus: Führen Sie die Datei *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

   Die Clientanwendung führt eine Schleife fünfmal aus und sendet dabei fünf Nachrichten an *SampleQ*.

Ergebnisse

Die Serviceanwendung empfängt die Nachrichten von *SampleQ* und zeigt `Hello World` auf dem Bildschirm fünfmal an.

Nächste Schritte

Beispiel für eine einfache Anforderung/Antwort-Verbindung zwischen Client und Server in WCF

In diesem Beispiel wird gezeigt, wie der angepasste IBM MQ-Kanal zum Starten eines WCF-Services(Windows Communication Foundation) aus einem WCF-Client in Form eines Anforderung/Antwort-Kanals verwendet wird.

Informationen zu diesem Vorgang

In diesem Service werden einige einfache Berechnungsmethoden bereitgestellt, mit denen zwei Zahlen addiert und subtrahiert werden und anschließend das Ergebnis ausgegeben wird. Der Client wurde vom Tool `svcutil` zum Abrufen der Service-Metadaten aus einem separat verfügbaren HTTP-Endpunkt generiert (siehe „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit Metadaten aus einem aktiven Service generieren” auf Seite 1349).

Das Beispiel wurde wie in in der folgenden Prozedur beschrieben mit bestimmten Ressourcennamen konfiguriert. Wenn Sie die Ressourcennamen ändern müssen, müssen Sie auch den zugehörigen Wert für die Clientanwendung in der Datei *MQ_INSTALLATION_PATH*`\Tools\wcf\samples\WCF\requestreply\client\app.config` und für die Serviceanwendung in der Datei *MQ_INSTALLATION_PATH*`\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` ändern. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ. Weitere Informationen zum Formatieren der JMS-Endpunkt-URI finden Sie unter *IBM MQ Transport for SOAP* in der IBM MQ-Produktdokumentation. Wenn Sie die Beispiellösung und die Beispielquelle ändern müssen, benötigen Sie eine integrierte Entwicklungsumgebung (IDE) wie beispielsweise Microsoft Visual Studio 8 oder höher.

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager mit der Bezeichnung *QM1*.
2. Erstellen Sie ein Warteschlangenziel mit der Bezeichnung *SampleQ*.
3. Erstellen Sie ein Warteschlangenziel mit der Bezeichnung *SampleReplyQ*.
4. Starten Sie den Service, damit der Listener auf Nachrichten warten kann: Führen Sie die Datei *MQ_INSTALLATION_PATH*`\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe` aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

5. Führen Sie den Client einmal aus: Führen Sie die Datei *MQ_INSTALLATION_PATH*\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

Ergebnisse

Nach der Ausführung des Clients wird der folgende Prozess gestartet und viermal wiederholt, womit insgesamt fünf Nachrichten in jede Richtung gesendet werden:

1. Der Client reiht eine Anforderungsnachricht in *SampleQ* ein und wartet auf eine Antwort.
2. Der Service empfängt die Anforderungsnachricht von *SampleQ*.
3. Der Service addiert und subtrahiert einige Werte mithilfe der Nachrichteninhalte.
4. Anschließend reiht der Service die Ergebnisse in eine Nachricht in *SampleReplyQ* ein und wartet, bis der Client eine neue Nachricht einreiht.
5. Der Client empfängt die Nachricht vom *SampleReplyQ* und zeigt die Ergebnisse auf dem Bildschirm an.

Nächste Schritte

Beispiel für die Verbindung von einem WCF-Client zu einem .NET-Service, der von IBM MQ gehostet wird

Beispielclientanwendungen und Beispiele für Service-Proxy-Anwendungen werden für .NET und Java bereitgestellt. Die Beispiele basieren auf einem Beispiel für Börsennotierung, bei dem eine Anforderung nach einer Börsennotierung ausgegeben und die Börsennotierung anschließend bereitgestellt wird.

Vorbereitende Schritte

Für das Beispiel muss der .NET SOAP over JMS-Service, auf dem die Umgebung gehostet ist, ordnungsgemäß in IBM MQ installiert und konfiguriert sein und der Zugriff darauf muss aus einem lokalen Warteschlangenmanager möglich sein.

Wenn der .NET SOAP over JMS-Service, auf dem die Umgebung gehostet ist, ordnungsgemäß in IBM MQ installiert und konfiguriert ist und der Zugriff darauf aus einem lokalen Warteschlangenmanager möglich ist, müssen zusätzliche Konfigurationsschritte ausgeführt werden.

1. Setzen Sie die Umgebungsvariable **WMQSOAP_HOME** auf das IBM MQ -Installationsverzeichnis, z. B. C:\Programme\IBM\MQ .
2. Stellen Sie sicher, dass der Java-Compiler `javac` verfügbar und in der Angabe für PATH enthalten ist.
3. Kopieren Sie die Datei `axis.jar` aus dem Verzeichnis `prereqs/axis` des Installationsimage in das IBM MQ -Produktionsverzeichnis, z. B. C:\Programme\IBM\MQ\java\lib\soap .
4. Fügen Sie *MQ_INSTALLATION_PATH*\Java\lib zur Angabe für PATH hinzu. Hierbei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem IBM MQ installiert ist. Beispiel: C:\Programme\IBM\MQ
5. Stellen Sie sicher, dass die Position von .NET in *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd richtig angegeben ist. Hierbei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem IBM MQ installiert ist. Beispiel: C:\Programme\IBM\MQ. Die Position von .NET kann angegeben werden. Beispiel: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Wenn die bisherigen Schritte abgeschlossen sind, testen Sie den Service und führen ihn aus:

1. Navigieren Sie zu Ihrem Soap over JMS-Arbeitsverzeichnis.
2. Geben Sie einen der folgenden Befehle zum Ausführen des Funktionstests ein und starten Sie den Listener für den Service:
   - Für .NET: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold , wobei *MQ_INSTALLATION_PATH* das Verzeichnis ist, in dem IBM MQ installiert ist.
   - Für AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold . Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem IBM MQ installiert ist.

Durch das Argument `hold` werden die Listeners auch nach Abschluss des Tests weiterhin ausgeführt.

Wenn während dieser Konfiguration Fehler gemeldet werden, können Sie alle Änderungen entfernen, damit die Prozedur folgendermaßen erneut gestartet werden kann:

1. Löschen Sie das generierte Verzeichnis für SOAP over JMS.
2. Löschen Sie den Warteschlangenmanager.

Informationen zu diesem Vorgang

Dieses Beispiel veranschaulicht eine Verbindung von einem WCF-Client zum Beispielservice .NET SOAP over JMS , der in IBM MQ bereitgestellt wird, in Form eines Einwegkanals. Der Service implementiert ein einfaches Beispiel für Börsennotierungen, das eine Textzeichenfolge an die Konsole ausgibt.

Der Client wurde mithilfe von WSDL zum Generieren von Clientdateien erstellt, wie unter „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit WSDL generieren" auf Seite 1350 beschrieben wird.

Das Beispiel wurde wie in in der folgenden Prozedur beschrieben mit bestimmten Ressourcennamen konfiguriert. Wenn Sie die Ressourcennamen ändern müssen, müssen Sie auch den zugehörigen Wert bei der Clientanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQNET\default\client\app.config` und bei der Serviceanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` ändern. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ. Weitere Informationen zum Formatieren der JMS-Endpunkt-URI finden Sie unter *IBM MQ Transport for SOAP* in der IBM MQ-Produktdokumentation.

Vorgehensweise

Führen Sie den Client einmal aus: Führen Sie die Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe` aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

Die Clientanwendung führt eine Schleife fünfmal aus und sendet dabei fünf Nachrichten an die Beispielwarteschlange.

Ergebnisse

Die Serviceanwendung erhält die Nachrichten aus der Beispielwarteschlange und zeigt `Hello World` auf dem Bildschirm fünfmal an.

Beispiel für eine Verbindung von einem WCF-Client zu einem Axis Java-Service, der von IBM MQ gehostet wird

Beispielclientanwendungen und Beispiele für Service-Proxy-Anwendungen werden für Java und .NET bereitgestellt. Die Beispiele basieren auf einem Beispiel für Börsennotierung, bei dem eine Anforderung nach einer Börsennotierung ausgegeben und die Börsennotierung anschließend bereitgestellt wird.

Vorbereitende Schritte

Für dieses Beispiel muss der .NET SOAP over JMS-Service, auf dem die Umgebung gehostet ist, ordnungsgemäß in IBM MQ installiert und konfiguriert sein und der Zugriff darauf muss aus einem lokalen Warteschlangenmanager möglich sein.

Wenn der .NET SOAP over JMS-Service, auf dem die Umgebung gehostet ist, ordnungsgemäß in IBM MQ installiert und konfiguriert ist und der Zugriff darauf aus einem lokalen Warteschlangenmanager möglich ist, müssen zusätzliche Konfigurationsschritte ausgeführt werden.

1. Setzen Sie die Umgebungsvariable **WMQSOAP_HOME** auf das IBM MQ -Installationsverzeichnis, z. B. `C:\Programme\IBM\MQ` .
2. Stellen Sie sicher, dass der Java-Compiler `javac` verfügbar und in der Angabe für PATH enthalten ist.

3. Kopieren Sie die Datei `axis.jar` aus dem Verzeichnis `prereqs/axis` des Installationsimage in das Installationsverzeichnis IBM MQ .
4. Fügen Sie *MQ_INSTALLATION_PATH*`\Java\lib` zur Angabe für PATH hinzu. Hierbei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem IBM MQ installiert ist. Beispiel: `C:\Programme\IBM\MQ`
5. Stellen Sie sicher, dass die Position von .NET in *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` richtig angegeben ist. Hierbei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem IBM MQ installiert ist. Beispiel: `C:\Programme\IBM\MQ`. Die Position von .NET kann angegeben werden. Beispiel: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Wenn die bisherigen Schritte abgeschlossen sind, testen Sie den Service und führen ihn aus:

1. Navigieren Sie zu Ihrem Soap over JMS-Arbeitsverzeichnis.
2. Geben Sie einen der folgenden Befehle zum Ausführen des Funktionstests ein und starten Sie den Listener für den Service:
   - Für .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , wobei *MQ_INSTALLATION_PATH* das Verzeichnis ist, in dem IBM MQ installiert ist.
   - Für AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` . Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem IBM MQ installiert ist.

Durch das Argument `hold` werden die Listeners auch nach Abschluss des Tests weiterhin ausgeführt.

Wenn während dieser Konfiguration Fehler gemeldet werden, können Sie alle Änderungen entfernen, damit die Prozedur folgendermaßen erneut gestartet wird:

1. Löschen Sie das generierte Verzeichnis für SOAP over JMS.
2. Löschen Sie den Warteschlangenmanager.

Informationen zu diesem Vorgang

Das Beispiel veranschaulicht eine Verbindung von einem WCF-Client zum Beispielservice 'Axis Java SOAP over JMS ', der in IBM MQ bereitgestellt wird, über einen unidirektionalen Kanal. Der Service implementiert ein einfaches Beispiel für Börsennotierungen, das eine Textzeichenfolge in eine Datei ausgibt, die im aktuellen Verzeichnis gespeichert wird.

Der Client wurde mithilfe von WSDL zum Generieren von Clientdateien erstellt, wie unter „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit WSDL generieren” auf Seite 1350 beschrieben wird.

Das Beispiel wurde wie in diesem Absatz beschrieben mit bestimmten Ressourcennamen konfiguriert. Wenn Sie die Ressourcennamen ändern müssen, müssen Sie auch den zugehörigen Wert in der Clientanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQAxis\default\client\app.config` und in der Serviceanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` ändern. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

Vorgehensweise

Führen Sie den Client einmal aus: Führen Sie die Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe` aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

Die Clientanwendung führt eine Schleife fünfmal aus und sendet dabei fünf Nachrichten an die Beispielwarteschlange.

Ergebnisse

Die Serviceanwendung erhält die Nachrichten aus der Beispielwarteschlange und fügt `Hello World` einer Datei im aktuellen Verzeichnis fünfmal hinzu.

Beispiel für eine Verbindung vom WCF-Client zum Java-Service, der von WebSphere Application Server gehostet wird

Es werden Beispielclientanwendungen und Beispiele für Service-Proxy-Anwendungen für WebSphere Application Server 6 bereitgestellt. Es wird auch ein Anfrage-/Antwort-Service bereitgestellt.

Vorbereitende Schritte

Für dieses Beispiel muss die folgende IBM MQ-Konfiguration verwendet werden:

Tabelle 195. Erforderliche IBM MQ-Konfiguration

Objekt	**Erforderlicher Name**
Warteschlangenmanager	QM1
Lokale Warteschlange	HelloWorld
Lokale Warteschlange	HelloWorldReply

Für dieses Beispiel muss außerdem eine Hosting-Umgebung für WebSphere Application Server 6 ordnungsgemäß installiert und konfiguriert sein. WebSphere Application Server 6 verwendet eine Verbindung im Bindungsmodus, um standardmäßig die Verbindung zu IBM MQ herzustellen. Deshalb muss WebSphere Application Server 6 auf dem gleichen System wie der Warteschlangenmanager installiert sein.

Nach der Konfiguration der WAS-Umgebung müssen die folgenden zusätzlichen Konfigurationsschritte ausgeführt werden:

1. Erstellen Sie die folgenden JNDI-Objekte im JNDI-Repository für WebSphere Application Server:
   a. Ein JMS-Warteschlangenziel mit der Bezeichnung `HelloWorld`
      - Legen Sie `jms/HelloWorld` als JNDI-Namen fest.
      - Legen Sie `HelloWorld` als Warteschlangenname fest.
   b. Eine JMS-Warteschlangenverbindungsfactory mit der Bezeichnung `HelloWorldQCF`
      - Legen Sie `jms/HelloWorldQCF` als JNDI-Namen fest.
      - Legen Sie `QM1` als Warteschlangenmanagernamen fest.
   c. Eine JMS-Warteschlangenverbindungsfactory mit der Bezeichnung `WebServicesReplyQCF`
      - Legen Sie `jms/WebServicesReplyQCF` als JNDI-Namen fest.
      - Legen Sie `QM1` als Warteschlangenmanagernamen fest.
2. Erstellen Sie einen Nachrichten-Listener-Port namens `HelloWorldPort` in WebSphere Application Server mit der folgenden Konfiguration:
   - Legen Sie `jms/HelloWorldQCF` als JNDI-Namen der Verbindungsfactory fest.
   - Legen Sie `jms/HelloWorld` als JNDI-Name des Ziels fest.
3. Installieren Sie die Anwendung `HelloWorldEJBEAR.ear` für den Web-Service folgendermaßen auf Ihrem WebSphere-Anwendungsserver:
   a. Klicken Sie auf **Applications** > **New Application** > **New Enterprise Application** (Anwendungen > Neue Anwendung > Neue Unternehmensanwendung).
   b. Navigieren Sie zu *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear`. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.
   c. Nehmen Sie keine Änderungen an den Standardoptionen im Assistenten vor und starten Sie den Anwendungsserver erneut, nachdem die Anwendung installiert wurde.

Testen Sie nach Abschluss der WAS-Konfiguration den Service, indem Sie ihn einmal ausführen:

1. Navigieren Sie zu Ihrem Soap over JMS-Arbeitsverzeichnis.

2. Geben Sie den folgenden Befehl zum Ausführen des Beispiels ein: *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\TestClient.exe`. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

Informationen zu diesem Vorgang

Im Beispiel wird eine Verbindung von einem WCF-Client zum WebSphere Application Server SOAP over JMS-Beispielservice gezeigt, der mit den in IBM MQ integrierten WCF-Beispielen bereitgestellt wird. In diesem Beispiel wird ein Anforderungs-/Antwortkanal verwendet. Nachrichten werden zwischen WCF und dem WebSphere Application Server mit IBM MQ-Warteschlangen übertragen. Der Service implementiert die Methode `HelloWorld(...)`, die eine Zeichenfolge empfängt und eine Begrüßung an den Client sendet.

Der Client wurde vom Tool 'svcutil' zum Abrufen der Service-Metadaten aus einem separat verfügbaren HTTP-Endpunkt generiert (siehe „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit Metadaten aus einem aktiven Service generieren” auf Seite 1349).

Das Beispiel wurde wie in in der folgenden Prozedur beschrieben mit bestimmten Ressourcennamen konfiguriert. Wenn Sie die Ressourcennamen ändern müssen, müssen Sie auch den zugehörigen Wert bei der Clientanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\default\client\app.config` und bei der Serviceanwendung in der Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` ändern. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

Der Service und Client basieren auf dem Service und Client, die im IBM Developer-Artikel *Building a JMS Web service using SOAP over JMS and WebSphere Studio* behandelt werden. Weitere Informationen zum Entwickeln von SOAP over JMS-Web-Services, die mit dem angepassten WCF-Kanal für IBM MQ kompatibel sind, finden Sie im Abschnitt https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html.

Vorgehensweise

Führen Sie den Client einmal aus: Führen Sie die Datei *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe` aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM MQ.

Die Clientanwendung startet beide Methoden des Service zur gleichen Zeit und sendet dabei zwei Nachrichten an die Beispielwarteschlange.

Ergebnisse

Die Serviceanwendung empfängt die Nachrichten von der Beispielwarteschlange und stellt für den Methodenaufruf `HelloWorld(...)` eine Antwort bereit, die von der Clientanwendung in der Konsole ausgegeben wird.

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: