9.0

Anwendungen für IBM MQ entwickeln

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen” auf Seite 1439 gelesen werden.

Diese Ausgabe bezieht sich auf Version 9 Release 0 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.

© Copyright International Business Machines Corporation 2007, 2023.

Inhaltsverzeichnis

Anwendungen entwickeln.....5
Konzepte für die Anwendungsentwicklung.....6
Von Anwendungen ausführbare Aktionen.....8
Anwendungsprogramme, die die MQI verwenden.....10
Objektorientierte Anwendungen.....11
IBM MQ-Nachrichten.....14
Microsoft Transaction Server-Anwendungen vorbereiten und ausführen.....47
IBM MQ mit WebSphere Application Server verwenden.....47
Konzepte für den Entwurf von IBM MQ-Anwendungen.....48
IBM MQ classes for Java oder IBM MQ classes for JMS verwenden.....51
Verfahren zum Entwerfen von Nachrichten.....52
Selektoren und Nachrichteneigenschaften.....53
Anwendungsdesign und Leistung.....53
Verfahren zum Entwerfen von anspruchsvolleren Anwendungen.....55
Entwurfs- und Leistungsaspekte für Anwendungen unter IBM i.....57
Linux on POWER Systems - Little Endian-Anwendungen.....59
Entwurfs- und Leistungsaspekte für Anwendungen unter z/OS.....59
IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS.....63
JMS- und Java-Anwendungen entwickeln.....76
IBM MQ classes for JMS verwenden.....77
IBM MQ classes for Java verwenden.....337
IBM MQ-Ressourcenadapter verwenden.....434
IBM MQ und WebSphere Application Server gemeinsam verwenden.....499
Paket IBM MQ Headers verwenden.....518
IBM MQ unter IBM i mit Java und JMS einrichten.....520
C++-Anwendungen entwickeln.....527
Beispielprogramme in C++.....530
Überlegungen zur Programmiersprache C++.....535
Messaging in C++.....539
IBM MQ C++-Programme erstellen.....545
.NET-Anwendungen entwickeln.....557
Erste Schritte mit IBM MQ classes for .NET.....558
IBM MQ .NET-Programme schreiben und implementieren.....573
Component Object Model-Schnittstelle verwenden (IBM MQ Automation Classes for ActiveX).....608
Design und Programmierung mithilfe von IBM MQ Automation-Klassen für ActiveX.....609
Verweis auf IBM MQ Automation Classes for ActiveX.....614
Tracing von IBM MQ Automation Classes for ActiveX.....685
ActiveX-Schnittstelle zur MQAI.....691
Beispielprogramme zur Einführung in IBM MQ Automation Classes for ActiveX.....700
AMQP-Clientanwendungen entwickeln.....705
MQ Light und AMQP (Advanced Message Queuing Protocol).....706
AMQP 1.0-Unterstützung.....707
AMQP- und IBM MQ-Nachrichtenfelder zuordnen.....708
Zuverlässigkeit der Nachrichtenübermittlung mit AMQP.....715
Topologien für AMQP-Clients mit IBM MQ.....717
REST-Anwendungen mit IBM MQ entwickeln.....722
Messaging mit der REST API.....723
Web-Services mit IBM MQ bridge for HTTP entwickeln.....728
MQI-Anwendungen mit IBM MQ entwickeln.....739
IBM MQ-Datendefinitionsdateien.....740
Prozedurale Anwendungen für die Warteschlangensteuerung erstellen.....743
Prozedurale Clientanwendungen schreiben.....953

Benutzerexits, API-Exits und installierbare IBM MQ-Services ... 978
Prozedurale Anwendung erstellen ... 1051
Prozedurale Programmfehler handhaben ... 1103
Multicast-Programmierung ... 1109
Codierung in C ... 1116
Codierung in Visual Basic ... 1119
Codierung in COBOL ... 1120
Codierung in Assemblersprache System/390 (Schnittstelle für Nachrichtenwarteschlangen) ... 1120
IBM MQ-Programme in RPG codieren (nur IBM i) ... 1123
Codierung in PL/I (nur unter z/OS) ... 1124
Prozedurale IBM MQ-Beispielprogramme verwenden ... 1124
Anwendungen für MQ Telemetry entwickeln ... 1300
IBM MQ Telemetry Transport-Beispielprogramme ... 1300
Konzepte zur Programmierung von MQTT-Clients ... 1302
Microsoft Windows Communication Foundation-Anwendungen mit IBM MQ entwickeln ... 1326
Einführung in die Verwendung des angepassten IBM MQ-Kanals für WCF mit .NET 3 ... 1327
Angepasste IBM MQ-Kanäle für WCF verwenden ... 1333
WCF-Beispiele verwenden ... 1354
Problembestimmung im angepassten WCF-Kanal für IBM MQ ... 1360
Web-Services mit IBM MQ entwickeln ... 1368
Web-Services mit IBM MQ-Transport für SOAP entwickeln ... 1369

Bemerkungen ... 1439
Informationen zu Programmierschnittstellen ... 1440
Marken ... 1441

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.

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

- LearnMQ *(Grundlagen lernen, Demo ausführen, App codieren, mehr erweiterte Lernprogramme nutzen)*
- MQ Developer-Downloads *(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 6
- „Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 48

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
- ActiveX (veraltet; verwenden Sie .NET)

Weitere Informationen hierzu finden Sie im Abschnitt „Objektorientierte Anwendungen” auf Seite 11.

.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)
- C + +
- Visual Basic

(siehe „IBM MQ .NET-Programme schreiben und implementieren” auf Seite 573).

ULW IBM MQ unterstützt auch die MQ Light-API, die das Protokoll OASIS AMQP 1.0 implementiert. Es gibt Messaging-APIs für folgende Sprachen:

- Node.js
- Ruby
- Java
- Python
- Maven (Projektentwurf; verwendet die Java-API)
- Gradle (Projektentwurf; verwendet die Java-API)

Weitere Informationen hierzu finden Sie im Abschnitt „AMQP-Clientanwendungen entwickeln” auf Seite 705.

Die folgenden programmiersprachenbezogenen Bindungen werden wie folgt bereitgestellt:

- eine Go-Bindung

© Copyright IBM Corp. 2007, 2023

- 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:

- V 9.0.4 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 722 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)
- IBM i RPG (nur IBM MQ for IBM i)
- z/OS PL/I (nur IBM MQ for z/OS)

Diese Sprachen rufen über die Schnittstelle für Nachrichtenwarteschlangen (MQI) Services zur Steuerung von Nachrichtenwarteschlangen auf. (siehe „MQI-Anwendungen mit IBM MQ entwickeln" auf Seite 739). 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.

Zugehörige Konzepte

„Microsoft Windows Communication Foundation-Anwendungen mit IBM MQ entwickeln" auf Seite 1326
Der angepasste WCF-Kanal (WCF = Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

Zugehörige Tasks

„Anwendungen für MQ Telemetry entwickeln" auf Seite 1300

Zugehörige Informationen

IBM Message Service Client for .NET

Konzepte für die Anwendungsentwicklung

Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

Machen Sie sich, bevor Sie mit Design und Schreiben von IBM MQ-Anwendungen beginnen, mit den IBM MQ-Basiskonzepten vertraut. Weitere Informationen erhalten Sie unter Technische Übersicht. Informationen zu den Anwendungsarten, die Sie für IBM MQ schreiben können, finden Sie im Abschnitt „Anwendungen für IBM MQ entwickeln” auf Seite 5.

Unter den folgenden Links erhalten Sie weitere Informationen zu konkreten IBM MQ-Konzepten für die Anwendungsentwicklung:

Zugehörige Konzepte

„MQI in einer Clientanwendung verwenden” auf Seite 954
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.

„Kanalexitprogramme für Messaging-Kanäle” auf Seite 1010
In dieser Themengruppe finden Sie Informationen zu IBM MQ-Kanalexitprogrammen für Nachrichtenkanäle.

„Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 48
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 743
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 953
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„MQI-Anwendungen mit IBM MQ entwickeln” auf Seite 739
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.

„Objektorientierte Anwendungen” auf Seite 11
IBM MQ unterstützt .NET, ActiveX, C++, Java und JMS. 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.

„IBM MQ classes for JMS verwenden” auf Seite 77
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

„Component Object Model-Schnittstelle verwenden (IBM MQ Automation Classes for ActiveX)” auf Seite 608
Bei IBM MQ Automation Classes for ActiveX (MQAX) handelt es sich um ActiveX-Komponenten, mit denen Klassen bereitgestellt werden, die Sie in Ihrer Anwendung für den Zugriff auf IBM MQ verwenden können.

„IBM MQ classes for Java verwenden” auf Seite 337
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.

„.NET-Anwendungen entwickeln” auf Seite 557
Mit IBM MQ classes for .NET kann ein Programm, das mit einem Framework zur .NET-Programmierung geschrieben wurde, eine Verbindung zu IBM MQ als IBM MQ MQI client oder eine direkte Verbindung zu einem IBM MQ-Server herstellen.

„C++-Anwendungen entwickeln” auf Seite 527
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 1051
Sie können eine IBM MQ-Anwendung in einer von mehreren prozeduralen Sprachen schreiben und die Anwendung auf mehreren unterschiedlichen Plattformen ausführen.

Zugehörige Tasks

„Web-Services mit IBM MQ entwickeln” auf Seite 1368
Sie können IBM MQ-Anwendungen für Web-Services mithilfe des IBM MQ-Transports für SOAP entwickeln.

„Prozedurale IBM MQ-Beispielprogramme verwenden” auf Seite 1124
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

Zugehörige Informationen

Transaktionsunterstützungsszenarios

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

Multi

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 IBM i IBM i-, TXSeries for AIX-, HP-UX-, Solaris- und Windows-Systeme verwenden.
- Message-Queuing in Encina für AIX-, HP-UX-, Solaris- und Windows-Systeme verwenden.
- Verwenden Sie Message-Queuing aus Tuxedo für AIX-, AT & T-, HP-UX-, Solaris-und Windows -Systeme.
- 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

z/OS

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. Beispiel:

- 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.
- In der IMS-Umgebung unterstützt IBM MQ for z/OS Programme zur Verarbeitung von Onlinenachrichten (MPPs), interaktive Direktaufrufprogramme (IFPs) und Programme zur Verarbeitung von Stapelnachrichten (BMPs). Beachten Sie beim Schreiben von DL/I-Programmen die Anweisungen in den Abschnitten „z/OS-Batchanwendungen erstellen” auf Seite 1089 und „Hinweise zu z/OS-Stapelprogrammen” auf Seite 756 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 finden Sie im 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 „Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927.

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

„Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 48
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 743
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 953
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„IBM MQ classes for JMS verwenden” auf Seite 77
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

„Component Object Model-Schnittstelle verwenden (IBM MQ Automation Classes for ActiveX)” auf Seite 608
Bei IBM MQ Automation Classes for ActiveX (MQAX) handelt es sich um ActiveX-Komponenten, mit denen Klassen bereitgestellt werden, die Sie in Ihrer Anwendung für den Zugriff auf IBM MQ verwenden können.

„IBM MQ classes for Java verwenden” auf Seite 337
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.

„.NET-Anwendungen entwickeln” auf Seite 557
Mit IBM MQ classes for .NET kann ein Programm, das mit einem Framework zur .NET-Programmierung geschrieben wurde, eine Verbindung zu IBM MQ als IBM MQ MQI client oder eine direkte Verbindung zu einem IBM MQ-Server herstellen.

„Microsoft Windows Communication Foundation-Anwendungen mit IBM MQ entwickeln” auf Seite 1326
Der angepasste WCF-Kanal (WCF = Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

„C++-Anwendungen entwickeln” auf Seite 527
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 1051
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 1124
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

Zugehörige Informationen

Sicherung

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.

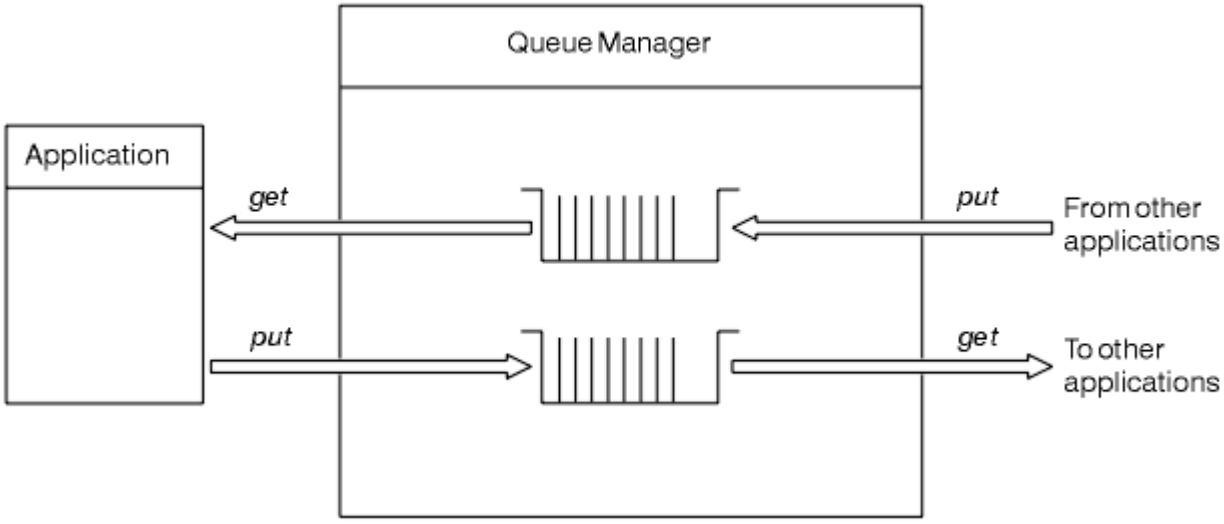


Abbildung 1. Warteschlangen, Nachrichten und Anwendungen

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 1051).
- 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.

Objektorientierte Anwendungen

IBM MQ unterstützt .NET, ActiveX, C++, Java und JMS. 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 12.

.NET

Informationen zum Codieren von .NET-Programmen mit den IBM MQ .NET-Klassen finden Sie unter .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.

ActiveX

IBM MQ ActiveX wird allgemein auch als MQAX bezeichnet. MQAX ist Bestandteil von IBM MQ for Windows. Die Unterstützung von ActiveX wurde auf Versionsebene IBM WebSphere MQ 6.0 stabilisiert. Informationen zum Codieren von Programmen mithilfe des IBM MQ -Objektmodells in ActiveX Component Object Model-Schnittstelle verwenden (WebSphere MQ Automation Classes for ActiveX).

V 9.0.0 Ab IBM MQ 9.0 wird die Unterstützung für MicrosoftActive X nicht mehr verwendet. IBM MQ-Klassen für .NET sind die empfohlene Ersatztechnologie. Weitere Informationen finden Sie unter .NET-Anwendungen entwickeln.

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).

Java
Informationen zum Codieren von Programmen mit dem IBM MQ-Objektmodell in Java finden Sie unter IBM MQ classes for Java verwenden. IBM führt keine weiteren Erweiterungen für IBM MQ classes for Java durch und sie werden funktional auf der Ebene stabilisiert, die im Lieferumfang von IBM MQ 8.0 enthalten ist. 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 „IBM MQ classes for Java oder IBM MQ classes for JMS verwenden” auf Seite 51.

JMS
IBM MQ stellt zudem Klassen bereit, die die Java Message Service-Spezifikation (JMS-Spezifikation) implementieren. Details zu den IBM MQ classes for JMS finden Sie im Abschnitt IBM MQ classes for JMS verwenden. 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 „IBM MQ classes for Java oder IBM MQ classes for JMS verwenden” auf Seite 51.

IBM Message Service Client for C/C++ und IBM Message Service Client 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 im Abschnitt Einführung in den IBM Message Service Client for .NET.

Zugehörige Konzepte

„MQI-Anwendungen mit IBM MQ entwickeln” auf Seite 739
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.

„Konzepte für die Anwendungsentwicklung” auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

Zugehörige Informationen

Technische Übersicht

Verweis auf die Anwendungsentwicklung

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.

Multi **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.

Rückgabecodes

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 14 stellt eine Nachricht dar und zeigt deren logische Aufteilung in Nachrichteneigenschaften und Anwendungsdaten.

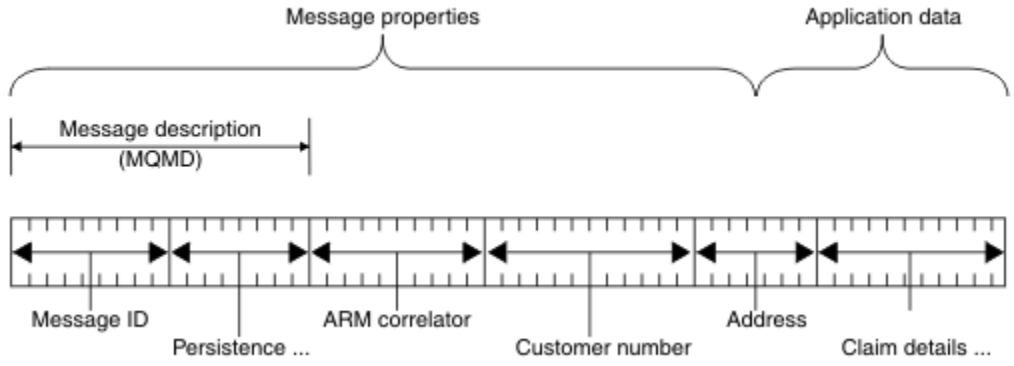


Abbildung 2. Darstellung einer Nachricht

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.

ULW Unter UNIX, 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 1070.

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 im Abschnitt „Die Daten der Nachricht" auf Seite 786.

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 45.

Es gibt verschiedene Versionen des Nachrichtendeskriptors. Weitere Informationen zur Gruppierung und Segmentierung von Nachrichten (siehe „Nachrichtengruppen" auf Seite 41) 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 Nachrichtenbestätigung 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*

Nicht unterstützt unter IBM MQ for z/OS.

Wenn Sie für eine segmentierte Nachricht die Generierung von Berichten anfordern, erhalten Sie unter Umständen mehr Berichte als bei einer nicht segmentierten Nachricht.

Segmentierte Nachrichten werden im Abschnitt „Nachrichtensegmentierung” auf Seite 822 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 1033). 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 1033.

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 771), 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: Eine JMS-Anwendung kann über den JMS-Aufruf `message.getStringProperty('myproperty')` auf die IBM MQ-Eigenschaft *usr.myproperty* (eine Zeichenfolge) zugreifen. 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 eine in einer JMS-Anwendung festgelegte Benutzereigenschaft in einer IBM MQ-Anwendung aufgerufen werden, indem Sie 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.

Wenn eine Nachricht an einen Warteschlangenmanager einer früheren Version als IBM WebSphere MQ 7.0 geleitet wird, zählen die Eigenschaften der Nachricht (mit Ausnahme der Eigenschaften im Nachrichtendeskriptor) zur Länge der Nachricht. Erhöhen Sie also entsprechend den Wert des Attributs *MaxMsgLength* von Kanälen, die in ein System vor IBM WebSphere MQ 7.0 führen, damit bei Bedarf mehr Daten für jede Nachricht gesendet werden können. Alternativ können Sie auch die maximale Nachrichtenlänge (*MaxMsgLength*) der Warteschlange oder des Warteschlangenmanagers verringern, damit die allgemeine Datenmenge, die über das System gesendet wird, gleich bleibt.

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 co-dierte Zeichenfolge)	Root .MQMD.ReplyToQ oder Root .MQMD.ReplyToQMgr oder jms.Rto
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

Mithilfe dieser Synonyme kann eine MQI-Anwendung ähnlich wie eine IBM MQ classes for JMS-Clien-tanwendung auf JMS-Eigenschaften zugreifen. Von diesen Eigenschaften können nur JMSCorrelation-ID, 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 " mq " (in beliebiger Groß-/Kleinschreibung) und nicht mit "mq_usr" beginnt, darf nur ein U+002E-Zeichen (".") enthalten. Mehrere "."-Zeichen sind in Eigenschaften mit diesen Präfixwerten nicht zulässig.
4. Zwischen zwei "."-Zeichen müssen andere Zeichen stehen. Ein Punkt, der keinen Inhalt abgrenzt, ist in der Hierarchie nicht möglich. Ebenso darf ein Eigenschaftsname nicht mit dem Zeichen "." (Punkt) enden.
5. Wenn eine Anwendung die Eigenschaft "a.b" und dann die Eigenschaft "a.b.c" festlegt, ist nicht er-kennbar, 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 erzwun-gen:

- 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*	*Field*

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 29 zeigt den Auswahlprozess unter Verwendung des Aufrufs MQSUB.

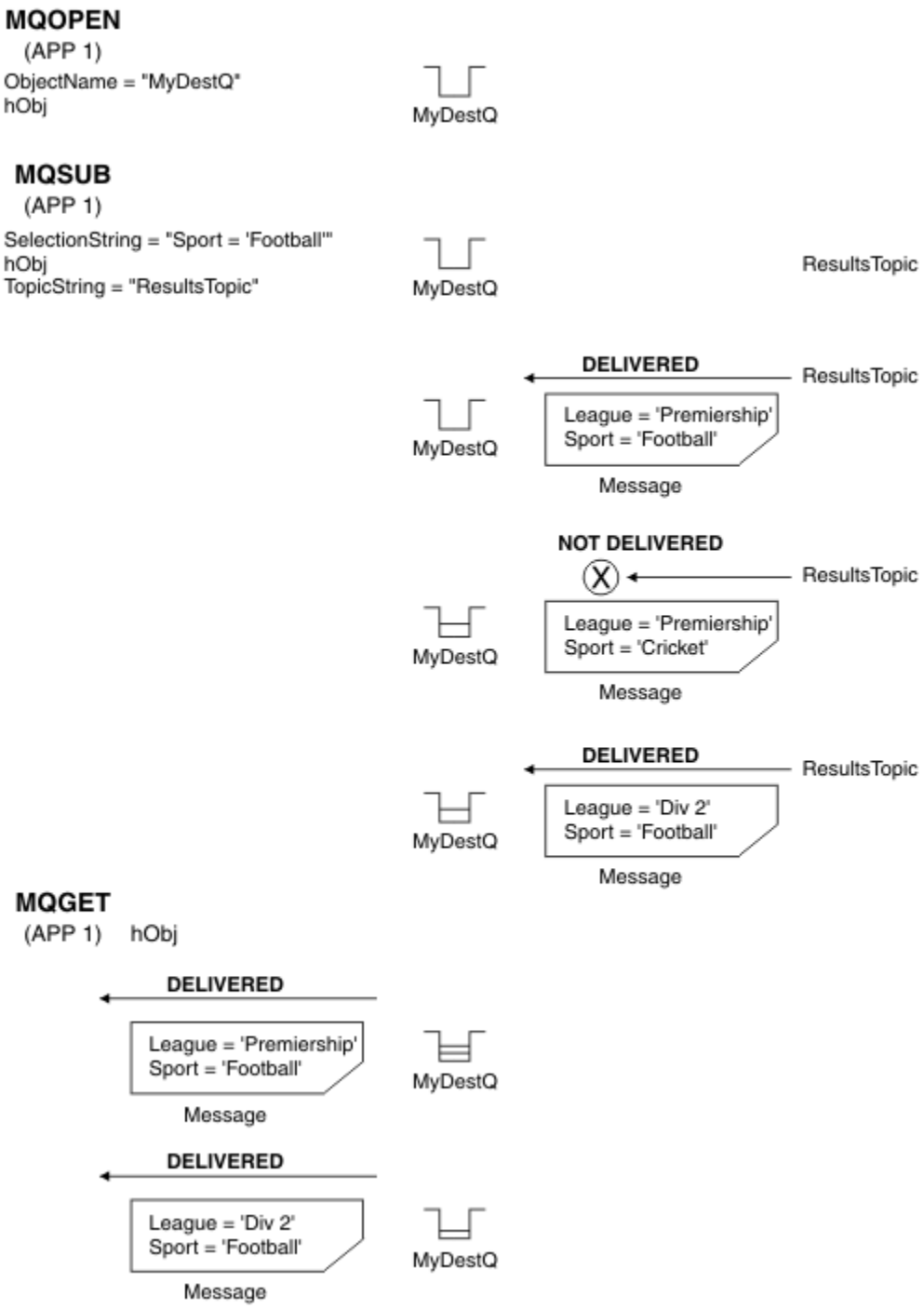


Abbildung 3. Auswahl mithilfe des Aufrufs MQSUB

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 30 zeigt den Auswahlprozess unter Verwendung des Aufrufs MQOPEN.

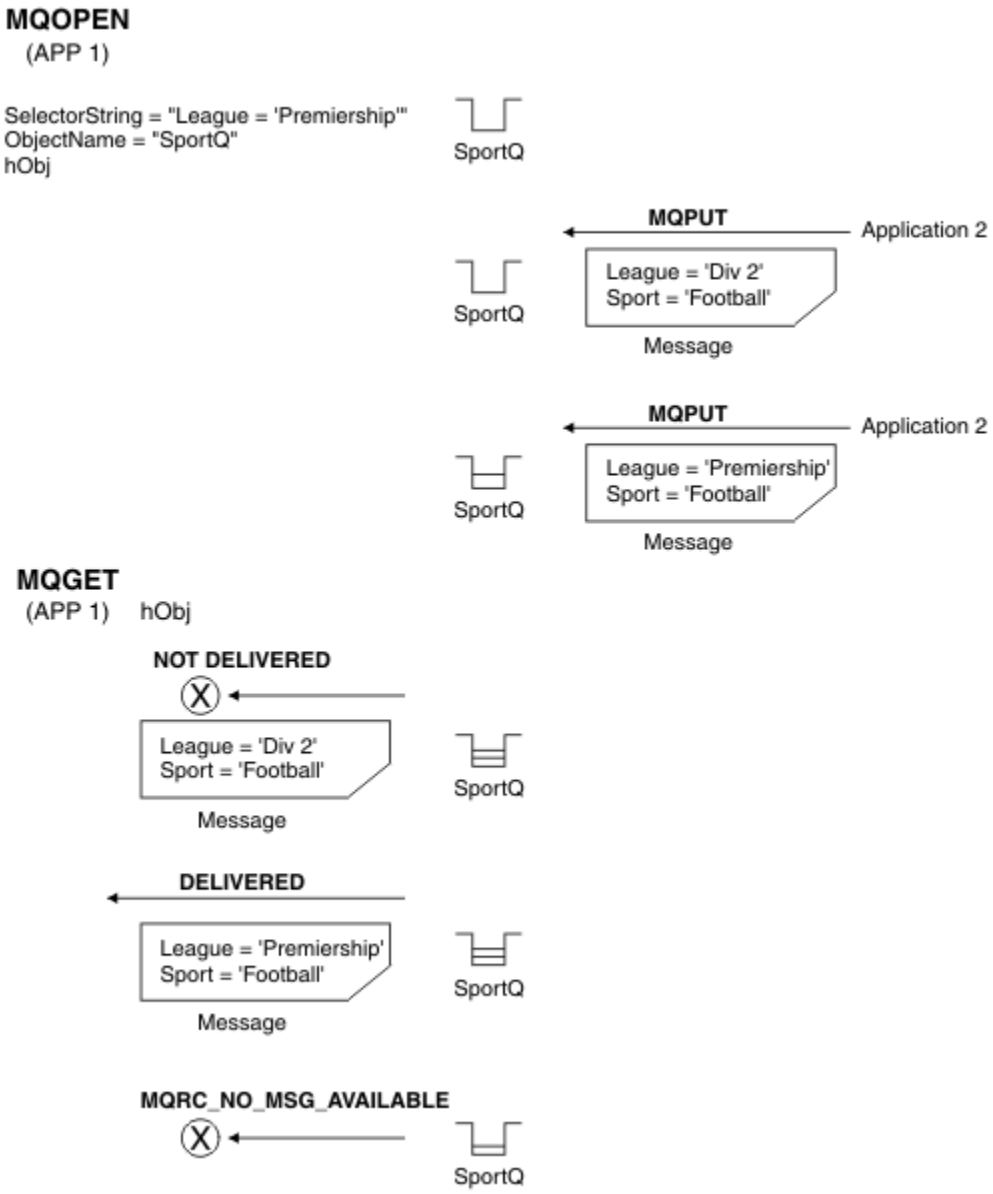


Abbildung 4. Auswahl mithilfe des Aufrufs MQOPEN

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 38

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 133.

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. Beispiel:

(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 38
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: "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 24 und „Einschränkungen bei Eigenschaftsnamen” auf Seite 25.

  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 133. 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. Falls logisch, lautet das Ergebnis der booleschen Operatoren 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. Falls logisch, lautet das Ergebnis der booleschen Operatoren A AND B (Forts.)		
Operator A	**Operator B**	**Ergebnis (A AND B)**
U	T	U
U	U	U
U	F	F

Tabelle 2. Falls logisch, lautet das Ergebnis der booleschen Operatoren 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. Falls logisch, lautet das Ergebnis der booleschen Operatoren 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 23
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 Informationen

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 31
Übersicht über das IBM MQ-Auswahlverhalten.

Zugehörige Informationen
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. V 9.0.0 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 Informationen

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 40 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.

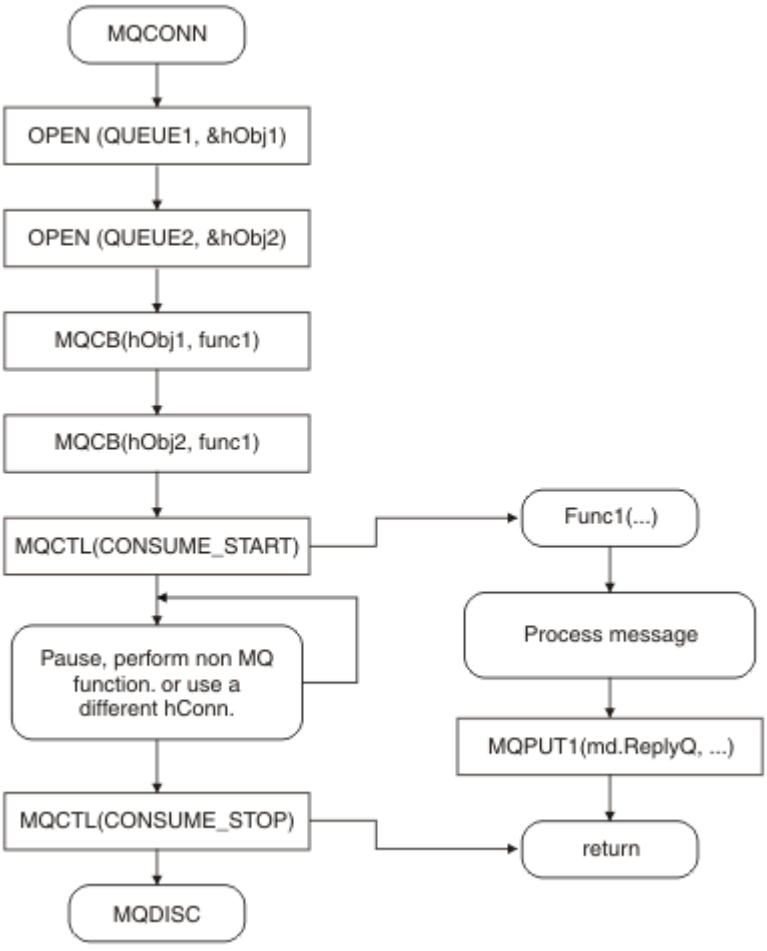


Abbildung 5. Nachrichtengesteuerte Standardanwendung, die Nachrichten aus zwei Warteschlangen verarbeitet

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 41Dieser 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.

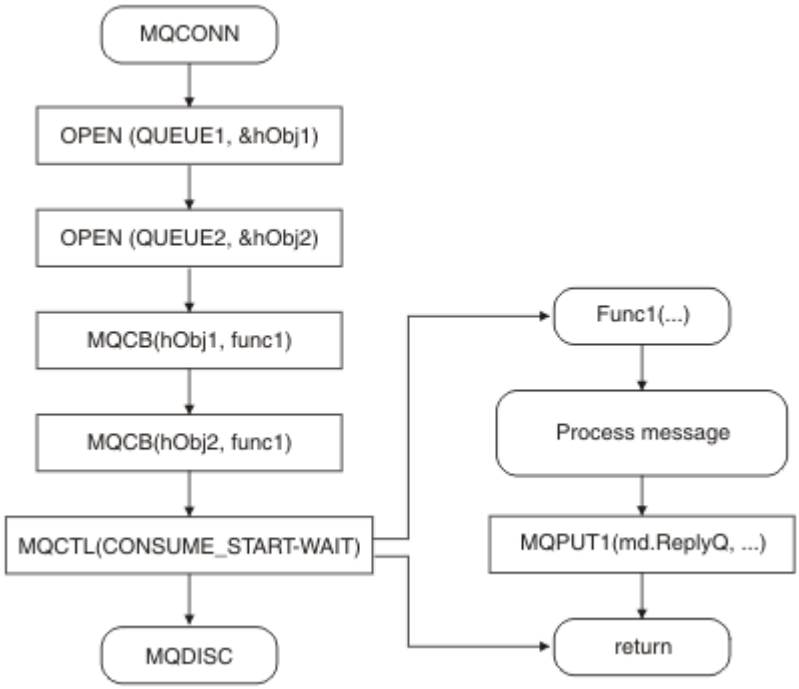


Abbildung 6. Nachrichtengesteuerte Einzelthread-Anwendung, die Nachrichten aus zwei Warteschlangen verarbeitet

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 803). 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 42 zeigt eine Gruppe logischer Nachrichten:

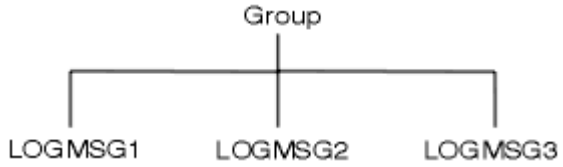


Abbildung 7. Gruppe logischer Nachrichten

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.

Segmentierung

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 im Abschnitt „Nachrichtensegmentierung” auf Seite 822.

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 43 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 43 zeigt eine Gruppe logischer Nachrichten, von denen einige segmentiert sind:

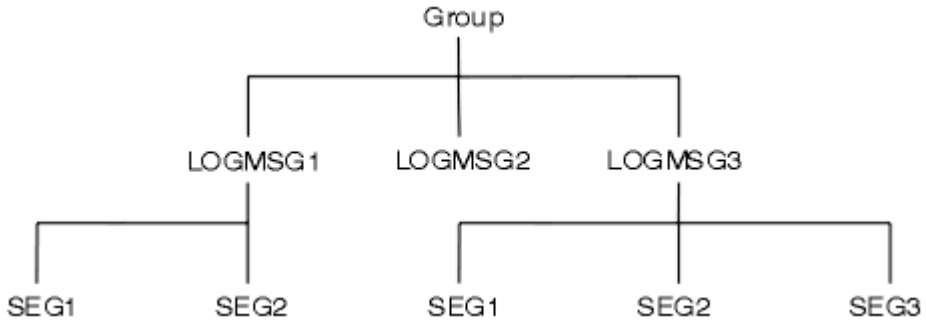


Abbildung 8. Segmentierte Nachrichten

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 822
In diesem Abschnitt erfahren Sie, wie Nachrichten segmentiert werden. Diese Funktion wird in IBM MQ for z/OS oder von Anwendungen, die IBM MQ classes for JMS verwenden, nicht unterstützt.

Zugehörige Verweise

„Logische und physische Reihenfolge" auf Seite 803
Nachrichten in Warteschlangen können (innerhalb jeder Prioritätsstufe) in *physischer* oder *logischer* Reihenfolge auftreten.

Zugehörige Informationen

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. Weitere Informationen hierzu finden Sie unter „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen" auf Seite 889.
- ULW z/OS 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 führen. 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 im Abschnitt „Prozedurale Programmfehler handhaben” auf Seite 1103.

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. Bei allen anderen Konfigurationen unter z/OSmüssen Sie das Attribut *HardenGetBackout* auf MQQA_BACKOUT_HARDENED; setzen, um sicherzustellen, dass der Rücksetzungszähler für private Warteschlangen auch nach einem Neustart des Warteschlangenmanagers erhalten bleibt; andernfalls behält der Warteschlangenmanager nicht für jede Nachricht einen genauen Rücksetzungszähler bei, wenn er erneut gestartet werden muss. 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 889.

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 1107).

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 788.

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).

Bei Nachrichten unter IBM MQ for Windows wird im Feld *AccountingToken* eine Sicherheits-ID (SID) für Windows-Systeme gespeichert. 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 788
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 778
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.

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 1081 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 553 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 1084 vorgehen, bei der Definition des Visual Basic-Projekts im Feld **Argumente für bedingte Kompilierung** jedoch `MqType=3` eingeben.
- Verwendet die Komponente IBM MQ Automation Classes for ActiveX (MQAX), definieren Sie die Umgebungsvariable GMQ_MQ_LIB mit dem Wert `mqic32xa.dll`.

  Sie können die Umgebungsvariable in der Anwendung definieren oder so, dass sie systemweit gilt. Eine systemweit geltende Definition kann allerdings zu nicht ordnungsgemäßem Verhalten von MQAX-Anwendungen führen, bei denen die Umgebungsvariable nicht innerhalb der Anwendung definiert wurde.

IBM MQ mit WebSphere Application Server verwenden

Die Informationen in diesem Abschnitt erläutern die Verwendung von IBM MQ mit WebSphere Application Server.

Anwendungen, die in Java geschrieben wurden und unter WebSphere Application Server ausgeführt werden, können die Java Messaging Service-Spezifikation (JMS-Spezifikation) zur Nachrichtenübertagung verwenden. 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.

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 von prozeduralen und objektorientierten 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. Beispiel:

```
#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 52.

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

Informationen zum automatischen Starten von Programmen oder zum Programmieren von Programmen, damit sie auf eine bestimmte Nachricht in einer Warteschlange warten (mittels der *Triggering*-Funktion von IBM MQ) finden Sie im Abschnitt „IBM MQ-Anwendungen durch Auslöser starten" auf Seite 901). Alternativ können Sie eine andere Instanz der Anwendung starten, wenn die Nachrichten in einer Warteschlange nicht schnell genug verarbeitet werden (mittels der *Instrumentierungsereignis*-Funktion von IBM MQ). Eine Beschreibung finden dazu finden Sie unter Instrumentierungsereignisse).

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 hierzu finden Sie unter „Anwendungen in der IBM MQ MQI client-Umgebung ausführen" auf Seite 962.

Anwendungsleistung
Entwurfsentscheidungen können sich auf die Anwendungsleistung auswirken. Vorschläge zur Leistungsoptimierung von IBM MQ-Anwendungen finden Sie unter „Anwendungsdesign und Leistung” auf Seite 53 IBM i und „Entwurfs- und Leistungsaspekte für Anwendungen unter IBM i” auf Seite 57 .

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 im Abschnitt „Verfahren zum Entwerfen von anspruchsvolleren Anwendungen” auf Seite 55.

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 889). 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 „Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927.

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

„Entwurfs- und Leistungsaspekte für Anwendungen unter z/OS” auf Seite 59
Der Anwendungsentwurf zählt zu den wichtigsten Faktoren hinsichtlich der Leistung. Die Informationen in diesem Abschnitt helfen Ihnen, einige dieser Entwurfsfaktoren besser zu verstehen.

„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 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

„Prozedurale Anwendungen für die Warteschlangensteuerung erstellen” auf Seite 743
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 953

Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„.NET-Anwendungen entwickeln” auf Seite 557
Mit IBM MQ classes for .NET kann ein Programm, das mit einem Framework zur .NET-Programmierung geschrieben wurde, eine Verbindung zu IBM MQ als IBM MQ MQI client oder eine direkte Verbindung zu einem IBM MQ-Server herstellen.

„C++-Anwendungen entwickeln” auf Seite 527
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 verwenden” auf Seite 77
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

„IBM MQ classes for Java verwenden” auf Seite 337
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.

„Component Object Model-Schnittstelle verwenden (IBM MQ Automation Classes for ActiveX)” auf Seite 608
Bei IBM MQ Automation Classes for ActiveX (MQAX) handelt es sich um ActiveX-Komponenten, mit denen Klassen bereitgestellt werden, die Sie in Ihrer Anwendung für den Zugriff auf IBM MQ verwenden können.

Zugehörige Informationen

IBM MQ - Technische Übersicht

IBM MQ classes for Java oder IBM MQ classes for JMS verwenden

Eine Java-Anwendung kann entweder IBM MQ classes for Java oder IBM MQ classes for JMS für den Zugriff auf IBM MQ-Ressourcen verwenden. Jede Methode hat ihre Vorteile.

Anmerkung: 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.

IBM MQ classes for Java bindet die MQI (Message Queue Interface, Schnittstelle für Nachrichtenwarteschlange), die native IBM MQ-API, ein und verwendet dasselbe Objektmodell wie andere objektorientierte Schnittstellen, während IBM MQ classes for Java Message Service die Java Message Service-Schnittstellen (JMS-Schnittstellen) von Oracle implementiert.

Wenn Sie mit IBM MQ in anderen Umgebungen als Java vertraut sind, die entweder prozedurale oder objektorientierte Sprachen verwenden, können Sie Ihr Wissen mittels IBM MQ classes for Java auf die Java-Umgebung übertragen. Außerdem können Sie sämtliche Funktionen von IBM MQ nutzen (in IBM MQ classes for JMS stehen sie nicht alle zur Verfügung).

Wenn Sie nicht mit IBM MQ vertraut sind oder bereits über Erfahrung mit JMS verfügen, fällt es Ihnen wahrscheinlich leichter, mit der bekannten JMS-API und mittels IBM MQ classes for JMS auf IBM MQ-Ressourcen zuzugreifen. JMS ist auch ein integraler Bestandteil der Java Platform, Enterprise Edition-Plattform (Java EE). Java EE-Anwendungen können Nachrichten unter Verwendung von Message-driven Beans (MDBs) asynchron verarbeiten. JMS ist zudem der Standardmechanismus für die Interaktion zwischen Java EE und Systemen zur asynchronen Nachrichtenübermittlung wie IBM MQ. Jeder Java EE-konforme Anwendungsserver muss einen JMS-Provider enthalten. Daher können Sie JMS für die Kommunikation zwischen verschiedenen Anwendungsservern verwenden oder eine Anwendung aus einem JMS-Provider in einen anderen Provider portieren, ohne die Anwendung ändern zu müssen.

„IBM MQ classes for Java verwenden” auf Seite 337
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.

„IBM MQ classes for JMS verwenden” auf Seite 77
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

Szenarios: WebSphere Application Server mit IBM MQ

Szenarios: WebSphere Application Server Liberty-Profil mit IBM MQ

Verfahren zum Entwerfen von Nachrichten

Berücksichtigen Sie beim Entwerfen von Nachrichten die nachfolgenden Aspekte.

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 UNIX 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.

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 827).
- 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 im Abschnitt „Signalisierung” auf Seite 828.
- Durch Erstellen eines Callback-Exits, der bei Eingang einer Nachricht ausgelöst wird; siehe „Asynchrone Verarbeitung von IBM MQ-Nachrichten” auf Seite 39.

- 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 45.

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 901. 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.

Multi 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 16 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 unter Handhabung von Nachrichtenaffinitäten.

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 57
- „Spezielle Leistungsprobleme” auf Seite 58

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 48.

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 mit der Nachrichten- und Korrelations-ID (*MsgId* und *CorrelId*) im Nachrichtendeskriptor eine bestimmte Nachricht angeben, muss der Warteschlangenmanager die gesamte Warteschlange durchsuchen, bis er die betreffende Nachricht findet. Diese Verwendung des MQGET-Aufrufs beeinträchtigt die Leistung Ihrer Anwendung.

Warteschlangen mit Nachrichten verschiedener Länge
Wenn die Warteschlange Nachrichten in unterschiedlichen Längen enthält, kann Ihre Anwendung die jeweilige Länge mit dem MQGET-Aufruf ermitteln. Dabei muss das Feld *BufferLength* auf null gesetzt werden, damit die Größe der Nachrichtendaten zurückgegeben wird, selbst 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 ist 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, belegen Sie möglicherweise den Speicherbereich, der mit den IBM MQ for IBM i-Warteschlangenmanagern verbunden 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 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 48

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.

Entwurfs- und Leistungsaspekte für Anwendungen unter z/OS

Der Anwendungsentwurf zählt zu den wichtigsten Faktoren hinsichtlich der Leistung. Die Informationen in diesem Abschnitt helfen Ihnen, einige dieser Entwurfsfaktoren besser zu verstehen.

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 folgenden Abschnitten werden einige Probleme erläutert, die speziell bei Programmen auftreten, welche MQI-Aufrufe ausführen.

Weitere Informationen zum Anwendungsentwurf finden Sie unter „Konzepte für den Entwurf von IBM MQ-Anwendungen" auf Seite 48.

Auswirkung der Nachrichtenlänge

Obwohl die Datenmenge von Nachrichten unter IBM MQ for z/OS 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. 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 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.

Daten für persistente Nachrichten werden in Protokollpuffer geschrieben. Diese Puffer wiederum werden in folgenden Fällen die Protokolldatensätze geschrieben:

- Ein Commit ist aufgetreten.
- Eine Nachricht wird außerhalb des Synchronisationspunkts abgerufen oder eingereiht.
- WRTHRSH-Puffer werden gefüllt.

Die Verarbeitung vieler Nachrichten in einer Arbeitseinheit kann weniger Ein-/Ausgabe verursachen, als wenn die Nachrichten einzeln für jede Arbeitseinheit verarbeitet werden oder außerhalb des Synchronisationspunkts liegen.

Bestimmte Nachrichten suchen

Normalerweise wird mit dem MQGET-Aufruf die erste Nachricht aus einer Warteschlange abgerufen. 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 der Anwendung aus, da IBM MQ möglicherweise die gesamte Warteschlange durchsuchen muss, um eine bestimmte Nachricht zu finden.

Mit dem Warteschlangenattribut **IndexType** können Sie angeben, dass der Warteschlangenmanager einen Index verwalten soll, damit die Geschwindigkeit von MQGET-Operationen in der Warteschlange erhöht werden kann. Die Leistung für die Indexverwaltung wird dadurch allerdings ein wenig beeinträchtigt. Generieren Sie einen Index also nur dann, wenn Sie in unbedingt verwenden müssen. Sie können wahlweise einen Index mit Nachrichten-IDs oder mit Korrelations-IDs erstellen oder für Warteschlangen, aus denen Nachrichten nacheinander abgerufen werden, keinen Index erstellen. Arbeiten Sie mit vielen verschiedenen Schlüsselwerten anstatt mit vielen gleichen Werten, zum Beispiel 'Balance1', 'Balance2' und 'Balance3' (anstelle von drei Werten mit 'Balance'). Bei gemeinsam genutzten Warteschlangen müssen Sie den richtigen **IndexType** verwenden. Details zum Warteschlangenattribut **IndexType** finden Sie unter Indextyp.

Verwenden Sie den Parameter QINDXBLD(NOWAIT) im CSQ6SYSP-Makro, um zu vermeiden, dass die Zeit für den Neustart des Warteschlangenmanagers durch die Verwendung von indizierten Warteschlangen beeinträchtigt wird. So kann der Neustart des Warteschlangenmanagers durchgeführt werden, ohne auf den Abschluss der Warteschlangenindexerstellung zu warten.

Eine ausführliche Beschreibung des Attributs **IndexType** und anderer Objektattribute finden Sie im Abschnitt Attribute von Objekten.

Warteschlangen mit Nachrichten verschiedener Länge

Rufen Sie eine Nachricht ab, indem Sie eine Puffergröße verwenden, die der erwarteten Größe der Nachricht entspricht. Wenn Ihnen ein Rückgabecode meldet, dass die Nachricht zu lang ist, verwenden Sie einen größeren Puffer. Schlägt der Abruf auch dann fehl, entspricht die zurückgegebene Datenlänge der Größe der nicht konvertierten Nachrichtendaten. Wenn Sie MQGMO_CONVERT im Aufruf MQGET angeben und die Datenmenge während der Konvertierung zunimmt, ist sie unter Umständen immer noch zu groß für den Puffer. Erhöhen Sie in diesem Fall noch einmal die Puffergröße.

Wenn Sie den Aufruf MQGET mit der Pufferlänge Null ausgeben, wird die Größe der Nachricht zurückgegeben. Die Anwendung erhält dann einen Puffer dieser Größe und kann den Aufruf erneut ausgeben. Wenn die Warteschlange von mehreren Anwendungen verarbeitet wird, wurde die Nachricht eventuell bereits von einer anderen Anwendung verarbeitet, als die ursprüngliche Anwendung den Aufruf erneut ausgab. Wenn Sie nur gelegentlich mit großen Nachrichten arbeiten, benötigen Sie für diese unter Umständen einen größeren Puffer, den Sie nach Verarbeitung der Nachrichten wieder freigeben. Probleme mit dem virtuellen Speicher sollten auf diese Weise verringert werden, wenn alle Anwendungen über große Puffer verfügen.

Wenn Ihre Anwendung keine Nachrichten einer festen Länge verwenden kann, können Sie zur Lösung dieses Problem auch den MQINQ-Befehl ausführen, um die maximale Nachrichtengröße festzustellen, die von der Warteschlange akzeptiert werden kann. Diesen Wert können Sie dann im MQGET-Aufruf verwenden. Die maximale Größe von Nachrichten für eine Warteschlange ist im Attribut **MaxMsgL** der Warteschlange gespeichert. Diese Methode könnte jedoch große Speichermengen verwenden, da der Wert von **MaxMsgL** bis zu 100 MB betragen kann, was dem von IBM MQ for z/OS zugelassenen Maximum entspricht.

Anmerkung: Sie können den Parameter **MaxMsgL** verkleinern, nachdem große Nachrichten in die Warteschlange gestellt wurden, zum Beispiel: Sie stellen eine Nachricht mit 100 MB ein und setzen dann den Wert des Parameters **MaxMsgL** auf 50 Bytes. Somit können weiterhin Nachrichten abgerufen werden, die größer sind als von der Anwendung erwartet.

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 hinsichtlich des Speichers und auch in der Form, dass Threads durch Tasks blockiert werden, die versuchen, Nachrichten abzurufen.

Wenn eine Warteschlange von mehreren Anwendungen verarbeitet wird, erhalten Sie in der Regel die beste Leistung, wenn Sie entweder

- 100 kurze Nachrichten (unter 1 KB) oder
- eine Nachricht für größere Nachrichten (100 KB)

für jeden Synchronisationspunkt verwenden. Wird die Warteschlange nur von einer Anwendung verarbeitet, müssen mehr Nachrichten pro Arbeitseinheit verwendet werden.

Sie können die Anzahl der Nachrichten, die eine Task in einer einzelnen Arbeitseinheit mit Wiederherstellung abrufen oder dort einstellen kann, mit dem Warteschlangenmanagerattribut **MAXUMSGS** begrenzen. Informationen zu diesem Attribut finden Sie in der Beschreibung des Befehls **ALTER QMGR** unter Scriptbefehle (MQSC).

Vorteile des MQPUT1-Aufrufs

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.

Wie viele Nachrichten kann ein Warteschlangenmanager enthalten?

Lokale Warteschlangen

Die Anzahl der lokalen Nachrichten, die ein Warteschlangenmanager enthalten kann, entspricht hauptsächlich der Größe der Seitengruppen. Bis zu 100 Seitengruppen sind möglich (es wird allerdings empfohlen, Seitengruppe 0 und Seitengruppe 1 für systembezogene Objekte und Warteschlangen zu verwenden). Sie können eine Seitengruppe mit erweitertem Format verwenden und die Kapazität einer Seitengruppe erhöhen.

Gemeinsam genutzte Warteschlangen

Die Kapazität für gemeinsam genutzte Warteschlangen hängt von der Größe der Coupling-Facility (CF) ab. IBM MQ verwendet CF-Listenstrukturen, deren grundlegende Speichereinheiten aus Einträgen und Elementen bestehen. Jede Nachricht wird als 1 Eintrag und mehrere Elemente gespeichert, die den zugehörigen MQMD und andere Nachrichtendaten enthalten. Die Anzahl der Elemente, die eine einzelne Nachricht verarbeitet, hängt von der Größe der Nachricht und bei CFLEVEL(5) von den während des MQPUT-Aufrufs gültigen Auslagerungsregeln ab. Wenn Nachrichtendaten entweder in Db2 oder SMDS ausgelagert werden, sind weniger Elemente erforderlich. Der Zugriff auf Nachrichtendaten verlangsamt sich, wenn die Nachricht ausgelagert wurde. Einen ausführlicheren Vergleich von Leistung und CPU-Aufwand im Zusammenhang mit der Nachrichtenauslagerung finden Sie im Performance Supportpac MP1H.

Faktoren, die sich auf die Leistung auswirken

Als Leistung wird u.a. bezeichnet, wie schnell Nachrichten verarbeitet werden oder wie viel CPU pro Nachricht benötigt wird.

Faktoren, die sich darauf auswirken, wie schnell Nachrichten verarbeitet werden

Bei persistenten Nachrichten wirkt sich die Geschwindigkeit der Protokolldateien am meisten aus. Die Geschwindigkeit der Protokolldaten hängt von dem DASD ab, in dem sie sich befinden. Achten Sie also darauf, Protokolldateien auf weniger verwendeten Datenträgern zu speichern, um Konflikte zu reduzieren. Einheitenübergreifendes Lesen und Schreiben der MQ-Protokolldaten verbessert die Protokollleistung, wenn mehrere Seiten pro Ein-/Ausgabe geschrieben werden. Auch eine Z High Performance Fibre-Verbindung (zHPF) wirkt sich erheblich auf die Leistung für die E/A-Antwortzeit aus, wenn das E/A-Subsystem aktiv ist.

Wenn das Abrufen und Einreihen einer Nachricht angefordert werden, ist der Zugriff auf die Warteschlange gesperrt, um die Integrität der Warteschlange zu wahren. Berücksichtigen Sie bei Ihrer Planung, dass die Warteschlange für die Dauer der gesamten Anforderung gesperrt ist. Bei einer Dauer von 100 Mikrosekunden pro Einstellvorgang und über 10.000 Anforderungen pro Sekunde kommt es also unter Umständen zu Verzögerungen. Möglicherweise erzielen Sie in der Praxis bessere Ergebnisse, doch dieses Beispiel eignet sich gut als allgemeine Regel. Zur Leistungsoptimierung können Sie verschiedene Warteschlangen verwenden.

Mögliche Ursachen hierfür können sein:

- Verwendung einer allgemeinen Antwortwarteschlange, die jede CICS-Transaktion verwendet
- Jede CICS-Transaktion erhält eine eindeutige Empfangswarteschlange für Antworten.
- Eine Empfangswarteschlange für Antworten für die CICS-Region und alle Transaktionen in der CICS-Region verwenden diese Warteschlange.

Die Antwort hängt von der Anzahl der Anforderungen pro Sekunde und der Antwortzeit der Anforderungen ab.

Nachrichten aus einer Seitengruppe werden langsamer gelesen als Nachrichten aus einem Pufferpool. Wenn mehr Nachrichten vorliegen, als der Pufferpool aufnehmen kann, laufen Sie auf den Datenträger über. Stellen Sie also sicher, dass der Pufferpool groß genug für Ihre kurzlebigen Nachrichten ist. Wenn Sie Nachrichten erst viele Stunden später verarbeiten möchten, laufen diese wahrscheinlich auf den Datenträger über und werden also langsamer abgerufen als aus einem Pufferpool.

Bei einer gemeinsam genutzten Warteschlange hängt die Geschwindigkeit der Nachrichten von der Geschwindigkeit der Coupling-Facility ab. Eine CF innerhalb des physischen Prozessors ist sehr wahrscheinlich schneller als eine externe CF. Die CF-Antwortzeit hängt vom Auslastungsgrad der CF ab. Beispiel: Als die CF in den Hursley-Systemen zu 17 % ausgelastet war, betrug die Antwortzeit 14 Mikrosekunden. Bei einer CF-Auslastung von 95 % betrug die Antwortzeit 45 Mikrosekunden.

Wenn Ihre MQ-Anforderungen die CPU sehr in Anspruch nehmen, kann sich dies darauf auswirken, wie schnell die Nachrichten verarbeitet werden, da die Anwendungen länger auf CPU-Kapazitäten warten müssen, wenn die logische Partition (LPAR) die CPU belegt.

Wie viel CPU pro Nachricht?

Im Allgemeinen benötigen größere Nachrichten mehr CPU-Kapazitäten. Vermeiden Sie daher große Nachrichten (x MB), wenn möglich.

Beim Abrufen bestimmter Nachrichten aus Warteschlangen sollte die Warteschlange indiziert sein, damit der Warteschlangenmanager die Nachricht direkt abrufen kann (und so unter Umständen vermeidet, dass die gesamte Warteschlange durchsucht werden muss). Ist die Warteschlange nicht indiziert, wird sie komplett auf die Nachricht durchsucht. Befinden sich also z. B. 1000 Nachrichten in der Warteschlange, müssen sie alle überprüft werden. Dies führt zu einer unnötigen CPU-Auslastung.

Kanäle, die TLS verwenden, verursachen aufgrund der Verschlüsselung der Nachricht zusätzliche Kosten.

In MQ V7 können Sie Nachrichten zusätzlich zur **CORRELID** oder der **MSGID** nach einer Selektorzeichenfolge auswählen. Da jede Nachricht geprüft werden muss, fallen bei vielen Nachrichten entsprechende hohe Kosten an.

Es ist effizienter, wenn die Anwendung OPEN PUT PUT CLOSE anstatt PUT1 PUT1 durchführt.

Auslösefunktion in CICS

Bei einer niedrigen Nachrichteneingangsrate für eine ausgelöste Warteschlange ist es effizient, zuerst die Auslösefunktion zu verwenden. Liegt die Nachrichteneingangsrate über 10 Nachrichten pro Sekunde, ist es hingegen effizienter, die erste Transaktion auszulösen, sie danach eine Nachricht verarbeiten zu lassen, die nächste Nachricht abzurufen usw. Wenn eine Nachricht nicht in einem kurzen Zeitraum eingetroffen ist (ungefähr zwischen 0,1 und 1 Sekunde), wird die Transaktion beendet. Bei einem hohen Durchsatz benötigen Sie unter Umständen mehrere Transaktionen, um Nachrichten zu verarbeiten und ein Auflaufen der Nachrichten zu vermeiden. Jede generierte Auslösenachricht muss eingereiht und abgerufen werden, wodurch sich die Kosten der Nachricht verdoppeln.

Wie viele Verbindungen oder gleichzeitig angemeldete Benutzer werden unterstützt?

Jede Verbindung verwendet virtuellen Speicher innerhalb des Warteschlangenmanagers. Je mehr Benutzer also gleichzeitig angemeldet sind, desto höher ist die Speicherbelegung. Wenn Sie einen sehr großen Pufferpool und eine hohe Anzahl an Benutzern benötigen, steht Ihnen unter Umständen nur eine eingeschränkte Menge an virtuellem Speicher zur Verfügung, sodass Sie die Größe Ihrer Pufferpools möglicherweise reduzieren müssen.

Wenn Sicherheit verwendet wird, speichert der Warteschlangenmanager Informationen innerhalb des Warteschlangenmanagers für einen langen Zeitraum zwischen. Dies wirkt sich auf die Menge an virtuellem Speicher aus, der innerhalb des Warteschlangenmanagers verwendet wird.

Der Kanalinitiator **CHINIT** kann bis zu 10.000 Verbindungen unterstützen. Die virtuelle Speicherkapazität ist begrenzt. Wenn eine Verbindung beispielsweise durch die Verwendung von TLS mehr Speicher belegt, erhöht sich die Speichermenge pro Verbindung. Das wiederum bedeutet, dass der Kanalinitiator **CHINIT** weniger Verbindungen unterstützen kann. Wenn Sie große Nachrichten verarbeiten, benötigen diese mehr Speicher für Puffer im **CHINIT** und der **CHINIT** kann daher weniger Nachrichten unterstützen.

Verbindungen mit einem fernen Warteschlangenmanager sind effizienter als Clientverbindungen. Beispiel: Jede MQ-Clientanforderung erfordert zwei Netzflüsse (einen für die Anforderung und einen für die Antwort). Bei einer Kanalverbindung zu einem fernen Warteschlangenmanager werden 50 Anforderungen über das Netz gesendet, bevor eine Antwort zurückgegeben wird. Wenn Sie ein großes Clientnetz in Betracht ziehen, kann es effizienter sein, einen Konzentratorwarteschlangenmanager in einem dezentralen Feld zu verwenden, wobei ein Kanal aus dem Konzentrator und ein Kanal in den Konzentrator führt.

Weitere Leistungsfaktoren

Die Größe der Protokolldateien sollte mindestens 1000 Zylinder betragen. Bei kleineren Protokollen ist unter Umständen der Prüfpunkt zu häufig aktiv. Auf einem ausgelasteten System sollte ein Prüfpunkt üblicherweise mindestens alle 15 Minuten aktiv sein. Bei sehr hohen Durchsätzen verringert sich dieses Intervall unter Umständen. Wenn ein Prüfpunkt auftritt, werden die Pufferpools durchsucht und 'alte' Nachrichten sowie geänderte Seiten werden auf den Datenträger geschrieben. Treten Prüfpunkte zu häufig auf, kann sich dies auf die Leistung auswirken. Zudem kann der Wert der Protokolllast ('LOGLOAD') Auswirkungen auf die Prüfpunktfrequenz haben. Wenn der Warteschlangenmanager abnormal beendet wird, muss er bei Neustart unter Umständen bis zu 3 vergangene Prüfpunkte lesen. Das beste Prüfpunktintervall liegt zwischen der Aktivität, wenn ein Prüfpunkt stattfindet, und der Menge der Protokolldaten, die bei einem Neustart des Warteschlangenmanagers unter Umständen gelesen werden müssen.

Der Start eines Kanals ist mit großem Aufwand verbunden. Es empfiehlt sich, einen Kanal zu starten und seine Verbindung aufrecht zu halten, anstatt ihn häufig zu starten und wieder zu stoppen.

Zugehörige Informationen

MP1H: IBM MQ for z/OS 9.0 Leistungsbericht

z/OS IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS

Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit IBM MQ.

- Informationen zur Verwendung von Synchronisationspunkten und MQI-Aufrufen in IMS-Anwendungen finden Sie unter „IMS-Anwendungen mit IBM MQ schreiben” auf Seite 64.
- Informationen zum Schreiben von Anwendungen, die die IBM MQ - IMS-Brücke verwenden, finden Sie unter „IMS-Brückenanwendungen schreiben” auf Seite 69.

Unter den nachfolgenden Links erhalten Sie weitere Informationen zu IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS:

- „IMS-Anwendungen mit IBM MQ schreiben” auf Seite 64
- „IMS-Brückenanwendungen schreiben” auf Seite 69

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

IMS-Anwendungen mit IBM MQ schreiben

Wenn Sie IBM MQ in IMS-Anwendungen verwenden, sind weitere Überlegungen zu berücksichtigen, z. B., welche MQ-API-Aufrufe und welche Mechanismen für Synchronisationspunkte verwendet werden können.

Unter den nachfolgenden Links erhalten Sie weitere Informationen zum Schreiben von IMS-Anwendungen unter IBM MQ for z/OS:

- „Synchronisationspunkte in IMS-Anwendungen” auf Seite 65
- „MQI-Aufrufe in IMS-Anwendungen” auf Seite 65

Einschränkungen

Es gibt Beschränkungen hinsichtlich der IBM MQ-API-Aufrufe, die eine Anwendung verwenden kann, die den IMS-Adapter verwendet.

Folgende IBM MQ-API-Aufrufe werden nicht in Anwendungen unterstützt, die den IMS-Adapter verwenden:

- MQCB
- MQCB_FUNCTION
- MQCTL

Zugehörige Konzepte

„IMS-Brückenanwendungen schreiben" auf Seite 69
Dieser Abschnitt enthält Informationen zum Schreiben von Anwendungen für die Verwendung der IBM MQ - IMS-Brücke.

Synchronisationspunkte in IMS-Anwendungen

Sie erstellen einen Synchronisationspunkt in einer IMS-Anwendung mithilfe von IMS-Aufrufen wie beispielsweise GU (Get Unique) eines primären E/A-Steuerblocks (IOPCB) und des CHKP (Prüfpunkt).

Wenn Sie alle Änderungen seit dem letzten Prüfpunkt zurücksetzen möchten, verwenden Sie den IMS-Aufruf ROLB (Rollback). Weitere Informationen finden Sie in folgender Dokumentation:

- IMS 13 Application Programming APG SC19-3646
- IMS 13 Application Programming APIs APR SC19-3647

Der Warteschlangenmanager ist Teilnehmer eines Protokolls für zweiphasige Festschreibung; Koordinator ist der IMS-Synchronisationspunktmanager.

Alle offenen Kennungen werden vom IMS-Adapter an einem Synchronisationspunkt geschlossen (außer in einer Stapel- oder nicht nachrichtengesteuerten BMP-Umgebung). Grund hierfür ist, dass ein anderer Benutzer die nächste Arbeitseinheit initiieren könnte und die Sicherheitsprüfung durch IBM MQ bei einem MQCONN-, MQCONNX- oder MQOPEN-Aufruf durchgeführt wird und nicht bei einem MQPUT- oder MQGET-Aufruf.

In einer Wait-for-Input-Umgebung (WFI) oder einer Pseudo-Wait-for-Input-Umgebung (PWFI) wird IBM MQ jedoch erst dann von IMS angewiesen, die Kennungen zu schließen, wenn die nächste Nachricht eintrifft oder ein QC-Statuscode an die Anwendung zurückgegeben wird. Wenn die Anwendung im IMS-Bereich wartet und solche Kennungen zu ausgelösten Warteschlangen gehören, findet keine Auslösung statt, da die Warteschlangen bereits offen sind. Aus diesem Grund sollen Anwendungen, die in einer WFI- oder PWFI-Umgebung ausgeführt werden, die Warteschlangenkennungen explizit mit MQCLOSE schließen, bevor sie den GU-Aufruf eines primären E/A-Steuerblocks (IOPCB) für die nächste Nachricht tätigen.

Wenn eine IMS-Anwendung (BMP oder MPP) einen MQDISC-Aufruf ausgibt, werden offene Warteschlangen geschlossen, jedoch kein impliziter Synchronisationspunkt eingerichtet. Wenn die Anwendung normaler beendet wird, werden alle geöffneten Warteschlangen geschlossen und eine implizite Festschreibung wird ausgeführt. Endet die Anwendung abnormal, werden alle geöffneten Warteschlangen geschlossen und eine implizite Zurücksetzung (Backout) wird ausgeführt.

MQI-Aufrufe in IMS-Anwendungen

Dieser Abschnitt enthält Informationen zur Verwendung von MQI-Aufrufen in Server- und Abfrageanwendungen.

In diesem Abschnitt wird die Verwendung von MQI-Aufrufen in folgenden IMS-Anwendungstypen behandelt:

- „Serveranwendungen" auf Seite 65
- „Abfrageanwendungen" auf Seite 68

Serveranwendungen

Nachfolgend sehen Sie die Gliederung des MQI-Serveranwendungsmodells:

```
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
```

Das Beispielprogramm CSQ4ICB3 zeigt, wie ein BMP mit diesem Modell in C/370 integriert wird. Das Programm stellt zunächst die Kommunikation mit IMS und dann mit IBM MQ her:

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

Return
```

Die IMS-Initialisierung ermittelt, ob das Programm als nachrichtengesteuerte oder als stapelorientiertes BMP aufgerufen wurde und steuert entsprechend die Verbindung zum IBM MQ-Warteschlangenmanager sowie die Warteschlangenkennung:

```
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
```

Die IBM MQ-Initialisierung stellt eine Verbindung zum Warteschlangenmanager her und öffnet die Warteschlange. Bei einem nachrichtengesteuerten BMP wird dies nach jedem IMS-Synchronisationspunkt aufgerufen, bei einem stapelorientierten BMP hingegen nur während des Programmstarts:

```
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
```

Die Implementierung des Servermodells in einem MPP wird von der Tatsache beeinflusst, dass das MPP eine einzelne Arbeitseinheit pro Aufruf verarbeitet. Wenn ein Synchronisationspunkt (GU) stattfindet, werden nämlich die Verbindung und die Warteschlangenkennungen geschlossen und die nächste IMS-Nachricht wird zugestellt. Diese Einschränkung kann zum Teil durch eine der folgenden Maßnahmen aufgehoben werden:

- **Verarbeitung vieler Nachrichten innerhalb einer einzelnen Arbeitseinheit**

  Hierzu gehören:

  – Nachricht lesen
  – Erforderliche Aktualisierungen verarbeiten
  – Antwort einreihen

  Dies findet in einer Schleife statt, bis alle Nachrichten verarbeitet wurden oder eine festgelegte maximale Anzahl von Nachrichten verarbeitet wurde, bis ein Synchronisationspunkt erreicht wird.

  Nur bestimmte Anwendungstypen (zum Beispiel eine einfache Datenbankaktualisierung oder Abfrage) können auf diese Weise bearbeitet werden. Obwohl die MQI-Antwortnachrichten mit der Berechtigung des Erstellers der zu bearbeitenden MQI-Nachricht eingereiht werden können, müssen die Sicherheitsüberlegungen zu allen IMS-Ressourcenaktualisierungen sorgfältig einbezogen werden.
- **Eine Nachricht pro Aufruf des MPP verarbeiten und die Mehrfachplanung des MPP sicherstellen, um alle verfügbaren Nachrichten zu verarbeiten.**

  Verwenden Sie das IBM MQ IMS-Auslösemonitorprogramm (CSQQTRMN) zur Planung der MPP-Transaktion, wenn Nachrichten in der IBM MQ-Warteschlange stehen und von keinen Anwendungen verarbeitet werden.

  Wenn der Auslösemonitor das MPP startet, werden der Name des Warteschlangenmanagers und der Name der Warteschlange an das Programm übergeben (siehe nachstehender COBOL-Codeextrakt):

```
* 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 '='.
```

Das Servermodell, bei dem es sich Erwartungen zufolge um eine Langzeittask handelt, wird besser in einem Stapelverarbeitungsbereich unterstützt, obwohl das BMP nicht mit CSQQTRMN ausgelöst werden kann.

Abfrageanwendungen

Eine typische IBM MQ-Anwendung, die eine Abfrage oder Aktualisierung initiiert, funktioniert wie folgt:

- Daten vom Benutzer zusammenstellen
- Mindestens eine IBM MQ-Nachricht einreihen
- Antwortnachricht abrufen (u. U. mit Wartezeit verbunden)
- Antwort an Benutzer übermitteln

Da Nachrichten, die in IBM MQ-Warteschlangen eingereiht wurden, für andere IBM MQ-Anwendungen erst zur Verfügung stehen, nachdem sie festgeschrieben wurden, müssen sie entweder außerhalb des Synchronisationspunkts eingereiht werden oder die IMS-Anwendung muss auf zwei Transaktionen aufgeteilt werden.

Wenn mit der Abfrage das Einreihen einer einzelnen Nachricht verbunden ist, können Sie die Option *no syncpoint* verwenden. Bei komplexeren Abfragen oder Ressourcenaktualisierungen entstehen jedoch unter Umständen Konsistenzprobleme, falls ein Fehler auftritt und Sie keinen Synchronisationspunkt verwenden.

Um dies zu vermeiden, können Sie IMS-MPP-Transaktionen mittels MQI-Aufrufen aufteilen, die einen Program-to-Program-Nachrichtenswitch verwenden. Informationen hierzu finden Sie unter *IMS/ESA Anwendungsprogrammierung: Datenkommunikation*. Auf diese Weise kann ein Abfrageprogramm in ein MPP implementiert werden:

```
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
```

IMS-Brückenanwendungen schreiben

Dieser Abschnitt enthält Informationen zum Schreiben von Anwendungen für die Verwendung der IBM MQ - IMS-Brücke.

Informationen zur IBM MQ-IMS-Bridge enthält der Abschnitt IMS-Bridge.

Unter den nachfolgenden Links erhalten Sie weitere Informationen zum Schreiben von IMS-Brückenanwendungen unter IBM MQ for z/OS:

- „Bearbeitung von Nachrichten durch die IMS-Bridge” auf Seite 69
- „IMS-Transaktionsprogramme in IBM MQ schreiben” auf Seite 951

Zugehörige Konzepte

„IMS-Anwendungen mit IBM MQ schreiben” auf Seite 64
Wenn Sie IBM MQ in IMS-Anwendungen verwenden, sind weitere Überlegungen zu berücksichtigen, z. B., welche MQ-API-Aufrufe und welche Mechanismen für Synchronisationspunkte verwendet werden können.

Bearbeitung von Nachrichten durch die IMS-Bridge

Wenn Sie mit der IBM MQ-IMS-Bridge Nachrichten an eine IMS-Anweisung senden, müssen Sie die Nachrichten in einem besonderen Format erstellen.

Sie müssen Ihre Nachrichten auch in IBM MQ-Warteschlangen einreihen, die mit einer Speicherklasse definiert wurden, welche die XCF-Gruppe und den Mitgliedsnamen des IMS-Zielsystems angibt. Diese werden als MQ-IMS-Bridge-Warteschlangen oder kurz als **Bridge-Warteschlangen** bezeichnet.

Die IBM MQ-IMS-Bridge erfordert exklusiven Eingabezugriff (MQOO_INPUT_EXCLUSIVE) für die Bridge-Warteschlange, wenn sie mit QSGDISP (QMGR) bzw. mit QSGDISP (SHARED) und der Option NOSHARE definiert ist.

Ein Benutzer muss sich nicht erst bei IMS anmelden, bevor er Nachrichten an eine IMS-Anwendung sendet. Die Benutzer-ID im Feld *UserIdentifier* der MQMD-Struktur wird für die Sicherheitsprüfung verwendet. Die Prüfungsstufe wird festgelegt, wenn IBM MQ die Verbindung zu IMS herstellt. Eine Beschreibung finden Sie im Abschnitt Anwendungszugriffssteuerung für die IMS-Bridge. So kann eine Pseudo-Anmeldung implementiert werden.

Die IBM MQ - IMS-Bridge akzeptiert folgende Nachrichtentypen:

- Nachrichten mit IMS-Transaktionsdaten und einer MQIIH-Struktur (siehe Beschreibung unter MQIIH):

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

  Anmerkung:

  1. Die eckigen Klammern [ ] stellen optionale Multisegmente dar.
  2. Setzen Sie das Feld *Format* der MQMD-Struktur auf MQFMT_IMS, wenn Sie die MQIIH-Struktur verwenden möchten.

- Nachrichten mit IMS-Transaktionsdaten, aber ohne MQIIH-Struktur:

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

IBM MQ validiert die Nachrichtendaten, um sicherzustellen, dass die Summe der LL-Bytes plus die Länge von MQIIH (falls vorhanden) der Nachrichtenlänge entspricht.

Wenn die IBM MQ - IMS-Bridge Nachrichten aus Bridge-Warteschlangen abruft, verarbeitet sie diese wie folgt:

- Enthält die Nachricht eine MQIIH-Struktur, verifiziert die Bridge den MQIIH (siehe MQIIH), erstellt die OTMA-Header und sendet die Nachricht an IMS. Der Transaktionscode wird in der Eingabenachricht angegeben. Handelt es sich dabei um einen LTERM, antwortet IMS mit einer DFS1288E-Nachricht. Stellt

der Transaktionscode einen Befehl dar, führt IMS den Befehl aus; andernfalls wird die Nachricht für die Transaktion in IMS eingereiht.

- Enthält die Nachricht IMS-Transaktionsdaten, aber keine MQIIH-Struktur, nimmt die IMS-Bridge Folgendes an:
  - Der Transaktionscode befindet sich in den Bytes 5 bis 12 der Benutzerdaten.
  - Die Transaktion ist im nicht interaktiven Modus.
  - Die Transaktion ist im Festschreibungsmodus 0 (commit-then-send).
  - Der Wert im Feld *Format* des MQMD wird bei der Eingabe als *MFSMapName* verwendet.
  - Der Sicherheitsmodus ist MQISS_CHECK.

  Die Antwortnachricht wird ebenfalls ohne MQIIH-Struktur erstellt. Sie übernimmt das *Format* für den MQMD aus *MFSMapName* der IMS-Ausgabe.

Die IBM MQ - IMS-Bridge verwendet eine oder zwei Transaktionspipes für jede IBM MQ-Warteschlange:

- Eine synchronisierte Transaktionspipe wird für alle Nachrichten verwendet, die Festschreibungsmodus 0 (COMMIT_THEN_SEND) verwenden (für diese wird im Statusfeld des Befehls IMS /DIS TMEMBER client TPIPE xxxx der Status SYN angezeigt).
- Für alle Nachrichten, die Festschreibungsmodus 1 (SEND_THEN_COMMIT) verwenden, wird eine nicht synchronisierte Transaktionspipe verwendet.

Die Transaktionspipes werden bei ihrer ersten Verwendung von IBM MQ erstellt. Eine nicht synchronisierte Transaktionspipe ist bis zum Neustart von IMS vorhanden. Synchronisierte Transaktionspipes sind bis zum Kaltstart von IMS vorhanden. Diese Transaktionspipes können Sie nicht selber löschen.

Weitere Informationen zur Bearbeitung von Nachrichten durch die IBM MQ - IMS-Bridge finden Sie in den folgenden Abschnitten:

- „IBM MQ-Nachrichten zu IMS-Transaktionstypen zuordnen” auf Seite 70
- „Nachricht kann nicht in die IMS-Warteschlange eingereiht werden” auf Seite 71
- „Rückkopplungscodes der IMS-Brücke” auf Seite 72
- „MWMD-Felder in Nachrichten von der IMS-Brücke” auf Seite 72
- „MQIIH-Felder in Nachrichten von der IMS-Brücke” auf Seite 73
- „Antwortnachrichten aus IMS” auf Seite 74
- „Alternative Antwort-PCBs in IMS-Transaktionen verwenden” auf Seite 74
- „Nicht erwartete Nachrichten aus IMS senden” auf Seite 74
- „Nachrichtensegmentierung” auf Seite 75
- „Datenkonvertierung” auf Seite 75

Zugehörige Konzepte

„IMS-Transaktionsprogramme in IBM MQ schreiben” auf Seite 951

Die Codierung, die für die Verarbeitung von IMS-Transaktionen durch IBM MQ erforderlich ist, hängt von dem von der IMS-Transaktion benötigten Nachrichtenformat und von dem Nachrichtenspektrum ab, das die Transaktion zurückgeben kann. Wenn Ihre Anwendung die Bildschirmformatinformationen von IMS verarbeitet, müssen Sie jedoch einige Punkte berücksichtigen.

IBM MQ-Nachrichten zu IMS-Transaktionstypen zuordnen

Eine Tabelle mit Erläuterungen zur Zuordnung von IBM MQ-Nachrichten zu IMS-Transaktionstypen.

Tabelle 4. IBM MQ-Nachrichten zu IMS-Transaktionstypen zuordnen		
IBM MQ-Nachrichtentyp	**Commit-then-send (mode 0) - verwendet synchronisierte IMS-Transaktionspipes**	**Send-then-commit (mode 1) - verwendet nicht synchronisierte IMS-Transaktionspipes**
Persistente IBM MQ-Nachrichten	• Wiederherstellbare Vollfunktion-Transaktionen • Nicht wiederherstellbare Transaktionen werden von IMS zurückgewiesen.	• Fastpath-Transaktionen • Dialogtransaktionen • Vollfunktion-Transaktionen
Nicht persistente IBM MQ-Nachrichten	• Nicht wiederherstellbare Vollfunktion-Transaktionen • Wiederherstellbare Transaktionen mit IMS V8 und APAR PQ61404 sowie allen späteren Versionen von IMS sind zulässig.	• Fastpath-Transaktionen • Dialogtransaktionen • Vollfunktion-Transaktionen

Anmerkung: IMS-Befehle können persistente IBM MQ-Nachrichten mit Commitmodus 0 nicht verwenden. Weitere Informationen hierzu enthält das *IMS/ESA-Benutzerhandbuch zu Open Transaction Manager Access*.

Nachricht kann nicht in die IMS-Warteschlange eingereiht werden
In diesem Abschnitt erfahren Sie, was zu tun ist, wenn eine Nachricht nicht in die IMS-Warteschlange eingereiht werden kann.

Wenn eine Nachricht nicht in die IMS-Warteschlange eingereiht werden kann, führt IBM MQ Folgendes aus:

- Wenn eine Nachricht nicht in IMS eingereiht werden kann, da sie ungültig ist, wird sie stattdessen in die Warteschlange für nicht zustellbare Nachrichten eingereiht und eine entsprechende Nachricht wird an die Systemkonsole gesendet.
- Lehnt IMS eine gültige Nachricht ab, sendet IBM MQ eine Fehlernachricht mit dem IMS-Prüfcode an die Systemkonsole. Die IBM MQ-Nachricht wird in die Warteschlange für nicht zustellbare Nachrichten eingereiht. Lautet der IMS-Prüfcode 001A, sendet IMS eine IBM MQ-Nachricht mit der Fehlerursache an die Empfangswarteschlange für Antworten.

  Anmerkung: Unter den oben genannten Umständen wird die Nachricht an die ursprüngliche IBM MQ-Warteschlange zurückgesendet, wenn sie von IBM MQ aus irgendeinem Grund nicht in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden kann. Eine Fehlernachricht wird an die Systemkonsole gesendet und es werden keine weiteren Nachrichten aus dieser Warteschlange gesendet.

  Führen Sie ***einen*** der folgenden Schritte aus, um die Nachrichten erneut zu senden:

  – Stopp und Neustart der Transaktionspipes in IMS entsprechend der Warteschlange
  – Ändern der Warteschlange in GET(DISABLED) und wieder in GET(ENABLED)
  – Stopp und Neustart von IMS oder von OTMA
  – Stopp und Neustart Ihres IBM MQ-Subsystems
- Lehnt IMS aus einem anderen Grund als ein Nachrichtenfehler ab, wird die IBM MQ-Nachricht an die ursprüngliche Warteschlange zurückgegeben. IBM MQ stoppt die Verarbeitung der Warteschlange und eine Fehlernachricht wird an die Systemkonsole gesendet.

  Wenn eine Ausnahmeberichtsnachricht erforderlich ist, reiht sie die Brücke mit der Berechtigung des Absenders die Empfangswarteschlange für Antworten ein. Kann die Nachricht nicht in die Warteschlange eingereiht werden, wird die Berichtsnachricht mit der Berechtigung der Brücke in die Warteschlange

für nicht zustellbare Nachrichten eingereiht. Kann sie nicht in die DLQ eingereiht werden, wird sie gelöscht.

Rückkopplungscodes der IMS-Brücke
Normalerweise werden IMS-Prüfcodes in IBM MQ-Konsolennachrichten wie z. B. CSQ2001I im Hexadezimalformat ausgegeben (z. B. Prüfcode 0x001F). IBM MQ-Rückkopplungscodes im Header einer nicht zustellbaren Nachricht, die in die Warteschlange für nicht zustellbare Nachrichten eingereiht wurde, werden in Dezimalzahlen angegeben.

Die Rückkopplungscodes der IMS-Brücke liegen im Zahlenbereich von 301 bis 399, der NACK-Prüfcode 0x001A liegt im Bereich von 600 bis 855. Sie werden von den IMS-OTMA-Prüfcodes wie folgt zugeordnet:

1. Der IMS-OTMA-Prüfcode im Hexadezimalformat wird in eine Dezimalzahl konvertiert.
2. 300 wird zu der Zahl addiert, die sich aus der Berechnung in 1ergibt und ergibt den IBM MQ *Feedback* -Code.
3. Der IMS-OTMA-Prüfcode 0x001A, Dezimalzahl 26, ist ein Sonderfall. Ein *Rückkopplungscode* im Bereich von 600 bis 855 wird generiert.
   a. Der IMS-OTMA-Ursachencode im Hexadezimalformat wird in eine Dezimalzahl konvertiert.
   b. Zu dem Ergebnis der Berechnung in a wird 600 hinzuaddiert, was den IBM MQ-*Rückkopplungscode* ergibt.

Informationen zu den IMS-OTMA-Prüfcodes finden Sie unter OTMA-Prüfcodes für NAK-Nachrichten.

MWMD-Felder in Nachrichten von der IMS-Brücke
Dieser Abschnitt enthält Informationen zu den MQMD-Feldern in Nachrichten von der IMS-Brücke.

Der MQMD der ursprünglichen Nachricht wird von IMS in den Benutzerdatenbereich der OTMA-Header übertragen. Stammt die Nachricht ursprünglich aus IMS, wird dieser vom IMS-Exit für die Zielauflösung erstellt. Der MQMD einer aus IMS empfangenen Nachricht wird wie folgt erstellt:

StrucID
"MD "

Version
MQMD_VERSION_1

Bericht
MQRO_NONE

MsgType
MQMT_REPLY

Verfall
Wenn MQIIH_PASS_EXPIRATION im Flagfeld des MQIIH gesetzt ist, enthält dieses Feld die verbleibende Ablaufzeit. Andernfalls wird es auf MQEI_UNLIMITED gesetzt.

Feedback
MQFB_NONE

Encoding
MQENC.Native (Codierung des z/OS-Systems)

CodedCharSetId
MQCCSI_Q_MGR (ID des codierten Zeichensatzes des z/OS-Systems)

Format
MQFMT_IMS, wenn das MQMD.Format der Eingabenachricht MQFMT_IMS ist, ansonsten IOPCB.MODNAME

Priority
MQMD.Priority der Eingabenachricht

Permanenz
Hängt vom Commit-Knoten ab: MQMD.Persistence der Eingabenachricht, wenn CM-1; Persistenz entspricht der Wiederherstellbarkeit der IMS-Nachricht, wenn CM-0

MsgId
MQMD.MsgId wenn MQRO_PASS_MSG_ID, ansonsten New MsgId (Standardeinstellung)

CorrelId
MQMD.CorrelId aus der Eingabenachricht, wenn MQRO_PASS_CORREL_ID, ansonsten MQMD.MsgId aus der Eingabenachricht (Standardeinstellung)

BackoutCount
0

ReplyToQ
Leerzeichen

ReplyToQMgr
Leerzeichen (beim MQPUT-Aufruf vom Warteschlangenmanager auf den Namen des lokalen Warteschlangenmanagers gesetzt)

UserIdentifier
MQMD.UserIdentifier der Eingabenachricht

AccountingToken
MQMD.AccountingToken der Eingabenachricht

ApplIdentityData
MQMD.ApplIdentityData der Eingabenachricht

PutApplType
MQAT_XCF, wenn kein Fehler vorliegt, ansonsten MQAT_BRIDGE

PutApplName
<XCFgroupName<XCFmemberName>, wenn kein Fehler vorliegt, ansonsten Name des Warteschlangenmanagers (QMGR name)

PutDate
Datum der Nachrichteneinreihung

PutTime
Uhrzeit, zu der die Nachricht eingereiht wurde

ApplOriginData
Leerzeichen

MQIIH-Felder in Nachrichten von der IMS-Brücke
Dieser Abschnitt enthält Informationen zu den MQIIH-Feldern in Nachrichten von der IMS-Brücke.

Der MQIIH einer aus IMS empfangenen Nachricht wird wie folgt erstellt:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat der Eingabenachricht, wenn MQIIH.ReplyToFormat nicht leer ist, ansonsten IOPCB.MODNAME

Flags
0

LTermOverride
LTERM-Name (Transaktionspipe) aus OTMA-Header

MFSMapName
Mapname aus OTMA-Header

ReplyToFormat
Leerzeichen

Authenticator
MQIIH.Authenticator der Eingabenachricht, wenn die Antwortnachricht in die Warteschlange einer MQ-IMS-Brücke eingereiht wird, ansonsten Leerzeichen.

TranInstanceId
Dialog-ID / Server-Token aus dem OTMA-Header, wenn im Dialog. In IMS-Versionen vor V14 enthält dieses Feld immer Null, wenn es sich nicht in einem Dialog befindet. Ab der IMS-Version 14 wird dieses Feld möglicherweise von IMS festgelegt, selbst wenn kein Dialog besteht.

TranState
"C", wenn im Dialog, ansonsten Leerzeichen

CommitMode
Commitmodus aus OTMA-Header ("0" oder "1")

SecurityScope
keinen

Reserved
keinen

Antwortnachrichten aus IMS
Wenn eine IMS-Transaktion den Befehl ISRT an ihren primären E/A-Steuerblock ausgibt, wird die Nachricht an das ursprüngliche logische Terminal (LTERM) oder an die Transaktionspipe (TPIPE) zurückgeleitet.

Sie werden in IBM MQ als Antwortnachrichten angezeigt. Antwortnachrichten aus IMS werden in die Empfangswarteschlange für Antworten eingereiht, die in der ursprünglichen Nachricht angegeben ist. Kann die Nachricht nicht in die Empfangswarteschlange für Antworten eingereiht werden, wird sie mit der Berechtigung der Brücke in die Warteschlange für nicht zustellbare Nachrichten eingereiht. Kann die Nachricht nicht in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden, wird sie an IMS gesendet, um mitzuteilen, dass die Nachricht nicht empfangen werden kann. Die Zuständigkeit für die Nachricht wird dann an IMS zurückgegeben. Wenn Sie Commitmodus 0 verwenden, werden Nachrichten aus dieser Transaktionspipe nicht an die Brücke gesendet und bleiben in der IMS-Warteschlange; d. h., bis zum Neustart werden keine weiteren Nachrichten gesendet. Wenn Sie Commitmodus 1 verwenden, können andere Vorgänge fortgesetzt werden.

Wenn die Antwort eine MQIIH-Struktur hat, lautet ihr Formattyp MQFMT_IMS; andernfalls wird ihr Formattyp vom IMS-MOD-Namen verwendet, der beim Einfügen der Nachricht verwendet wurde.

Alternative Antwort-PCBs in IMS-Transaktionen verwenden
Wenn eine IMS-Transaktion alternative Antwort-PCBs verwendet (Ausgabe von ISRT-Befehlen an den alternativen primären Steuerblock oder Ausgabe eines CHNG-Aufrufs an einen änderbaren primären Steuerblock), wird der Pre-Routing-Exit (DFSYPRX0) aufgerufen, um festzustellen, ob die Nachricht umgeleitet werden sollte.

Soll die Nachricht umgeleitet werden, wird der Exit für die Zielauflösung (DFSYDRU0) aufgerufen, um das Ziel zu bestätigen und die Headerinformationen vorzubereiten. Informationen zu diesen Exitprogrammen finden Sie unter OTMA-Exits in IMS verwenden und Pre-Routing-Exit DFSYPRX0.

Solange keine Aktion in den Exits vorgenommen wird, werden alle Ausgaben aus IMS-Transaktionen, die in einem IBM MQ-Warteschlangenmanager initialisiert wurden - ob nun Ausgaben an den primären E/A-Steuerblock oder an einen alternativen primären Steuerblock -, an denselben Warteschlangenmanager zurückgegeben.

Nicht erwartete Nachrichten aus IMS senden
Zum Senden von Nachrichten aus IMS an eine IBM MQ-Warteschlange müssen Sie eine IMS-Transaktion aufrufen, die den Befehl ISRT an einen alternativen primären Steuerblock (ALTPCB) ausgibt.

Sie müssen Pre-Routing-Exits und Exits für die Zielauflösung schreiben, um nicht erwartete Nachrichten aus IMS weiterzuleiten und die OTMA-Benutzerdaten zu erstellen, damit der MQMD der Nachricht ordnungsgemäß erstellt werden kann. Informationen zu diesen Exitprogrammen finden Sie unter Pre-Routing-Exit DFSYPRX0 und Benutzer-Exit für die Zielauflösung.

Anmerkung: Die IBM MQ - IMS-Brücke weiß nicht, ob es sich bei einer von ihr empfangenen Nachricht um eine Antwort um oder eine nicht erwartete Nachricht handelt. Sie bearbeitet die Nachricht in allen Fällen gleich und erstellt den MQMD und den MQIIH der Antwort auf Grundlage der OTMA-Benutzerdaten, die mit der Nachricht empfangen werden.

Nicht erwartete Nachrichten können neue Transaktionspipes erstellen. Wenn beispielsweise eine vorhandene IMS-Transaktion auf ein neues logisches Terminal wechselt (z. B. PRINT01), aber die Implementierung die Bereitstellung der Ausgabe über OTMA erfordert, wird eine neue Transaktionspipe erstellt (in diesem Beispiel PRINT01). Standardmäßig handelt es sich hierbei um eine nicht synchronisierte Transaktionspipe. Wenn die Implementierung eine wiederherstellbare Nachricht erfordert, legen Sie das Ausgabeattribut des Exits für die Zielauflösung fest. Weitere Informationen hierzu finden Sie im Handbuch *IMS Customization Guide*.

Nachrichtensegmentierung
Sie können IMS-Transaktionen so definieren, dass Eingaben in einzelnen oder mehreren Segmenten erwartet werden.

Die ursprüngliche IBM MQ-Anwendung muss die Benutzereingabe nach der MQIIH-Struktur als mindestens ein LLZZ-Datensegment konstruieren. Alle Segmente einer IMS-Nachricht müssen in einer einzelnen IBM MQ-Nachricht enthalten sein, die mit einem einzelnen MQPUT-Aufruf gesendet wird.

Die maximale Länge eines LLZZ-Datensegments wird durch IMS/OTMA (32767 Byte) definiert. Die Gesamtlänge der IBM MQ-Nachricht setzt sich aus der Summe der LL-Bytes plus der Länge der MQIIH-Struktur zusammen.

Alle Segmente der Antwort sind in einer einzelnen IBM MQ-Nachricht enthalten.

Es gibt eine weitere Beschränkung der Begrenzung auf 32 KB bei Nachrichten mit dem Format MQFMT_IMS_VAR_STRING. Wenn die Daten in einer ASCII-gemischten CCSID-Nachricht in eine EBCDIC-gemischte CCSID-Nachricht konvertiert werden, wird bei jedem Wechsel zwischen SBCS- und DBCS-Zeichen immer ein DBCS-Endebyte bzw. ein DBCS-Startbyte hinzugefügt. Die Einschränkung auf 32 KB gilt für die maximale Größe der Nachricht: Da das LL-Feld in der Nachricht 32 KB nicht überschreiten kann, darf die Nachrichtengröße maximal 32 KB einschließlich aller DBCS-Endezeichen und DBCS-Startzeichen betragen. Die Anwendung, mit der die Nachricht erstellt wird, muss dies unterstützen.

Datenkonvertierung
Die Datenkonvertierung wird entweder von der Funktion der verteilten Steuerung von Warteschlangen (die u.U. alle notwendigen Exits aufruft) oder vom gruppeninternen Warteschlangensteuerungsagenten vorgenommen (dieser unterstützt keine Verwendung von Exits), wenn eine Nachricht in eine Zielwarteschlange mit definierten XCF-Informationen für ihre Speicherklasse eingereiht wird. Die Datenkonvertierung findet nicht statt, wenn eine Nachricht mittels Publish/Subscribe an eine Warteschlange übermittelt wird.

Benötigte Exits müssen der Funktion für die verteilte Steuerung von Warteschlangen in der von der CSQXLIB-DD-Anweisung referenzierten Datei zur Verfügung stehen. Sie können also Nachrichten von jeder IBM MQ-Plattform unter Verwendung der IBM MQ - IMS-Bridge an eine IMS-Anwendung senden.

Kommt es zu Konvertierungsfehlern, wird die Nachricht unkonvertiert in die Warteschlange eingereiht; dies führt letztlich dazu, dass sie von der IBM MQ - IMS-Bridge als Fehler behandelt wird, da die Bridge das Headerformat nicht erkennen kann. Tritt ein Konvertierungsfehler auf, wird eine Fehlernachricht an die z/OS-Konsole gesendet.

Ausführliche Informationen zur Datenkonvertierung allgemein finden Sie unter „Datenkonvertierungsexits schreiben” auf Seite 1033.

Nachrichten an die IBM MQ - IMS-Bridge senden

Um sicherzustellen, dass die Konvertierung ordnungsgemäß ausgeführt wird, müssen Sie dem Warteschlangenmanager das Format der Nachricht mitteilen.

Wenn die Nachricht eine MQIIH-Struktur aufweist, muss das *Format* im MQMD auf das integrierte Format MQFMT_IMS gesetzt werden und das *Format* im MQIIH auf den Namen des Formats, das Ihre Nachrichtendaten beschreibt. Ist kein MQIIH vorhanden, setzen Sie das *Format* im MQMD auf Ihren Formatnamen.

Wenn es sich bei Ihren Daten (mit Ausnahme der LLZZs) nur um Zeichendaten (MQCHAR) handelt, verwenden Sie das integrierte Format MQFMT_IMS_VAR_STRING als Ihren Formatnamen (im MQIIH oder im MQMD). Andernfalls verwenden Sie Ihren eigenen Formatnamen. In diesem Fall müssen Sie ein Datenkonvertierungsexit für Ihr Format bereitstellen. Zusätzlich zu den eigentlichen Daten muss der Exit die Konvertierung der LLZZs in Ihrer Nachricht ausführen (muss aber keinen MQIIH am Anfang der Nachricht bearbeiten).

Wenn Ihre Anwendung *MFSMapName* verwendet, können Sie stattdessen Nachrichten mit MQFMT_IMS verwenden und den Mapnamen definieren, der im MQIIH-Feld MFSMapName an die IMS-Transaktion übermittelt wurde.

Nachrichten von der IBM MQ - IMS-Bridge empfangen

Ist eine MQIIH-Struktur in der ursprünglichen Nachricht vorhanden, die Sie an IMS senden, ist auch eine in der Antwortnachricht vorhanden.

So stellen Sie die ordnungsgemäße Konvertierung Ihrer Antwort sicher:

- Wenn Ihre ursprüngliche Nachricht eine MQIIH-Struktur enthält, geben Sie das gewünschte Format für Ihre Antwortnachricht im MQIIH-Feld *ReplytoFormat* der ursprünglichen Nachricht an. Dieser Wert wird in das MQIIH-Feld *Format* der Antwortnachricht eingetragen. Dies ist besonders nützlich, wenn alle Ihre Ausgabedaten das Format LLZZ<Zeichendaten> haben.
- Wenn Ihre ursprüngliche Nachricht keine MQIIH-Struktur enthält, geben Sie das gewünschte Format für Ihre Antwortnachricht als MFS MOD-Name im Befehl ISRT an, den Sie an den primären E/A-Steuerblock der IMS-Anwendung ausgeben.

JMS- und Java-Anwendungen entwickeln

IBM MQ stellt zwei Java-Sprachenschnittstellen bereit: IBM MQ classes for Java Message Service und IBM MQ classes for Java.

Innerhalb von IBM MQ gibt es zwei alternative APIs, die in Java-Anwendungen verwendet werden können:

IBM MQ classes for JMS
IBM MQ classes for Java Message Service (JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Die Java Platform, Enterprise Edition Connector Architecture (JCA) stellt eine Standardmethode für die Verbindung von Anwendungen, die in einer Java EE-Umgebung aktiv sind, mit einem Enterprise Information System (EIS) wie IBM MQ oder Db2 bereit.

IBM MQ classes for Java
IBM MQ classes for Java ermöglichen Ihnen die Verwendung von IBM MQ in einer Java-Umgebung. IBM MQ classes for Java ermöglichen einer Java-Anwendung eine Verbindung mit IBM MQ als IBM MQ-Client oder eine direkte Verbindung mit einem IBM MQ-Warteschlangenmanager.

Anmerkung:

Die IBM MQ classes for Java sind auf der Stufe, die im Lieferumfang von IBM MQ 8.0enthalten ist, funktional stabilisiert. Weitere Informationen finden Sie im Abschnitt Stabilisierung von WebSphere MQ-Klassen für Java.

Die IBM MQ classes for Java werden in IMS nicht unterstützt.

Die IBM MQ classes for Java werden in WebSphere Application Server 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 verwenden

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

Die JMS-Spezifikation definiert eine Gruppe von Schnittstellen, mit denen Anwendungen Messaging-Operationen ausführen können. Die aktuelle Version der Spezifikation ist JMS 2.0. Das Paket 'javax.jms' definiert 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.

Die JMS-Spezifikation erwartet, dass es sich bei den ConnectionFactory- und Destination-Objekten um verwaltete Objekte handelt. Ein Administrator erstellt und wartet verwaltete Objekte in einem zentralen Repository, die dann von einer JMS-Anwendung über Java Naming Directory Interface (JNDI) abgerufen werden. IBM MQ classes for JMS unterstützen die Verwendung von verwalteten Objekten und ein Administrator kann verwaltete Objekte entweder mit dem IBM MQ-JMS-Verwaltungstool oder mit IBM MQ Explorer erstellen und warten.

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

Die Vorgängerreleases der IBM MQ classes for JMS enthalten Erweiterungen, die in Objekten wie MQConnectionFactory-, MQQueue- und MQTopic-Objekten implementiert werden. 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. In diesem Release von IBM MQ classes for JMS wurden diese Erweiterungen, die unter dem Begriff IBM MQ JMS-Erweiterungen bekannt sind, beibehalten. Sie können alle Anwendungen, die diese Erweiterungen verwenden, weiterhin ohne Änderung nutzen.

IBM JMS-Erweiterungen

Dieses Release der IBM MQ classes for JMS stellt eine generisch gehaltene Gruppe von Erweiterungen für die JMS-API zur Verfügung, die sich nicht speziell auf IBM MQ als Messaging-System beziehen. Diese Erweiterungen sind unter dem Begriff 'IBM JMS-Erweiterungen' bekannt und haben die folgende allgemeine Zielsetzung:

- Bereitstellung einer verbesserten Konsistenz über verschiedene IBM JMS-Provider hinweg
- Vereinfachung des Schreibens einer Bridge-Anwendung zwischen zwei IBM Messaging-Systemen
- Vereinfachung der Portierung einer Anwendung aus einem IBM JMS-Provider an einen anderen Provider

Die Erweiterungen stellen ähnliche Funktionen bereit wie IBM Message Service Client for C/C++ und IBM Message Service Client for .NET.

Ab IBM MQ 8.0 werden die IBM MQ classes for JMS mit Java 7 entwickelt. Die Laufzeitumgebung von Java 7 unterstützt die Ausführung früherer Klassendateiversionen.

V 9.0.0.6 IBM MQ 9.0.5 war das letzte Continuous Delivery Release für IBM MQ 9.0. Ab IBM MQ 9.0.0 Fix Pack 6 werden daher die Javadoc-Informationen für IBM MQ classes for JMS dahingehend aktualisiert, dass das Verhalten von IBM MQ classes for JMS nur noch für Funktionen berücksichtigt wird, die für Long Term Support-Kunden zur Verfügung stehen.

Zugehörige Konzepte

„JMS-Modell” auf Seite 129
Das Modell JMS definiert eine Gruppe von Schnittstellen, die Java -Anwendungen verwenden können, um Messaging-Operationen auszuführen. IBM MQ classes for JMS als JMS -Provider definiert, wie JMS -Objekte mit IBM MQ -Konzepten in Beziehung stehen. Die JMS-Spezifikation erwartet, dass es sich bei bestimmten JMS-Objekten um verwaltete Objekte handelt. In JMS 2.0 wurde eine vereinfachte API eingeführt. Die klassische API aus JMS 1.1 wurde jedoch ebenfalls beibehalten.

„Funktionalität von JMS 2.0 nutzen” auf Seite 315
JMS 2.0 bietet mehrere neue Funktionalitätsbereiche für die IBM MQ classes for JMS.

Zugehörige Informationen

IBM MQ Java-Sprachenschnittstellen

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.

IBM MQ classes for JMS ist eine von zwei alternativen APIs, die Java-Anwendungen für den Zugriff auf IBM MQ-Ressourcen verwenden können. Die andere API ist IBM MQ classes for Java. Auch wenn vorhandene Anwendungen, die die IBM MQ classes for Java verwenden, weiterhin vollständig unterstützt werden, sollten neue Anwendungen die IBM MQ classes for JMS verwenden (siehe „Auswahl der API” auf Seite 79).

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.

- 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 mithilfe von JNDI Verbindungsfactorys und Ziele abrufen, die als verwaltete Objekte gespeichert werden, und nur die im Paket javax.jms enthaltenen Schnittstellen zur Durchfüh-

rung von Messaging-Operationen verwenden. 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 in Java EE möglich war, die IBM MQ classes for Java 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 im IBM technischen Hinweis WebSphere MQ -Java-Schnittstellen in J2EE/JEE -Umgebungen verwenden .

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 lassen sich nun einfacher entweder eigenständig oder innerhalb von Softwareverwaltungsframeworks wie Maven implementieren.Weitere Informationen hierzu finden Sie in der IBM Technote Obtaining the WebSphere MQ classes for JMS (WebSphere MQ-Klassen für JMS erhalten).

Auswahl der API

Neue Anwendungen sollten statt der IBM MQ classes for Java besser die IBM MQ classes for JMS verwenden.

IBM MQ classes for JMS bieten sowohl Zugriff auf die Punkt-zu-Punkt- als auch auf die 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. Auf die MQMD- und MQ-Nachrichtennutzdaten kann uneingeschränkt zugegriffen werden. 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.

Neue Funktionen von IBM MQ, z. B. asynchroner Konsum und automatische Verbindungswiederherstellung, stehen nur in den IBM MQ classes for JMS zur Verfügung, nicht in den IBM MQ classes for Java. Vorhandene Anwendungen, die die IBM MQ classes for Java verwenden, werden weiterhin vollständig unterstützt.

Wenn Sie Zugriff auf IBM MQ-Funktionen benötigen, die in den IBM MQ classes for JMS nicht verfügbar sind, können Sie eine RFE-Anfrage (Request for Enhancement, Verbesserungsanfrage) stellen. IBM kann Ihnen dann sagen, ob die Implementierung innerhalb der IBM MQ classes for JMS möglich ist oder ob es eine Best Practice gibt, der Sie folgen können. 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.

Zugehörige Informationen
IBM RFE-Übergabeprozess
Prüfprozess für JMS-Java-Spezifikation
WebSphere MQ-Java-Schnittstellen in J2EE/JEE-Umgebungen verwenden
WebSphere MQ-Klassen für JMS abrufen
PCF-Nachrichten mithilfe von JMS senden
Tracing von IBM MQ classes for JMS-Anwendungen
Fehlerbehebung für Java und JMS

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 UNIX and Linux System Services (USS) für z/OS.

V 9.0.0 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.

V 9.0.0 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 eine solche Konformität manuell aktivieren (siehe erneut die Diskussion unter FIPS 140-kompatibler Modus für SunJSSE für einige Hinweise), aber IBM kann derzeit keine Anleitung dazu bereitstellen.

Sie können Internet Protocol Version 6-Adressen (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.

Installation und Konfiguration von IBM MQ classes for JMS

In diesem Abschnitt werden die Verzeichnisse und Dateien beschrieben, die bei der Installation von IBM MQ classes for JMS erstellt werden. Außerdem erfahren Sie, wie Sie IBM MQ classes for JMS nach der Installation konfigurieren.

Zugehörige Konzepte

„Dies wird für IBM MQ classes for JMS installiert" auf Seite 82
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.

„IBM MQ classes for JMS-Anwendungen unter dem Java Security Manager ausführen" auf Seite 98
IBM MQ classes for JMS können bei aktiviertem Java-Sicherheitsmanager 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.

„IBM MQ-Ressourcenadapter verwenden" auf Seite 434
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.

„Konfiguration für Anwendungen der IBM MQ classes for JMS nach der Installation" auf Seite 100
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.

„Punkt-zu-Punkt-Installationsprüftest (IVT) für IBM MQ classes for JMS" auf Seite 103
Mit den IBM MQ classes for JMS wird ein Programm für einen Punkt-zu-Punkt-Installationsprüftest (Installation Verification Test, IVT) bereitgestellt. 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.

„Publish/Subscribe-Installationsprüftest (IVT) für IBM MQ classes for JMS" auf Seite 108
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.

„Ressourcenadapter für abgehende Kommunikation konfigurieren" auf Seite 470
Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

„Unterstützung von OSGi" auf Seite 116
OSGi stellt ein Framework bereit, das die Implementierung von Anwendungen als Bundles unterstützt. Mit den IBM MQ classes for JMS werden neun OSGi-Bundles bereitgestellt.

Zugehörige Tasks

„Installation des Ressourcenadapters überprüfen" auf Seite 490
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.

Zugehörige Verweise

„Mit IBM MQ classes for JMS bereitgestellte Scripts" auf Seite 115
Es werden mehrere Scripts bereitgestellt, die Sie bei allgemeinen Tasks unterstützen, die bei der Verwendung von IBM MQ classes for JMS ausgeführt werden müssen.

Zugehörige Informationen

Fehlerbehebung bei Problemen mit IBM MQ classes for JMS

Problembestimmung für den IBM MQ-Ressourcenadapter

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 in diesem Abschnitt 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.
- Zudem wird auch das Einschließen der Datei `com.ibm.mq.allclient.jar` oder der IBM MQ classes for JMS innerhalb von Anwendungsarchiven (z. B. Unternehmensanwendungsarchive oder EAR-Dateien) nicht unterstützt.

Daher sollten Sie die Bündelung von IBM MQ-JAR-Dateien in Ihren Anwendungen (EAR-Dateien in WebSphere Application Server) vermeiden, da andernfalls unerwartete Probleme auftreten können, die mit der Ausführung von nicht gepatchtem Code einer früheren Version verbunden sind.

Installationsverzeichnisse

In Tabelle 5 auf Seite 82 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**
AIX UNIX 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*`/mqm/V9R0M0/java`
	MQ_INSTALLATION_PATH`/opt/mqm/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, 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 87.
- Zusätzliche Scripts (diese werden im Abschnitt „Mit IBM MQ classes for JMS bereitgestellte Scripts" auf Seite 115 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:

  – ULW Unter UNIX, Linux, and Windows enthält dieses Unterverzeichnis die einzelnen HTML-Seiten.

  – IBM i Unter IBM i befinden sich die HTML-Seiten in einer Datei mit dem Namen wmqjms_javadoc.jar.

  – z/OS Unter z/OS befinden sich die HTML-Seiten in einer Datei mit dem Namen wmqjms_javadoc.jar.
- Unterstützung von OGSi. OSGi-Bundles werden im Verzeichnis java\lib\OSGi installiert und im Abschnitt „Unterstützung von OSGi" auf Seite 116 beschrieben.
- Den IBM MQ-Ressourcenadapter, der auf jedem Anwendungsserver implementiert werden kann, der mit Java Platform, Enterprise Edition 7 (Java EE 7) konform ist.

  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 434.
- 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 anderen IBM MQ-Komponenten gehören:

- Der IBM MQ-Transport für SOAP, der einen JMS-Transport für SOAP bereitstellt, wird im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib\soap installiert. Weitere Informationen zum IBM MQ-Transport für SOAP finden Sie im Abschnitt „Web-Services mit IBM MQ-Transport für SOAP entwickeln" auf Seite 1369.

  Ab IBM MQ 9.0 wird IBM MQ Transport for SOAP nicht mehr verwendet.

V 9.0.5 V 9.0.0.3 Die Datei JSON4J.jar und das Paket com.ibm.msg.client.mqlight werden von den IBM MQ classes for Java und IBM MQ classes for JMS nicht benötigt. Ab IBM MQ 9.0.0 Fix Pack 3 und IBM MQ 9.0.5 werden daher folgende Änderungen an der Datei com.ibm.mq.allclient.jar vorgenommen:

- Die Referenz auf die Datei JSON4J.jar wird aus der Klassenpfadanweisung in der Manifestdatei für die Datei com.ibm.mq.allclient.jar entfernt.
- Das Paket com.ibm.msg.client.mqlight ist nicht mehr in der Datei com.ibm.mq.allclient.jar enthalten.

Beispielanwendungen

Mit IBM MQ classes for JMS werden einige Beispielanwendungen bereitgestellt. In Tabelle 6 auf Seite 84 sehen Sie, wo die Beispielanwendungen auf den einzelnen Plattformen installiert werden.

Tabelle 6. Beispielverzeichnisse

Plattform	Directory
AIX UNIX 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*/mqm/V9R0M0/java/samples/jms
	MQ_INSTALLATION_PATH/opt/mqm/samp/jms

MQ_INSTALLATION_PATH steht 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.

Im Abschnitt „Umgebungsvariablen einrichten" auf Seite 85 wird der Klassenpfad beschrieben, der für die Ausführung einfacher Anwendungen der IBM MQ classes for JMS 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 „Konfigurationsdatei für IBM MQ classes for JMS" auf Seite 89 beschrieben.

Verschiebbare JAR-Dateien

Innerhalb eines Unternehmens können die folgenden Dateien auf Systeme verschoben werden, die IBM MQ classes for JMS ausführen müssen:

- `-com.ibm.mq.allclient.jar`
- `-com.ibm.mq.traceControl.jar`
- `-jms.jar`
- `-fscontext.jar`
- `-providerutil.jar`
- Sicherheitsprovider Bouncy Castle und JAR-Dateien für CMS-Unterstützung

Die Dateien `fscontext.jar` und `providerutil.jar` sind erforderlich, wenn Ihre Anwendung unter Verwendung eines Dateisystemkontexts JNDI-Suchvorgänge ausführt.

Der Sicherheitsprovider Bouncy Castle und die JAR-Dateien für CMS-Unterstützung sind erforderlich. Weitere Informationen finden Sie im Abschnitt Unterstützung für JREs anderer Anbieter. Die folgenden JAR-Dateien sind erforderlich:

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- V 9.0.0.12 `bcutil-jdk15on.jar`

Die Datei `com.ibm.mq.allclient.jar` enthält die IBM MQ classes for JMS, die IBM MQ classes for Java sowie die PCF-und Headerklassen. Wenn Sie diese Datei an eine neue Position verschieben, müssen

Sie unbedingt entsprechende Schritte ergreifen, damit diese neue Position mit neuen Fixpacks zu IBM MQ gewartet wird. Sorgen Sie zudem dafür, dass die Verwendung dieser Datei dem IBM Support bekannt ist, wenn Sie einen vorläufigen Fix erhalten.

Sie können die Version der Datei `com.ibm.mq.allclient.jar` mit folgendem Befehl bestimmen:

```
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:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.0.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Die Datei `com.ibm.mq.traceControl.jar` dient zur dynamischen Tracesteuerung für IBM MQ classes for JMS-Anwendungen. Weitere Informationen finden Sie im Abschnitt Trace in einem aktiven Prozess mithilfe von WebSphere MQ-Klassen für Java und WebSphere MQ-Klassen für JMS steuern.

Zugehörige Informationen

Probleme beim Implementieren des Ressourcenadapters

Umgebungsvariablen einrichten

Damit Sie Anwendungen der IBM MQ classes for JMS kompilieren und ausführen können, muss die Einstellung Ihrer CLASSPATH-Umgebungsvariablen die JAR-Datei (Java Java Archive) für die IBM MQ classes for JMS enthalten. Abhängig von den gegebenen Voraussetzungen müssen Sie Ihrem Klassenpfad möglicherweise weitere JAR-Dateien hinzufügen. Für die Ausführung der zusammen mit IBM MQ classes for JMS bereitgestellten Scripts müssen weitere Umgebungsvariablen festgelegt werden.

Informationen zu diesem Vorgang

Wichtig: Die Einstellung der Java-Option *-Xbootclasspath* zum Einschließen der IBM MQ classes for JMS wird nicht unterstützt.

Wenn Sie Anwendungen der IBM MQ classes for JMS kompilieren und ausführen möchten, verwenden Sie die jeweilige CLASSPATH-Einstellung für Ihre Plattform (siehe Tabelle 7 auf Seite 86). Diese Einstellung enthält das samples-Verzeichnis, sodass Sie die IBM MQ classes for JMS-Beispielanwendungen kompilieren und ausführen können. Alternativ zur Verwendung der Umgebungsvariablen können Sie den Klassenpfad auch im **java**-Befehl angeben.

Tabelle 7. CLASSPATH-Einstellung für die Kompilierung und Ausführung von Anwendungen der IBM MQ classes for JMS einschließlich der Beispielanwendungen	
Plattform	**CLASSPATH-Einstellung**
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Solaris Linux HP-UX HP-UX, Linux und Solaris	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/samples/jms/samples:

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Das Manifest der JAR-Datei `com.ibm.mqjms.jar` enthält Verweise auf die meisten für IBM MQ classes for JMS-Anwendungen erforderlichen JAR-Dateien; daher müssen diese JAR-Dateien dem Klassenpfad nicht hinzugefügt werden. 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 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 JMS installiert ist.

MQ_JAVA_LIB_PATH
Diese Umgebungsvariable gibt das Verzeichnis an, in dem die IBM MQ classes for JMS-Bibliotheken gespeichert sind (siehe Tabelle 8 auf Seite 88).

Prozedur

- Windows
  Führen Sie unter Windowsnach der Installation von IBM MQden Befehl **setmqenv**aus.
  Wenn dieser Befehl nicht zuerst ausgeführt wird, erhalten Sie bei Ausgabe des Befehls **dspmqver** eventuell die folgende Fehlernachricht:

  V 9.0.2
  ```
  AMQ8351: IBM MQ Java environment has not been configured
  correctly, or the IBM MQ JRE feature has not been installed.
  ```

  Anmerkung: V 9.0.2 Wenn IBM MQ Java Runtime Environment (JRE) nicht installiert ist, müssen Sie mit diesem Fehler rechnen.
- Auf allen anderen Plattformen richten Sie die Umgebungsvariablen manuell ein:
  - Linux UNIX Um bei Verwendung einer 32-Bit-JVM auf UNIX-, - oder Linux-Systemen die Umgebungsvariablen einzustellen, können Sie das Script 'setjmsenv' verwenden.
  - Linux UNIX Um bei Verwendung einer 64-Bit-JVM auf einem UNIX oder Linux-System die Umgebungsvariablen festzulegen, können Sie das Script 'setjmsenv64' verwenden. 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.

  Die Scripts 'setjmsenv' und 'setjmsenv64' können auf verschiedene Arten verwendet werden: Sie können sie als Basis für die Festlegung der erforderlichen Umgebungsvariablen nutzen, die in der Tabelle aufgeführt sind, oder sie mit einem Texteditor 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 durch Eingabe von `. ./setjmsenv` oder `. ./setjmsenv64` starten.

  IBM i Unter IBM i müssen Sie die Umgebungsvariable QIBM_MULTI_THREADED auf Y setzen. Sie können dann Multithread-Anwendungen genau wie bei der Ausführung von Einzelthread-Anwendungen ausführen. Weitere Informationen hierzu finden Sie im Abschnitt IBM MQ mit Java und JMS konfigurieren.

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.

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 9.0 installierte Bibliothek 'mqjbnd' kann zum Herstellen einer Verbindung zu jedem beliebigen Warteschlangenmanager für IBM MQ 9.0 (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 8. 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 HP-UX Linux Linux (Plattformen POWER, x86-64 und zSeries s390x) Solaris Solaris (Plattformen x86-64 und SPARC)	*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*/mqm/V8R0M0/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 `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:

```
export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
```

```
export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIB
RARY_PATH
```

```
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.

Konfigurationsdatei für IBM MQ classes for JMS
In einer Konfigurationsdatei für die IBM MQ classes for JMS sind Eigenschaften angegeben, die zur Konfiguration der IBM MQ classes for JMS verwendet 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.

Die Konfigurationsdatei für die IBM MQ classes for JMS hat das Format einer standardmäßigen Java-Eigenschaftendatei. Im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS befindet sich eine Beispielkonfigurationsdatei mit dem Namen `jms.config`. 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 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 JMS 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
```

In diesem Befehl ist als Konfigurationsdatei für die IBM MQ classes for JMS die Datei `D:\mydir\mjms.config` im lokalen Windows-System angegeben.

Wenn eine Anwendung gestartet wird, lesen die IBM MQ classes for JMS 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 JMS die Standardwerte für alle Eigenschaften.

Eine Konfigurationsdatei für IBM MQ classes for JMS 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 JMS verwendet 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.

Falls erforderlich, können Sie jedes Attribut in einer IBM MQ MQI client-Konfigurationsdatei überschreiben, indem Sie es als Eigenschaft in einer Konfigurationsdatei für IBM MQ classes for JMS 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 JMS 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 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 relevant. Wenn Sie andere Attribute angeben oder außer Kraft setzen, hat dies keine Wirkung. Beachten Sie insbesondere, dass die `ChannelDefinitionFile` und `ChannelDefinitionDirectory` in der Zeilengruppe CHANNELS der Clientkonfigurationsdatei nicht verwendet werden. Einzelheiten zur Verwendung der CCDT mit den IBM MQ classes for JMS finden Sie im Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden” auf Seite 273.

Tabelle 9. 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
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 des IBM MQ MQI clients finden Sie im Abschnitt Client mithilfe einer Konfigurationsdatei konfigurieren.

Zeilengruppe für den Java-Standardumgebungstrace
Über die Zeilengruppe für die Java-Standardumgebungs-Traceeinstellungen können Sie die Tracefunktion der IBM MQ classes for JMS 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:

- Bei IBM MQ classes for JMS für IBM MQ 9.0.0 Fix Pack 1 oder früher wird der Trace in eine Datei mit dem Namen `mqjms_%PID%.trc` geschrieben.
- V 9.0.0.2 Ab IBM MQ 9.0.0 Fix Pack 2 wird der Trace in eine Datei namens `mqjava_%PID%.trc` geschrieben, falls die Anwendung die IBM MQ classes for JMS aus der JAR-Datei `com.ibm.mqjms.jar` geladen hat.
- V 9.0.0.2 Ab IBM MQ 9.0.0 Fix Pack 2 wird der Trace in eine Datei mit dem Namen `mqjavaclient_%PID%.trc` geschrieben, wenn die Anwendung die IBM MQ classes for JMS aus der verschiebbaren JAR-Datei `com.ibm.mq.allclient.jar` geladen hat.
- V 9.0.0.10 Ab IBM MQ 9.0.0 Fix Pack 10 wird der Trace in eine Datei namens `mqjava_%PID%.cl%u.trc` geschrieben, falls die Anwendung die IBM MQ classes for JMS aus der JAR-Datei `com.ibm.mqjms.jar` geladen hat.
- V 9.0.0.10 Ab IBM MQ 9.0.0 Fix Pack 10 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.

Dabei steht *%PID%* für die Prozess-ID der Anwendung, für die ein Trace erstellt wird, und *%u* für die eindeutige Nummer, durch die Dateien zwischen Threads unterschieden werden, die das Trace unter verschiedenen Java-Klassenladeprogrammen ausführen.

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.

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 JMS 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 JMS 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 JMS 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 JMS 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 JMS 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 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 unmittelbar, nachdem von den IBM MQ classes for JMS eine FDC-Datei generiert wurde, eine JavaCore-Datei 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 die IBM MQ classes for JMS verwendet, über den normalen IBM MQ-Messaging-Provider-Modus eine Verbindung zu einem IBM MQ-Warteschlangenmanager herstellt, kann die Eigenschaft `receiveConversionCCSID` festgelegt werden, um den CCSID-Standardwert in der MQMD-Struktur, die zum Empfangen von Nachrichten vom Warteschlangenmanager verwendet wird, außer Kraft zu setzen. 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. Eine Zurücksetzung dieser Änderungen ist eventuell für einige JMS 2.0-Anwendungen erforderlich sowie für einige Anwendungen, die die JMS 1.1-API verwenden, jedoch auf die IBM MQ classes for JMS von IBM MQ 8.0 angewendet werden.

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`
Nachdem eine Anwendung eine Byte- oder Datenstromnachricht gesendet hat, können die IBM MQ classes for JMS ab IBM MQ 8.0.0 Fix Pack 2 den Status der soeben gesendeten Nachricht auf schreib- oder lesegeschützt 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 321
Mit der JMS 2.0-Spezifikation werden Änderungen bestimmter Verhaltensweisen eingeführt. Ab 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.

STEPLIB-Konfiguration für IBM MQ classes for JMS unter z/OS
Unter z/OS muss die zur Laufzeit verwendete STEPLIB die Bibliotheken IBM MQ SCSQAUTH und SCSQANLE enthalten. Geben Sie diese Bibliotheken in der Start-JCL oder unter Verwendung der Datei `.profile` an.

Ab UNIX and Linux System Services können Sie diese mithilfe einer Zeile in Ihrem `.profile` hinzufügen, wie im folgenden Code-Snippet gezeigt, indem Sie `thlqual` durch das übergeordnete Dateiqualifikationsmerkmal ersetzen, das Sie bei der Installation von IBM MQausgewählt haben:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In anderen Umgebungen müssen Sie normalerweise die Start-JCL so bearbeiten, dass sie SCSQAUTH und SCSQANLE in der STEPLIB-Verkettung enthält:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS und Softwareverwaltungstools
Mit den IBM MQ classes for JMS 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 JMS 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.

Ab IBM MQ 8.0 müssen die folgenden JAR-Dateien im Repository installiert werden:

- `com.ibm.mq.allclient.jar`.
- Die Datei `jms.jar` ist erforderlich, wenn Sie die IBM MQ classes for JMS verwenden.
- Die Datei `fscontext.jar` ist erforderlich, wenn Sie die IBM MQ classes for JMS verwenden und auf verwaltete JMS-Objekte zugreifen, die in einem JNDI-Dateisystemkontext gespeichert sind.
- Die Datei `providerutil.jar` ist erforderlich, wenn Sie die IBM MQ classes for JMS verwenden und auf verwaltete JMS-Objekte zugreifen, die in einem JNDI-Dateisystemkontext gespeichert sind.

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 „Dies wird für IBM MQ classes for JMS installiert" auf Seite 82 und Unterstützung für JREs anderer Anbieter.

IBM MQ classes for JMS-Anwendungen unter dem Java Security Manager ausführen

IBM MQ classes for JMS können bei aktiviertem Java-Sicherheitsmanager 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 befindet sich diese Datei im Verzeichnis `lib/security/java.policy` (relativ zum JRE-Verzeichnis). Sie können die Richtlinienkonfigurationsdatei entweder mit Ihrem bevorzugten Editor oder mit dem Programm policytool, das mit Ihrer JRE bereitgestellt wurde, bearbeiten.

Wichtig:

Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Für IBM MQ 9.0 und höhere Releases umfasst dies die die in diesem Thema erwähnten Java-Systemeigenschaftsnamen (**`com.ibm.mq.jms.*`**). Die vorherigen Systemeigenschaftsnamen funktionieren auch weiterhin.

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.
- PropertyPermission (read) für die Eigenschaften **`com.ibm.mq.jms.allowlist`**, **`com.ibm.mq.jms.allowlist.discover`** und **`com.ibm.mq.jms.allowlist.mode`**.

Für Continuous Deliverywird die Zulassungsliste ClassName von IBM MQ 9.0.1unterstützt. Weitere Informationen finden Sie unter „Zulassungslistenkonzepte” auf Seite 121.

V 9.0.0.1 Im Release Long Term Support wird die Zulassungsliste ClassName mit APAR IT14385und ab IBM MQ 9.0.0 Fix Pack 1unterstützt.

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* für die Speicherposition 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.

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.

Eine Anwendung, die in WebSphere Application Server unter z/OS ausgeführt wird, kann entweder im Bindungsmodus oder im Clientmodus eine Verbindung zu einem Warteschlangenmanager herstellen. Bei anderen Umgebungen unter z/OS ist die Herstellung einer Verbindung zu einem Warteschlangenmanager jedoch nur im Bindungsmodus möglich. 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 87.

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_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- 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 763.

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.

ULW Unter UNIX, 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

Mit den IBM MQ classes for JMS wird ein Programm für einen Punkt-zu-Punkt-Installationsprüftest (Installation Verification Test, IVT) bereitgestellt. 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.

Für die Ausführung des Programms des Installationsprüftests wird ein Script zur Verfügung gestellt. Auf Systemen mit UNIX and Linux heißt das Script 'IVTRun' und unter Windows heißt es 'IVTRun.bat'. Es befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS.

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, müssen Sie zuerst den Warteschlangenmanager wie im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1135 beschrieben einrichten. Beachten Sie, dass für den zu verwendenden Kanal standardmäßig der Wert SYSTEM.DEF.SVRCONN übernommen und als zu verwendende Warteschlange SYSTEM.DEFAULT.LOCAL.QUEUE festgelegt ist. Geben Sie dann folgenden Befehl ein:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

Auf z/OS-Systemen wird zwar kein entsprechendes Script bereitgestellt, Sie können den Installationsprüftest aber im Bindungsmodus ausführen, indem Sie die Java-Klasse mit folgendem Befehl direkt aufrufen:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Der Klassenpfad muss com.ibm.mqjms.jar. enthalten

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, 2023. 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 80. 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. Auf Systemen mit UNIX and Linux heißt das Script 'IVTSetup' und unter Windows heißt es 'IVTSetup.bat'. Es befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS. 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. Auf Systemen mit UNIX and Linux heißt das Script 'IVTTidy' und unter Windows heißt es 'IVTTidy.bat'. Es befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS.

Problembestimmung beim Punkt-zu-Punkt-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 einrichten” auf Seite 85 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, für den entweder `Bindings` oder `Client` angegeben wird.

  hostname (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. Details hierzu finden Sie unter „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1135.

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. Auf Systemen mit UNIX and Linux heißt das Script 'PSIVTRun' und unter Windows heißt es 'PSIVTRun.bat'. Es befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS.

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 1135 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, 2023. All
Rights Reserved.
IBM MQ classes for Java(tm) 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 80. 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. Auf Systemen mit UNIX and Linux heißt das Script 'IVTSetup' und unter Windows heißt es 'IVTSetup.bat'. Es befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS. 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. Auf Systemen mit UNIX and Linux heißt das Script 'IVTTidy' und unter Windows heißt es 'IVTTidy.bat'. Es befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS.

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 einrichten” auf Seite 85 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, für den entweder `Bindings` oder `Client` angegeben wird.

hostname (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 im Abschnitt „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms" auf Seite 1135.

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 im Abschnitt „Anwendungen der IBM MQ classes for JMS schreiben" auf Seite 129. Die Beispiele könnten sich in zukünftigen Releases von IBM MQ ändern.

In Tabelle 10 auf Seite 112 sehen Sie, wo die Beispielanwendungen der IBM MQ classes for JMS auf den einzelnen Plattformen installiert werden:

Tabelle 10. Installationsverzeichnisse für die Beispielanwendungen der IBM MQ classes for JMS

Plattform	Directory
UNIX UNIX 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 11 auf Seite 112 dargestellt.

Tabelle 11. IBM MQ classes for JMS-Beispielanwendungen

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.

Tabelle 11. IBM MQ classes for JMS-Beispielanwendungen (Forts.)	
Name der Beispielanwendung	**Beschreibung**
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.
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.

Tabelle 11. IBM MQ classes for JMS-Beispielanwendungen (Forts.)	
Name der Beispielanwendung	**Beschreibung**
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.
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 12 auf Seite 115 zeigt den Speicherort des Scripts auf den einzelnen Plattformen:

Tabelle 12. Speicherort des Scripts `runjms`	
Plattform	**Directory**
UNIX UNIX 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 das Script `runjms` zum Aufrufen einer Beispielanwendung zu verwenden:

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 zum Beispiel die folgenden Befehle ein, um das JmsBrowser-Beispiel 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 82
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 bereitgestellte Scripts

Es werden mehrere Scripts bereitgestellt, die Sie bei allgemeinen Tasks unterstützen, die bei der Verwendung von IBM MQ classes for JMS ausgeführt werden müssen.

Tabelle 13 auf Seite 116 listet alle Scripts und ihre Verwendungen auf. Die Scripts befinden sich im Unterverzeichnis `bin` des Installationsverzeichnisses der IBM MQ classes for JMS.

Tabelle 13. Mit IBM MQ classes for JMS bereitgestellte Scripts	
Versorgungsbetrieb	**Verwenden Sie**
Cleanup [1]	Dieses Script steht aus Gründen der Kompatibilität mit Vorgängerreleases noch zur Verfügung, bleibt jedoch ohne Funktion. Eine 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 103).
JMSAdmin [1]	Führt das IBM MQ-JMS-Verwaltungstool aus (siehe Beschreibung im Abschnitt Verwaltungstool starten).
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 108).
PSReportDump.class	Diese Klasse steht aus Gründen der Kompatibilität mit Vorgängerreleases noch zur Verfügung, bleibt jedoch ohne Funktion.
setjmsenv	Legt die Umgebungsvariablen für die Ausführung einer Anwendung der IBM MQ classes for JMS in einer 32-Bit-Version der Java Virtual Machine (JVM) auf Systemen mit UNIX and Linux fest (siehe Beschreibung im Abschnitt „Umgebungsvariablen einrichten" auf Seite 85).
setjmsenv64	Legt die Umgebungsvariablen für die Ausführung einer Anwendung der IBM MQ classes for JMS in einer 64-Bit-JVM auf Systemen mit UNIX and Linux fest (siehe Beschreibung im Abschnitt „Umgebungsvariablen einrichten" auf Seite 85).

Anmerkung:

1. Unter Windows hat der Dateiname die Erweiterung .bat.

Unterstützung von OSGi

OSGi stellt ein Framework bereit, das die Implementierung von Anwendungen als Bundles unterstützt. Mit den IBM MQ classes for JMS werden neun OSGi-Bundles 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 dynamische und skalierbare Weise.

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.osgi.allclient_*version_number*.jar
Diese JAR-Datei ermöglicht es Anwendungen, sowohl die IBM MQ classes for JMS als auch die IBM MQ classes for Java zu verwenden. Darüber hinaus enthält sie den Code für die Behandlung von PCF-Nachrichten.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
Diese JAR-Datei stellt die Voraussetzungen für `com.ibm.mq.osgi.allclient_`*`version_number`*`.jar` bereit. Dabei steht *`version_number`* für die Versionsnummer der IBM MQ-Installation.

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. Für Produktversionen vor IBM MQ 8.0 gilt, die Einschränkung, entweder IBM MQ classes for JMS oder IBM MQ classes for Java zu verwenden, allerdings 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 Klassenladeprogrammen 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.

IBM MQ classes for JMS separat abrufen

Die IBM MQ classes for JMS sind in einer selbstextrahierenden JAR-Datei verfügbar, die Sie von Fix Central herunterladen können, wenn Sie nur die JAR-Dateien der IBM MQ classes for JMS zur Implementierung in einem Software-Management-Tool oder zur Verwendung mit eigenständigen Clientanwendungen abrufen möchten.

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, der es ausführt, Schreibzugriff auf das Verzeichnis hat, in dem die Dateien abgelegt werden sollen.

Informationen zu diesem Vorgang

Vor IBM MQ 8.0 waren die IBM WebSphere MQ classes for Java oder IBM WebSphere MQ classes for JMS nicht als separater Download verfügbar. Wenn Sie für IBM WebSphere MQ 7.5 oder früher Anwendungen der Sprache Java entwickeln und ausführen, die entweder die IBM WebSphere MQ classes for Java oder die IBM WebSphere MQ classes for JMS verwenden, müssen Sie sie installieren, indem Sie entweder eine vollständige Serverinstallation durchführen oder eines der Client-SupportPacs auf dem System, auf dem die Anwendung entwickelt wird, und auf dem System, auf dem die Anwendung ausgeführt werden soll, installieren. Bei dieser Installation werden wesentlich mehr Dateien als die Dateien der IBM WebSphere MQ classes for Java und IBM WebSphere MQ classes for JMS installiert.

Ab IBM MQ 8.0 sind jedoch folgende Dateien in einer selbstextrahierenden JAR-Datei verfügbar, was die Größe des Downloads und der Installation minimiert und die Zeit, die für die Durchführung der Installation erforderlich ist, verkürzt:

- IBM MQ classes for JMS
- IBM MQ classes for Java
- IBM MQ-Ressourcenadapter
- IBM MQ-OSGi-Bundles

Bei Ausführung der JAR-Datei wird die IBM MQ-Lizenzvereinbarung angezeigt, die akzeptiert werden muss. Es wird nach einem Verzeichnis gefragt, in dem die IBM MQ classes for Java, IBM MQ classes for JMS, der Ressourcenadapter und OSGi-Bundles installiert werden sollen. Wenn das ausgewählte Installationsverzeichnis nicht vorhanden ist, wird es erstellt und die Programmdateien werden installiert. Falls das Verzeichnis jedoch vorhanden ist, wird ein Fehler gemeldet und es werden keine Dateien installiert.

Vorgehensweise

1. Laden Sie die IBM MQ Java-JAR-Datei von Fix Central herunter.

   Geben Sie den Ausdruck `"Java"` im Feld **Text Search** (Textsuche) ein, um nach der neuesten Version, die für den Download verfügbar ist, zu suchen. Der Name der Datei, die heruntergeladen werden muss, hat das Format *V.R.M.F*`-WS-MQ-Install-Java-All.jar`, wobei *V.R.M.F* für die Produktversionsnummer steht, z. B. `9.0.0.0`.

   Wenn Sie die Datei nicht finden können, stellen Sie sicher, dass als **Product Selected** (Ausgewähltes Produkt) `WebSphere MQ` und als **Version** `9.0` angegeben sind.
2. Starten Sie die Installation aus dem Verzeichnis, in das Sie die Datei heruntergeladen haben.

   Geben Sie zum Starten der Installation einen Befehl in folgendem Format ein:

   ```
   java -jar V.R.M.F-WS-MQ-Install-Java-All.jar
   ```

   Dabei steht *V.R.M.F* für die Produktversionsnummer, z. B. `9.0.0.0`, und *V.R.M.F*`-WS-MQ-Install-Java-All.jar` für den Namen der Datei, die von Fix Central heruntergeladen wurde.

   Verwenden Sie beispielsweise den folgenden Befehl, um IBM MQ classes for JMS für IBM MQ 9.0.0.0zu installieren:

   ```
   java -jar 9.0.0.0-WS-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 9.0verwenden, extrahieren oder installieren können, müssen Sie die
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 sofort nach der Auswahl von x wird 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.

      Nach der Anzeige der zusätzlichen Lizenzbedingungen oder sofort nach der Auswahl von x wird 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.

      Alternativ können Sie 2 auswählen, um die Installation sofort zu beenden.

      Wenn Sie 1 auswählen, wird folgende Nachricht angezeigt:

      ```
      Enter directory for product files or leave blank to accept the default value.
      The default target directory is H:\WMQ

      Target directory for product files?
      ```

4. Geben Sie das Installationsverzeichnis für den Ressourcenadapter an:

   - Wenn die Produktdateien an der Standardposition installiert werden sollen, drücken Sie die Eingabetaste, ohne einen Wert anzugeben.
   - Wenn Sie die Produktdateien an einer anderen Position als der Standardposition installieren möchten, geben Sie den Namen des Verzeichnisses an, in dem die Produktdateien installiert werden sollen, und drücken Sie dann die Eingabetaste, um die Installation zu starten.

Der angegebene Verzeichnisname darf noch nicht vorhanden sein. Andernfalls wird beim Starten der Installation ein Fehler gemeldet und es werden keine Dateien installiert.

Wenn das ausgewählte Installationsverzeichnis noch nicht vorhanden ist, wird es erstellt und die Programmdateien werden in dem Verzeichnis installiert. Während der Installation wird ein neues Verzeichnis mit dem Namen `wmq` in dem von Ihnen ausgewählten Installationsverzeichnisses erstellt. Es werden drei Unterverzeichnisse, `JavaEE`, `JavaSE` und `OSGi`, mit folgenden Inhalten im Verzeichnis `wmq` erstellt:

```
.\JavaEE:
wmq.jmsra.ivt.ear
wmq.jmsra.rar

.\JavaSE:
com.ibm.mq.allclient.jar
com.ibm.mq.traceControl.jar
fscontext.jar
```

```
jms.jar
providerutil.jar

.\OSGi:
com.ibm.mq.osgi.allclient_V.R.M.F.jar
com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar
```

Dabei steht *V.R.M.F* wird die Versions-, Release-, Modifikations- und Fixpacknummer.

V 9.0.5 V 9.0.0.3 Vor IBM MQ 9.0.0 Fix Pack 3 und IBM MQ 9.0.5 enthielten die im Verzeichnis JavaSE installierten Dateien auch die Datei JSON4J.jar. Diese JAR-Datei ist jedoch nicht erforderlich und wird daher aus der Datei *V.R.M.F*-WS-MQ-Install-Java-All.jar aus IBM MQ 9.0.0 Fix Pack 3 und IBM MQ 9.0.5entfernt. Außerdem gibt es ab IBM MQ 9.0.0 Fix Pack 3 und IBM MQ 9.0.5zwei Änderungen an der com.ibm.mq.allclient.jar file:

- Die Referenz auf die Datei JSON4J.jar wird aus der Klassenpfadanweisung in der Manifestdatei für die Datei com.ibm.mq.allclient.jar entfernt.
- Das Paket com.ibm.msg.client.mqlight ist nicht mehr in der Datei com.ibm.mq.allclient.jar enthalten.

Wenn die Installation abgeschlossen ist, wird eine Bestätigungsnachricht wie im folgenden Beispiel angezeigt:

```
Extracting files to H:\WMQ\wmq
Successfully extracted all product files.
```

V 9.0.0.1 Verwendung einer Zulassungsliste in IBM MQ classes for JMS

Der Mechanismus zur Serialisierung und Deserialisierung von Java-Objekten wurde als potenzielles Sicherheitsrisiko erkannt. Die Verwendung einer Zulassungsliste in IBM MQ classes for JMS bietet einen gewissen Schutz gegen einige Serialisierungsrisiken.

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 erwähnenswerte Serialisierungsanwendung findet sich in Java Message Service-(JMS-)ObjectMessage, wo mithilfe einer Serialisierung beliebige Objekte eingebunden und übertragen werden.

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.

Verwendung einer Zulassungsliste in IBM MQ classes for JMS

Unter

- „Zulassungslistenkonzepte” auf Seite 121 mit einer Übersicht über die Zulassungslistenfunktion
- „JMS-Zulassungsliste einrichten und verwenden” auf Seite 124 mit Informationen zum Einrichten einer Zulassungsliste
- „Verwendung einer Zulassungsliste in WebSphere Application Server” auf Seite 126 mit Informationen zum Einrichten einer Zulassungsliste in WebSphere Application Server

Zugehörige Konzepte

„IBM MQ classes for JMS-Anwendungen unter dem Java Security Manager ausführen” auf Seite 98
IBM MQ classes for JMS können bei aktiviertem Java-Sicherheitsmanager 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.

V 9.0.0.1 *Zulassungslistenkonzepte*

In den IBM MQ classes for JMS bewirkt die Unterstützung von Zulassungslisten für Klassen in der Implementierung der JMS-ObjectMessage-Schnittstelle eine potenzielle Minderung der Sicherheitsrisiken, die durch den Mechanismus zur Serialisierung und Deserialisierung von Java-Objekten entstehen können.

Verwendung einer Zulassungsliste in IBM MQ classes for JMS

Wichtig:

Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Für IBM MQ 9.0 und höhere Releases umfasst dies die die in diesem Thema erwähnten Java-Systemeigenschaftsnamen (**`com.ibm.mq.jms.*`**). Die vorherigen Systemeigenschaftsnamen funktionieren auch weiterhin.

IBM MQ classes for JMS unterstützen die Verwendung einer Zulassungsliste für Klassen in der Implementierung der JMS-ObjectMessage-Schnittstelle.

In der Zulassungsliste wird definiert, welche Java-Klassen mit ObjectMessage.setObject() serialisiert und mit ObjectMessage.getObject() deserialisiert werden können.

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.

Zulassungsliste erstellen

Wichtig: Die IBM MQ classes for JMS 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
: 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

- 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 der Eigenschaft com.ibm.mq.jms.allowlist property angegeben 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

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.*` für `com.ibm.mq.MQMessage` zutreffen, aber nicht für `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 eine durch Kommas getrennte Liste von Klassennamen ist, wird das Zeichen als normales Zeichen behandelt, da angenommen wird, dass das Betriebssystem oder die Shell es nicht verarbeitet, weil es sich um das Standardkommentarzeichen in vielen UNIX- oder Linux-Shells handelt.

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.
- PropertyPermission (read) für die Eigenschaften **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover** und **com.ibm.mq.jms.allowlist.mode**.

Zusatzinformationen

V 9.0.0.1

Weitere Informationen zu Zulassungslisten finden Sie in den Abschnitten „JMS-Zulassungsliste einrichten und verwenden” auf Seite 124 und „Verwendung einer Zulassungsliste in WebSphere Application Server” auf Seite 126.

Zugehörige Konzepte

„IBM MQ classes for JMS-Anwendungen unter dem Java Security Manager ausführen” auf Seite 98
IBM MQ classes for JMS können bei aktiviertem Java-Sicherheitsmanager 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.

V 9.0.0.1 *JMS-Zulassungsliste einrichten und verwenden*

Hier erfahren Sie, wie eine Zulassungsliste funktioniert und wie Sie mit der in den IBM MQ classes for JMS enthaltenen Funktionalität eine Zulassungslistendatei mit einer Liste der ObjectMessages-Typen generieren, die eine Anwendung verarbeiten kann.

Vorbereitende Schritte

Wichtig:

Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Für IBM MQ 9.0 und höhere Releases umfasst dies die die in diesem Thema erwähnten Java-Systemeigenschaftsnamen (**com.ibm.mq.jms.***). Die vorherigen Systemeigenschaftsnamen funktionieren auch weiterhin.

Vor Beginn dieser Task sollten Sie die Informationen im Abschnitt „Zulassungslistenkonzepte” auf Seite 121 gelesen und zur Kenntnis genommen haben.

Informationen zu diesem Vorgang

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 wurde.
```

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 nur den Eintrag `java.net.`URL enthält, und die Anwendung mit der festgelegten Java-Systemeigenschaft

```
-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 aus, die ObjectMessages verarbeitet, wobei die folgenden Java-Systemeigenschaften angegeben sind:

   ```
   -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.

V 9.0.0.1 *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. Für IBM MQ 9.0 und höhere Releases umfasst dies die die in diesem Thema erwähnten Java-Systemeigenschaftsnamen (**com.ibm.mq.jms.***). 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. Die Funktion wurde dem Ressourcenadapter als Teil des APAR IT14385 hinzugefügt.

Weitere Informationen zur Verwendung der beiden Produkte finden Sie unter „IBM MQ und WebSphere Application Server gemeinsam verwenden” auf Seite 499.

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-Zulassungsliste einrichten und verwenden” auf Seite 124 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 *Generische JVM-Argumente* im Abschnitt Java Virtual Machine-Einstellungen.

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
```

wird der Anwendungsserver für die Verwendung des Modus *Discover* (Erkennen) konfiguriert und Details zu den JMS-ObjectMessages, die der Anwendungsserver verarbeitet, werden in der Datei `C:\allowlist.txt` aufgezeichnet.

Die folgende Einstellung:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

bewirkt, dass der Anwendungsserver die Datei `C:/allowlist.txt` lädt und anhand der Informationen in dieser Datei die Zulassungsliste bestimmt.

Zugehörige Konzepte

„IBM MQ classes for JMS-Anwendungen unter dem Java Security Manager ausführen” auf Seite 98
IBM MQ classes for JMS können bei aktiviertem Java-Sicherheitsmanager 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 8.0wurden Zeichenfolgekonvertierungen in IBM MQ classes for JMS durch Aufrufen der Methoden `java.nio.charset.Charset.decode(ByteBuffer)` und `Charset.encode(CharBuffer)` durchgeführt.

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.

Konfiguration

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.

UnmappableCharacter-Einstellungen für Aktions-und Ersatzbyte in IBM MQ -Klassen für 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 341
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.

Anwendungen der IBM MQ classes for JMS schreiben

Nach einer kurzen Einführung in das JMS-Modell finden Sie in diesem Abschnitt eine ausführliche Anleitung für das Schreiben von Anwendungen der IBM MQ classes for JMS.

JMS-Modell

Das Modell JMS definiert eine Gruppe von Schnittstellen, die Java -Anwendungen verwenden können, um Messaging-Operationen auszuführen. IBM MQ classes for JMS als JMS -Provider definiert, wie JMS -Objekte mit IBM MQ -Konzepten in Beziehung stehen. Die JMS-Spezifikation erwartet, dass es sich bei bestimmten JMS-Objekten um verwaltete Objekte handelt. In JMS 2.0 wurde eine vereinfachte API eingeführt. Die klassische API aus JMS 1.1 wurde jedoch ebenfalls beibehalten.

Die JMS-Spezifikation und das Paket javax.jms definieren eine Reihe von Schnittstellen für Java-Anwendungen zur Ausführung von Messaging-Operationen.

Ab IBM MQ 8.0 unterstützt das Produkt die Version JMS 2.0 des JMS-Standards, Mit dieser Version wurde eine vereinfachte API eingeführt. Die klassische API aus JMS 1.1 wurde jedoch ebenfalls beibehalten.

Vereinfachte API

In JMS 2.0 wurde die vereinfachte API eingeführt. Die domänenspezifischen und domänenunabhängigen Schnittstellen aus JMS 1.1 wurden jedoch ebenfalls beibehalten. 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.

JMSProducer
: 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.

JMSConsumer
: 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. Mithilfe von JMS 2.0 können Sie wie folgt Eigenschaften festlegen und eine Nachricht senden:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

In JMS 2.0 wurden außerdem 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:

Ziel
: 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.

Sitzungen
: 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 131 sind diese Objekte und ihre Beziehungen dargestellt.

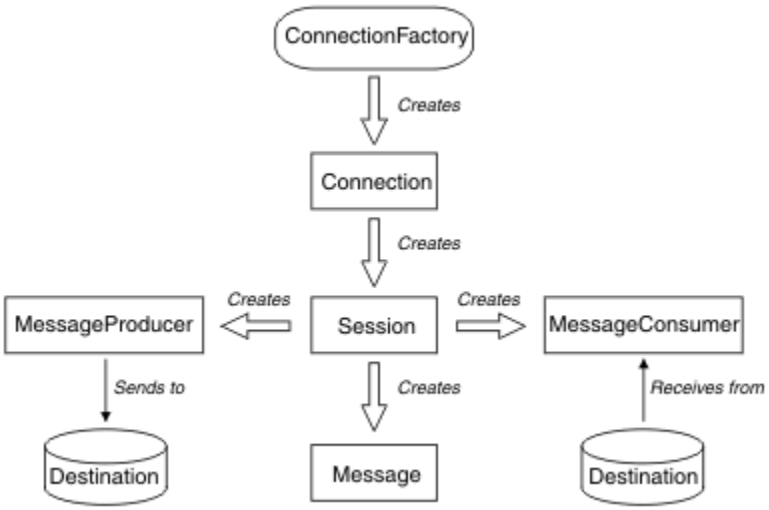


Abbildung 9. JMS-Objekte und ihre Beziehungen

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.

JMS unterstützt zwei Arten von Messaging:

- Punkt-zu-Punkt-Messaging
- Publish/Subscribe-Messaging

Diese Arten des Messaging werden auch als *Messaging-Domänen* bezeichnet. In einer Anwendung können beide Arten kombiniert werden. In einer Punkt-zu-Punkt-Domäne ist das Ziel eine Warteschlange, in einer Publish/Subscribe-Domäne ist es ein Thema.

Bei JMS-Versionen vor JMS 1.1 wird bei der Programmierung für die Punkt-zu-Punkt-Domäne eine Gruppe von Schnittstellen und Methoden verwendet, während bei der Programmierung für die Publish/Subscribe-Domäne eine andere Gruppe verwendet wird. Auch wenn sich beide ähneln, sind sie völlig getrennt voneinander. Ab JMS 1.1 können Sie eine gemeinsame Gruppe von Schnittstellen und Methoden verwenden, die beide Messaging-Domänen unterstützen. Diese gemeinsamen Schnittstellen bieten eine domänenunabhängige Darstellung der Messaging-Domänen. In Tabelle 14 auf Seite 131 sehen Sie eine Übersicht über die domänenunabhängigen JMS-Schnittstellen und deren domänenspezifischen Entsprechungen.

Tabelle 14. 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
Ziel	Warteschlange	Thema
Sitzungen	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher

Tabelle 14. Domänenunabhängige und domänenspezifische JMS-Schnittstellen (Forts.)		
Domänenunabhängige Schnittstellen	**Domänenspezifische Schnittstellen für Punkt-zu-Punkt-Domäne**	**Domänenspezifische Schnittstellen für Publish/Subscribe-Domäne**
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

Da in JMS 2.0 2.0 alle domänenspezifischen Schnittstellen beibehalten wurden, können bestehende Anwendungen diese Schnittstellen nach wie vor verwenden. Für neue Anwendungen sollten Sie jedoch die domänenunabhängigen Schnittstellen aus JMS 1.1 oder die vereinfachte API aus JMS 2.0 verwenden.

In IBM MQ classes for JMS bestehen folgende Beziehungen zwischen denJMS-Objekten und den Konzepten von IBM MQ:

- 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 der IBM MQ classes for JMS gelten alle üblichen Regeln vonIBM 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 ist die Implementierung der Destination-Schnittstelle eine abstrakte Superklasse von Queue und Topic. Das bedeutet, dass 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 können daher Objekte der folgenden Typen verwaltete Objekte sein:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- Warteschlange
- Thema
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Zugehörige Konzepte

„Funktionalität von JMS 2.0 nutzen” auf Seite 315
JMS 2.0 bietet mehrere neue Funktionalitätsbereiche für die IBM MQ classes for JMS.

Zugehörige Informationen
IBM MQ Java-Sprachenschnittstellen

JMS-Nachrichten

JMS-Nachrichten setzen sich aus Header, Eigenschaften und Hauptteil zusammen. JMS definiert fünf Typen für den Nachrichtenhauptteil.

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.

Properties
Jede Nachricht enthält eine integrierte Funktion zur Unterstützung anwendungsdefinierter Eigenschaftswerte. Die Eigenschaften bieten einen effizienten Mechanismus zum Filtern anwendungsdefinierter Nachrichten.

Body
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.

Object
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 von Java 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*'

Dabei ist *message_id* eine Zeichenfolge, die eine IBM MQ -Standard-Nachrichten-ID enthält. Anschließend 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*'

Dabei ist *correlation_id* eine Zeichenfolge, die eine standardmäßige IBM MQ -Korrelations-ID enthält. Die IBM MQ -Klassen für JMS verwenden 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:000000000000000000000000000000000000000000000000'
```

an, wird angenommen, dass jede Nachricht in der Warteschlange mit dem Selektor übereinstimmt, und an die Anwendung zurückgegeben.

Weitere Informationen zu MQMO_MATCH_MSG_ID und MQMO_MATCH_CORREL_ID **MatchOptions**finden Sie im Artikel 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 138 ist dargestellt, wie die Struktur einer JMS-Nachricht in eine IBM MQ-Nachricht (und umgekehrt) transformiert wird:

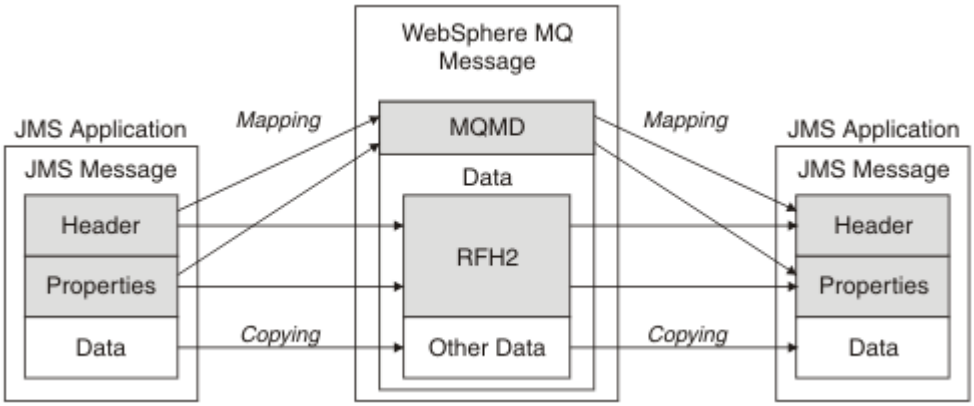


Abbildung 10. Verfahren für die Transformation von Nachrichten zwischen JMS und IBM MQ unter Verwendung des MQRFH2-Headers

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.)

Die NameValueCCSID erhält Werte aus der folgenden Tabelle:

Tabelle 15. Mögliche Werte für das NameValueCCSID-Feld

CCSID	**Bedeutet**
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 in einer JMS-Nachricht mit einem `MQRFH2`-Header immer vorhanden.

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 16. `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 im Ordner `mcd` hinzu.

Ordner `jms`

`jms` enthält JMS-Headerfelder sowie JMSX-Eigenschaften, die im `MQMD`-Header nicht vollständig ausgedrückt werden können. Der Ordner `jms` ist immer in einem JMS-`MQRFH2`-Header enthalten.

Ordner `usr`

`usr` enthält anwendungsdefinierte JMS-Eigenschaften, die der Nachricht zugeordnet sind. Der Ordner `usr` ist nur vorhanden, wenn von einer Anwendung eine anwendungsdefinierte Eigenschaft gesetzt wurde.

Ordner `mqext`

`mqext` enthält folgende 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 17. 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 im 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 18. 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 im Ordner mqps hinzu.

In Tabelle 19 auf Seite 141 ist eine vollständige Liste der Eigenschaftsnamen aufgeführt.

Tabelle 19. Von JMS verwendete MQRFH2-Ordner und Eigenschaften				
Name des JMS-Felds	**Java-Typ**	**MQRFH2-Ordnername**	**Eigenschaftsname**	**Typ/Werte**
JMSDestination	Ziel	jms	Dst	Zeichenfolge

Tabelle 19. Von JMS verwendete MQRFH2-Ordner und Eigenschaften (Forts.)				
Name des JMS-Felds	**Java-Typ**	**MQRFH2-Ordnername**	**Eigenschaftsname**	**Typ/Werte**
JMSExpiration	long	jms	Erw	i8
JMSPriority	Int	jms	Pri	i4
JMSDeliveryMode	Int	jms	Dlv	i4
JMSCorrelationID	Zeichenfolge	jms	Cid	Zeichenfolge
JMSReplyTo	Ziel	jms	Rto	Zeichenfolge
JMSTimestamp	long	jms	Tms	i8
JMSType	Zeichenfolge	mcd	Type, Set, Fmt	Zeichenfolge
JMSXGroupID	Zeichenfolge	jms	Gid	Zeichenfolge
JMSXGroupSeq	Int	jms	Reihenfolge	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 Endtag </foldername> können Leerzeichen als Füllzeichen folgen. Jedes Triplet wird mithilfe einer XML-ähnlichen Syntax wie folgt codiert:

```
<name dt='datatype'>value</name>
```

Das dt='datatype'-Element ist optional. Es wird bei vielen Eigenschaften weggelassen, da der Datentyp vordefiniert ist. Wenn es enthalten ist, muss mindestens ein Leerzeichen vor dem dt=-Tag enthalten sein.

name
Der Name der Eigenschaft; siehe Tabelle 19 auf Seite 141.

datatype
Muss nach dem Umsetzen einem der in Tabelle 20 auf Seite 143 aufgeführten Datentypen entsprechen.

value
Eine Zeichenfolgedarstellung des zu übertragenden Werts, wobei die Definitionen in Tabelle 20 auf Seite 143 verwendet werden.

Ein Nullwert wird mit folgender Syntax codiert:

```
<name dt='datatype' xsi:nil='true'></name>
```

Verwenden Sie nicht `xsi:nil='false'`.

Tabelle 20. Eigenschaftsdatentypen

Datentyp	**Definition**
Zeichenfolge	Jede Zeichenfolge mit Ausnahme von < und &
boolean	Das Zeichen `0` oder 1 (`0` = falsch, 1 = wahr)
bin.hex	Durch Hexadezimalziffern dargestellte Oktette
i1	Eine durch die Ziffern `0..9` ausgedrückte Zahl mit optionalem Vorzeichen (keine Bruchzahl und kein Exponent). Muss im Bereich -128 bis einschließlich 127 liegen.
i2	Eine durch die Ziffern `0..9` ausgedrückte Zahl mit optionalem Vorzeichen (keine Bruchzahl und kein Exponent). Muss im Bereich -32768 bis einschließlich 32767 liegen.
i4	Eine durch die Ziffern `0..9` ausgedrückte Zahl mit optionalem Vorzeichen (keine Bruchzahl und kein Exponent). Muss im Bereich -2147483648 bis einschließlich 2147483647 liegen.
i8	Eine durch die Ziffern `0..9` ausgedrückte Zahl mit optionalem Vorzeichen (keine Bruchzahl und kein Exponent). Muss im Bereich -9223372036854775808 bis einschließlich 9223372036854775807 liegen.
Int	Eine durch die Ziffern `0..9` ausgedrückte Zahl mit optionalem Vorzeichen (keine Bruchzahl und kein Exponent). 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 >
- `'` für das Zeichen '
- `"` 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 21 auf Seite 144 sind die JMS-Headerfelder aufgelistet, während in Tabelle 22 auf Seite 144 die JMS-Eigenschaften aufgeführt sind, die den MQMD-Feldern direkt zugeordnet werden. Tabelle 23 auf

Seite 144 enthält eine Liste der providerspezifischen Eigenschaften und der MQMD-Felder, denen sie zugeordnet werden.

Tabelle 21. Zuordnung der JMS-Headerfelder zu MQMD-Feldern

JMS-Headerfeld	**Java-Typ**	**MQMD-Feld**	**C-Typ**
JMSDeliveryMode	int	Permanenz	MQLONG
JMSExpiration	long	Verfall	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	Zeichenfolge	MsgID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Zeichenfolge	CorrelId	MQBYTE24

Tabelle 22. Zuordnung der JMS-Eigenschaften zu MQMD-Feldern

JMS-Eigenschaft	**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 23. 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
JMS_IBM_Report_Pass_Msg_ID	int	Bericht	MQLONG

Tabelle 23. Zuordnung der JMS-Provider-spezifischen Eigenschaften zu MQMD-Feldern (Forts.)			
JMS-Provider-spezifische Eigenschaft	**Java-Typ**	**MQMD-Feld**	**C-Typ**
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 145	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	Zeichenfolge	CodedCharacterSetId „2" auf Seite 145	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 24 auf Seite 146 zeigt, wie die JMS-Headerfelder beim Senden (send()) oder Veröffentlichen (publish()) zu MQMD-/MQRFH2-Feldern zugeordnet werden. Tabelle 25 auf Seite 146 zeigt, wie JMS-Eigenschaften beim Senden (send()) oder Veröffentlichen (publish()) zu MQMD-/MQRFH2-Feldern zugeordnet werden. Tabelle 26 auf Seite 146 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 24. Zuordnung der Felder abgehender Nachrichten

Name des JMS-Headerfelds	MQMD-Feld für Übertragung	Kopfzeile	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 25. Zuordnung der JMS-Eigenschaften abgehender Nachrichten

JMS-Eigenschaftsname	MQMD-Feld für Übertragung	Kopfzeile	Festgelegt durch
JMSXUserID	UserIdentifier		Sendemethode
JMSXAppID	PutApplName		Sendemethode
JMSXDeliveryCount			Nur Empfang
JMSXGroupID	GroupId	MQRFH2	Nachrichtenobjekt
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Nachrichtenobjekt

Tabelle 26. Zuordnung der JMS-Provider-spezifischen Eigenschaften abgehender Nachrichten

Name der JMS-Provider-spezifischen Eigenschaft	MQMD-Feld für Übertragung	Kopfzeile	Festgelegt durch
JMS_IBM_Report_Exception	Bericht		Nachrichtenobjekt

Tabelle 26. Zuordnung der JMS-Provider-spezifischen Eigenschaften abgehender Nachrichten (Forts.)			
Name der JMS-Provider-spezifischen Eigenschaft	**MQMD-Feld für Übertragung**	**Kopfzeile**	**Festgelegt durch**
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 210 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 210 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

IBM MQ weist allen Nachrichten, die von JMS gesendet werden, eindeutige Nachrichten-IDs zu. 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, die MQMD.MessageID anzugeben, können Sie dies mit einer der im Abschnitt „Nachrichtendeskriptor über eine Anwendung der IBM MQ classes for JMS lesen und schreiben” auf Seite 229 beschriebenen Erweiterungen von IBM MQ JMS 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 Binär-

daten konvertiert und anschließend im Feld 'MQMD.CorrelId' festgelegt. Im MQRFH2 wird kein CorrelId-Wert codiert.

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 Anwendung muss den Wert dieser Eigenschaften auf die MQRO_-Standardkonstanten von IBM MQ setzen (diese sind in com.ibm.mq.MQC enthalten). Um also beispielsweise eine Zustellungsbestätigung (Confirmation of Delivery, COD) mit ausführlichen Daten anzufordern, muss die Anwendung JMS_IBM_Report_COD auf den Wert CMQC.MQRO_COD_WITH_FULL_DATA setzen.

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

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 27 auf Seite 151 zeigt, wie JMS-Headerfelder beim Abrufen (get()) oder Empfangen (receive()) zu MQMD/MQRFH2-Feldern zugeordnet werden. Tabelle 28 auf Seite 151 zeigt, wie JMS-Eigenschaftsfelder beim Abrufen (get()) oder Empfangen (receive()) zu MQMD/MQRFH2-Feldern zugeordnet werden. In Tabelle 29 auf Seite 152 ist die Zuordnung von JMS-Provider-spezifischen Eigenschaften dargestellt.

Tabelle 27. 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 151
JMSDeliveryMode	Persistence „2" auf Seite 151	jms.Dlv „2" auf Seite 151
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate „2" auf Seite 151 PutTime „2" auf Seite 151	jms.Tms „2" auf Seite 151
JMSCorrelationID	CorrelId „2" auf Seite 151	jms.Cid „2" auf Seite 151
JMSReplyTo	ReplyToQ „2" auf Seite 151 ReplyToQMgr „2" auf Seite 151	jms.Rto „2" auf Seite 151
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 28. Eigenschaftszuordnung bei eingehenden Nachrichten

JMS-Eigenschaftsname	**MQMD-Feld für Abruf**	**MQRFH2-Feld für Abruf**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	

Tabelle 28. Eigenschaftszuordnung bei eingehenden Nachrichten (Forts.)		
JMS-Eigenschaftsname	**MQMD-Feld für Abruf**	**MQRFH2-Feld für Abruf**
JMSXGroupID	GroupId „1" auf Seite 152	jms.Gid „1" auf Seite 152
JMSXGroupSeq	MsgSeqNumber „1" auf Seite 152	jms.Seq „1" auf Seite 152

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 29. 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 152	Encoding	
JMS_IBM_Character_Set „1" auf Seite 152	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 153 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.

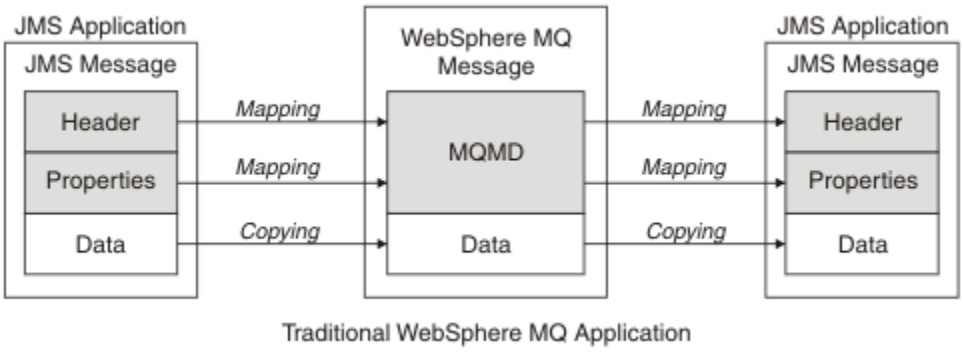


Abbildung 11. JMS-Nachrichtentransformation in IBM MQ-Nachrichten ohne MQRFH2-Header

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 20 auf Seite 143 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 20 auf Seite 143 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 162). 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 der Nachrichtentyp kein JMS-Nachrichtentyp ist, sondern `MQ` lautet, 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 156
2. „Anwendungsdatenkonvertierung” auf Seite 157
3. „Warteschlangenmanager-Datenkonvertierung” auf Seite 158
4. „Nachrichtenkanal-Datenkonvertierung” auf Seite 159

Wo sollte die Konvertierung stattfinden?
Im Abschnitt „Methode für die Nachrichtenkonvertierung wählen: receiver makes good” auf Seite 159 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 30 auf Seite 160 aufgelistet.

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 eben-

[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.

falls 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 169.

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 162 und Abbildung 15 auf Seite 162 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 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.

[2] Ausnahme: Daten, die mit writeUTF geschrieben werden, beginnen mit einem aus 2 Bytes bestehenden Längenfeld.

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`*`);`.

Weitere Informationen hierzu finden Sie unter „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 178.

Warteschlangenmanager-Datenkonvertierung

In IBM MQ 7.0kann die Codepagekonvertierung 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 161. 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 158.

Vor 7.0wurden Konvertierungen immer vom JMS -Client durchgeführt. Die JMS-Client-Datenkonvertierung ist auf die Konvertierung von Zahlen- und Textfolgen beschränkt, deren Typ und Länge dem JMS-Client bekannt sind. Datenstrukturen können nicht konvertiert werden; siehe „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 178. Wenn die Konvertierung in 7.0bis Fixpack 7.0.1.5vom WS-Manager durchgeführt werden kann, wird sie immer vom Warteschlangenmanager ausgeführt. Ab 7.0.1.5 wird das Standardkonvertierungsverhalten auf 6.0zurückgesetzt und alle Konvertierungen werden vom JMS -Client ausgeführt. Ab 7.0.1.5 bzw. 7.0.1.4 mit APAR IC72897 können Sie eine neue Zieloption (`WMQ_RECEIVE_CONVERSION`) festlegen, mit der Sie steuern, wo die Konvertierung durchgeführt wird. Mit der Option `WMQ_RECEIVE_CCSID` können Sie die Zielcodepage festlegen; siehe Abbildung 12 auf Seite 158.

```
((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 JMS-fremden Empfänger bezieht sich das Beispiel auf IBM MQ-Versionen vor und

nach V7.0. Bei IBM MQ 7.0kann der Konvertierungsexit auch für ein empfangendes JMS -Programm aufgerufen werden.

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 178.

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 UNIX. `windows-1252` ist ein Einzelbytezeichensatz und `UTF-8` ist ein Mehrbytezeichensatz.

Normalerweise übernehmen Sie beim Austausch von Nachrichten mit anderen JMS-Anwendungen die Standardwerte `UTF-8` und `Native` der Zieleigenschaften für den Zeichensatz und die Codierung. 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.

Weitere Informationen hierzu finden Sie unter „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 178. Dort sind Beispiele für die Konvertierung von Nachrichten enthalten, die mit JMS-fremden Anwendungen ausgetauscht werden.

Beispiele

Tabelle mit Nachrichtentypen und Konvertierungstypen

Tabelle 30. 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 30. 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. Codeausschnitt aus amqsget0.c

Text in einer JMSBytesMessage senden und empfangen

Der Code in Abbildung 14 auf Seite 162 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 162). 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. Seit 7.0verwenden JMS -Clients, die Nachrichten empfangen, auch die Datenkonvertierung des Warteschlangenmanagers. Ab 7.0.1.5 bzw. 7.0.1.4 mit APAR IC72897 ist die Warteschlangenmanager-Datenkonvertierung optional.

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 165 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 MQ in 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 166.

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 166.

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 folgender Ausnahmebedingung: ClassCastException: com.ibm.jms.JMSBytesMessage kann nicht in javax.jms.TextMessage umgesetzt werden.

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 168. Details hierzu finden Sie unter „Nachrichtenkonvertierung und -codierung beim JMS-Client" auf Seite 169. Jedes primitive Element ist mit Tags versehen; siehe „JMS-Nachrichtenhauptteil" auf Seite 153.

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 166 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 166 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 168. Details hierzu finden Sie unter „JMS-Nachrichtentypen und Konvertierung" auf Seite 162.

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 166.

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.

Die Art, wie eine Nachricht empfangen wird, wird nicht durch die Einstellung von WMQ_TARGET_DEST beeinflusst. Dabei spielt es keine Rolle, ob eine Nachricht einen Hauptteil im MQ-Stil oder im JMS-Stil enthält.

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. Für die Verwendung des Pakets com.ibm.mq.header mit JMS ist ein Code wie im Beispiel des Abschnitts „Nachrichten mithilfe von DataInputStream und DataOutputStream lesen und schreiben” auf Seite 167 erforderlich.

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 JMSStreamMessage und JMSMapMessage senden

Text in einer JMSBytesMessage senden und empfangen

Der Code in Abbildung 21 auf Seite 167 sendet eine Zeichenfolge in einer BytesMessage. Der Einfachheit halber wird in diesem Beispiel eine einzelne Zeichenfolge gesendet, für die eine JMSTextMessage geeig-

neter 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 167). 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 167 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` mittels `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 168 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 mittels 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 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
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar

Tabelle 31. 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. Seit 7.0verwenden JMS -Clients, die Nachrichten empfangen, auch die Datenkonvertierung des Warteschlangenmanagers. Ab 7.0.1.5 bzw. 7.0.1.4 mit APAR IC72897 ist die Warteschlangenmanager-Datenkonvertierung optional.

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 32 auf Seite 170 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 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
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 173 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 173 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 173 mit dem falschen Codeausschnitt in Abbildung 25 auf Seite 171. 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 174 zeigt eine Anwendung, die eine Zahl im Zielformat codiert und diese in einer JMSStreamMessage sendet. Vergleichen Sie das Beispiel in Abbildung 30 auf Seite 174 mit dem Beispiel in Abbildung 31 auf Seite 174. 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 174 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 173 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 UNIX UTF-8 verwendet wird. Der Austausch zwischen Windows und UNIX erfordert, dass Sie eine explizite Codepage für den Austausch von Text als Bytes 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 174 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 174). 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. Seit 7.0verwenden JMS -Clients, die Nachrichten empfangen, auch die Datenkonvertierung des Warteschlangenmanagers. Ab 7.0.1.5 bzw. 7.0.1.4 mit APAR IC72897 ist die Warteschlangenmanager-Datenkonvertierung optional.

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. Seit 7.0verwenden JMS -Clients, die Nachrichten empfangen, auch die Datenkonvertierung des Warteschlangenmanagers. Ab 7.0.1.5 bzw. 7.0.1.4 mit APAR IC72897 ist die Warteschlangenmanager-Datenkonvertierung optional.

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. Die Konvertierungsfunktion des Warteschlangenmanagers war erst ab 7.0für eine JMS -Anwendung verfügbar, die eine Nachricht empfängt.

Sie können die Warteschlangenmanager-Konvertierung vor 7.0mit einer JMS -Clientanwendung verwenden, die eine Nachricht sendet. Der JMS-Client erstellt einen formatierten Datensatz und legt die Attribute `CodedCharacterSetId`, `Encoding` und `Format` in Übereinstimmung mit den in der Nachricht enthaltenen Daten fest. Eine empfangende JMS-fremde Anwendung liest die Nachricht unter Verwendung von `GMO_CONVERT` und veranlasst den Aufruf eines benutzerdefinierten Datenkonvertierungsexits. Der Datenkonvertierungsexit ist eine gemeinsam genutzte Bibliothek, die den Namen hat, der im Feld `Format` festgelegt ist.

Seit 7.0kann der Warteschlangenmanager Nachrichten konvertieren, die an JMS -Clients gesendet werden. In den Versionen 7.0.0.0 bis einschließlich 7.0.1.4 wird die Warteschlangenmanager-Konvertierung immer für JMS- Clients aufgerufen. Ab 7.0.1.5 bzw. ab 7.0.1.4 mit Anwendung von APAR IC72897 wird die Warteschlangenmanager-Konvertierung gesteuert, indem die Zieleigenschaft `WMQ_RECEIVE_CONVERSION` auf `WMQ_RECEIVE_CONVERSION_QMGR` oder `WMQ_RECEIVE_CONVERSION_CLIENT_MSG` gesetzt wird. `WMQ_RECEIVE_CONVERSION_CLIENT_MSG` ist die Standardeinstellung, die dem Verhalten von IBM WebSphere MQ 6.0entspricht, das die Warteschlangenmanager-Datenkonvertierung für JMS -Clients nicht unterstützt hat. 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 177.

```
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. Codeausschnitt 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 1034
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 137.

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 schreiben” auf Seite 181 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 179 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 181 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 181.
   - 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 181 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. Lesen Sie die empfohlenen Änderungen für die Verwendung des Beispielheaders `RECORD.h` in Abbildung 38 auf Seite 180. 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 181.
   - 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. In:

```
/* 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. Seit 7.0verwenden JMS -Clients, die Nachrichten empfangen, auch die Datenkonvertierung des Warteschlangenmanagers. Ab 7.0.1.5 bzw. 7.0.1.4 mit APAR IC72897 ist die Warteschlangenmanager-Datenkonvertierung optional.

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.

Zugehörige Informationen

Dienstprogramm zum Einrichten eines Konvertierungsexitcodes

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 189 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 7.0ab 7.0.1.5muss die Warteschlangenmanager-Konvertierung aktiviert werden. Standardmäßig ist sie inaktiviert. In 7.0ist die Warteschlangenmanager-Konvertierung bis 7.0.1.4 standardmäßig aktiviert und diese Codezeile verursacht einen 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 = -16166172327505617l2L;
   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 MQ mit `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 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;
```

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 javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Ergebnisse

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 180 in „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 178.
  - 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 Abbildung 39 auf Seite 189.

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 mit dem Namen `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 Abbildungen Abbildung 39 auf Seite 189 bis Abbildung 44 auf Seite 193 aufgelisteten Klassen sind auch in einer ZIP-Datei verfügbar; laden Sie die Datei jm25529_.zip oder jm25529_.tar.gz herunter.

```
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());
    }
}
```

Abbildung 39. TryMyRecord

```
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; }
}
```

Abbildung 40. RECORD

```
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; }
}
```

Abbildung 41. MyRecord

```
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); }
    }
```

Abbildung 42. EndPoint

```
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); }
}
```

Abbildung 43. MyProducer

```
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();  }
}
```

Abbildung 44. MyConsumer

Verbindungsfactorys und Ziele in einer Anwendung der IBM MQ classes for JMS erstellen und konfigurieren

Eine Anwendung der IBM MQ classes for JMS kann Verbindungsfactorys und Ziele erstellen, indem sie diese als verwaltete Objekte aus einem JNDI-Namensbereich (JNDI = Java Naming and Directory Interface) abruft. Dabei verwendet sie die IBM JMS-Erweiterungen oder die IBM MQ-JMS-Erweiterungen. 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 Logikablauf einer JMS-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 wie im Abschnitt Objekte mit dem JMS -Verwaltungstool konfigurierenbeschrieben oder IBM MQ Explorer wie im Abschnitt JMS -Objekte mit IBM MQ Explorerkonfigurierenbeschrieben verwenden, um Verbindungsfactorys und Ziele als verwaltete Objekte in einem JNDI-Namensbereich zu erstellen und 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.

IBM JMS-Erweiterungen verwenden
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.

IBM MQ JMS-Erweiterungen verwenden
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 Informationen
JMS-Ressourcen konfigurieren

JNDI für das Abrufen von verwalteten Objekten in einer JMS-Anwendung verwenden
Für den Abruf von verwalteten Objekten aus einem JNDI-Namensbereich (JNDI = Java Naming and Directory Interface) muss eine JMS-Anwendung einen Ausgangskontext erstellen und dann die Objekte mit der Methode lookup() abrufen.

Bevor eine Anwendung verwaltete Objekte aus einem JNDI-Namensbereich abrufen kann, muss ein Administrator diese verwalteten Objekte zuerst erstellen. Der Administrator kann das IBM MQ-JMS-Verwaltungstool oder IBM MQ Explorer zum Erstellen und Verwalten von verwalteten Objekten in einem JNDI-Namensbereich verwenden. Weitere Informationen finden Sie im Abschnitt Verbindungsfactorys und Ziele in einem JNDI-Namensbereich konfigurieren.

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:

```
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 Informationen

JMS-Objekte mit IBM MQ Explorer konfigurieren

JMS-Objekte mit dem Verwaltungstool konfigurieren

JMS-Ressourcen in WebSphere Application Server konfigurieren

IBM JMS-Erweiterungen verwenden

IBM MQ classes for JMS enthalten eine Gruppe von Erweiterungen für die JMS-API. Diese werden als 'IBM JMS-Erweiterungen' bezeichnet. Eine Anwendung kann mithilfe dieser Erweiterungen dynamisch zur Laufzeit Verbindungsfactorys und Ziele erstellen sowie die Eigenschaften der Objekte der IBM MQ classes for JMS festlegen. 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

Die Pakete befinden sich im Verzeichnis `com.ibm.mqjms.jar` im Verzeichnis *`MQ_INSTALLATION_PATH`*`/java/lib`.

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 für die Festlegung der Eigenschaften von Objekten der IBM MQ classes for JMS

- 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 für den Abruf von Versionsinformationen zu IBM MQ classes for JMS

Für die dynamische Erstellung von Verbindungsfactorys und Zielen zur Laufzeit sowie für die Festlegung und den Abruf ihrer Eigenschaften stellen die IBM JMS-Erweiterungen eine alternative Gruppe von Schnittstellen für die 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 enthält die Definitionen von Konstanten, die eine Anwendung beim Festlegen der Eigenschaften von Objekten der IBM MQ classes for JMS unter Verwendung der IBM JMS-Erweiterungen nutzen kann. Die Schnittstelle enthält Konstanten für den IBM MQ-Messaging-Provider und JMS-Konstanten, die unabhängig von einem bestimmten Messaging-Provider eingesetzt werden können.

Bei den nachfolgenden Codebeispielen wird vorausgesetzt, dass die folgenden Importanweisungen ausgeführt wurden:

```
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:

```
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 197.

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 210. 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 197 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 210. 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 197 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.

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.

Die Instanzen der Klassen im Paket com.ibm.mq.jms übernehmen auch die Methoden der JmsProperty-Context-Schnittstelle. Daher kann eine Anwendung diese Methoden zum Festlegen der Eigenschaften von MQConnectionFactory-, MQQueue- und MQTopic-Objekten verwenden.

Wenn eine Anwendung ein Objekt der IBM MQ classes for JMS erstellt, werden alle Eigenschaften automatisch mit Standardwerten 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 für diese Eigenschaft nicht gültig ist, lösen die IBM MQ classes for JMS die Ausnahmebedingung 'JMSException' 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 lösen die Ausnahmebedingung 'NumberFormatException' für primitive Datentypen aus und geben für referenzierte Datentypen 'null' zurück.

Neben den vordefinierten Eigenschaften eines Objekts der IBM MQ classes for JMS kann eine Anwendung eigene Eigenschaften festlegen. Diese anwendungsdefinierten Eigenschaften werden von IBM MQ classes for JMS ignoriert.

Weitere Informationen zu den Eigenschaften von Objekten der IBM MQ classes for JMS finden Sie im Abschnitt Eigenschaften von Objekten der IBM MQ classes for JMS.

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 JmsPropertyContext-Schnittstelle verwendet, um die Eigenschaft eines Objekts der IBM MQ classes for JMS festzulegen oder abzurufen, kann der Wert der Eigenschaft implizit aus einem Datentyp in einen anderen Datentyp 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 unterstützten Konvertierungen sind in Tabelle 33 auf Seite 199 dargestellt.

Tabelle 33. 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
Int	long, String
long	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 aufweist, lösen die IBM MQ classes for JMS die Ausnahmebedingung 'NumberFormatException' aus.
- Wenn eine Anwendung eine nicht unterstützte Konvertierung versucht, lösen die IBM MQ classes for JMS die Ausnahmebedingung 'MessageFormatException' aus.

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.

  Vorzeichen
  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 aufweist, lösen die IBM MQ classes for JMS die Ausnahmebedingung 'NumberFormatException' aus.
- 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.

  Vorzeichen
  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 aufweist, lösen die IBM MQ classes for JMS die Ausnahmebedingung 'NumberFormatException' aus.
- 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 197 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 JmsPropertyContext-Schnittstelle verwendet um die Eigenschaften von Objekten der IBM MQ classes for JMS festzulegen und abzurufen, kann die Anwendung die Namen und Werte von Eigenschaften auf eine beliebige 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 Eigenschaftsnamen und -werte als Literalwerte, die in der Schnittstelle com.ibm.msg.client.wmq.WMQConstants definiert sind. Dies ist das definierte kanonische Format für Eigenschaftsnamen und -werte. Als allgemeine Regel gilt Folgendes: Wenn eine Anwendung Eigenschaften mithilfe einer der beiden anderen Verfahren für die Angabe von Eigenschaftsnamen und -werten festlegt, müssen IBM MQ classes for JMS die Namen und Werte aus dem angegebenen Eingabeformat in das kanonische Format konvertieren. Ähnlich gilt, dass IBM MQ classes for JMS die Namen aus dem angegebenen Eingabeformat in das kanonische Format und die Werte aus dem kanonischen Format in das erforderliche Ausgabeformat konvertieren müssen, wenn eine Anwendung Eigenschaften mithilfe einer der beiden anderen Verfahren für die Angabe von Eigenschaftsnamen und -werten abruft. Wenn diese Konvertierungen durchgeführt werden müssen, kann sich das auf die Leistung auswirken.

Eigenschaftsnamen und -werte, die von Ausnahmebedingungen, in Tracedateien oder im Protokoll der IBM MQ classes for JMS zurückgegeben werden, haben immer das kanonische Format.

Map-Schnittstelle verwenden

Die JmsPropertyContext-Schnittstelle ist eine Erweiterung der java.util.Map-Schnittstelle. Daher kann eine Anwendung die Methoden der Map-Schnittstelle verwenden, um auf die Eigenschaften eines Objekts der IBM MQ classes for JMS 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 203.

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.

Für eine Echtzeitverbindung mit einem Broker kann eine Anwendung den folgenden Code verwenden:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_DIRECT);
factory.setHostName("HOST2");
factory.setPort(1507);
```

Dieser Code setzt voraus, dass der Broker auf einem System mit dem Hostnamen HOST2 ausgeführt wird und an der Portnummer 1507 empfangsbereit ist.

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 210. Eine alternative Methode für das Festlegen der Eigenschaften eines MQQueue-Objekts ist im Abschnitt „Eigenschaften von Zielen festlegen" auf Seite 205 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 210. Eine alternative Methode für das Festlegen der Eigenschaften eines MQTopic-Objekts ist im Abschnitt „Eigenschaften von Zielen festlegen" auf Seite 205 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.

Verbindung in einer JMS-Anwendung erstellen

Um eine Verbindung zu erstellen, verwendet eine JMS-Anwendung ein ConnectionFactory-Objekt zur Erstellung eines Connection-Objekts. Anschließend startet sie 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();
```

Sitzung in einer JMS-Anwendung erstellen

Eine JMS-Anwendung verwendet für die Erstellung einer Sitzung 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.

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 210.

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 210.

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, erstellen IBM MQ classes for JMS ein Thema mit einem Namen, der mit den Zeichen TEMP/*tempTopicPrefix* beginnt. Dabei ist *tempTopicPrefix* der Wert der Eigenschaft TEMPTOPICPREFIX der Verbindungsfactory.

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:

Name_des_Warteschlangenmanagers
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.

Warteschlangenname
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 der Platzhalterzeichen +, #, * und ? in Themennamen. Themennamen, die diese Zeichen enthalten, können zu unerwarteten Ergebnissen führen, wenn Sie diese subskribieren. Siehe Themenzeichenfolgen verwenden.

***propertyName1*, *propertyName2*, ...**
Die Namen der Eigenschaften des Queue- oder Topic-Objekts, das durch die Anwendung erstellt wird. In Tabelle 34 auf Seite 212 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 34 auf Seite 212 1 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 34 auf Seite 212 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 34. 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 238.

Tabelle 34. 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 für das Abrufen von verwalteten Objekten in einer JMS-Anwendung verwenden” auf Seite 194
- „IBM JMS-Erweiterungen verwenden” auf Seite 195
- „Verwendung der IBM MQ JMS-Erweiterungen” auf Seite 203

- „Ziele in einer JMS-Anwendung erstellen” auf Seite 209

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 133.

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 unter „Nachrichten asynchron in IBM MQ classes for JMS einreihen” auf Seite 312.

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 für das Abrufen von verwalteten Objekten in einer JMS-Anwendung verwenden” auf Seite 194
- „IBM JMS-Erweiterungen verwenden” auf Seite 195
- „Verwendung der IBM MQ JMS-Erweiterungen” auf Seite 203
- „Ziele in einer JMS-Anwendung erstellen” auf Seite 209

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 133.

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 IBM WebSphere MQ 7 -Warteschlangenmanager 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:

```
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: A synchronous method call is not permitted when a session is being used asynchronous-
ly:'method name'
```

(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 221.

V 9.0.0.2 V 9.0.2 *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.

Für Continuous Delivery gilt ab IBM MQ 9.0.2, dass beim Lesen einer Nachricht aus einer Warteschlange, die einen RFH2-Header mit dem Ordner MQPS enthält, der Wert, der dem Schlüssel Sud (sofern vorhanden) zugeordnet ist, als Zeichenfolgeeigenschaft zu dem JMS-Nachrichtenobjekt hinzugefügt wird, das an die IBM MQ classes for JMS-Anwendung zurückgegeben wird. Um den Abruf dieser Eigenschaft aus der Nachricht zu ermöglichen, kann die Konstante JMS_IBM_SUBSCRIPTION_USER_DATA in der Schnittstelle JmsConstants zusammen mit der Methode javax.jms.Message.getStringProperty(java.lang.String) verwendet werden, um die Subskriptionsbenutzerdaten abzurufen.

Ab IBM MQ 9.0.0 Fix Pack 2 ist die Konstante JMS_IBM_SUBSCRIPTION_USER_DATA auch in Long Term Support verfügbar.

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:

```
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 138

Zugehörige Informationen

Verwaltungssubskription definieren

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. Es kann sich beispielsweise um eine Nachricht eines unerwarteten Typs handeln oder sie enthält Informationen, die von der Logik der Anwendung 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 verschiebt IBM MQ classes for JMS die Nachricht anschließend in die mit dem Attribut **BOQNAME** angegebene Warteschlange. Wenn die Nachricht nicht in die Rücksetzwarteschlange eingereiht werden kann, wird sie entweder in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers verschoben oder verworfen, je nach den Berichtsoptionen der Nachricht.

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:

- Bei Verwendung der IBM MQ classes for JMS für IBM MQ 9.0.0 Fix Pack 5 und früher gilt: wenn die lokale Warteschlange oder die Zielwarteschlange für eine Aliaswarteschlange eine Clusterwarteschlange ist, wird sie von den IBM MQ classes for JMS mit den Optionen `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` und `MQOO_FAIL_IF_QUIESCING` geöffnet. 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.
- V 9.0.0.6 Bei Verwendung der IBM MQ classes for JMS für IBM MQ 9.0.0 Fix Pack 6 und höher muss der Benutzer, der die empfangende Anwendung ausführt, unabhängig vom Typ der Warteschlange über Abfragezugriff auf die lokale Warteschlange verfügen.

Damit nicht verarbeitbare Nachrichten entweder in eine Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten oder in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers verschoben werden können, 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.
- Ist für die Nachricht die Berichtsoption `MQRO_DEAD_LETTER_Q` angegeben, 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.
- Ist für die Nachricht die Berichtsoption `MQRO_DEAD_LETTER_Q` angegeben, 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 326
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 in der Lage sein, Ausnahmebedingungen zu 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. JMSException ist die Stammklasse für Ausnahmebedingungen, die durch JMS-Methoden ausgelöst werden. Das Abfangen von JMSException-Ausnahmebedingungen ist eine allgemeine Methode für die Handhabung aller Ausnahmebedingungen, die sich auf JMS beziehen.

Jede JMSException-Ausnahmebedingung beinhaltet die folgenden Informationen:

- Eine providerpezifische Ausnahmebedingungsnachricht, die eine Anwendung erhält, wenn sie die Methode Throwable.getMessage() aufruft.

- Einen Provider-spezifischen Fehlercode, den eine Anwendung erhält, wenn sie die Methode JMSException.getErrorCode() aufruft.
- Eine verlinkte Ausnahmebedingung. Eine von einem JMS-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. Eine Anwendung ruft eine verlinkte Ausnahmebedingung durch Aufruf der Methode JMSException.getLinkedException() oder Throwable.getCause() ab.

Bei den meisten der von IBM MQ classes for JMS ausgelösten Ausnahmebedingungen handelt es sich um Instanzen der Unterklassen von JMSException. Diese Unterklassen implementieren die Schnittstelle com.ibm.msg.client.jms.JmsExceptionDetail, die die folgenden weiteren Informationen bereitstellt:

- Eine Erläuterung der Ausnahmebedingungsnachricht, die eine Anwendung nach Aufruf der Methode JmsExceptionDetail.getExplanation() erhält.
- Eine empfohlene Benutzeraktion auf die Ausnahmebedingung, die eine Anwendung nach Aufruf der Methode JmsExceptionDetail.getUserAction() erhält.
- Die Schlüssel für die Einfügungen in der Ausnahmebedingungsnachricht. Eine Anwendung erhält einen Iterator für alle Schlüssel durch Aufruf der Methode JmsExceptionDetail.getKeys().
- Die Nachrichteneinfügungen in der Ausnahmebedingungsnachricht. Eine Nachrichteneinfügung kann zum Beispiel aus dem Namen der Warteschlange bestehen, die die Ausnahmebedingung verursacht hat, und nützlich für die Anwendung sein, um Zugang zu diesem Namen zu bekommen. Eine Anwendung erhält die einem bestimmten Schlüssel entsprechende Nachrichteneinfügung durch Aufruf der Methode JmsExceptionDetail.getValue().

Alle Methoden der Schnittstelle JmsExceptionDetail können 'null' melden, wenn keine Einzelangaben 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 von javax.jms.InvalidDestinationException und implementiert die Schnittstelle com.ibm.msg.client.jms.JmsExceptionDetail.

Verlinkte Ausnahmebedingungen

Eine verlinkte Ausnahmebedingung liefert weitere Informationen zu einem Laufzeitproblem. Daher sollte eine Anwendung bei jeder ausgelösten JMSException-Ausnahmebedingung die verlinkte Ausnahmebedingung überprüfen. 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(). Bei einer JMSException-Ausnahmebedingung nimmt die Methode getLinkedException() eine Delegierung an die Methode Throwable.getCause() vor.

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
|
+- -->
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 Ausnahmebedingung, eine Instanz einer Unterklasse von JMSException, wird von der allgemeinen Schicht in IBM MQ classes for JMS ausgelöst.
- Die nächste Ausnahmebedingung, eine Instanz von com.ibm.mq.MQException, wird durch den IBM MQ-Messaging-Provider ausgelöst.
- Die nächste Ausnahmebedingung, eine Instanz von com.ibm.mq.jmqi.JmqiException, wird durch die allgemeine Java-Schnittstelle zur MQI ausgelöst.
- Die letzte Ausnahmebedingung, eine Instanz von java.net.ConnectionException, wird durch die Java-Klassenbibliothek ausgelöst.

Weitere Informationen zu der Schichtarchitektur von IBM MQ classes for JMS finden Sie im Abschnitt IBM MQ-Klassen für JMS-Architektur.

Bei Verwendung von Code wie dem folgenden kann eine Anwendung diese Kette durchlaufen, um alle entsprechenden Informationen zu extrahieren:

```
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();
}
}
```

Wir weisen darauf hin, dass eine Anwendung immer die Art der Ausnahmebedingung in einer Kette überprüfen sollte, da diese variieren kann und verschiedene Arten der Ausnahmebedingung verschiedene Informationen einbinden können.

IBM MQ-spezifische Informationen zu einem Problem abrufen

Instanzen von com.ibm.mq.MQException und com.ibm.mq.jmqi.JmqiException binden IBM MQ-spezifische Informationen zu einem Problem ein.

Eine MQException-Ausnahmebedingung bindet die folgenden Informationen ein:

- Einen Beendigungscode, den eine Anwendung erhält, wenn sie die Methode getCompCode() aufruft
- Einen Ursachencode, den eine Anwendung erhält, wenn sie die Methode getReason() aufruft

Eine JmqiException-Ausnahmebedingung bindet ebenfalls einen Beendigungscode und einen Ursachencode ein. Zusätzlich bindet eine JmqiException-Ausnahmebedingung jedoch die Informationen in eine Nachricht vom Typ AMQ *nnnn* oder CSQ *nnnn* ein, falls eine solche Nachricht der Ausnahmebedingung zugeordnet ist. Durch Aufrufen der entsprechenden Ausnahmebedingungsmethoden kann eine Anwendung die verschiedenen Bestandteile der Nachricht erhalten, wie Wertigkeit, Erläuterung und Benutzeraktion.

Beispiele zur Verwendung der in diesem Abschnitt erwähnten Methoden finden Sie im Beispielcode „Verlinkte Ausnahmebedingungen” auf Seite 225.

Upgrade von Vorgängerversionen von IBM MQ classes for JMS durchführen

Im Vergleich zu früheren Versionen von IBM MQ classes for JMSwurden die meisten Fehlercodes und Ausnahmebedingungsnachrichten in IBM WebSphere MQ 7.0geändert. Der Grund für diese Änderungen ist, dass IBM MQ classes for JMS ab IBM WebSphere MQ 7.0eine Schichtarchitektur hat und Ausnahmebedingungen von verschiedenen Ebenen im Code ausgelöst werden.

Versuchte eine Anwendung bei Vorgängerversionen von IBM MQ classes for JMS beispielsweise, sich mit einem nicht vorhandenen Warteschlangenmanager zu verbinden, wurde eine JMSException-Ausnahmebedingung mit den folgenden Informationen ausgelöst:

```
MQJMS2005: Failed to create MQQueueManager for 'localhost:QM_test'.
```

Diese Ausnahmebedingung enthielt eine verlinkte MQException-Ausnahmebedingung mit den folgenden Informationen:

```
MQJE001: Completion Code 2, Reason 2058
```

Im Vergleich dazu löst IBM MQ classes for JMS in IBM WebSphere MQ 7.0 unter denselben Umständen eine JMSException-Ausnahmebedingung mit den folgenden Informationen aus:

```
Message : JMSWMQ0018: Failed to connect to queue manager 'QM_test' with
connection mode 'Client' and host name 'localhost'.
Class : class com.ibm.msg.client.jms.DetailedJMSException
Error Code : JMSWMQ0018
Explanation : null
User Action : Check the queue manager is started and if running in client mode,
check there is a listener running. Please see the linked exception
for more information.
```

Diese Ausnahmebedingung enthält eine verlinkte MQException-Ausnahmebedingung mit den folgenden Informationen:

```
Message : JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED')
          reason '2058' ('MQRC_Q_MGR_NAME_ERROR').
Class : class com.ibm.mq.MQException
Completion Code : 2
Reason Code : 2058
```

Wenn Ihre Anwendung Ausnahmebedingungsnachrichten, die von der Methode Throwable.getMessage() zurückgegeben werden, oder Fehlercodes, die von der Methode JMSException.getErrorCode() zurückgegeben werden, analysiert oder testet und Sie ein Upgrade von einem Release vor IBM WebSphere MQ 7.0durchführen, muss Ihre Anwendung wahrscheinlich geändert werden, um IBM MQ classes for JMS in IBM WebSphere MQ 7.0verwenden zu können.

Listener für Ausnahmebedingungen

Eine Anwendung kann einen Listener für Ausnahmebedingungen bei einem Connection-Objekt (Verbindungsobjekt) registrieren. Wenn anschließend ein Problem auftritt, das dazu führt, dass die Verbindung nicht verwendet werden kann, übermitteln IBM MQ classes for JMS eine Ausnahmebedingung an den Listener für Ausnahmebedingungen, indem sie die zugehörige Methode onException() aufrufen. Die 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.

Um in Versionen ab IBM MQ 8.0.0 Fix Pack 2 das Verhalten bei aktuellen JMS-Anwendungen beizubehalten, die einen JMS-MessageListener und einen JMS-ExceptionListener konfigurieren, und um sicherzustellen, dass die IBM MQ classes for JMS mit der JMS-Spezifikation konsistent sind, wird der Standardwert für die Eigenschaft 'ASYNC_EXCEPTIONS JMS ConnectionFactory' in ASYNC_EXCEPTIONS_CONNECTIONBROKEN für die IBM MQ classes for JMS geändert. Folglich werden standardmäßig nur Ausnahmebedingungen an den JMS-ExceptionListener einer Anwendung übermittelt, die Fehlercodes von Verbindungsunterbrechungen entsprechen.

V 9.0.0.1 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 wurde, wird bei allen Ausnahmebedingungen vom Typ 'Verbindung unterbrochen' aufgerufen, und zwar unabhängig davon, ob die Anwendung synchrone oder asynchrone Nachrichtenkonsumenten verwendet.
- Ein von der Anwendung registrierter ExceptionListener wird aufgerufen, wenn ein von einer JMS-Sitzung verwendeter TCP/IP-Socket nicht ordnungsgemäß funktioniert.
- Ausnahmebedingungen während der Nachrichtenübermittlung, die keine unterbrochene Verbindung zur Ursache haben (zum Beispiel MQRC_GET_INHIBITED), werden an den ExceptionListener der Anwendung übermittelt, wenn die Anwendung asynchrone Nachrichtenkonsumenten verwendet und für die von der Anwendung verwendete JMS-ConnectionFactory die Eigenschaft ASYNC_EXCEPTIONS auf den Wert ASYNC_EXCEPTIONS_ALL gesetzt ist.

Anmerkung: Ein ExceptionListener wird für eine Ausnahmebedingung vom Typ 'Verbindung unterbrochen' nur einmal aufgerufen. Dies gilt auch dann, wenn zwei TCP/IP-Verbindungen (einmal von einer JMS-Verbindung und einmal von einer JMS-Sitzung) unterbrochen sind.

Bei allen anderen Arten von Problemen löst der aktuelle JMS-API-Aufruf eine JMSException-Ausnahmebedingung aus.

Wenn eine Anwendung keinen Listener für Ausnahmebedingungen bei einem Connection-Objekt registriert, werden sämtliche Ausnahmebedingungen, die eigentlich an den Listener für Ausnahmebedingungen übermittelt worden wären, in das Protokoll der IBM MQ classes for JMS für JMS geschrieben.

Zugehörige Informationen

IBM MQ-Klassen für JMS-Architektur

ASYNCEXCEPTION

Über eine Anwendung der IBM MQ classes for JMS 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. Die IBM MQ classes for JMS stellen Nachrichtenattribute zur Verfügung, mit denen JMS-Anwendungen MQMD-Felder festlegen und somit die Steuerung von IBM MQ-Anwendungen durch JMS-Anwendungen ermöglichen.

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 232.

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 35. 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 36. 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

Tabelle 36. Eigenschaftsnamen, Werte und Festlegungsmethoden (Forts.)

Eigenschaft	Gültige Werte im Verwaltungstool (Standardwerte sind fett dargestellt)	Gültige Werte in Programmen	Festlegungsmethode
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 36. Eigenschaftsnamen, Werte und Festlegungsmethoden (Forts.)			
Eigenschaft	**Gültige Werte im Verwaltungstool (Standardwerte sind fett dargestellt)**	**Gültige Werte in Programmen**	**Festlegungsmethode**
WMQ_MQMD _MESSAGE_CONTEXT	• **DEFAULT** 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 37. Eigenschaftsnamen, Beschreibungen und Typen

Eigenschaft	Beschreibung	Java-Typ	Link zur ausführlichen Beschreibung
JMS_IBM_MQMD_Report	Optionen für Berichtsnachrichten	Integer	Report
JMS_IBM_MQMD_MsgType	Nachrichtentyp	Integer	MsgType
JMS_IBM_MQMD_Expiry	Nachrichtenlebensdauer	Integer	Expiry
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	Priority
JMS_IBM_MQMD_Persistence	Nachrichtenpersistenz	Integer	Persistence
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 37. 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 der Nachrichteneinreihung	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.

Über eine Anwendung unter Verwendung von IBM MQ classes for JMS auf IBM MQ-Nachrichtendaten zugreifen
Sie können innerhalb einer Anwendung unter Verwendung von IBM MQ classes for JMS auf die vollständigen IBM MQ-Nachrichtendaten zugreifen. 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 JMSBytesMessage senden, können Sie beim Erstellen der IBM MQ die Standardeinstellungen für den JMS-Nachrichtenhauptteil überschreiben. 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 den JMS-Header MQRFH2 nicht enthält. 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 45 auf Seite 236 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 45. Inkonsistente Codierung von MQMD und Nachrichtendaten

Beide Codeausschnitte in Abbildung 46 auf Seite 237 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 46. 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 38. 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 39. 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.MODEL anstelle von SYSTEM.DEFAULT.MODEL.QUEUE verwenden. 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 273.

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. Der Abschnitt „TLS CipherSpecs und Cipher-Suites in IBM MQ classes for JMS" auf Seite 244 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 Clientkanäle (Client Channel Definition Table, CCDT) verwenden. Weitere Informationen finden Sie unter „Definitionstabelle für den Clientkanal mit IBM MQ classes for JMS verwenden" auf Seite 273.

SSLFIPSREQUIRED, Objekteigenschaft
Wenn eine Verbindung eine CipherSuite verwenden soll, die vom IBM Java JSSE-FIPS-Anbieter (IBMJSSEFIPS) unterstützt wird, setzen Sie die Eigenschaft SSLFIPSREQUIRED der Verbindungsfactory auf YES.

Der Standardwert dieser Eigenschaft ist NO. Dies bedeutet, dass eine Verbindung jede beliebige CipherSuite 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 Informationen
Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden
Federal Information Processing Standards (FIPS) für UNIX, 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.

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.

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.

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 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites

CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	nein

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLSv1.2	nein

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECD-HE_RSA_3DES_EDE_CBC_SHA256	SSL_ECD-HE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLSv1.2	nein

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	nein

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_3DES_EDE_CBC_SHA „1" auf Seite 268	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLSv1	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLSv1	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLSv1.2	ja

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1	nein

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLSv1.2	nein

Tabelle 40. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	nein

Anmerkungen:

1. 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.

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

Sie können konfigurieren, ob Ihre Anwendung die Standardzuordnungen zwischen IBM Java-Cipher-Suites und IBM MQ-CipherSpecs oder die Zuordnungen zwischen Oracle-Cipher-Suites und IBM MQ-CipherSpecs verwendet. In jedem Fall können Sie TLS-Cipher-Suites verwenden, unabhängig davon, ob Ihre Anwendung eine IBM JRE oder eine Oracle-JRE verwendet. Welche Zuordnungen verwendet werden, steuert die Java-Systemeigenschaft `com.ibm.mq.cfg.useIBMCipherMappings`. Für diese Eigenschaft sind folgende Werte möglich:

true
Die Zuordnungen zwischen IBM Java-Cipher-Suites und IBM MQ-CipherSpecs werden verwendet.

Dies ist der Standardwert.

false
Die Zuordnungen zwischen Oracle-Cipher-Suites und IBM MQ-CipherSpecs werden verwendet.

Weitere Informationen zur Verwendung von IBM MQ Java und TLS-Verschlüsselungen finden Sie im MQdev-Blogbeitrag MQ Java, TLS-Verschlüsselungen, Nicht-IBM JREs & APARs IT06775, IV66840, IT09423, IT10837.

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 41. 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 269	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. 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.

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 MQConnectionFactorieskönnen Sie die Eigenschaften **SENDEXIT**, **RECEXIT** und **SECEXIT** mit dem Verwaltungstool von IBM MQ JMS oder IBM MQ Explorerfestlegen. 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. 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 42 auf Seite 272 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 42. IBM MQ-Exitverzeichnis

Plattform	Directory
Linux UNIX UNIX 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. Informationen hierzu finden Sie unter Anwendungen, die die IBM MQ-Klassen für JMS verwenden, unter Java Security Manager ausführen.

Die Schnittstellen `MQSendExit`, `MQReceiveExit`und `MQSecurityExit` , die mit früheren Versionen als IBM WebSphere MQ 7.0 bereitgestellt werden, 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 1010. Sie müssen Kanalexitprogramme, die in C oder C++ geschrieben wurden, in dem Verzeichnis speichern, das in Tabelle 42 auf Seite 272 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 273.

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 273.

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 „Konfigurationsdatei für IBM MQ classes for JMS" auf Seite 89 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 270.

  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 automatische Wiederholung einer Clientverbindung ist nur für eigenständige Anwendungen der IBM MQ classes for JMS verfügbar. Die Verwendung der automatischen Wiederholung einer Clientverbindung innerhalb von Anwendungsservern der Java Platform, Enterprise Edition wird nicht unterstützt.

Automatische JMS-Clientverbindungswiederholung mit CONNECTIONNAMELIST
Wenn eine eigenständige Anwendung der IBM MQ classes for JMS eine Verbindungsfactory verwendet, für die die Eigenschaft CONNECTIONNAMELIST definiert ist, kann die Anwendung die automatische Clientverbindungswiederholung verwenden.

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 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 43. Mögliche Werte für die Eigenschaft CLIENTRECCECTOPTIONS

CLIENTRECONNECTOPTIONS	**Verhalten der IBM MQ classes for JMS**
ANY	Der Wert der Eigenschaft CONNECTIONNAMELIST wird zum Öffnen einer Verbindung zu einer Kombination aus Hostname und Port sowie zur Verbindung mit 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 "*".
ASDEF	Verwenden Sie den Wert von DefRecon, um herauszufinden, ob die automatische Clientverbindungswiederholung verfügbar ist.

Tabelle 43. Mögliche Werte für die Eigenschaft CLIENTRECCECTOPTIONS (Forts.)	
CLIENTRECONNECTOPTIONS	**Verhalten der IBM MQ classes for JMS**
INAKTIVIERT	Es erfolgt keine automatische Clientverbindungswiederholung und es wird eine JMS-Ausnahme (JMSException) an die Anwendung zurückgegeben.
QMGR	Der Wert der Eigenschaft CONNECTIONNAMELIST wird zum Öffnen einer Verbindung zu einer Kombination aus Hostname und Port sowie zur Verbindung mit dem über die Eigenschaft QMANAGER angegebenen Warteschlangenmanager verwendet.

Bei der automatischen Clientverbindungswiederholung stellen die IBM MQ classes for JMS 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.

Dieser Prozess der automatischen Clientverbindungswiederholung wird 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.

Verbindung zu Multi-Instanz-Warteschlangenmanagern über CONNECTIONNAMELIST herstellen
Die automatische Clientverbindungswiederholung kann von den Anwendungen der IBM MQ classes for JMS verwendet werden, die eine Verbindung zu einem Multi-Instanz-Warteschlangenmanager herstellen.

Wenn die von einer Anwendung verwendete Warteschlangenmanagerinstanz nicht mehr verfügbar ist, können die IBM MQ classes for JMS automatisch versuchen, für die Anwendung eine Verbindung zu der Standby-Instanz herzustellen. Während der automatischen Clientverbindungswiederholung wird die Anwendung blockiert und fortgesetzt, wenn die IBM MQ classes for JMS eine Verbindung zum Standby-Warteschlangenmanager herstellen.

Definieren Sie die folgenden Eigenschaften in der Verbindungsfactory, die von der Anwendung der IBM MQ classes for JMS verwendet wird, um die automatische Clientverbindungswiederholung für einen Multi-Instanz-Warteschlangenmanager zu aktivieren:

CHANNEL
Der Name eines auf dem Warteschlangenmanager definierten Serververbindungskanals.

QMANAGER

Der Name des Multi-Instanz-Warteschlangenmanagers.

CONNECTIONNAMELIST=host1(port1), host2(port2).

Der erste Eintrag der Liste muss den Hostnamen und Port angeben, die für den Kontakt zur primären Warteschlangenmanagerinstanz verwendet werden. Der zweite Eintrag sollte den Hostnamen und Port des Systems enthalten, auf dem sich die Standby-Warteschlangenmanagerinstanz befindet.

CLIENTRECONNECTOPTIONS=QMGR.

Mit dieser Angabe wird sichergestellt, dass die IBM MQ classes for JMS versuchen, wieder eine Verbindung zu einem Warteschlangenmanager herzustellen, der denselben Namen wie der Warteschlangenmanager hat, mit dem die Anwendung zuvor verbunden war.

Automatische JMS-Clientverbindungswiederholung mit CCDTs
Wenn eine eigenständige Anwendung der IBM MQ classes for JMS eine Verbindungsfactory verwendet, für die die Eigenschaft CCDTURL definiert ist, kann die Anwendung die automatische Clientverbindungswiederholung verwenden.

Das Verhalten der Funktion für die automatische Clientverbindungswiederholung, die von den IBM MQ classes for JMS bereitgestellt wird, hängt von den folgenden 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 über eine Verbindungsfactory, deren Eigenschaft TRANSPORT auf BIND, DIRECT oder DIRECTHTTP gesetzt ist, eine Verbindung zu einem Warteschlangenmanager herstellen, 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 CCDTURL der JMS-Verbindungsfactory (Kurzname CCDT)

Die Eigenschaft CCDTURL zeigt auf die Definitionstabelle für Clientkanal, die von den IBM MQ classes for JMS für die Verbindung zu einem Warteschlangenmanager 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 über das CLIENT-Transportprotokoll eine Verbindung zu einem Warteschlangenmanager herstellt, 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	Die über die Eigenschaft CCDTURL angegebene Definitionstabelle für Clientkanal wird geöffnet, ein Eintrag in der Tabelle ausgewählt und mit diesem Eintrag ein Clientverbindungskanal zu einem Warteschlangenmanager gestartet. 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	Die über die Eigenschaft CCDTURL angegebene Definitionstabelle für Clientkanal wird geöffnet und es werden die Einträge in der Tabelle gesucht, die dem Namen des Warteschlangenmanagers entsprechen, der über die Eigenschaft QMANAGER angegeben wurde. Anschließend wird mit diesen Einträgen ein Clientverbindungskanal zu dem Warteschlangenmanager gestartet.

Bei der automatischen Clientverbindungswiederholung stellen die IBM MQ classes for JMS anhand der Definitionstabelle für Clientkanal, 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.

Dieser Prozess der automatischen Clientverbindungswiederholung wird fortgesetzt, bis die IBM MQ classes for JMS erfolgreich wieder eine Verbindung zu dem in der Eigenschaft QMANAGER angegebenen Warteschlangenmanager herstellen können.

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 wird verhindert, 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, die die Funktion für die automatische Verbindungswiederholung verwenden können.

Verbindung zu Multi-Instanz-Warteschlangenmanagern über CCDTs herstellen
Von Anwendungen der IBM MQ classes for JMS, die eine Verbindung zu einem Multi-Instanz-Warteschlangenmanager herstellen, kann die automatische Clientverbindungswiederholung verwendet werden.

Wenn die von einer Anwendung verwendete Warteschlangenmanagerinstanz nicht mehr verfügbar ist, können die IBM MQ classes for JMS automatisch versuchen, für die Anwendung eine Verbindung zu der Standby-Instanz herzustellen. Während der automatischen Clientverbindungswiederholung wird die Anwendung blockiert und fortgesetzt, wenn die IBM MQ classes for JMS eine Verbindung zum Standby-Warteschlangenmanager herstellen.

Definieren Sie die folgenden Eigenschaften in der Verbindungsfactory, die von der Anwendung der IBM MQ classes for JMS verwendet wird, um die automatische Clientverbindungswiederholung für einen Multi-Instanz-Warteschlangenmanager zu aktivieren:

QMANAGER=Der Name des Multi-Instanz-Warteschlangenmanagers.

CCDTURL=URI

Der URI für eine Clientkanal-Definitionstabelle mit zwei Einträgen für den Multi-Instanz-Warteschlangenmanager - einem für die primäre Instanz und einem für die Standby-Instanz.

Automatische Clientverbindungswiederholung in Java SE- und Java EE-Umgebungen verwenden
Informationen zur IBM MQ-Funktion "Automatische Wiederherstellung der Clientverbindung" sowie zu Multi-Instanz-Warteschlangenmanagern in Java SE- und Java EE-Umgebungen.

Multi-Instanz-Warteschlangenmanager sind Instanzen desselben Warteschlangenmanagers, die auf verschiedenen Servern konfiguriert sind. 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.

Zugehörige Informationen
Warteschlangenmanager mit mehreren Instanzen
Automatische Clientverbindungswiederholung

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** in IBM WebSphere MQ 7.0.1 Fix Pack 3 und höher 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.

Eigenschaft CONNECTIONNAMELIST

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** verwendet, um eine Clientverbindung zu erstellen, versuchen die IBM MQ classes for JMS , eine Verbindung zu jedem Host in Listenreihenfolge 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 IBM MQ-Ursachencode `MQRC_QMGR_NOT_AVAILABLE` zurück.

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.

Eigenschaft CCDTURL

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 Clientkanal-Definitionstabelle mit IBM MQ Classes for JMS verwenden.

Eigenschaft CLIENTRECONNECTOPTIONS zum Aktivieren der automatischen Clientverbindungswiederholung in IBM MQ classes for JMS verwenden

Die Eigenschaft **CLIENTRECONNECTOPTIONS** wird verwendet, um die automatische Clientverbindungswiederholung in IBM MQ classes for JMSzu aktivieren. 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.

ANY
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 IBM MQ-Messaging-Provider stellen Unterstützung für die automatische Clientverbindungswiederholung bereit.

Für die Bereitstellung der automatischen Clientverbindungswiederholung in einer Java EE-Umgebung sind folgende Optionen verfügbar:

- Aktivierungsspezifikation
- WebSphere Application Server-Listenerports
- Enterprise JavaBeans und webbasierte Anwendungen
- Innerhalb von Clientcontainern ausgeführte Anwendungen

Anmerkung: Eine automatische Clientverbindungswiederholung mit Aktivierungsspezifikationen unter Verwendung der von IBM MQ classes for JMS bereitgestellten Funktionalität wird 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.

Dieser 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 447.

Die Verwendung der automatischen Clientverbindungswiederholung in der Methode `onMessage()` einer MDB-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.

Unterstützung für automatische Clientverbindungswiederholung in Java EE-Umgebungen
In Java EE-Umgebungen, wie z. B. WebSphere Application Server, bieten der IBM MQ-Ressourcenadapter und der WebSphere Application Server IBM MQ-Messaging-Provider Unterstützung für die automatische Wiederherstellung der Clientverbindung. In einigen Konfigurationen ist diese Unterstützung jedoch eingeschränkt.

Der in Java EE-Umgebungen bereitstellbare IBM MQ-Ressourcenadapter und der WebSphere Application Server IBM MQ-Messaging-Provider verwenden die IBM MQ classes for JMS zur Kommunikation mit den IBM MQ-Warteschlangenmanager.

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 automatische Wiederherstellung der Clientverbindung in Java EE-Umgebungen

	Eigenschaft CONNECTIONNAMELIST	**Eigenschaft CCDTURL**	**Eigenschaft CLIENTRECONNECTOPTIONS**	**Alternative Methode der automatischen Wiederherstellung der Clientverbindung**
Aktivierungsspezifikationen	Unterstützt mit Einschränkungen	Unterstützt mit Einschränkungen	Nicht unterstützt	Java EE-Umgebung und Aktivierungsspezifikationen stellen einen eigenen Verbindungswiederherstellungsmechanismus bereit
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. MDBs können aus ihrer **onMessage()**-Methode heraus 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 aus ihrer onMessage()-Methode heraus weitere 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 289.

Eigenschaft CONNECTIONNAMELIST

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 Eigenschaft **reconnectionRetryCount** des IBM MQ-Ressourcenadapters definiert die Anzahl der aufeinanderfolgenden Verbindungswiederholungen, bevor eine Aktivierungsspezifikation beendet wird und ein manueller Neustart erforderlich ist.

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.

Eigenschaft CCDTURL

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.

Hat die Aktivierungsspezifikation alle Einträge der CCDT ausprobiert, ohne dass die Verbindung mit dem Warteschlangenmanager gelingt, startet die Spezifikation den nächsten Verbindungsversuch erst wieder nach der in der Eigenschaft **reconnectionRetryInterval** des IBM MQ-Ressourcenadapters angegebenen Zeit.

Die Eigenschaft **reconnectionRetryCount** des IBM MQ-Ressourcenadapters definiert die Anzahl der aufeinanderfolgenden Verbindungswiederholungen, bevor eine Aktivierungsspezifikation beendet wird und ein manueller Neustart erforderlich ist.

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.

CLIENTRECONNECTOPTIONS

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 aus ihrer onMessage()-Methode heraus 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" auf Seite 289.

Für WebSphere Application Server-Listenerports gilt Folgendes:

- **CONNECTIONNAMELIST** und **CCDTURL** werden mit Einschränkungen unterstützt
- **CLIENTRECONNECTOPTIONS** wird nicht unterstützt

CONNECTIONNAMELIST

Listener-Ports verwenden beim Herstellen einer Verbindung zu IBM MQ JMS-Verbindungspools. Sie unterliegen daher den Einschränkungen dieser Pools. Weitere Informationen hierzu finden Sie im Thema „Aktivierungsspezifikationen" auf Seite 284.

Wenn keine Verbindungen mehr frei sind und die maximal erlaubte Anzahl an Verbindungen für diese Verbindungsfactory noch nicht erreicht ist, wird über die**CONNECTIONNAMELIST** versucht, eine neue Verbindung mit IBM MQ herzustellen.

Wenn keines der in der **CONNECTIONNAMELIST** genannten IBM MQ-Systeme verfügbar ist, stoppt der Listenerport.

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.

CCDTURL

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.

CLIENTRECONNECTOPTIONS

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
Enterprise JavaBean-Anwendungen (EJBs) sowie innerhalb des Webcontainers ausgeführte Anwendungen (beispielsweise Servlets) verwenden zur Erstellung einer Verbindung mit einem IBM MQ-Warteschlangenmanager eine JMS-Verbindungsfactory.

Für EJBs und webbasierte Anwendungen gelten folgende Einschränkungen:

- **CONNECTIONNAMELIST** und **CCDTURL** werden mit Einschränkungen unterstützt
- **CLIENTRECONNECTOPTIONS** wird nicht unterstützt

CONNECTIONNAMELIST

Wenn die Java EE-Umgebung einen Verbindungspool für JMS-Verbindungen bereitstellt, finden Sie in „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden" auf Seite 290 Informationen darüber, wie dies das Verhalten der Eigenschaft **CONNECTIONNAMELIST** beeinflusst.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, Die Anwendung verwendet die Eigenschaft **CONNECTIONNAMELIST** auf dieselbe 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.

CCDTURL

Wenn die Java EE -Umgebung einen Verbindungspool für JMS -Verbindungen bereitstellt, finden Sie unter „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden" auf Seite 290 Informationen dazu, wie sich dies auf das Verhalten der Eigenschaft **CCDTURL** auswirkt.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, Die Anwendung verwendet die Eigenschaft **CCDTURL** auf dieselbe 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.

CLIENTRECONNECTOPTIONS

Die Eigenschaft **CLIENTRECONNECTOPTIONS** muss daher für alle von EJBs oder webbasierten Anwendungen verwendeten JMS-Verbindungsfactorys auf *DISABLED* gesetzt sein.

Anwendungen, die beim Ausfall des verwendeten Warteschlangenmanagers eine automatische Herstellung der Verbindung mit einem neuen Warteschlangenmanager erfordern, müssen ihre eigene Verbindungswiederholungslogik implementieren. Weitere Informationen hierzu finden Sie unter „Verbindungswiederherstellungslogik in einer Java EE-Anwendung implementieren" auf Seite 291.

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 innerhalb dieser Umgebungen verwenden zur Herstellung der Verbindung mit einem IBM MQ-Warteschlangenmanager eine JMS-Verbindungsfactory.

Für Anwendungen innerhalb von Clientcontainern gilt Folgendes:

- **CONNECTIONNAMELIST** und **CCDTURL** werden vollständig unterstützt
- **CLIENTRECONNECTOPTIONS** wird vollständig unterstützt

CONNECTIONNAMELIST

Wenn die Java EE-Umgebung einen Verbindungspool für JMS-Verbindungen bereitstellt, finden Sie in „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden" auf Seite 290 Informationen darüber, wie dies das Verhalten der Eigenschaft **CONNECTIONNAMELIST** beeinflusst.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, Die Anwendung verwendet die Eigenschaft **CONNECTIONNAMELIST** auf dieselbe Weise wie Java SE -Anwendungen.

CCDTURL

Wenn die Java EE -Umgebung einen Verbindungspool für JMS -Verbindungen bereitstellt, finden Sie unter „CONNECTIONNAMELIST oder CCDT in Verbindungspools verwenden" auf Seite 290 Informationen dazu, wie sich dies auf das Verhalten der Eigenschaft **CCDTURL** auswirkt.

Wenn die Java EE-Umgebung keinen JMS-Verbindungspool bereitstellt, Die Anwendung verwendet die Eigenschaft **CCDTURL** auf dieselbe 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 zum ersten Mal zum Erstellen einer Verbindung verwendet wird, verwendet die Java EE -Umgebung entweder die **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.

Wenn eine Verbindung aus der Verbindungsfactory erstellt wird, gibt die Java EE -Umgebung die Verbindung aus dem Verbindungspool zurück, anstatt die Eigenschaften **CONNECTIONNAMELIST** oder **CCDTURL** zum Erstellen einer neuen Verbindung zu verwenden.

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 **CONNECTION-NAMELIST** oder **CCDTURL** , um eine neue Verbindung zu IBM MQherzustellen.

Aufgrund der Verwendung von **CONNECTIONNAMELIST** und CCDTs zum Erstellen von JMS -Verbindungen ist es auch möglich, einen Verbindungspool zu haben, der Verbindungen zu verschiedenen IBM MQ -Systemen enthält.

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 unter `hostname1(port1)` ist jetzt verfügbar, sodass eine JMS -Verbindung zu 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, die beim Ausfall des verwendeten Warteschlangenmanagers eine automatische Herstellung der Verbindung mit einem neuen Warteschlangenmanager erfordern, müssen ihre eigene Verbindungswiederholungslogik implementieren.

Nachfolgend erhalten Sie nähere Informationen darüber, wie Sie dies erreichen:

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 schließt eine Ruhemodus-und Wiederholungsschleife ein, um die nicht sofortige Verbindungswiederholung zu verarbeiten. In der Praxis können dadurch 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
Die Verwendung einer Form des Verbindungspoolings außerhalb von Java EE trägt dazu bei, die Gesamtlast zu reduzieren, die beispielsweise durch einige eigenständige Anwendungen, die Frameworks verwenden oder in Cloudumgebungen implementiert werden, sowie durch eine größere Anzahl von Clientverbindungen in `QueueManagers` verursacht wird, was zu einer Erhöhung 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 stellen Verbindungspoolingfunktionen bereit, die von MDB-Anwendungen (Message-driven Bean), Enterprise JavaBeans 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 47 auf Seite 295 zeigt, wie Aktivierungsspezifikationen und Listenerports eine JMS-Verbindung einrichten und diese Verbindung zur Überwachung eines Ziels auf Nachrichten im Normalmodus verwenden.

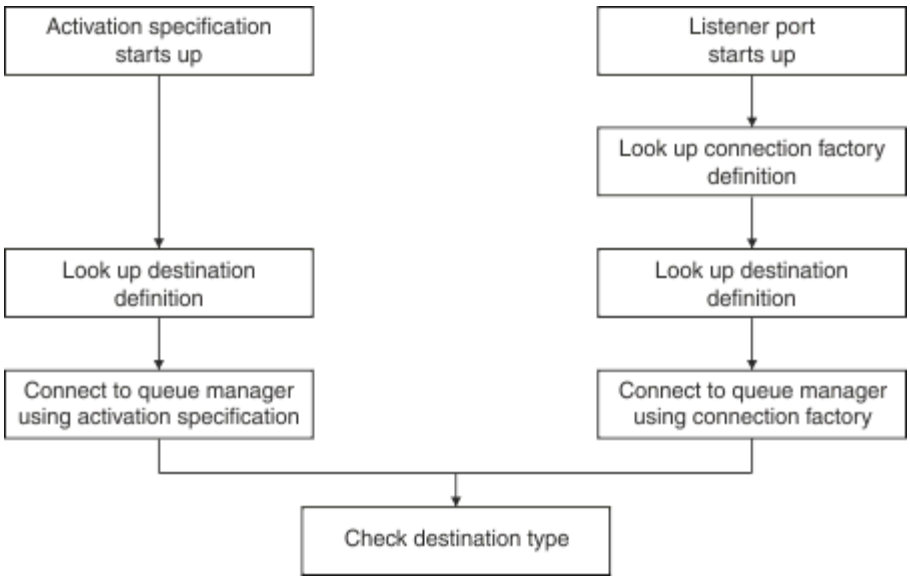


Abbildung 47. Normalmodus

Bei Verwendung des IBM MQ-Messaging-Providers können Anwendungen, aus denen Nachrichten versendet werden (wie Enterprise JavaBeans und Servlets), und die Listenerport-Komponente für Message-driven Beans (MDBs) diese Verbindungspools nutzen.

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 300 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 303 wird erläutert, was mit diesen Verbindungen passiert, wenn die Anwendung bzw. der Listenerport sie nicht mehr benötigt.

Im Abschnitt „Beispiele für Poolwartungsthreads” auf Seite 304 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 48 auf Seite 297 zeigt die Beziehung zwischen diesen Objekten.

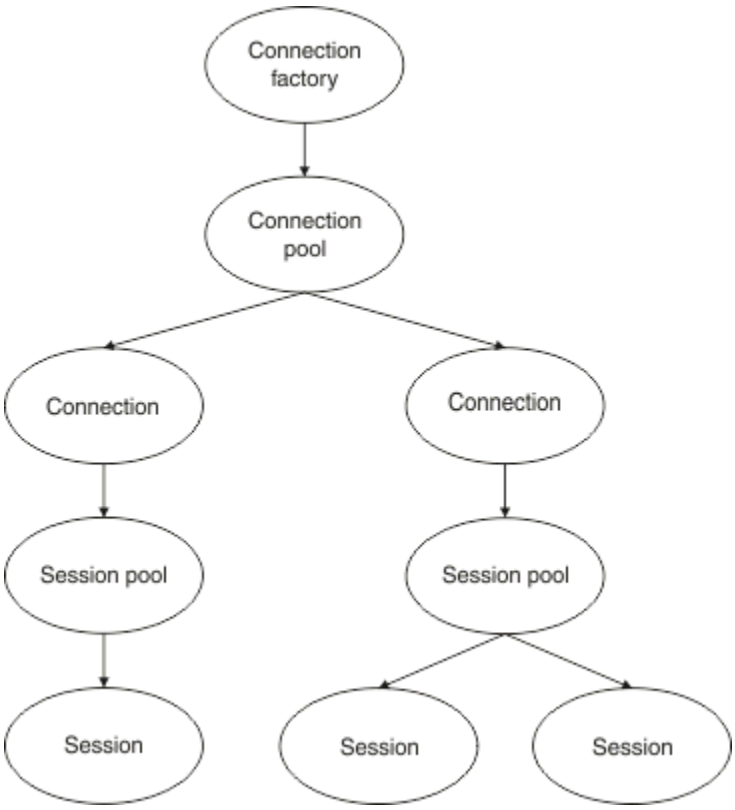


Abbildung 48. Verbindungs- und Sitzungspools

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 302.

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 304 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.

Stellen Sie sich nun vor, ein EJB erstellt aus der Verbindungsfactory `jms/CF1` eine JMS-Verbindung mit IBM MQ und versucht anschließend durch Aufruf von `Connection.createSession()` eine JMS-Sitzung über diese Verbindung zu erstellen.

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 thresholdfestgelegte` 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 49 auf Seite 300 zeigt, wie der Verbindungspool für WebSphere Application Server 7.5 und 8.0funktioniert.

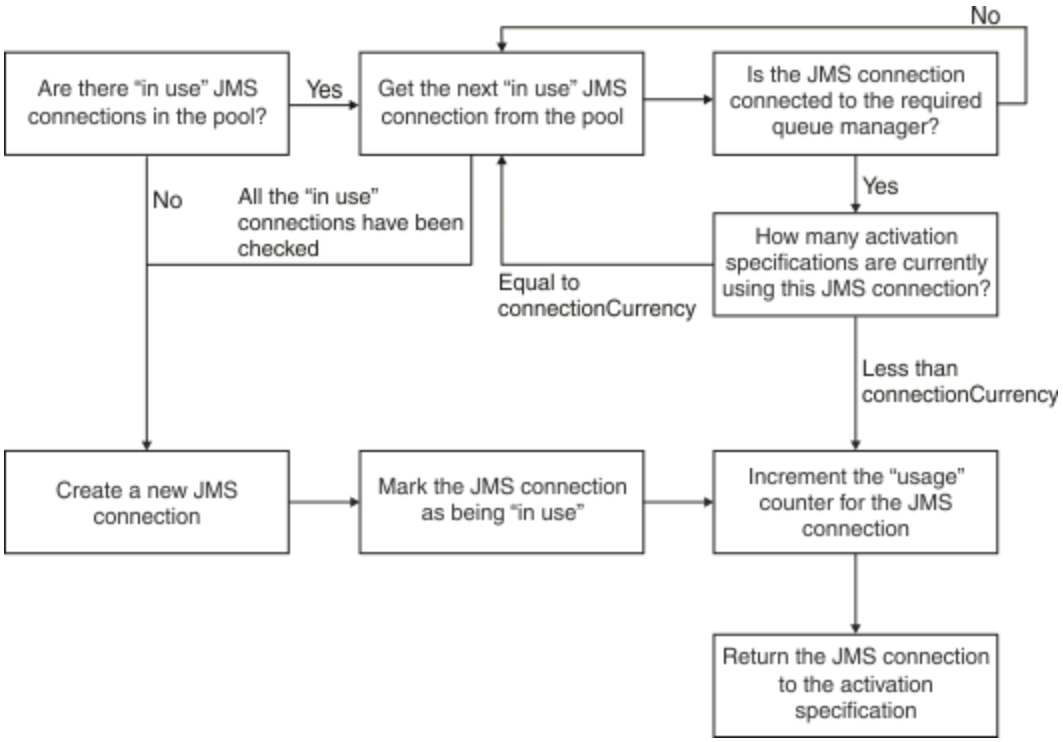


Abbildung 49. WebSphere Application Server 7.5 und 8.0 -Funktionsweise des Verbindungspools

Abbildung 50 auf Seite 301 zeigt, wie der Verbindungspool für WebSphere Application Server 8.5funktioniert.

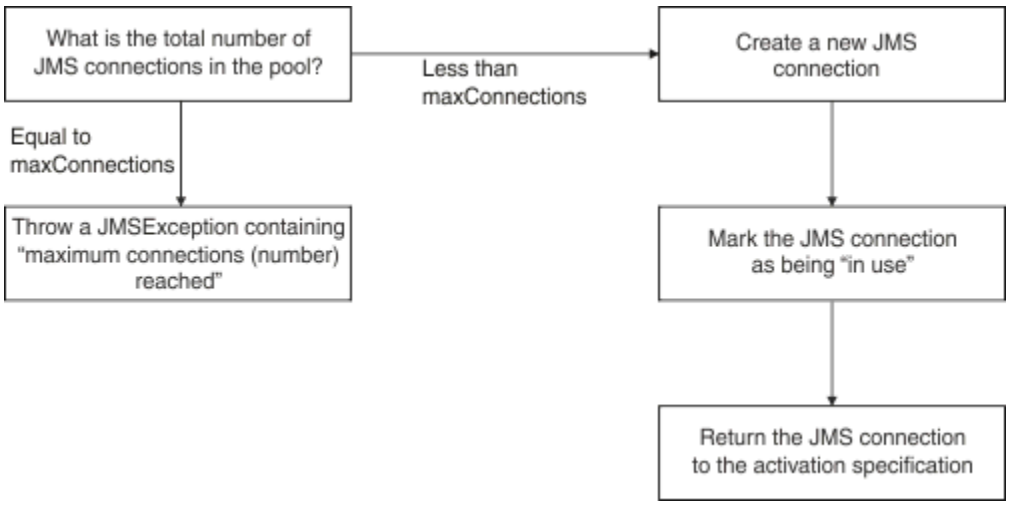


Abbildung 50. WebSphere Application Server 8.5 -Funktionsweise des Verbindungspools

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/CF1`mit `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/CF1`herzustellen. 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: Wenn Sie **Aged timeout** mit JMS -Verbindungen innerhalb von EJBs verwenden wollen, stellen Sie sicher, dass alle JMS-Anforderungen explizit in der JMS -Sitzung festgeschrieben werden, bevor die EJB-Methode, die die JMS -Operationen ausführt, beendet wird.

Beispiele für Poolwartungsthreads
In diesem Abschnitt wird die Funktionsweise des Poolverwaltungsthreads anhand eines Beispiels für Enterprise Java Beans (EJBs) veranschaulicht. 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 302.

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** (Zeitlimit für nicht verwendete Verbindungen) ist 300 Sekunden.

Nach dem Starten des Anwendungsservers wird die MethodesendMessage() 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 vorangegangene Beispiel zeigte, wie der Verwaltungsthread die Eigenschaften **Reap time** und **Unused timeout** zur Vermeidung veralteter Verbindungen verwendet, wenn die Eigenschaft **Aged timeout** auf 0 (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 und findet bei der Untersuchung des freien Pools für 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 301 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 Poolverwaltungsthread aktiviert und stellt bei der Untersuchung des freien Pools für `jms/CF1` fest, dass sich die Verbindung `c1` bereits länger im freien Pool befindet, als der Wert der Eigenschaft **`Unused timeout`** (Zeitlimit für nicht verwendete Verbindungen) der Verbindungsfactory 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`. Auch diese Verbindung befindet sich schon länger im Pool, als die Eigenschaft **`Unused timeout`** (Zeitlimit für nicht verwendete Verbindungen) zulässt. Da `c2` nun aber die einzige Verbindung im Pool ist, würde ein Schließen dieser Verbindung wiederum bedeuten, dass im Pool weniger als die geforderte Anzahl an Verbindungen verbliebe. Deswegen lässt der Verbindungsmanager die Verbindung `c2` geöffnet.

JMS-Verbindungen und IBM MQ
Informationen zur Verwendung von IBM MQ als JMS-Provider.

BINDINGS-Transport verwenden

Wenn eine Verbindungsfactory für die Verwendung des Bindungstransports konfiguriert wurde, richtet jede JMS -Verbindung einen Dialog (auch bekannt als **hconn**) mit IBM MQein. Dieser Dialog verwendet für die Kommunikation mit dem Warteschlangenmanager die Interprozesskommunikation (bzw. gemeinsam genutzten Speicher).

CLIENT-Transport verwenden

Wenn eine Verbindungsfactory des IBM MQ -Messaging-Providers für die Verwendung des Clienttransports konfiguriert wurde, stellt jede Verbindung, die aus dieser Factory erstellt wird, einen neuen Dialog (auch bekannt als**hconn**) mit IBM MQher.

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 im Abschnitt TCP/IP-Verbindung in IBM MQ classes for JMS gemeinsam nutzen.

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.

Angenommen, Sie haben zwei Verbindungsfactorys, `jms/CF1` und `jms/CF2` , die so konfiguriert wurden, dass sie über denselben IBM MQ -Channel eine Verbindung zu demselben 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 Informationen

IBM MQ classes for JMS verwenden

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
- Sitzungen
- 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 folgender:

- 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 verwenden" auf Seite 77
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

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 innerhalb derselben Java-Laufzeitumgebung ausgeführt werden und die IBM MQ classes for JMS oder den IBM MQ-Ressourcenadapter für die Verbindung mit einem Warteschlangenmanager unter Verwendung des CLIENT-Transports verwenden, können so festgelegt werden, dass sie dieselbe Kanalinstanz gemeinsam nutzen.

Zwischen Kanalinstanzen und TCP/IP-Verbindungen besteht eine Eins-zu-eins-Beziehung. Für jede Kanalinstanz wird eine TCP/IP-Verbindung erstellt.

Wenn ein Kanal mit dem Parameter **SHARECNV** definiert ist und für diesen ein höherer Wert als 1 festgelegt ist, kann diese Anzahl an Dialogen eine Kanalinstanz gemeinsam nutzen. Damit eine Verbindungsfactory oder Aktivierungsspezifikation diese Funktion verwenden kann, müssen Sie die Eigenschaft **SHARECONVALLOWED** auf YES setzen.

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 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 JMS verwendet wird, wird bei einer neuen Verbindungsanforderung nicht immer automatisch eine bereits hergestellte Verbindung gemeinsam genutzt.

Anzahl der Kanalinstanzen 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_value* der Wert der Eigenschaft **maxPoolDepth** und *SHARECNV_value* ist der Wert der Eigenschaft **SHARECNV** bei dem Kanal, der von der Aktivierungsspezifikation verwendet wird.

Sonstige JMS-Anwendungen

Anzahl der Kanalinstanzen = (*jms_connections* + *jms_sessions*) / *SHARECNV_value*

Dabei ist *JMS-Verbindungen* die Anzahl der Verbindungen, die von der Anwendung erstellt werden, *JMS-Sitzungen* die Anzahl der JMS-Sitzungen, die von der Anwendung erstellt werden, und *SHARECNV_wert* ist der Wert der Eigenschaft **SHARECNV** in dem Kanal, der von der Aktivierungsspezifikation verwendet wird.

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. Bei dem Kanal, der von der Anwendung für die Verbindung mit dem Warteschlangenmanager verwendet wird, 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. Die Aktivierungsspezifikation ist mit dem Wert 10 für die Eigenschaft **maxPoolDepth** konfiguriert. Der Kanal, für dessen Verwendung die Aktivierungsspezifikation konfiguriert wurde, weist den Wert 10 für die Eigenschaft **SHARECNV** auf. 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. Die Aktivierungsspezifikation wird mit der Eigenschaft **maxPoolDepth** konfiguriert, die auf 5 gesetzt ist. Für den Kanal, dessen Verwendung für 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, entspricht 6, da die Eigenschaft **SHARECNV** bei dem 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 504

In IBM WebSphere MQ 7.0 wurde als neue Funktion die gemeinsame Dialognutzung eingeführt. Dank dieser Funktion 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:

- der JMS-Zieleigenschaft PUTASYNCALLOWED (Kurzname - PAALD).

  PUTASYNCALLOWED steuert, ob JMS-Anwendungen Nachrichten asynchron einreihen können, wenn 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 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. Außerdem werden die Werte aufgelistet, die für eine Aktivierung der Funktionalität der asynchronen Put-Operationen erforderlich sind:

Tabelle 46. PUTASYNCALLOWED- und DEFPRESP-Eigenschaften zur Bestimmung, ob Nachrichten asynchron eingereiht werden

JMS-Zieleigenschaft	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**PUTASYNCALLOWED = AS_DEST oder AS_Q_DEF oder AS_T_DEF**
IBM MQ-Warteschlange-neigenschaft			
DEFPRESP=SYNC	Funktionalität der asynchronen Put-Operation nicht aktiviert	Funktionalität der asynchronen Put-Operation aktiviert	Funktionalität der asynchronen Put-Operation nicht aktiviert
DEFPRESP=ASYNC	Funktionalität der asynchronen Put-Operation nicht aktiviert	Funktionalität der asynchronen Put-Operation aktiviert	Funktionalität der asynchronen Put-Operation aktiviert

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.

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:

- der 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. Außerdem werden die Werte aufgelistet, die für eine Aktivierung der Vorauslesefunktion erforderlich sind:

Tabelle 47. Eigenschaften READAHEADALLOWED und DEFREADA zur Festlegung, ob das Vorauslesen beim Abrufen oder Durchsuchen von nicht persistenten Nachrichten außerhalb einer Transaktion verwendet wird

IBM MQ-Zieleigenschaft	**READAHEADALLOWED= YES**	**READAHEADALLOWED= NO**	**AS_DEST oder AS_Q_DEF or AS_T_DEF**
IBM MQ-Warteschlangeneigenschaft			
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.

Funktionalität von JMS 2.0 nutzen
JMS 2.0 bietet mehrere neue Funktionalitätsbereiche für die IBM MQ classes for JMS.

Beim Entwickeln einer JMS-Anwendung für IBM MQ 8.0 oder höher müssen Sie gegebenenfalls die Auswirkung dieser Funktionalität auf Ihren Warteschlangenmanager berücksichtigen.

Zugehörige Informationen
IBM MQ Java-Sprachenschnittstellen

Zustellungsverzögerung durch JMS 2.0
In JMS 2.0 können Sie beim Senden einer Nachricht eine Zustellungsverzögerung angeben. 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.

Ab IBM MQ 8.0 wird die Zustellungsverzögerung über eine 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 ange-

geben 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 beziehen sich gleichermaßen auf `MessageProducers` und `JMSProducers`, im Falle von `JMSProducers` werden jedoch Laufzeitausnahmen (`JMSRuntimeExceptions`) 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.
- Eine Zustellungsverzögerung gemeinsam mit einer Nachrichtenzusammenfassung wird nicht unterstützt. Beim Versuch, diese Kombination zu verwenden, kommt es zu einer JMSException-Ausnahme 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.

Berechtigung

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 Auf Multiplatforms ist die Warteschlange SYSTEM.DDELAY.LOCAL.QUEUE 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 wird SYSTEM.DDELAY.LOCAL.QUEUE 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 für 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.

Die Zustellungsverzögerung wird von OPMODE nicht geschützt. Wenn Sie die Zustellungsverzögerung mit einem IBM MQ 8.0 -Warteschlangenmanager verwenden und anschließend alle Nachrichten auf dem SYSTEM.DDELAY.LOCAL.QUEUE -Warteschlangen werden marogezont, wenn Sie sie nicht manuell bearbeiten.

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

Bei der Zustellungsverzögerung wird das Verhalten der Ablaufeigenschaft MQMD.Expiry beibehalten. 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 abgeschaltet wird, ist ein Neustart möglich, indem das PUT-Attribut der Staging-Warteschlange von ENABLED auf DISABLED und anschließend wieder auf ENABLED gesetzt wird. Sollten Sie den Zustellungsverzögerungsprozessor aus irgendeinem Grund stoppen müssen, setzen Sie das PUT-Attribut der Staging-Warteschlange auf DISABLED.

Multi 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. Hierbei handelt es sich um eine Abweichung von der JMS 2.0-Spezifikation.

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
In IBM MQ 8.0 oder höher gibt es 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, indem 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

Mit der JMS 2.0-Spezifikation werden gemeinsam genutzte Subskriptionen eingeführt. Auf diese Weise können Nachrichten von einer Themensubskription von mehreren Konsumenten gemeinsam genutzt werden. Jede Nachricht von der Subskription wird nur einem der Konsumenten für die betreffende Subskription zugestellt. Gemeinsam genutzte Subskriptionen werden über den betreffenden Aufruf an die JMS 2.0-API 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 JMS 2.0-Spezifikation definiert kein Standardverfahren zum Ansteuern einer MDB aus einer shared-Subscription, sodass IBM MQ 8.0 oder höher zu diesem Zweck die Aktivierungsspezifikationseigenschaft 'sharedSubscription' bereitstellt. Weitere Informationen zu dieser Eigenschaft finden Sie in den Abschnitten „Ressourcenadapter für eingehende Kommunikation konfigurieren” auf Seite 450 und „Beispiele für Definition der Eigenschaft sharedSubscription” auf Seite 469.

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 notwendig, über die normale JMS-Konfiguration hinausgehend separat Objekte auf dem Warteschlangenmanager zu erstellen, alle erforderlichen Objekte werden dynamisch erstellt.

Entscheidung zwischen gemeinsam genutzten oder geklonten Subskriptionen

Bei der Entscheidung, ob gemeinsam genutzte oder geklonte Subskriptionen verwendet werden sollen, sind die Vorteile von beiden zu berücksichtigen. Wenn möglich, sollten Sie gemeinsam genutzte Subskriptionen verwenden, da es sich hierbei um ein durch die Spezifikation definiertes Verhalten und nicht um eine IBM MQ-spezifische Erweiterung handelt.

In der folgenden Tabelle sind einige Punkte aufgeführt, die bei der Entscheidung für gemeinsam genutzte oder geklonte Subskriptionen zu beachten sind:

Tabelle 48. Überlegungen bei der Auswahl zwischen gemeinsam genutzten und geklonten Subskriptionen

Gemeinsam genutzte Subskriptionen	**Geklonte Subskriptionen**
Gemeinsam genutzte Subskriptionen sind eine Standardkomponente der JMS 2.0-Spezifikation.	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.

Versuche zur Änderung der gemeinsamen Nutzung einer vorhandenen Subskription

V9.0.0.1

Wenn eine Subskription als gemeinsam genutzt erstellt wird, kann die gemeinsame Nutzung später nicht aufgehoben werden. Dasselbe gilt auch umgekehrt.

Ab IBM MQ 9.0.0 Fix Pack 1 gilt aufgrund einer Aktualisierung des IBM MQ-Warteschlangenmanagers, dass, wenn eine JMS 2.0-Anwendung eine permanente, nicht gemeinsam genutzte Subskription erstellt und eine andere Anwendung, die keine JMS 2.0-Anwendung ist, dann versucht, die Subskription fortzusetzen, der folgende Ursachencode zurückgegeben wird:

```
2432 (MQRC_SUB_ALREADY_EXISTS
```

Zugehörige Verweise

„Beispiele für Definition der Eigenschaft sharedSubscription" auf Seite 469
Sie können die Eigenschaft sharedSubscription einer Aktivierungsspezifikation in einer `server.xml`-Datei von WebSphere Application Server Liberty definieren. Alternativ können Sie die Eigenschaft in einer Message-driven Bean (MDB) mithilfe von Annotationen definieren.

Zugehörige Informationen

Subskribenten und Subskriptionen

Subskriptionspermanenz

Gemeinsam genutzte Subskriptionen in JMS 2.0 verwenden

CLONESUPP

Eigenschaft SupportMQExtensions

Mit der JMS 2.0-Spezifikation werden Änderungen bestimmter Verhaltensweisen eingeführt. Ab 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.

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 mit dem Wert *-1* angegeben werden, dass für die Nachricht die Standardpriorität der Warteschlange verwendet werden soll. 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.

Die Eigenschaft `com.ibm.mq.jms.SupportMQExtensions` ist eine boolesche Eigenschaft, die in `com.ibm.mqjms.jar` enthalten ist. 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

„Konfigurationsdatei für IBM MQ classes for JMS" auf Seite 89
In einer Konfigurationsdatei für die IBM MQ classes for JMS sind Eigenschaften angegeben, die zur Konfiguration der IBM MQ classes for JMS verwendet werden.

Zugehörige Verweise

„Zur Konfiguration des JMS-Clientverhaltens verwendete Eigenschaften" auf Seite 96
Mit diesen Eigenschaften können Sie das Verhalten des JMS-Clients konfigurieren.

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 434 finden Sie praxisbezogene Verbindungsinformationen.

IBM MQ classes for JMS unterstützen die Anwendungsserverfunktionen (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 JMS-Spezifikation ermöglicht einem Anwendungsserver mithilfe der Schnittstelle `ConnectionConsumer` eine nahtlose Integration in eine JMS-Implementierung. 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 322
- „Allgemeine Grundsätze für das Publish/Subscribe-Messaging unter Verwendung von ASF" auf Seite 323
- „Nachrichten aus der Warteschlange in ASF entfernen" auf Seite 324
- Behandlung von nicht verarbeitbaren Nachrichten in ASF. (siehe „Nicht verarbeitbare Nachrichten in IBM MQ classes for JMS behandeln" auf Seite 221).

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 324). 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 Emp-

fang 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 der Standardwert.

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 327. 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 221.

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 221
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 326 und „Ursachen- und Feedbackcodes in ASF" auf Seite 327.

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-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 51 auf Seite 329 werden die Grundsätze der ServerSessionPool- und ServerSession-Funktionalität zusammengefasst.

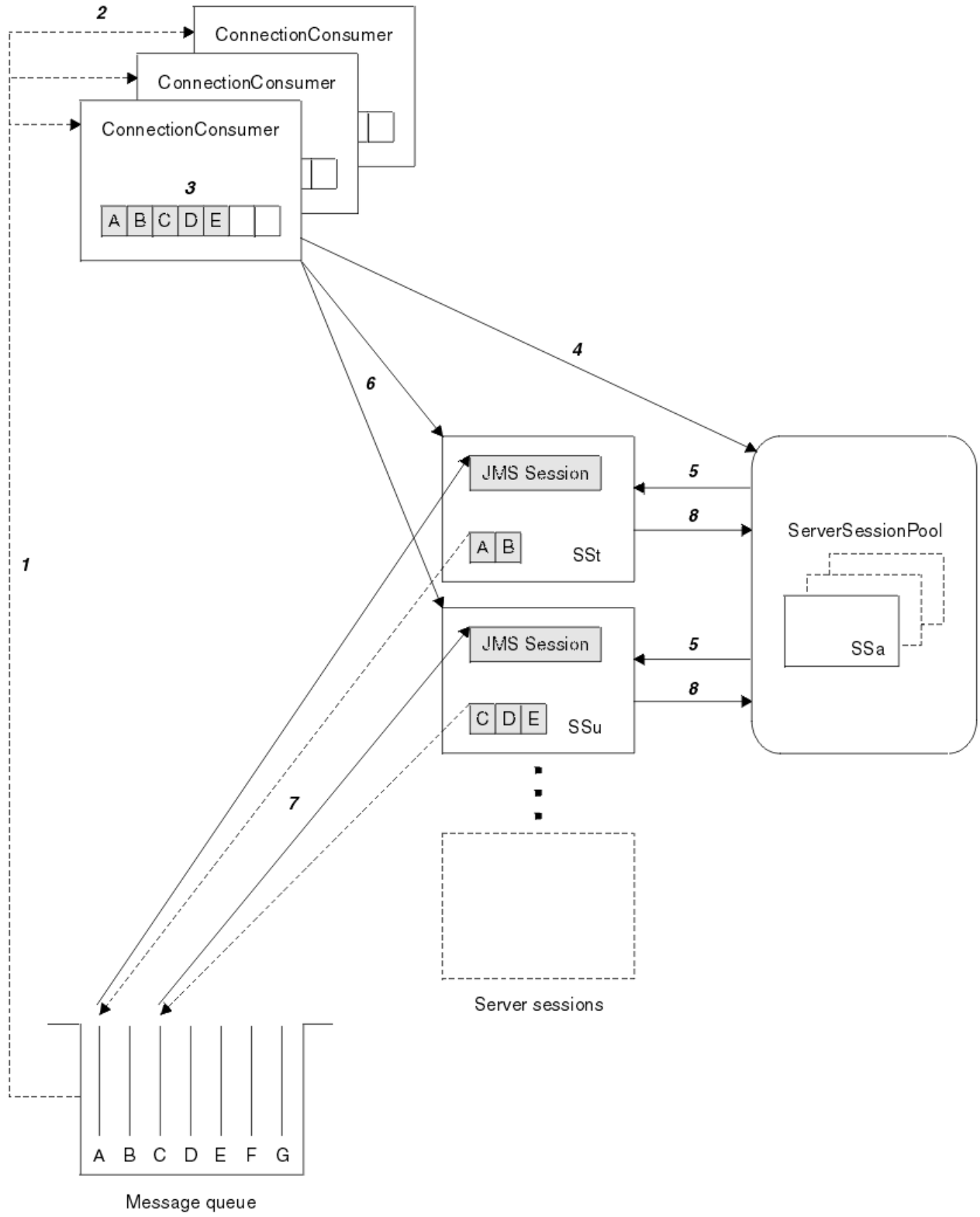


Abbildung 51. ServerSessionPool- und ServerSession-Funktionalität

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 diese 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 IBM MQ classes for JMS in einem CICS OSGi JVM-Server verwenden

In IBM MQ 8.0 ist Unterstützung für die Verwendung der IBM MQ classes for JMS in bestimmten Versionen des CICS Open Services Gateway initiative (OSGi) Java Virtual Machine (JVM)-Servers hinzugekommen.

Achtung: Überprüfen Sie die Systemvoraussetzungen für das CICS-System, das Ihr Unternehmen verwendet. Ausführliche Informationen finden Sie unter Detailed system requirements for CICS Transaction Server.

Dieser Abschnitt gibt eine Einführung in die Einrichtung der IBM MQ classes for JMS in einer JVM-Serverumgebung.

Weitere Informationen zum Einrichten und Konfigurieren Ihres Systems finden Sie unter Using IBM MQ classes for JMS in einem OSGi JVM-Server in der Dokumentation zuCICS .

Allgemeine Einschränkungen

Die folgenden Einschränkungen gelten, wenn die IBM MQ classes for JMS auf einem CICS OSGi JVM-Server verwendet werden:

- Verbindungen im Clientmodus werden nicht unterstützt.
- Verbindungen werden nur zu Warteschlangenmanagern von IBM WebSphere MQ 7.1oder IBM MQ 8.0 oder höher unterstützt. Das Attribut **PROVIDERVERSION** in der Verbindungsfactory darf entweder nicht angegeben sein oder es muss ein Wert größer oder gleich sieben angegeben werden.
- Die Verwendung von XA-Verbindungsfactorys wie z. B. `com.ibm.mq.jms.MQXAConnectionFactory` wird nicht unterstützt.
- Die Verwendung der IBM MQ classes for JMS in einem CICS OSGi JVM-Server wird nur in CICS 5.2 oder höher unterstützt. Bei Verwendung von CICS 5.2 sollten Sie APAR PI32151 anwenden.

V 9.0.1 Vor IBM MQ 9.0.1wird die Verwendung von IBM MQ classes for JMS in einer Liberty JVM-Serverumgebung nicht unterstützt.

Zugehörige Informationen

Eigenschaft JMS **PROVIDERVERSION** konfigurieren

z/OS V 9.0.1 IBM MQ classes for JMS in einem CICS Liberty-JVM-Server verwenden

Ab IBM MQ 9.0.1 können Java-Programme, die in einem CICS Liberty-JVM-Server ausgeführt werden, mithilfe der IBM MQ classes for JMS auf IBM MQ zugreifen.

Sie müssen eine IBM MQ 9.0.1 oder eine höhere Version des IBM MQ-Ressourcenadapters verwenden, die über Fix Central erhältlich ist (siehe „Ressourcenadapter in Liberty installieren” auf Seite 443).

Es sind zwei Varianten von Liberty Profile-JVMs in CICS 5.3 und höher verfügbar. Für die möglichen Typen von Verbindungen zu IBM MQ gelten folgende Einschränkungen:

CICS Liberty Standard

- Der IBM MQ-Ressourcenadapter kann eine Verbindung zu einer beliebigen in Betrieb befindlichen Version von IBM MQ im CLIENT-Modus herstellen.
- Der IBM MQ-Ressourcenadapter kann eine Verbindung zu einer beliebigen in Betrieb befindlichen Version von IBM MQ for z/OS im BINDINGS-Modus herstellen, wenn keine CICS-Verbindung (aktive CICS-MQCONN-Ressourcendefinition) zum selben Warteschlangenmanager aus derselben CICS-Region besteht.

CICS Liberty Integrated

- Der IBM MQ-Ressourcenadapter kann eine Verbindung zu einer beliebigen in Betrieb befindlichen Version von IBM MQ im CLIENT-Modus herstellen.
- Verbindungen im BINDINGS-Modus werden nicht unterstützt.

Weitere Informationen sowie Details zur Einrichtung und Konfiguration Ihres Systems finden Sie im Abschnitt Using IBM MQ classes for JMS in a Liberty JVM server in der CICS-Dokumentation.

IBM MQ classes for JMS in IMS verwenden

Mit IBM MQ 8.0.0 Fix Pack 4 wurde Unterstützung für die Verwendung der IBM MQ classes for JMS in IMS Version 13 und höher hinzugefügt.

Überprüfen Sie die Systemvoraussetzungen für das IMS-System, das Ihr Unternehmen verwendet. Weitere Informationen finden Sie unter Allgemeine Planungsinformationen für IMS 13 .

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 336.

Anmerkung: Ähnliche Einschränkungen gelten für die traditionellen (JMS 1.0.2) domänenspezifischen Schnittstellen, die hier jedoch nicht speziell beschrieben werden.

Unterstützte IMS-abhängige Regionen

Folgende abhängige Regionstypen werden unterstützt:

- MPR
- BMP
- IFP
- JMP (nur 31-Bit-Versionen von Java Virtual Machines (JVMs), 64-Bit-Versionen von JVMs werden nicht unterstützt)
- JBP (nur 31-Bit-Versionen von JVMs, 64-Bit-Versionen von JVMs werden nicht unterstützt)

Sofern in den folgenden Abschnitten nicht anders angegeben, ist das Verhalten der IBM MQ classes for JMS in allen Regionen dasselbe.

Unterstützte Java Virtual Machines

Für IBM MQ classes for JMS ist Java Platform, Standard Edition 7 (Java SE 7) oder höher erforderlich.

Sonstige Einschränkungen

Bei Verwendung der IBM MQ classes for JMS in einer IMS-Umgebung gelten folgende Einschränkungen:

- Verbindungen im Clientmodus werden nicht unterstützt.
- Es werden nur Verbindungen zu Warteschlangenmanagern von IBM MQ 8.0 unter Verwendung des IBM MQ-Messaging-Providers `Normal` oder des `Version 8`-Modus unterstützt.

Das Attribut **PROVIDERVERSION** in der Verbindungsfactory darf entweder nicht angegeben sein oder es muss ein Wert größer oder gleich sieben angegeben werden.

- Die Verwendung von XA-Verbindungsfactorys wie z. B. `com.ibm.mq.jms.MQXAConnectionFactory` wird nicht unterstützt.

Zugehörige Informationen

IBM MQ für IMS definieren

IMS-Adapter für Verwendung mit IBM MQ classes for JMS einrichten

IBM MQ classes for JMS nutzen 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, nachdem Sie den IMS-Adapter konfiguriert haben, die folgende Prozedur aus.

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. DFSJVMEV könnte beispielsweise wie folgt aussehen, wobei die letzte Zeile das Verzeichnis angibt, das die nativen Bibliotheken der IBM MQ classes for JMS enthält:

```
LIBPATH=>
/java/java71_31/J7.1/bin/j9vm:>
/java/java71_31/J7.1/bin:>
/ims13/dbdc/imsjava/classic/lib:>
/ims13/dbdc/imsjava/lib:>
/mqm/V8R0M0/java/lib
```

2. Fügen Sie die IBM MQ classes for JMS zum Klassenpfad der JVM hinzu, die von Ihrer IMS-abhängigen Region verwendet wird, indem Sie die Option `java.class.path` aktualisieren.

   Folgen Sie dabei den Anweisungen in Member DFSJVMMS des IMS PROCLIB-Datasets.

   Sie können beispielsweise Folgendes verwenden, wobei die Zeile in Fettdruck die Aktualisierung angibt:

```
-Djava.class.path=/ims13/dbdc/imsjava/imsutm.jar:/ims13/dbdc/imsjava/imsudb.jar:
/mqm/V8R0M0/java/lib/com.ibm.mq.allclient.jarLIBPATH_SUFFIX=MQ_INSTALLATION_PATH
```

   Anmerkung: Von den vielen verschiedenen JAR-Dateien, die im Verzeichnis mit den IBM MQ classes for JMS verfügbar sind, benötigen Sie nur die Datei `com.ibm.mq.allclient.jar`.

3. Führen Sie einen Stopp und Neustart aller IMS-abhängigen Regionen durch, die die IBM MQ classes for JMS verwenden werden.

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 in einer Anwendung der IBM MQ classes for JMS erstellen und konfigurieren” auf Seite 193.

Beachten Sie, dass diese drei Methoden alle in einer IMS-Umgebung gültig sind.

Zugehörige Informationen

IMS-Adapter einrichten

IBM MQ für IMS definieren

Transaktionsverhalten

Nachrichten, die in einer IMS-Umgebung durch die IBM MQ classes for JMS gesendet und empfangen werden, sind immer der IMS-Arbeitseinheit (UOW) zugeordnet, die in der aktuellen Task aktiv ist.

Diese Arbeitseinheit kann nur abgeschlossen werden, indem die Methoden commit oder rollback für eine Instanz des `com.ibm.ims.dli.tm.Transaction` -Objekts aufgerufen werden oder indem die IMS -Task normal beendet wird. In diesem Fall wird die Arbeitseinheit implizit festgeschrieben. 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. Der Aufruf der folgenden Methoden löst im Sitzungsfall eine der `IllegalStateException`-Ausnahmen aus:

- `javax.jms.Session.commit()`
- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

`IllegalStateRuntimeSession` im Kontext von JMS :

- `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-UOW empfangen, es sei denn, für die Verbindungsfactory, durch die die Sitzung bzw. der JMS-Kontext erstellt wurde, wurde die Eigenschaft SYNCPOINTALLGETS festgelegt.
- 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.

Weitere Informationen hierzu finden Sie unter „Synchronisationspunkte in IMS-Anwendungen” auf Seite 65.

Zur Veranschaulichung dieses Punkts soll der folgende Code dienen, der in einer JMP-Umgebung ausgeführt wird. Der zweite Aufruf von `mp.send()` führt zu einem `JMSException` , da der `messageQueue.getUnique(inputMessage)` -Code dazu führt, dass alle offenen IBM MQ -Verbindungen und Objektkennungen 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 nativer Code, die beide IBM MQ-Funktionalität verwenden, verzahnt sind und wenn die Verbindung zu IBM MQ nicht geschlossen wird, bevor der native Code oder der Java-Code verlassen wird.

Im folgenden Pseudocode ist beispielsweise ein Verbindungshandle zu einem Warteschlangenmanager ursprünglich im Java-Code unter Verwendung der IBM MQ classes for JMS erstellt worden. 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 generell möglich, dass nativer und Java-Code IBM MQ-Verbindungshandles gemeinsam nutzen (z. B. würde das vorherige Beispiel funktionieren, wenn es keinen MQDISC-Aufruf gäbe), aber das bewährte Verfahren besteht darin, alle Verbindungshandles zu schließen, bevor vom Java- zum 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. Der Versuch, einen JMS -Kontext zu erstellen, während ein Benutzername und ein Kennwort angegeben werden, führt dazu, dass ein `JMSRuntimeException` ausgelöst wird.

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 Informationen
Sicherheit unter z/OS einrichten

JMS-API-Einschränkungen

Aus Sicht der JMS-Spezifikation behandeln die IBM MQ classes for JMS einen IMS wie einen Java EE-konformen Anwendungsserver, auf dem immer gerade eine JTA-Transaktion bearbeitet wird.

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 333 beschriebenen Beschränkungen folgende Einschränkungen.

Einschränkungen der klassischen API

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` löst immer eine `JMSException`aus.
- `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` löst immer eine `JMSException`aus.
- 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:

Die IBM MQ classes for Java sind auf der Stufe, die im Lieferumfang von IBM MQ 8.0enthalten ist, funktional stabilisiert. Weitere Informationen finden Sie im Abschnitt Stabilisierung von WebSphere MQ-Klassen für Java.

Die IBM MQ classes for Java werden in IMS nicht unterstützt.

Die IBM MQ classes for Java werden in WebSphere Application Server 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 zwei alternativen APIs, die Java-Anwendungen für den Zugriff auf IBM MQ-Ressourcen verwenden können. Die andere API ist IBM MQ classes for JMS.

Ab IBM MQ 8.0 werden die IBM MQ classes for Java mit Java 7 entwickelt.

Die Laufzeitumgebung von Java 7 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 362.

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 347 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 362.

Zugehörige Informationen

IBM MQ Java-Sprachenschnittstellen

Tracing von IBM MQ classes for Java-Anwendungen

Fehlerbehebung für Java und JMS

IBM MQ classes for JMS verwenden

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: Auch wenn vorhandene Anwendungen, die die IBM MQ classes for Java verwenden, weiterhin vollständig unterstützt werden, sollten neue Anwendungen die IBM MQ classes for JMS verwenden. Neue Funktionen von IBM MQ, z. B. asynchroner Konsum und automatische Verbindungswiederherstellung, stehen nur in den IBM MQ classes for JMS zur Verfügung, nicht in den IBM MQ classes for Java. Weitere Informationen finden Sie unter „Gründe für die Verwendung von IBM MQ classes for JMS" auf Seite 78.

Anmerkung: IBM MQ classes for Java werden funktional auf der mit IBM MQ 8.0 bereitgestellten Stufe eingefroren. Weitere Informationen finden Sie im Abschnitt Stabilisierung von WebSphere MQ-Klassen für Java.

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, UNIX and Linux System Services (USS)

Wenn TLS-Verbindungen Verschlüsselungsmodule verwenden müssen, die gemäß FIPS 140-2 zertifiziert wurden, benötigen Sie den JSSE-FIPS-Provider von IBM Java (IBM JSSEFIPS). 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:
  - 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

Da IBM MQ und WebSphere Application Server auf derselben Maschine installiert werden können, können sich die Hauptversionen des Warteschlangenmanagers und des in WebSphere Application Server ausgelieferten IBM MQ-Ressourcenadapters (RA) unterscheiden. WebSphere Application Server 7.0, in dem ein IBM MQ-RA der Version 7.0.1 enthalten ist, kann beispielsweise auf derselben Maschine installiert sein wie ein Warteschlangenmanager der IBM WebSphere MQ 6.

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 ver-

wendet, 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 8.0wurden Zeichenfolgekonvertierungen in IBM MQ classes for Java durch Aufrufen der Methoden `java.nio.charset.Charset.decode(ByteBuffer)` und `Charset.encode(CharBuffer)` durchgeführt.

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.

Konfiguration

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 127
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.

Installation und Konfiguration von IBM MQ classes for Java

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.

Unterstützung für Standardnachrichtenheader, z. B. Programmable Command Format (PCF), ist in der JAR-Datei `com.ibm.mq.headers.jar` enthalten.

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 339. Weitere Informationen finden Sie unter Using WebSphere MQ Java Interfaces in J2EE/JEE Environments.

Wichtig: Mit Ausnahme der verschiebbaren JAR-Dateien, die in diesem Abschnitt 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. Zudem wird auch das Einschließen der Datei `com.ibm.mq.allcli-`

ent.jar oder der IBM MQ classes for Java innerhalb von Anwendungsarchiven (z. B. Unternehmensanwendungsarchive oder EAR-Dateien) nicht unterstützt.

V 9.0.5 V 9.0.0.3 Die Datei JSON4J.jar und das Paket com.ibm.msg.client.mqlight werden von den IBM MQ classes for Java und IBM MQ classes for JMS nicht benötigt. Ab IBM MQ 9.0.0 Fix Pack 3 und IBM MQ 9.0.5 werden daher folgende Änderungen an der Datei com.ibm.mq.allclient.jar vorgenommen:

- Die Referenz auf die Datei JSON4J.jar wird aus der Klassenpfadanweisung in der Manifestdatei für die Datei com.ibm.mq.allclient.jar entfernt.
- Das Paket com.ibm.msg.client.mqlight ist nicht mehr in der Datei com.ibm.mq.allclient.jar enthalten.

Verschiebbare JAR-Dateien

Innerhalb eines Unternehmens können die folgenden Dateien zu Systemen verschoben werden, die IBM MQ classes for Java-Anwendungen ausführen müssen:

- com.ibm.mq.allclient.jar
- com.ibm.mq.traceControl.jar
- Sicherheitsprovider Bouncy Castle und JAR-Dateien für CMS-Unterstützung

Der Sicherheitsprovider Bouncy Castle und die JAR-Dateien für CMS-Unterstützung sind erforderlich. Weitere Informationen finden Sie im Abschnitt Unterstützung für JREs anderer Anbieter. Die folgenden JAR-Dateien sind erforderlich:

- bcpkix-jdk15on.jar
- bcprov-jdk15on.jar
- V 9.0.0.12 bcutil-jdk15on.jar

Die Datei com.ibm.mq.allclient.jar enthält die IBM MQ classes for JMS, die IBM MQ classes for Java sowie die PCF-und Headerklassen. Wenn Sie diese Datei an eine neue Position verschieben, müssen Sie unbedingt entsprechende Schritte ergreifen, damit diese neue Position mit neuen Fixpacks zu IBM MQ gewartet wird. Sorgen Sie zudem dafür, dass die Verwendung dieser Datei dem IBM Support bekannt ist, wenn Sie einen vorläufigen Fix erhalten.

Sie können die Version der Datei com.ibm.mq.allclient.jar mit folgendem Befehl bestimmen:

```
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:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.0.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.0.0.0
Level:      p000-L140428.1
```

```
Build Type: Production
Location:   file:/C: /Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Die Datei `com.ibm.mq.traceControl.jar` dient zur dynamischen Tracesteuerung für IBM MQ classes for JMS-Anwendungen. Weitere Informationen finden Sie im Abschnitt Trace in einem aktiven Prozess mithilfe von WebSphere MQ-Klassen für Java und WebSphere MQ-Klassen für JMS steuern.

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 344 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
Solaris, Linux, HP-UX, HP-UX, Solaris, Linux, HP-UX, HP-UX, Linux und Solaris	*MQ_INSTALLATION_PATH*/java/lib
IBM i	/QIBM/ProdData/mqm/java/lib
Windows, 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 344 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/
Solaris, Linux, HP-UX, HP-UX, Solaris, Linux, HP-UX, HP-UX, Linux und Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i	/QIBM/ProdData/mqm/java/samples

Tabelle 50. Beispielverzeichnisse (Forts.)	
Plattform	**Directory**
Windows Windows Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS z/OS 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 (oder neuer) Java Runtime Environment (JRE). Eine geeignete JRE wird mit IBM MQ installiert. In Tabelle 51 auf Seite 345 werden die Installationsverzeichnisse dieser JRE aufgeführt. Wenn Sie Java -Programme wie die bereitgestellten Beispiele unter Verwendung dieser JRE ausführen möchten, rufen Sie entweder explizit *JRE_LOCATION*`/bin/java` auf oder fügen Sie *JRE_LOCATION*`/bin` zur PATH-Umgebung (oder funktional entsprechend) für Ihre Plattform hinzu, wobei *JRE_LOCATION* das in Tabelle 51 auf Seite 345angegebene Verzeichnis ist.

Tabelle 51. JRE-Verzeichnisse	
Plattform	**Directory**
AIX AIX AIX	*MQ_INSTALLATION_PATH*/java/jre
Solaris Linux HP-UX Solaris Linux HP-UX HP-UX, Linux und Solaris	*MQ_INSTALLATION_PATH*/java/jre
IBM i IBM i IBM i	/QIBM/ProdData/mqm/java/jre
Windows Windows Windows	*MQ_INSTALLATION_PATH*\java\jre
z/OS z/OS 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 CLASSPATH-Umgebungsvariablen bereitgestellt werden.

Wichtig: Das Festlegen der Java-Option *-Xbootclasspath*, um die IBM MQ classes for Java einzuschließen, wird nicht unterstützt.

In Tabelle 52 auf Seite 346 ist die entsprechende CLASSPATH-Einstellung zu sehen, die verwendet werden soll, damit die jeweilige Plattform IBM MQ classes for Java-Anwendungen (einschließlich der Beispielanwendungen) ausführen kann.

Tabelle 52. CLASSPATH-Einstellung für die Ausführung von IBM MQ classes for Java-Anwendungen (einschließlich der IBM MQ classes for Java-Beispielanwendungen)	
Plattform	**CLASSPATH-Einstellung**
AIX AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Solaris Linux HP-UX Solaris Linux HP-UX HP-UX, Linux und Solaris	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Windows Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V8R0M0/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.

UNIX Unter UNIX können Sie die Umgebungsvariablen mit dem Script **setjmsenv** (bei einer 32-Bit-JVM) bzw. **setjmsenv64** (bei einer 64-Bit-JVM) festlegen.

Linux UNIX Unter UNIX and Linux befinden sich diese Scripts im Verzeichnis *MQ_INSTALLATION_PATH*/java/bin.

IBM i Unter IBM i muss für die Umgebungsvariable QIBM_MULTI_THREAD der Wert Y festgelegt sein. Sie können dann Multithread-Anwendungen genau wie bei der Ausführung von Einzelthread-Anwendungen ausführen. Weitere Informationen hierzu finden Sie im Abschnitt IBM MQ mit Java und JMS konfigurieren.

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 344.

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 347 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	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
HP-UX Linux (Plattformen POWER, x86-64 und zSeries s390x) Solaris (Plattformen x86-64 und SPARC)	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
Linux (x86-Plattform)	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\Java\lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*\Java\lib64 (64-Bit-Bibliotheken)
z/OS z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (32-Bit- und 64-Bit-Bibliotheken)

Anmerkung:

1. Solaris Linux HP-UX AIX Verwenden Sie unter AIX, HP-UX, Linux (Power-Plattform) oder Solaris entweder die 32-Bit- 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.osgi.allclient_*version_number*.jar
Diese JAR-Datei ermöglicht es Anwendungen, sowohl die IBM MQ classes for JMS als auch die IBM MQ classes for Java zu verwenden. Darüber hinaus enthält sie den Code für die Behandlung von PCF-Nachrichten.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
Diese JAR-Datei stellt die Voraussetzungen für `com.ibm.mq.osgi.allclient_`*`version_number`*`.jar` bereit.

Dabei steht *`version_number`* für die Nummer der installierten Version 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 die Bundles `com.ibm.mq.osgi.allclient_8.0.0.0.jar` und `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` für neue Anwendungen. 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 Versionen vor IBM MQ 8.0gilt die Einschränkung für die Verwendung von IBM MQ classes for JMS oder IBM MQ classes for Java .

Es werden ebenfalls neun weitere Bundles im Unterverzeichnis `java/lib/OSGi` der 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 von IBM MQ classes for Java unter z/OS*

Unter z/OS muss die zur Laufzeit verwendete STEPLIB die Bibliotheken IBM MQ SCSQAUTH und SCSQANLE enthalten.

Aus UNIX and Linux System Services können Sie diese Bibliotheken, wie im Beispiel unten gezeigt, über eine Zeile in `.profile` hinzufügen. Dabei muss `thlqual` durch das Dataset-Qualifikationsmerkmal der übergeordneten Ebene ersetzt werden, das Sie bei der Installation von IBM MQ ausgewählt haben:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

In anderen Umgebungen müssen Sie üblicherweise die Start-JCL bearbeiten, damit diese SCSQAUTH bei der STEPLIB-Verkettung einschließt:

```
 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.

V 9.0.3 V 9.0.0.2 Ab IBM MQ 9.0.3 und IBM MQ 9.0.0 Fix Pack 2 wird eine Beispielkonfigurationsdatei namens `mqjava.config` im Unterverzeichnis `bin` des Installationsverzeichnisses von IBM MQ classes for Java bereitgestellt. 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. URLs der folgenden Typen werden unterstützt: `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. Beachten Sie insbesondere, dass die `ChannelDefinitionFile` und `ChannelDefinitionDirectory` in der Zeilengruppe CHANNELS der Clientkonfigurationsdatei nicht verwendet werden. Einzelheiten zur Verwendung der CCDT mit den IBM MQ classes for Java finden Sie im Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for Java verwenden” auf Seite 367.

Tabelle 54. Welche Zeilengruppe der Clientkonfigurationsdatei enthält welches Attribut

Zeilengruppe	Attribut
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	Put1DefaultAlwaysSync
Zeilengruppe CHANNELS der Clientkonfigurationsdatei	PasswordProtection

Tabelle 54. Welche Zeilengruppe der Clientkonfigurationsdatei enthält welches Attribut (Forts.)	
Zeilengruppe	**Attribut**
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 Konfiguration des IBM MQ MQI clients finden Sie im Abschnitt Client mithilfe einer Konfigurationsdatei konfigurieren.

Zugehörige Informationen

Tracefunktion für WebSphere MQ-Klassen für Java-Anwendungen

Zeilengruppe für den Java-Standardumgebungstrace

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:

- Bis IBM MQ classes for Java für IBM MQ 9.0.0 Fix Pack 1 wird der Trace in eine Datei namens `mqjms_%PID%.trc` geschrieben.
- V 9.0.0.2 Ab IBM MQ 9.0.0 Fix Pack 2 wird der Trace in eine Datei mit dem Namen `mqjava_%PID%.trc` geschrieben, wenn die Anwendung die IBM MQ classes for Java aus der JAR-Datei `com.ibm.mq.jar` geladen hat.
- V 9.0.0.2 Ab IBM MQ 9.0.0 Fix Pack 2 wird der Trace in eine Datei mit dem Namen `mqjavaclient_%PID%.trc` geschrieben, wenn die Anwendung die IBM MQ classes for Java aus der verschiebbaren JAR-Datei `com.ibm.mq.allclient.jar` geladen hat.
- V 9.0.0.10 Ab IBM MQ 9.0.0 Fix Pack 10 wird der Trace in eine Datei mit dem Namen `mqjava_%PID%.cl%u.trc` geschrieben, wenn die Anwendung die IBM MQ classes for Java aus der JAR-Datei `com.ibm.mq.jar` geladen hat.

- V 9.0.0.10 Ab IBM MQ 9.0.0 Fix Pack 10 wird der Trace in eine Datei mit dem Namen `mqjavaclient_%PID%.cl%u.trc` geschrieben, wenn die Anwendung die IBM MQ classes for Java aus der verschiebbaren JAR-Datei `com.ibm.mq.allclient.jar` geladen hat.

Dabei steht *%PID%* für die Prozess-ID der Anwendung, für die ein Trace erstellt wird, und *%u* für die eindeutige Nummer, durch die Dateien zwischen Threads unterschieden werden, die das Trace unter verschiedenen Java-Klassenladeprogrammen ausführen.

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 „Dies wird für IBM MQ classes for Java installiert" auf Seite 342 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.

Details hierzu finden Sie unter „Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter Multiplatforms” auf Seite 1135.

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 IBM MQ classes for Java verfügbar sein, wenn die Ausführung unter Java Security Managererfolgt.

Ein IBM MQ-Warteschlangenmanager kann Benachrichtigungen an verbundene Clients senden, die einen kontrollierten Abschluss von Dialogen (Verbindungskennungen) anfordern, z. B. wenn der Warteschlangenmanager in den Quiescemodus versetzt wird. Wenn ein Thread innerhalb eines Java-Clients eine dieser Benachrichtigungen empfängt und gleichzeitig ein anderer Thread im Client einen neuen Datenaustausch anfordert, kann ein Deadlock auftreten, da beide Threads Zugriff auf die interne "connectionsLock" im Objekt RemoteConnectionSpecification benötigen.

V 9.0.5 V 9.0.0.3 Ab IBM MQ 9.0.0 Fix Pack 3 und IBM MQ 9.0.5 ist der Deadlock im IBM MQ Java -Client behoben. 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";
```

```
V 9.0.0.3 // 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.

IBM MQ classes for Java-Anwendungen unter CICS Transaction Server ausführen
Eine IBM MQ classes for Java-Anwendung kann unter CICS Transaction Server als Transaktion ausgeführt werden.

Gehen Sie wie folgt vor, um eine IBM MQ classes for Java-Anwendung unter CICS Transaction Server als Transaktion für z/OS auszuführen:

1. Definieren Sie die Anwendung und Transaktion für CICS, indem Sie die bereitgestellte CEDA-Transaktion verwenden.
2. Stellen Sie sicher, dass der IBM MQ CICS-Adapter in Ihrem CICS-System installiert ist. z/OS (Weitere Informationen finden Sie im Abschnitt IBM MQ mit CICS verwenden.
3. Stellen Sie sicher, dass die in CICS angegebene JVM-Umgebung die entsprechenden CLASSPATH- und LIBPATH-Einträge enthält.
4. Rufen Sie die Transaktion durch Verwendung eines normalen Prozesses auf.

Weitere Informationen zur Ausführung von CICS Java-Transaktionen finden Sie in Ihrer CICS-Systemdokumentation.

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 1135beschrieben. 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 337.

   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 345 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 347).

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 im Abschnitt „Anwendungen der IBM MQ classes for Java schreiben” auf Seite 362. Die Beispiele könnten sich in zukünftigen Releases von IBM MQ Java ändern.

In Tabelle 55 auf Seite 359 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**
UNIX UNIX 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 359 zeigt die Gruppen von Beispielanwendungen, die mit IBM MQ classes for Java bereitgestellt werden.

Tabelle 56. IBM MQ classes for Java-Beispielanwendungen	
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	Zeigt die Verwendung der API für Nachrichteneigenschaften, die in IBM WebSphere MQ 7.0 eingeführt wurde.
MQPubSubApiSample.java	Zeigt die Verwendung der Publish/Subscribe-API, die in IBM WebSphere MQ 7.0 eingeführt wurde.
MQSample.java	Einfaches Programm zur Demonstration des Einreihens von Nachrichten in eine Warteschlange und des Abrufens aus einer Warteschlange.
MQSampleMessageManager.java	Dienstprogrammklasse für die Nachrichtenbehandlung in den Java-Basisbeispielen von IBM MQ>.
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 359 zeigt den Speicherort des Scripts auf den einzelnen Plattformen:

Tabelle 57. Speicherort des Scripts `runjms`	
Plattform	**Directory**
UNIX UNIX Linux Linux	*MQ_INSTALLATION_PATH*/java/bin/runjms
Windows Windows	*MQ_INSTALLATION_PATH*\java\bin\runjms.bat

Tabelle 57. Speicherort des Scripts `runjms` (Forts.)	
Plattform	**Directory**

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 das Script `runjms` zum Aufrufen einer Beispielanwendung zu verwenden:

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 82
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.

IBM MQ classes for Java-Probleme lösen

Führen Sie zuerst das Installationsprüfprogramm aus. Sie müssen möglicherweise ebenfalls die Tracefunktion verwenden.

Wenn ein Programm nicht erfolgreich vollständig ausgeführt wird, verwenden Sie das Installationsprüfprogramm und befolgen Sie die Hinweise der Diagnosenachrichten. Dieses Programm wird unter „IBM MQ classes for Java-Installation überprüfen” auf Seite 357 beschrieben.

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 *com.ibm.mq.commonservices*-Eigenschaftendatei (weitere Informationen erhalten Sie unter 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 347).

Weitere Informationen zur Verwendung der Tracefunktion finden Sie unter Trace für Anwendungen erstellen, die die IBM MQ classes for Java verwenden.

z/OS V 9.0.4 MQ Adv. VUE Java- und JMS-Clientkonnektivität zu Stapelanwendungen unter z/OS

Eine Anwendung von JMS oder IBM MQ classes for Java unter z/OS kann über eine Clientverbindung eine Verbindung zu einem Warteschlangenmanager unter z/OS, der das Attribut **ADVCAP** (ENABLED) besitzt, herstellen.

Der Wert **ADVCAP** (ENABLED) gilt nur für einen z/OS -Warteschlangenmanager, der als IBM MQ Advanced for z/OS, Value Unit Edition lizenziert ist (siehe IBM MQ -Produkt-IDs und -Exportinformationen), bei dessen Ausführung **QMGRPROD** auf ADVANCEDVUEgesetzt ist.

Im Abschnitt DISPLAY QMGR finden Sie weitere Informationen zu **ADVCAP** und im Abschnitt START QMGR weitere Informationen zu **QMGRPROD**.

Beachten Sie, dass Stapelbetrieb die einzige unterstützte Umgebung ist; es gibt keine Unterstützung für JMS for CICS oder JMS for IMS.

Eine IBM MQ classes for JMS-oder IBM MQ classes for Java-Anwendung unter z/OS kann über eine Clientmodusverbindung keine Verbindung zu einem Warteschlangenmanager herstellen, der nicht unter z/OSausgeführt wird, oder zu einem WS-Manager ohne die Option **ADVCAP** (ENABLED) .

Wenn eine JMS-Anwendung versucht, eine Verbindung im Clientmodus herzustellen, aber nicht dazu berechtigt ist, wird die Ausnahmebedingungsnachricht JMSFMQ0005 ausgegeben.

Wenn eine Anwendung von IBM MQ classes for Java unter z/OS versucht, eine Verbindung im Clientmodus herzustellen, aber nicht dazu berechtigt ist, wird MQRC_ENVIRONMENT_ERROR zurückgegeben.

Unterstützung von Advanced Message Security (AMS)

V 9.0.5

Ab IBM MQ 9.0.5 können IBM MQ classes for JMS- oder IBM MQ classes for Java-Clientanwendungen AMS verwenden, wenn Sie eine Verbindung zu IBM MQ Advanced for z/OS, Value Unit Edition-Warteschlangenmanagern auf fernen z/OS-Systemen herstellen.

Ein neuer Keystore-Typ, `jceracfks`, wird in `keystore.conf` nur unter z/OS unterstützt, wobei Folgendes gilt:

- 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
```

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 '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.
- Die automatische Wiederherstellung einer Clientverbindung 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 763.

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 101

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, auf Port 1414 eine Verbindung zu einem IBM MQ-Listener herzustellen. 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 jedoch diese Information nicht verfügbar ist, wird der Text `WebSphere 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 '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 programmier-

technischen 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 sie ein `MQQueueManager-Objekt` anhand eines Konstruktors erstellen, der ein URL-Objekt als Parameter nimmt. 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 744.

Wenn Ihre Anwendung Verbindungspooling nutzt, erhalten Sie weitere Informationen unter „Standardverbindungspool in IBM MQ classes for Java steuern" auf Seite 389.

- 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 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 382. 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 385.

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. Beispiel:

```
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. Beispiel:

```
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 Konstruktoren. 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. Beispiel:

```
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 im Abschnitt „Klasse MQMessage” auf Seite 647.

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.

Sie finden allgemeine Informationen zur Funktion 'Vorauslesen' im .

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. Wenn Sie eine interne Nachrichten-Header-Klasse wie MQRFH, MQRFH2, MQCIH, MQDEAD, MQIIH oder MQXMIT verwenden möchten, stellen Sie vor der Weitergabe der Nachricht an die Klasse sicher, dass der Cursor sich an der richtigen Position befindet.

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 24.

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 ac-

cessQueue() 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-Klassen für 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 Kennungen durch mehrere Threads beschränkt wird. IBM MQ classes for Java lockern diese Beschränkung. Dort kann ein Warteschlangenmanagerobjekt (und seine zugehörige Warteschlange sowie seine Themen- 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.

In IBM MQ classes for Java gibt es keine direkte funktionale Entsprechung zu dem IBM MQ-Kanal. 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 385 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 385 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 exitPath-Zeilengruppe 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 , HP-UX, Linux und Solaris	`/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 367.

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 WebSphere MQ 7.0 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.

Externe Exitklassen verwenden

In Versionen vor IBM WebSphere MQ 7.0 wurden drei Klassen für die Verwendung von Kanalexits bereitgestellt, die in anderen Sprachen als Java geschrieben wurden:

- MQExternalSecurityExit zur Implementierung der MQSecurityExit-Schnittstelle
- MQExternalSendExit zur Implementierung der MQSendExit-Schnittstelle
- MQExternalReceiveExit zur Implementierung der MQReceiveExit-Schnittstelle

Diese Klassen können nach wie vor verwendet werden, die neue Methode wird jedoch bevorzugt.

Um einen Sicherheitsexit verwenden zu können, der nicht in Java geschrieben wurde, musste eine Anwendung zuerst ein MQExternalSecurityExit-Objekt erstellen. Die Anwendung gab als Parameter im MQExternalSecurityExit-Konstruktor den Namen der Bibliothek mit dem Sicherheitsexit, den Namen des Einstiegspunkts für den Sicherheitsexit und die Benutzerdaten an, die beim Aufruf des Sicherheitsexits an diesen übergeben werden sollten. Kanalexitprogramme, die nicht in Java geschrieben wurden, wurden in dem in Tabelle 58 auf Seite 385 genannten Verzeichnis gespeichert.

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, Definitionen bestehen, wovon jede der Name einer Java-Klasse oder eines C-Programms im Format `Bibliothek(Funktion)` sein kann.

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));
:
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.addConnection-PoolToken()-Methoden. Wenn Sie den Pool inaktivieren möchten, verwenden Sie MQEnvironment.remove-ConnectionPoolToken().

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 391).

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 MQPoolTokens 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 MQSimpleConnection-Manager.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 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 den 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. Beispiel:

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`.

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 müssen, 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.

Es folgt ein Beispiel für einen **make**-Befehl:

```
make db2
```

Beachten Sie dabei Folgendes:

- 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 lautet 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.

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 Verwalten. Dort finden Sie Informationen zur Erstellung einer XAResourceManager-Zeilengruppe. Jeder SwitchFile-Eintrag in einer neuen XAResourceManager-Zeilengruppe muss genau wie zuvor für eine Db2- oder Oracle-Datenbank beschrieben angegeben werden. 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 367.

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 401 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 TLS mit dem CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA anfordert:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Obwohl der Kanal über die CipherSpec 'TLS_RSA_WITH_AES_128_CBC_SHA' verfügt, muss die Java-Anwendung die Cipher-Suite 'SSL_RSA_WITH_AES_128_CBC_SHA' angeben. Im Abschnitt „TLS CipherSpecs und Cipher-Suites in IBM MQ classes for Java” auf Seite 401 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 hierzu finden Sie im Abschnitt „Definitionstabelle für den Clientkanal mit IBM MQ classes for Java verwenden” auf Seite 367.

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 Informationen
Federal Information Processing Standards (FIPS) für UNIX, 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";
```

kann die Verbindung nur erfolgreich hergestellt werden, wenn der Warteschlangenmanager ein Zertifikat vorlegt, dessen allgemeiner Name mit QMGR. beginnt. Außerdem muss es mindestens zwei Namen der Organisationseinheit enthalten, wovon der erste 'IBM' und der zweite 'WebSphere' lauten 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 Klasse 'java.security.cert.CertStore' 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. Beispiel:

```
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. Beispiel:

```
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 401.

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.

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-Sui-

tes. Weitere Informationen zur Konfiguration Ihrer Anwendung für eine Oracle Java-JRE finden Sie im Abschnitt „Anwendung für Oracle IBM Java- oder Oracle Java-Cipher-Suite-Zuordnungen konfigurieren" auf Seite 427.

In der Tabelle ist auch das für die Kommunikation verwendete Protokoll angegeben sowie, ob die Cipher-Suite dem FIPS 140-2-Standard entspricht.

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.

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	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLSv1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECD-HE_RSA_AES_256_CBC_SHA384	SSL_ECD-HE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLSv1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_3DES_EDE_CBC_SHA „1" auf Seite 426	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLSv1	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLSv1	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLSv1.2	ja

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**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	TLSv1	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)

CipherSpec	**Äquivalente Cipher-Suite (IBM JRE)**	**Äquivalente Cipher-Suite (Oracle JRE)**	**Protokoll**	**FIPS 140-2-konform**
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLSv1.2	nein

Tabelle 59. Von IBM MQ unterstützte CipherSpecs und äquivalente Cipher-Suites (Forts.)				
CipherSpec	**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	TLSv1.2	nein

Anmerkungen:

1. 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.

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

Sie können konfigurieren, ob Ihre Anwendung die Standardzuordnungen zwischen IBM Java-Cipher-Suites und IBM MQ-CipherSpecs oder die Zuordnungen zwischen Oracle-Cipher-Suites und IBM MQ-CipherSpecs verwendet. In jedem Fall können Sie TLS-Cipher-Suites verwenden, unabhängig davon, ob Ihre Anwendung eine IBM JRE oder eine Oracle-JRE verwendet. Welche Zuordnungen verwendet werden, steuert die Java-Systemeigenschaft `com.ibm.mq.cfg.useIBMCipherMappings`. Für diese Eigenschaft sind folgende Werte möglich:

true
Die Zuordnungen zwischen IBM Java-Cipher-Suites und IBM MQ-CipherSpecs werden verwendet.

Dies ist der Standardwert.

false
Die Zuordnungen zwischen Oracle-Cipher-Suites und IBM MQ-CipherSpecs werden verwendet.

Weitere Informationen zur Verwendung von IBM MQ Java und TLS-Verschlüsselungen finden Sie im MQdev-Blogbeitrag MQ Java, TLS-Verschlüsselungen, Nicht-IBM JREs & APARs IT06775, IV66840, IT09423, IT10837.

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 427	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. 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 unter „IBM MQ classes for Java-Bibliotheken” auf Seite 347.

Zugehörige Informationen

Traceerstellung für Anwendungen für IBM MQ classes for Java

Traceerstellung für IBM MQ-Ressourcenadapter ausführen

z/OS V 9.0.4 MQ Adv. VUE *Java- und JMS-Clientkonnektivität zu Stapelanwendungen unter z/OS*

Eine JMS-oder IBM MQ classes for Java-Anwendung unter z/OS kann über eine Clientverbindung eine Verbindung zu einem Warteschlangenmanager unter z/OSherstellen, der über das Attribut **ADVCAP**(ENABLED) verfügt.

Der Wert **ADVCAP**(ENABLED) gilt nur für einen z/OS -Warteschlangenmanager, der als IBM MQ Advanced for z/OS, Value Unit Edition lizenziert ist (siehe IBM MQ -Produkt-IDs und -Exportinformationen), bei dessen Ausführung **QMGRPROD** auf ADVANCEDVUEgesetzt ist.

Im Abschnitt DISPLAY QMGR finden Sie weitere Informationen zu **ADVCAP** und im Abschnitt START QMGR weitere Informationen zu **QMGRPROD**.

Beachten Sie, dass Stapelbetrieb die einzige unterstützte Umgebung ist; es gibt keine Unterstützung für JMS for CICS oder JMS for IMS.

Eine IBM MQ classes for JMS-oder IBM MQ classes for Java-Anwendung unter z/OS kann über eine Clientmodusverbindung keine Verbindung zu einem Warteschlangenmanager herstellen, der nicht unter z/OSausgeführt wird, oder zu einem WS-Manager ohne die Option **ADVCAP**(ENABLED) .

Wenn eine JMS-Anwendung versucht, eine Verbindung im Clientmodus herzustellen, aber nicht dazu berechtigt ist, wird die Ausnahmebedingungsnachricht JMSFMQ0005 ausgegeben.

Wenn eine Anwendung von IBM MQ classes for Java unter z/OS versucht, eine Verbindung im Clientmodus herzustellen, aber nicht dazu berechtigt ist, wird MQRC_ENVIRONMENT_ERROR zurückgegeben.

Unterstützung von Advanced Message Security (AMS)

V 9.0.5

Ab IBM MQ 9.0.5 können IBM MQ classes for JMS- oder IBM MQ classes for Java-Clientanwendungen AMS verwenden, wenn Sie eine Verbindung zu IBM MQ Advanced for z/OS, Value Unit Edition-Warteschlangenmanagern auf fernen z/OS-Systemen herstellen.

Ein neuer Keystore-Typ, `jceracfks`, wird in `keystore.conf` nur unter z/OS unterstützt, wobei Folgendes gilt:

- 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
```

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 430 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 430); 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. Das Verhalten ist so, als ob ein Windows-, UNIX -oder Linux IBM MQ -Warteschlangenmanager verwendet wird, mit Ausnahme der folgenden geringfügigen Einschränkungen und Variationen.

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 Wiederherstellung einer Clientverbindung wird von den IBM MQ classes for Java nicht 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.

Diverse Unterschiede zwischen IBM MQ classes for Java unter z/OS und auf anderen Plattformen
IBM MQ for z/OS verhält sich gegenüber IBM MQ auf anderen Plattformen in manchen Bereichen unterschiedlich.

BackoutCount
Ein z/OS-Warteschlangenmanager gibt für BackoutCount höchstens den Wert 255 zurück. Dies gilt selbst dann, wenn die Nachricht häufiger als 255 Mal zurückgesetzt wurde.

Standardpräfix für dynamische Warteschlangen
Wenn unter Verwendung einer Bindungsverbindung eine Verbindung mit einem z/OS-Warteschlangenmanager besteht, lautet das Standardpräfix der dynamischen Warteschlange CSQ.*. Andernfalls lautet das Standardpräfix der dynamischen Warteschlange AMQ.*.

MQQueueManager-Konstruktor
Die Clientverbindung wird unter z/OS nicht unterstützt. Der Versuch, mit Clientoptionen eine Verbindung herzustellen, führt zu einer MQ-Ausnahmebedingung (MQException) mit MQCC_FAILED und MQRC_ENVIRONMENT_ERROR.

Der MQQueueManager-Konstruktor kann auch mit dem Fehler MQRC_CHAR_CONVERSION_ERROR fehlschlagen (falls er die Konvertierung zwischen den Codepages IBM-1047 und ISO8859-1 nicht initialisieren kann). Denkbar ist auch der Fehler MQRC_UCS2_CONVERSION_ERROR (falls er die Konvertierung zwischen der Codepage des Warteschlangenmanagers und Unicode nicht initialisieren kann). Wenn Ihre Anwendung mit einem dieser Ursachencodes fehlschlägt, stellen Sie sicher, dass die

Komponente mit den Ressourcen für Landessprachen (National Language Resources) der Language Environment installiert ist. Sorgen Sie außerdem dafür, dass die richtigen Konvertierungstabellen verfügbar sind.

Die Konvertierungstabellen für Unicode werden als Teil der Zusatzfunktion C/C++ für z/OS installiert. Weitere Informationen zur Aktivierung von UCS-2-Konvertierungen finden Sie im Handbuch *z/OS C/C++ Programming Guide* (SC09-4765).

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 unter UNIX, Linux oder Windows im Bindungsmodus verwendet werden. Andernfalls schlägt sie mit MQRC_ENVIRONMENT_ERROR fehl.

Weitere Informationen finden Sie unter „JTA/JDBC-Koordination unter Verwendung von IBM MQ classes for Java” auf Seite 392.

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 MQDistribution-

List. 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 tat-

sächlichen 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 „Diverse Unterschiede zwischen IBM MQ classes for Java unter z/OS und auf anderen Plattformen" auf Seite 431 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

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 implementiert die Schnittstellen von JCA 1.7 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 auf einem Anwendungsserver ausgeführt werden, auf dem der Ressourcenadapter implementiert wurde. Sie ermöglichen Anwendungen, die auf einem Anwendungsserver ausgeführt werden, bei ihrem Zugriff auf die Ressourcen eines IBM MQ-Warteschlangenmanagers die Verwendung der API der IBM MQ classes for Java.

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 339.

Zu verwendende Version des Ressourcenadapters

Welche Version des Ressourcenadapters zu verwenden ist, hängt von der Java EE-Version (Java Platform, Enterprise Edition) des Anwendungsservers ab, die Sie verwenden.

Java EE 7

Der Ressourcenadapter von IBM MQ 8.0 und IBM MQ 9.0 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 436).

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. Mithilfe des Ressourcenadapters aus IBM MQ 8.0 oder höher kann sich eine Anwendung entweder unter Verwendung des Transports BINDINGS oder CLIENT mit einem Warteschlangenmanager von IBM WebSphere MQ 7.0 oder höher oder bei ausschließlicher Verwendung des CLIENT-Transports mit einem Warteschlangenmanager von IBM WebSphere MQ 6.0 verbinden.

Wichtig: Der Ressourcenadapter von IBM MQ 8.0 oder höher kann nur auf einem Anwendungsserver bereitgestellt werden, der JMS 2.0 unterstützt.

Java EE 5 und Java EE 6

Der Ressourcenadapter von IBM WebSphere MQ 7.5 unterstützt Java EE Connector Architecture (JCA) v1.5 und stellt Unterstützung für JMS 1.1 bereit. Um eine vollständige Integration mit WebSphere Application Server Libertyzu ermöglichen, wird der IBM WebSphere MQ 7.5 -Ressourcenadapter von IBM WebSphere MQ 7.5.0 Fix Pack 2auf APAR IC92914 aktualisiert. Dieser Ressourcenadapter behält die vollständige Kompatibilität mit anderen Anwendungsservern von Java EE 5 und höher bei (siehe WebSphere MQ -Ressourcenadapter 7.1 und höher).

Ressourcenadapter mit WebSphere Application Server traditional verwenden

V 9.0.0 Der Ressourcenadapter von IBM MQ 9.0 ist in WebSphere Application Server traditional 9.0 vorinstalliert. Deshalb muss kein neuer Ressourcenadapter installiert werden.

Anmerkung: Ein Ressourcenadapter von IBM MQ 9.0 kann im Transportmodus CLIENT oder BINDINGS eine Verbindung zu jedem in Betrieb befindlichen IBM MQ-Warteschlangenmanager herstellen.

Ressourcenadapter mit WebSphere Application Server Liberty verwenden

Wenn Sie von WebSphere Application Server Liberty aus eine Verbindung zu IBM MQ herstellen möchten, müssen Sie den IBM MQ-Ressourcenadapter verwenden. Da Liberty den IBM MQ-Ressourcenadapter nicht enthält, müssen Sie ihn separat aus Fix Central beziehen. Welche Version des Ressourcenadapters zu verwenden ist, hängt von der Java EE-Version des Anwendungsservers ab.

Weitere Informationen zum Herunterladen und Installieren des Ressourcenadapters finden Sie im Abschnitt „Ressourcenadapter in Liberty installieren” auf Seite 443.

Zugehörige Konzepte

„Ressourcenadapter für eingehende Kommunikation konfigurieren” auf Seite 450
Definieren Sie die Eigenschaften eines oder mehrerer ActivationSpec-Objekte, um die eingehende Kommunikation zu konfigurieren.

„Ressourcenadapter für abgehende Kommunikation konfigurieren” auf Seite 470
Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

„IBM MQ classes for JMS verwenden” auf Seite 77
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) stellen den JMS-Provider dar, der in IBM MQ bereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die IBM MQ classes for JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

„IBM MQ classes for Java verwenden” auf Seite 337

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 Informationen

Anwendungsserver für die Verwendung der neuesten Wartungsstufe des Ressourcenadapters konfigurieren

Problembestimmung für den IBM MQ-Ressourcenadapter

Referenzinformationen für WebSphere Application Server Version 8.5.5

IBM MQ-Ressourcenadapter warten

JMS-Anwendungen zur Verwendung des IBM MQ-Messaging-Providers im Liberty-Profil implementieren

Unterstützungserklärung für IBM MQ-Ressourcenadapter

Der mit IBM MQ 8.0 oder höher ausgelieferte Ressourcenadapter implementiert die JMS 2.0-Spezifikation. 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 der zertifizierten Anwendungsserver wird auf der Oracle-Website gepflegt.

Implementierung in WebSphere Application Server Liberty

Da WebSphere Liberty 8.5.5 Fix Pack 6 oder höher und WebSphere Application Server Liberty 9.0 oder höher für Java EE 7 zertifizierte Anwendungsserver sind, kann der Ressourcenadapter von IBM MQ 8.0 oder höher darin bereitgestellt werden.

WebSphere Application Server Liberty verfügt über die Funktion wmqJmsClient-1.1, die die Arbeit mit JMS 1.1-Ressourcenadaptern ermöglicht, und über die Funktion wmqJmsClient-2.0, die die Arbeit mit JMS 2.0Ressourcenadaptern ermöglicht. Der Ressourcenadapter von IBM MQ 8.0 oder höher muss mit der Funktion wmqJmsClient-2.0 implementiert werden.

Informationen zu dieser Konfiguration finden Sie im Szenario Connecting WebSphere Application Server Liberty profile to IBM MQ.

Implementierung in WebSphere Application Server traditional

WebSphere Application Server traditional 9.0 wird mit einem bereits installierten IBM MQ 9.0-Ressourcenadapter geliefert. Der Ressourcenadapter kann eine Verbindung zu allen Warteschlangenmanagern herstellen, die in einer unterstützten Version von IBM MQ oder IBM WebSphere MQ ausgeführt werden. Weitere Informationen finden Sie im Abschnitt „Konnektivität zu IBM MQ 8.0 -und 9.0 -Warteschlangenmanagern” auf Seite 437.

Der Ressourcenadapter von IBM MQ 9.0 kann nicht in früheren Versionen von WebSphere Application Server bereitgestellt werden, weil diese Versionen nicht Java EE 7-zertifiziert sind.

Ressourcenadapter mit anderen Anwendungsservern verwenden

Bei allen anderen Java EE 7-konformen Anwendungsservern können Probleme, die nach dem erfolgreichen Abschluss des Installationsprüftests (IVT) für den IBM MQ-Ressourcenadapter auftreten, an IBM berichtet werden, damit der IBM MQ-Produkttrace und sonstige IBM MQ-Diagnoseinformationen untersucht werden können. Wenn der IVT für den IBM MQ-Ressourcenadapter nicht erfolgreich ausgeführt werden kann, werden auftretende Probleme wahrscheinlich durch eine falsche Implementierung oder falsche Ressourcendefinitionen verursacht, die für den jeweiligen Anwendungsserver spezifisch sind. Die Probleme müssen mit Hilfe der Anwendungsserverdokumentation und/oder der Unterstützungsorganisation für den betreffenden 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 IBM MQ 9.0-Client unterstützt wird. Diese JREs werden in den System-

voraussetzungen für IBM MQ aufgelistet. (Wählen Sie den Betriebssystem- oder Komponentenbericht aus, den Sie einsehen möchten, und folgen Sie dann dem **Java**-Link, auf der Registerkarte **Unterstützte Software**.)

Konnektivität zu IBM MQ 8.0 -und 9.0 -Warteschlangenmanagern

Der vollständige Funktionsumfang von JMS 2.0 ist verfügbar, wenn eine Verbindung zu einem Warteschlangenmanager der IBM MQ 8.0 oder 9.0 über den IBM MQ 9.0 -Ressourcenadapter hergestellt wird, der in einem zertifizierten Java EE 7 -Anwendungsserver implementiert wurde. Um diese Funktionalität nutzen zu können, muss der Ressourcenadapter die Verbindung zum Warteschlangenmanager im normalen Modus des IBM MQ-Messaging-Providers herstellen. Weitere Informationen finden Sie unter **PROVIDERVERSION** nicht angegeben.

Konnektivität zu Warteschlangenmanagern von IBM WebSphere MQ 7.5 oder früher

Es besteht Unterstützung für die Implementierung des IBM MQ 9.0-Ressourcenadapters in einem Java EE 7-zertifizierten Anwendungsserver, der JMS 2.0 unterstützt, und für eine Verbindung dieses Ressourcenadapters zu einem Warteschlangenmanager, der IBM WebSphere MQ 7.5 oder früher ausführt. Die verfügbare Funktionalität wird durch die Leistungsmerkmale des Warteschlangenmanagers begrenzt. Weitere Informationen finden Sie im Abschnitt ../com.ibm.mq.con.doc/q123360_.dita#q123360_.

MQ-Erweiterungen

Mit der JMS 2.0-Spezifikation werden Änderungen bestimmter Verhaltensweisen eingeführt. Da IBM MQ 8.0 und 9.0 diese Spezifikation implementieren, gibt es Verhaltensänderungen zwischen diesen beiden Releases und früheren Versionen von IBM WebSphere MQ. IBM MQ 8.0 und 9.0 IBM MQ classes for JMS enthalten Unterstützung für die Java -Systemeigenschaft `com.ibm.mq.jms.SupportMQExtensions` , die, wenn sie auf TRUEgesetzt ist, bewirkt, dass diese beiden Releases diese Verhaltensweisen auf die von IBM WebSphere MQ 7.5 oder früher zurücksetzen. Der Standardwert für die Eigenschaft ist FALSE.

Der IBM MQ 9.0 -Ressourcenadapter 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 321.

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 Application Server 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. In der IBM Technote Using WebSphere MQ Java Interfaces in J2EE/JEE Environments finden Sie Details und Hinweise zur Verwendung der IBM MQ classes for Java in Java EE.
- Bei Ausführung im WebSphere Application Server 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 in IBM WebSphere MQ 7.0.1 eingeführte Funktionalität zur Verbindungswiederholung 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.
- Eine erneute Authentifizierung von JMS-Verbindungen, wie in Abschnitt 9.1.9 der Spezifikation von Java EE Connector Architecture Version 1.7 definiert, 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 Informationen

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

Federal Information Processing Standards (FIPS) für UNIX, Linux und 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 oder IBM WebSphere MQ nicht mit WebSphere Application Server 6.0 oder 6.1.

WebSphere Application Server 7.0 und WebSphere Application Server 8.0 enthalten eine Version des Ressourcenadapters von IBM WebSphere MQ 7.0.

WebSphere Application Server 8.5.5 enthält eine Version des Ressourcenadapters von IBM WebSphere MQ 7.1.

V 9.0.0 WebSphere Application Server traditional 9.0 enthält eine Version des IBM MQ 9.0-Ressourcenadapters. Der Ressourcenadapter von IBM MQ 9.0 kann nicht in früheren Versionen von WebSphere Application Server bereitgestellt werden, weil diese Versionen nicht Java EE 7-zertifiziert sind.

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 in WebSphere Application Server Liberty WebSphere Application Server 8.5.5 Fix Pack 2 oder höher installiert werden. Verwenden Sie dazu entweder die Funktion wmqJmsClient-1.1 oder wmqJmsClient-2.0, abhängig davon, 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.
- V 9.0.0 Vor IBM MQ 9.0 war das Feature Advanced Message Security (AMS) nicht in der Messaging-Funktion von IBM MQ Liberty enthalten. Allerdings wird AMS mit einem Ressourcenadapter von IBM MQ 9.0 unterstützt.

Einschränkungen bei Verwendung der Liberty-Funktionen

Mit Liberty WebSphere Application Server 8.5.5 Fix Pack 2 bis einschließlich WebSphere Application Server 8.5.5 Fix Pack 5 war nur die Funktion wmqJmsClient-1.1 verfügbar und konnte nur JMS 1.1 verwendet werden.Liberty Mit WebSphere Application Server 8.5.5 Fix Pack 6 wurde die Funktion wmqJmsClient-2.0 hinzugefügt, damit JMS 2.0 verwendet werden konnte.

Welche Funktion Sie verwenden müssen, hängt jedoch von der Version des Ressourcenadapters ab, die Sie verwenden:

- Der Ressourcenadapter von IBM WebSphere MQ 7.5.0 Fix Pack 6 und ab IBM WebSphere MQ 7.5 kann nur mit Funktion wmqJmsClient-1.1 verwendet werden.
- 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.

Einschränkungen bei Verwendung der generischen JCA-Unterstützung

Wenn Sie die generische JCA-Unterstützung verwenden, gelten folgende Einschränkungen:

- Sie müssen bei Verwendung der generischen JCA-Unterstützung die JMS-Version angeben:
  - JMS 1.1 und JCA 1.6 dürfen nur mit dem Ressourcenadapter von IBM WebSphere MQ 7.5.0 Fix Pack 6 und ab IBM WebSphere MQ 7.5 verwendet werden.
  - 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. Unter z/OS muss der IBM MQ-Ressourcenadapter mit der Funktion wmqJmsClient-1.1 oder wmqJmsClient-2.0 ausgeführt werden.
- Die Position des Ressourcenadapters wird mit dem folgenden XML-Element angegeben:

```
<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:

```
<!-- 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 classloaderProviderRef zurück zum Ressourcenadapter über die ID `mqJms`; dies dient dazu, das Laden IBM MQ-spezifischer Klassen zu erlauben.

Einschränkungen bei der Traceerstellung mit generischer JCA -Unterstützung

Traceerstellung und Protokollierung sind nicht in das Liberty-Tracesystem integriert. Stattdessen muss der IBM MQ -Ressourcenadaptertrace entweder mit Java -Systemeigenschaften oder mit einer IBM MQ classes for JMS -Konfigurationsdatei aktiviert werden, wie in Traceerstellung für IBM MQ -Klassen für JMS-Anwendungenbeschrieben. Details zum Festlegen von Java -Systemeigenschaften in Libertyfinden Sie in der WebSphere Application Server 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.

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) mit dem Namen `wmq.jmsra.rar` bereitgestellt. 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 `wmq.jmsra.rar` mit IBM MQ classes for JMS in dem Verzeichnis installiert, das in Tabelle 61 auf Seite 442 angegeben ist.

Tabelle 61. Verzeichnis mit der Datei 'wmq.jmsra.rar' auf den einzelnen Plattformen

Plattform	**Directory**
UNIX 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 Application Server Liberty	Da WebSphere Application Server Liberty den IBM MQ-Ressourcenadapter nicht enthält, müssen Sie ihn separat aus Fix Central beziehen.
Anderer Java EE-Anwendungsserver	Beziehen Sie den Ressourcenadapter separat von Fix Central, wie für WebSphere Application Server Liberty.

Prozedur

- Wenn Sie eine Verbindung mit IBM MQ von WebSphere Application Server Liberty oder einem anderen Java EE-Anwendungsserver aus herstellen, laden Sie den IBM MQ-Ressourcenadapter herunter und installieren Sie ihn (siehe Beschreibung in „Ressourcenadapter in Liberty installieren” auf Seite 443).
- Linux UNIX

  Stellen Sie bei Bindings-Verbindungen auf Systemen mit UNIX 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 87.

  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 Application Server Liberty konfiguriert würde:

  ```
  <wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
  ```

Transaktionen werden sowohl im Clientmodus als auch im Bindungsmodus unterstützt.

Ressourcenadapter in Liberty installieren

Wenn Sie von WebSphere Application Server Liberty oder anderen Java EE-Anwendungsservern aus eine Verbindung zu IBM MQ herstellen möchten, 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.

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 Downloaddatei 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-InstallRA.jar`).

Nachdem Sie den Ressourcenadapter heruntergeladen und installiert haben, können Sie ihn in WebSphere Application Server Liberty konfigurieren.

Vorgehensweise

1. Laden Sie den IBM MQ-Ressourcenadapter von Fix Central herunter:
   a) Klicken Sie auf **Find Product** (Produkt suchen) und geben Sie die Informationen für Ihre IBM MQ-Installation in folgenden Feldern ein:
      - Geben Sie im Feld **Product Selector** (Produktauswahl) `MQ` ein und wählen Sie dann **WebSphere MQ** in der angezeigten Liste aus.
      - Klicken Sie im Feld **Installed Version** (Installierte Version) auf den Pfeil und wählen Sie dann die Versionsnummer in der angezeigten Liste aus, z. B. **9.0.0.0**.
      - Klicken Sie im Feld **Platform** (Plattform) auf den Pfeil und wählen Sie Ihre Plattform aus, z. B. **Windows 64-Bit, x86**.

      Klicken Sie auf **Weiter** .
   b) Stellen Sie sicher, dass **Browse for Fixes** (Nach Fixes suchen) aktiviert ist, inaktivieren Sie unter **Additional Query options** (Weitere Abfrageoptionen) die Option **Show fixes that apply to this version** (Für diese Version geltende Fixes anzeigen) und wählen Sie **Show fixes that get me to this version** (Fixes anzeigen, die mich zu dieser Version führen) aus. Klicken Sie dann auf **Continue** (Weiter).

Fix Central sucht nach den verfügbaren Fixes für das Produkt, die Version und die Plattform, die Sie ausgewählt haben, z. B. **WebSphere, WebSphere MQ (9.0.0.0, Windows 64-bit, x86)**.

c) Suchen Sie den Ressourcenadapter in der angezeigten Liste mit den verfügbaren Fixes.

Beispiel:

```
release level: 9.0.0.0-IBM-MQ-Install-Java-All
9.0.0.0 MQ Resource Adapter for use with Application Servers
```

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 steht *V.R.M.F* für die Versions-, Release-, Modifikations- und Fixpacknummer und `V.R.M.F-IBM-MQ-Java-InstallRA.jar` für den Namen der Datei, die von Fix Central heruntergeladen wurde.

Verwenden Sie beispielsweise den folgenden Befehl, um den IBM MQ -Ressourcenadapter für IBM MQ 9.0.0.0zu installieren:

```
java -jar 9.0.0.0-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 9.0verwenden, extrahieren oder installieren können, müssen Sie die
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 ein neues Verzeichnis mit dem Namen `wmq` im ausgewählten Installationsverzeichnis erstellt. Anschließend werden folgende Dateien im Verzeichnis `wmq` installiert:
   - das Installationsprüftestprogramm `wmq.jmsra.ivt`
   - die IBM MQ-RAR-Datei `wmq.jmsra.rar`
5. Konfigurieren Sie den Ressourcenadapter in WebSphere Application Server Liberty.

   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 9.0-Ressourcenadapter arbeiten zu können.

   Welche Funktion Sie hinzufügen (wmqJmsClient-1.1 oder wmqJmsClient-2.0), hängt von der Version des Ressourcenadapters ab, die Sie installiert haben. Der IBM MQ 9.0-Ressourcenadapter muss mit der Funktion wmqJmsClient-2.0 implementiert werden. Weitere Informationen finden Sie unter „Zu verwendende Version des Ressourcenadapters” auf Seite 435.

   b) Fügen Sie eine Referenz zu der Datei `wmq.jmsra.rar` hinzu, die Sie installiert haben.

   Anmerkung: Wenn für Liberty-Versionen bis einschließlich WebSphere Application Server 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 Profile 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.

   Eine Beispielkonfiguration für die Unterstützung von Servlets und MDBs und mit JNDI könnte wie folgt aussehen:

   ```
    <featureManager>
       <feature>wmqJmsClient-1.1</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"/>
   ```

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 447 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 450.
- 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 470.
- 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 470.
- Optional: Definieren Sie einen Implementierungsplan für den Ressourcenadapter.

  Die RAR-Datei des IBM MQ-Ressourcenadapters enthält eine Datei mit dem Namen `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 240.

- 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 „Konfigurationsdatei für IBM MQ classes for JMS" auf Seite 89). 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 490.

  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. Nähere Informationen finden Sie unter „Konfiguration von WebSphere Application Server traditional" auf Seite 449, wenn Sie WebSphere Application Server traditional verwenden, bzw. unter „Konfiguration von WebSphere Application Server Liberty" auf Seite 449, wenn Sie WebSphere Application Server Liberty verwenden. 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 448 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.
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).
V 9.0.0 V 9.0.0 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 321.
V 9.0.0 V 9.0.0 nativeLibraryPath	Zeichenfolge	<leer>	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 UNIX 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.

Konfiguration von WebSphere Application Server traditional

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.

Konfiguration von WebSphere Application Server Liberty

Der Ressourcenadapter wird in der Datei `server.xml` durch XML-Elemente konfiguriert, wie das nachfolgende Beispiel zeigt:

```
<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.

Neue ActivationSpec-Eigenschaften in JMS 2.0

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.

Gehen wir von der Situation aus, dass in JNDI eine Verbindungsfactory definiert ist und der JNDI-Name dieses Objekts in der Eigenschaft 'connectionFactoryLookup' einer Aktivierungsspezifikation angegeben ist. Alle Eigenschaften der Verbindungsfactory, die in JNDI definiert sind, werden bevorzugt vor den Eigenschaften in Tabelle 64 auf Seite 451 verwendet.

Wenn ein Ziel in JNDI definiert ist und dessen JNDI-Name in der Eigenschaft 'destinationLookup' der Aktivierungsspezifikation angegeben ist, werden die Werte dieser Eigenschaft bevorzugt vor den in Tabelle 65 auf Seite 462 angegebenen Werten verwendet. Weitere Informationen zur Verwendung dieser beiden Eigenschaften finden Sie im Abschnitt „Eigenschaften ' connectionFactoryLookup' und 'destinationLookup' der Aktivierungsspezifikation” auf Seite 466.

Eigenschaften zur Herstellung einer JMS-Verbindung mit einem IBM MQ-Warteschlangenmanager

Alle Eigenschaften in Tabelle 64 auf Seite 451 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
brokerSubQueue [1]	Zeichenfolge	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Konsument nicht permanenter Nachrichten die Nachrichten empfängt

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
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
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

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
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 466.

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.
failIfQuiesce	Boolesch	• **true** • false	Gibt an, ob Aufrufe bestimmter Methoden fehlschlagen, wenn der Warteschlangenmanager gerade stillgelegt wird
headerCompression	Zeichenfolge	• **NONE** • 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.

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
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	• **NONE** • Eine Liste mit einem oder mehreren der folgenden Werte, die durch Leerzeichen voneinander getrennt sind: RLE ZLIBFAST ZLIBHIGH	Eine Liste der Verfahren, die zum Komprimieren von Nachrichtendaten in einer Verbindung verwendet werden können
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.

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
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.
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.

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
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 ihr Socket nicht für die gemeinsame Nutzung freigeben. • YES - Eine Clientverbindung kann ihr Socket für die gemeinsame Nutzung freigeben.	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	Gibt an, ob eine TLS-Verbindung eine CipherSuite verwenden muss, die vom JSSE-FIPS-Provider von IBM Java unterstützt wird (IBMJSSEFIPS).
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 IBM MQ classes for JMS in IBM WebSphere MQ 7.0verwendet werden. Sie hat keine Auswirkung auf eine Anwendung, die mit einem Warteschlangenmanager der IBM WebSphere MQ 7.0 verbunden ist, es sei denn, die Eigenschaft **providerVersion** wurde auf eine niedrigere Versionsnummer als 7 gesetzt.
2. Sie finden wichtige Informationen zur Verwendung der Eigenschaft 'sslFipsRequired' im Abschnitt „Einschränkungen beim IBM MQ-Ressourcenadapter” auf Seite 438.
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 271.

Eigenschaften zur Erstellung eines JMS-Verbindungskonsumenten

Anmerkung: Die Eigenschaften **destination** und **destinationType** müssen explizit definiert werden. Alle anderen Eigenschaften in Tabelle 65 auf Seite 462 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
Ziel	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 466.
destinationType	Zeichenfolge	• javax.jms.Queue • javax.jms.Topic	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.
maxPoolDepth	Int	• **10** • Eine positive Ganzzahl	Die maximale Anzahl der Serversitzungen im Sitzungspool des Servers, die vom Verbindungskonsumenten genutzt werden.

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
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 Application Server Liberty mit der entsprechenden Version des wmqJms-Client-Features ausgeführt. Weitere Informationen finden Sie im Abschnitt „Liberty und der IBM MQ-Ressourcenadapter" auf Seite 439.
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 die MDB Vorauslesen verwenden darf, um nicht persistente Nachrichten des Ziels vor dem Empfang in einen internen Puffer zu schreiben.
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 469.
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 (wahr) - Die Eigenschaft mit dem Namen 'destination" wird als Name eines javax.jms.Queue- oder javax.jms.Topic-Objekts im JNDI-Namensbereich des Anwendungsservers interpretiert.	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** festlegen, für die Eigenschaft **channel** aber den Standardwert SYSTEM.DEF.SVRCONN übernehmen, wird keine Ausnahmebedingung ausgelöst. In diesem Fall wird die durch die Eigenschaft **ccdtURL** angegebene Definitionstabelle für den Clientkanal zum Starten einer JMS-Verbindung verwendet.

Eigenschaften ' connectionFactoryLookup' und 'destinationLookup' der Aktivierungsspezifikation

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 451 und Tabelle 65 auf Seite 462) 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 451 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. Die JMS 2.0 -Spezifikation besagt jedoch, dass jede aus einer ConnectionFactory erstellte JMS -Verbindung eine eindeutige **ClientID**haben sollte. 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 Flag **UseJndi** auf den Wert `true`gesetzt ist, wird der in der Zieleigenschaft angegebene Text als JNDI -Name betrachtet und ein Zielobjekt mit diesem JNDI -Namen wird in JNDIgesucht.

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**.

Die Eigenschaft useJNDI ist in IBM MQ 9.0 veraltet, da die Eigenschaft **destinationLookup** die JMS 2.0 -Spezifikation ist, die der Ausführung derselben Funktion entspricht.

ActivationSpec-Eigenschaften ohne Entsprechungen in IBM MQ classes for JMS

Die meisten Eigenschaften eines ActivationSpec-Objekts entsprechen den Eigenschaften von IBM MQ classes for JMS-Objekten oder -Parametern von IBM MQ classes for JMS-Methoden. Es gibt jedoch drei Optimierungseigenschaften und eine Nutzbarkeitseigenschaft, die keine funktionalen Entsprechungen in IBM MQ classes for JMS haben:

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 462.

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:

```
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 441.

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 221.

Zugehörige Informationen

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

Federal Information Processing Standards (FIPS) für UNIX, Linux und Windows

JMS-Ressourcen in WebSphere Application Server konfigurieren

Beispiele für Definition der Eigenschaft sharedSubscription
Sie können die Eigenschaft sharedSubscription einer Aktivierungsspezifikation in einer `server.xml`-Datei von WebSphere Application Server Liberty definieren. Alternativ können Sie die Eigenschaft in einer Message-driven Bean (MDB) mithilfe von Annotationen definieren.

Beispiel: Definition in einer `server.xml`-Datei von Liberty

Das folgende Beispiel zeigt, wie eine Aktivierungsspezifikation in einer `server.xml`-Datei von WebSphere Application Server 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:

```
/**
 * 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

„Klone und gemeinsam genutzte Subskriptionen” auf Seite 319
In IBM MQ 8.0 oder höher gibt es zwei Methoden, mehreren Konsumenten Zugriff auf dieselbe Subskription zu erteilen: die Verwendung geklonter Subskriptionen oder die Verwendung gemeinsam genutzter Subskriptionen.

Zugehörige Informationen

Subskribenten und Subskriptionen

Subskriptionspermanenz

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:

```
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

Zur Konfiguration der abgehenden Kommunikation definieren Sie JCA-Ressourcen (Java EE Connector Architecture) 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 War-

teschlange, 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 471 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).
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.
brokerPubQueue [1]	Zeichenfolge	• **SYSTEM.BROKER.DEFAULT.STREAM** • Ein Warteschlangenname	Der Name der Warteschlange, an die veröffentlichte Nachrichten gesendet werden (die Datenstromwarteschlange).
brokerQueueManager [1]	Zeichenfolge	• **"" (leere Zeichenfolge)** • Name eines Warteschlangenmanagers	Der Namen des Warteschlangenmanagers, für den der Broker aktiv ist.
brokerSubQueue [1]	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.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
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.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
failIfQuiesce	Boolesch	• **true** • false	Gibt an, ob Aufrufe bestimmter Methoden fehlschlagen, wenn der Warteschlangenmanager gerade stillgelegt wird.
headerCompression	Zeichenfolge	• **NONE** • 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.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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	• **NONE** • Eine Liste mit einem oder mehreren der folgenden Werte, die durch Leerzeichen voneinander getrennt sind: RLE ZLIBFAST ZLIBHIGH	Eine Liste der Verfahren, die zum Komprimieren von Nachrichtendaten in einer Verbindung verwendet werden können.
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.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
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.
pubAckInterval [1]	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.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
rescanInterval [1]	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.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
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 ihr Socket nicht für die gemeinsame Nutzung freigeben. • YES - Eine Clientverbindung kann ihr Socket für die gemeinsame Nutzung freigeben.	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 [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.	Diese Eigenschaft steuert die Richtlinie für den Nachrichtenabruf eines TopicSubscriber-Objekts.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
sslFipsRequired [2]	Boolesch	• **false** • true	Gibt an, ob eine TLS-Verbindung eine CipherSuite verwenden muss, die vom JSSE-FIPS-Provider von IBM Java (IBMJSSEFIPS) unterstützt wird.
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.

Tabelle 66. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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 **SUBSTORE** 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.
targetClientMatching	Boolesch	• **true** • 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.

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	Bei der Erstellung von temporären Themen generiert JMS eine Topic-Zeichenfolge in der Form "TEMP/ *TEMPTOPICPREFIX* / *eindeutige_ID* " oder (falls für die Eigenschaft der Standardwert übernommen wurde) lediglich in der Form "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. Diese Eigenschaft kann zusammen mit IBM WebSphere MQ classes for JMS in IBM WebSphere MQ 7.0 verwendet werden, wirkt sich jedoch nicht auf eine Anwendung aus, die mit einem IBM WebSphere MQ 7.0 -Warteschlangenmanager verbunden ist, es sei denn, die Eigenschaft providerVersion ist auf eine Versionsnummer kleiner als 7 gesetzt.
2. Sie finden wichtige Informationen zur Verwendung der Eigenschaft 'sslFipsRequired' im Abschnitt „Einschränkungen beim IBM MQ-Ressourcenadapter” auf Seite 438.
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 271.

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 483 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.

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
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.
failIfQuiesce	Zeichenfolge	• **true** • 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 238bestimmt.	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 IBM MQ-Anwendung ohne JMS.	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 487 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 488 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 [1]	Zeichenfolge	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Verbindungskonsument permanente Subskriptionsnachrichten empfängt.
brokerDurSubQueue [1]	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 Eigenschaft **BROKEDURRSUBQ** in der IBM MQ Explorer -Dokumentation.
brokerPubQueue [1]	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 [1]	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 [1]	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.

Anmerkung:

1. Diese Eigenschaft kann mit IBM WebSphere MQ classes for JMS in IBM WebSphere MQ 7.0 verwendet werden, wirkt sich jedoch nicht auf eine Anwendung aus, die mit einem IBM WebSphere MQ 7.0 -Warteschlangenmanager verbunden ist, es sei denn, die Eigenschaft providerVersion des Objekts ConnectionFactory ist auf eine Versionsnummer kleiner als 7 gesetzt.

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 Informationen

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

Federal Information Processing Standards (FIPS) für UNIX, Linux, and Windows

JMS-Ressourcen in WebSphere Application Server konfigurieren

V 9.0.0.5 *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 Application Server 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

- Verwenden Sie in WebSphere Application Server traditionaldie Administrationskonsole, um die Eigenschaft **targetClientMatching** als angepasste Eigenschaft in der IBM MQ -Aktivierungsspezifikation zu definieren:
  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 Application Server Liberty die Eigenschaft **targetClientMatching** in der Definition einer Aktivierungsspezifikation in der Datei `server.xml` an.
  Beispiel:

```
<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 209
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 470
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 Programm des Installationsprüftests (IVT) wird als Unternehmensarchivdatei (EAR-Datei) mit dem Namen `wmq.jmsra.ivt.ear` bereitgestellt. Diese Datei wird mit IBM MQ classes for JMS in demselben Verzeichnis installiert wie die RAR-Datei '`wmq.jmsra.rar`' des IBM MQ-Ressourcenadapters. Informationen zur Installationsposition dieser Dateien finden Sie im Abschnitt „IBM MQ-Ressourcenadapter installieren” auf Seite 441.

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 Application Server Liberty festlegen, fügen Sie der Datei `server.xml` des Anwendungsservers die folgenden Einträge hinzu:

```
<!-- 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/
   ```

   In Abbildung 52 auf Seite 492 ist die erste Seite des IVT-Programms dargestellt.

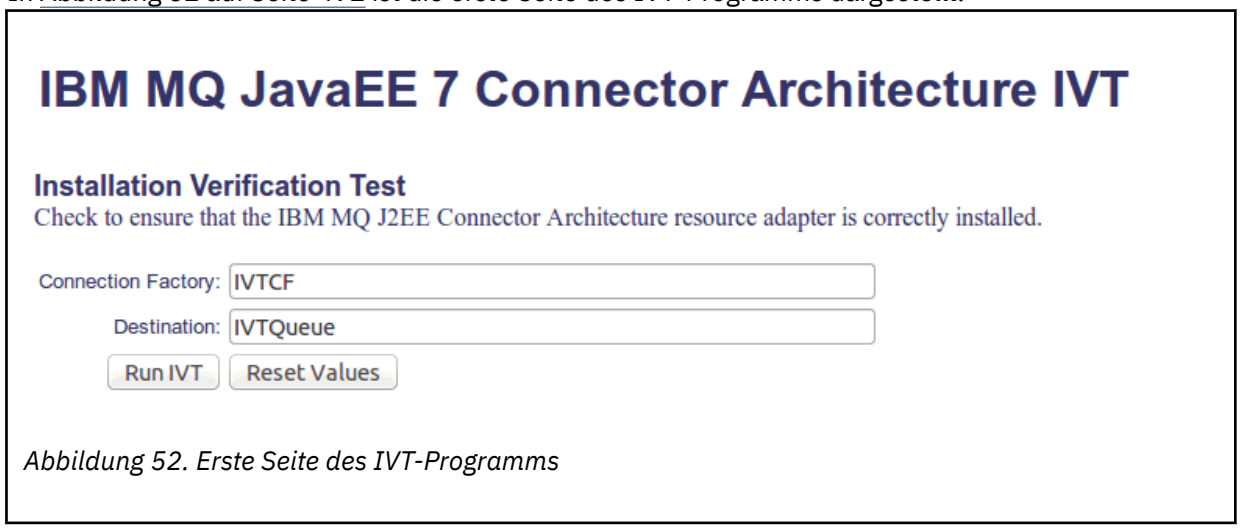
   

   Abbildung 52. Erste Seite des IVT-Programms

2. Wenn Sie den Test ausführen möchten, klicken Sie auf **Run IVT** (IVT ausführen).

   In Abbildung 53 auf Seite 493 ist die Seite dargestellt, die angezeigt wird, wenn der IVT erfolgreich verlaufen ist.

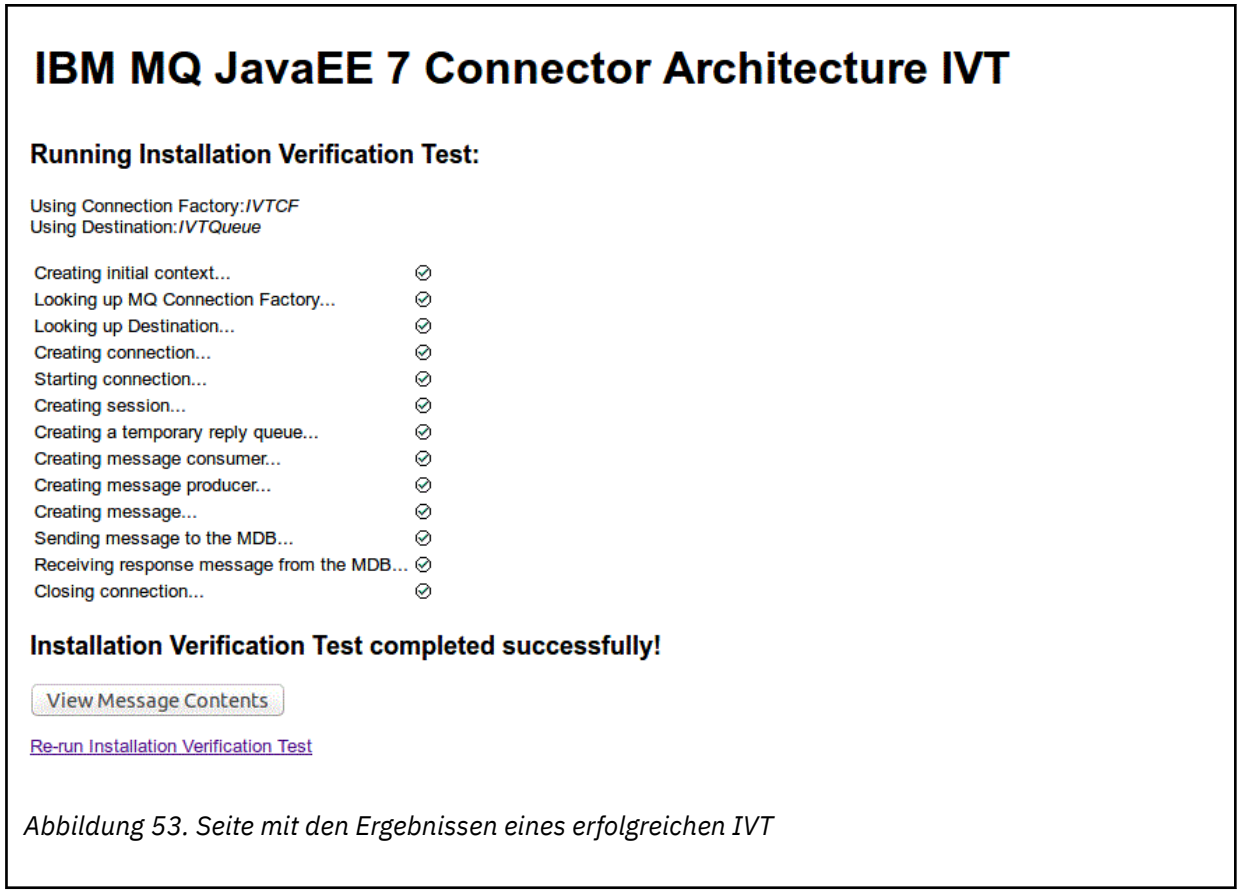


Abbildung 53. Seite mit den Ergebnissen eines erfolgreichen IVT

Wenn der IVT fehlschlägt, wird eine Seite angezeigt, die derjenigen in Abbildung 54 auf Seite 493 ähnelt. Wenn Sie weitere Informationen zur Fehlerursache erhalten möchten, klicken Sie auf **View Stack Trace** (Stack-Trace anzeigen).

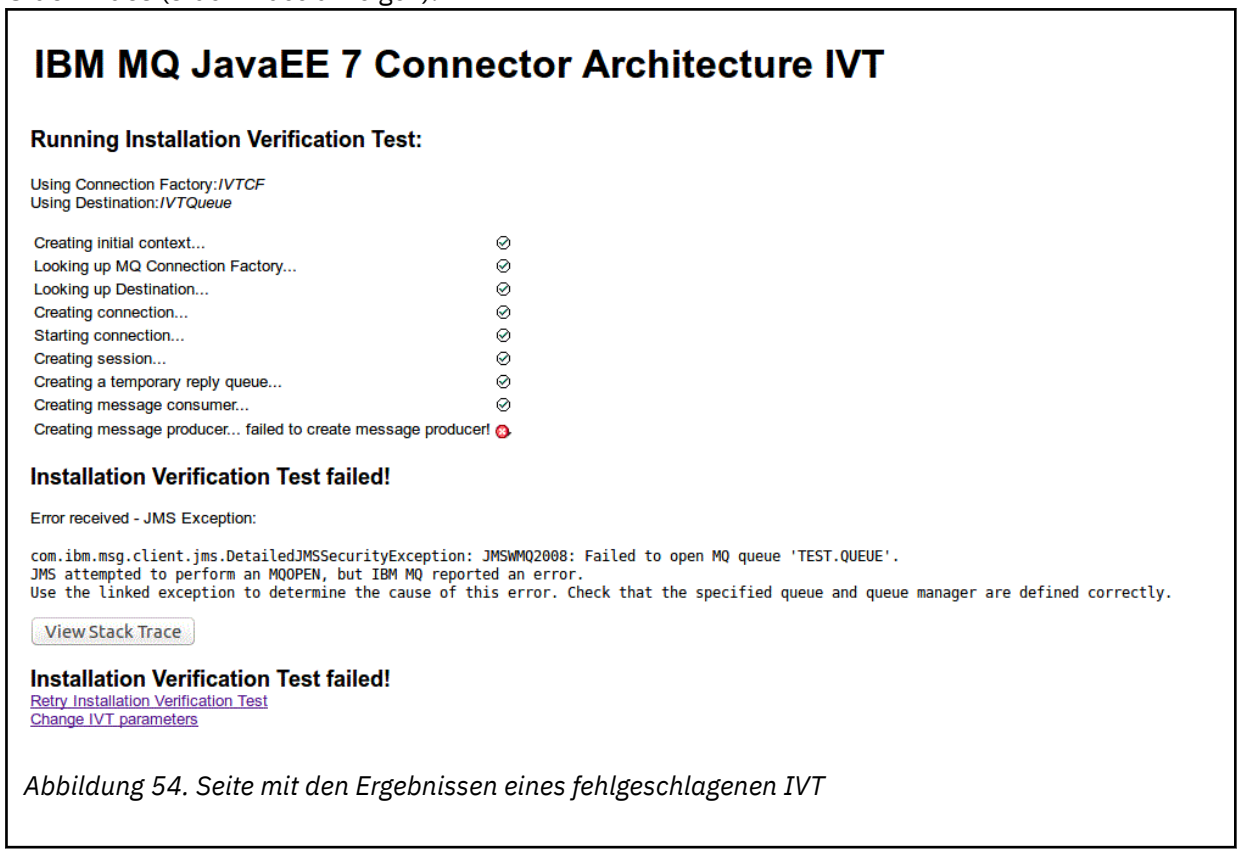


Abbildung 54. Seite mit den Ergebnissen eines fehlgeschlagenen IVT

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.

Informationen zu diesem Vorgang

Wichtig: Diese Anweisungen beziehen sich auf GlassFish Server Version 4.

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, wobei *GlassFish* das Installationsverzeichnis von GlassFish Server Version 4 ist.
   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 **Browse** (Durchsuchen) und navigieren Sie zur Speicherposition 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 **Connection Definition** (Verbindungsdefinition) `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 diese 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 `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:

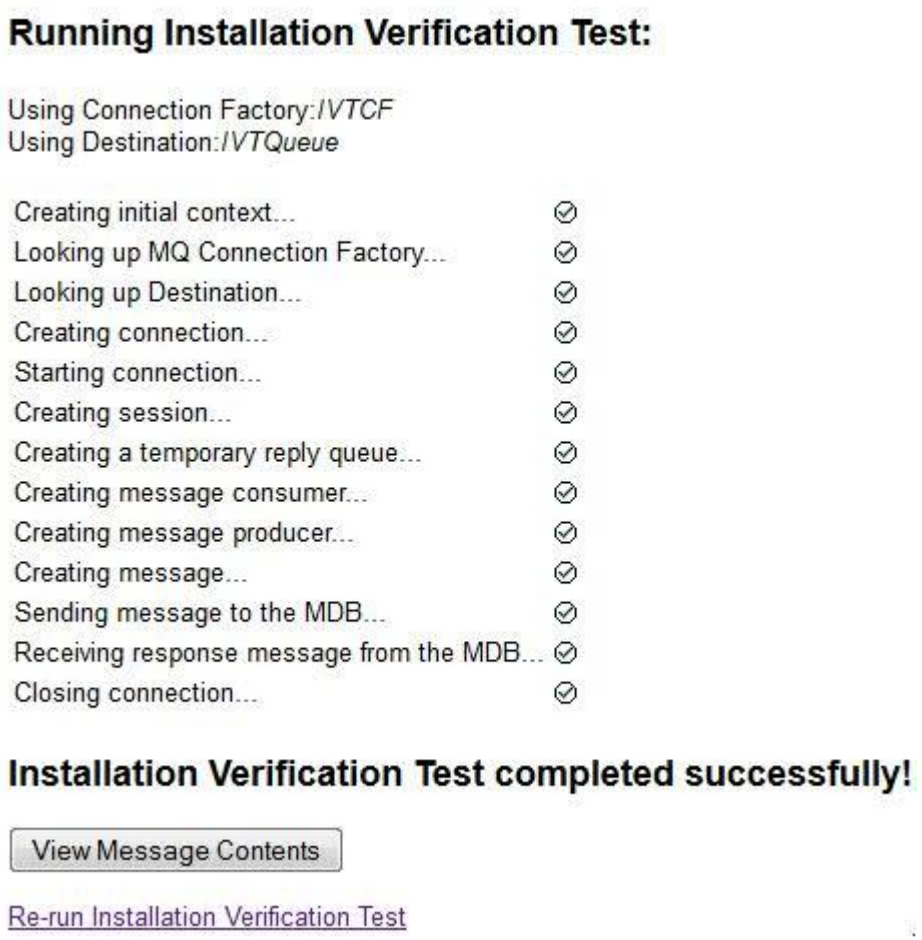


Abbildung 55. Erfolgreiche IVT-Ausgabe

V 9.0.0.1 V 9.0.2 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.

Informationen zu diesem Vorgang

Wichtig: Diese Anweisungen gelten für Wildfly V10.

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 1135 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 folgendes 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. Stellen Sie den Ressourcenadapter auf Ihrem Server bereit, indem Sie die Datei wmq.jmsra.rar in das Verzeichnis *WildFly_Home*/standalone/deployments kopieren.
4. Stellen Sie die IVT-Anwendung bereit, 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 öffnen, zum Verzeichnis *WildFly_Home*/bin navigieren und folgenden Befehl ausführen:

```
standalone.bat -c standalone-full.xml
```

6. Führen Sie die IVT-Anwendung aus.

   Weitere Informationen finden Sie im Abschnitt „Installation des Ressourcenadapters überprüfen” auf Seite 490. 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 Server können die JMS-Messaging-Anwendungen (Java Message Service) Ihr IBM MQ-System als externen Anbieter von JMS-Messaging-Ressourcen verwenden.

Informationen zu diesem Vorgang

Anwendungen, die in Java geschrieben wurden und unter WebSphere Application Server ausgeführt werden, können die Java Messaging Service-Spezifikation (JMS-Spezifikation) zur Nachrichtenübertagung verwenden. 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 Informationen

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 sich überlegen, welche Versionen von IBM MQ mit WebSphere Application Server verwendet werden können, müssen Sie verstehen, wie die im Anwendungsserver ausgeführten JMS-Anwendungen (Java Message Service) 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 Application Server 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 finden Sie unter IBM MQ-Bibliotheken und WebSphere Application Server for 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.

Ermittlung der in WebSphere Application Server installierten Version des IBM MQ-Ressourcenadapters

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

WebSphere Application Server -Versionen vor 7.0 können so konfiguriert werden, dass die IBM WebSphere MQ classes for JMS in einer externen IBM WebSphere MQ -Installation verwendet wird, um eine Verbindung zu einem Warteschlangenmanager herzustellen, indem die WebSphere -Variable MQ_INSTALL_ROOT festgelegt wird.

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 9.0 bereitgestellten nativen Bibliotheken geladen werden. Weitere Informationen finden Sie im Abschnitt „Java Native Interface-(JNI-)Bibliotheken konfigurieren” auf Seite 87.
   - z/OS Unter z/OS müssen Sie die richtigen IBM MQ-Bibliotheken in der WebSphere Application Server-STEPLIB-Verkettung angeben.

     Unter IBM MQ-Bibliotheken und WebSphere Application Server for z/OS STEPLIB finden Sie die erforderlichen Details zu den IBM MQ-Bibliotheken.

     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 ein WebSphere Application Server 8.5mit einem IBM WebSphere MQ 7.1 -Ressourcenadapter eine Bindungsverbindung 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 436.

In der folgenden Tabelle werden die Transporttypen gezeigt, mit denen eine Verbindung zu IBM MQ aus allen Versionen von WebSphere Application Server hergestellt werden kann.

Version von IBM MQ oder IBM WebSphere MQ	BINDINGS-Transport	CLIENT-Transport
IBM MQ 9.0	Unterstützt. • IBM MQ 9.0 muss auf demselben System installiert sein wie der Anwendungsserver. • WebSphere Application Server muss so konfiguriert sein, dass die mit IBM MQ 9.0 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.	Unterstützt
IBM MQ 8.0	Unterstützt. • IBM MQ 8.0 muss auf demselben System installiert sein wie der Anwendungsserver. • WebSphere Application Server muss so konfiguriert sein, dass die mit IBM MQ 8.0 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.	Unterstützt
IBM WebSphere MQ 7.5	Unterstützt. • IBM WebSphere MQ 7.5 muss auf demselben System installiert sein wie der Anwendungsserver. • WebSphere Application Server muss so konfiguriert sein, dass die mit IBM WebSphere MQ 7.5	Unterstützt

Version von IBM MQ oder IBM WebSphere MQ	BINDINGS-Transport	CLIENT-Transport
	bereitgestellten nativen Bibliotheken geladen werden. • z/OS Wenn Sie unter z/OS eine WebSphere Application Server-Verbindungsfactory mit einem IBM WebSphere MQ-Warteschlangenmanager im Bindungsmodus verbinden möchten, müssen Sie die richtigen IBM WebSphere MQ-Bibliotheken in der WebSphere Application Server-STEPLIB-Verkettung angeben.	
IBM WebSphere MQ 7.1	Unterstützt. • IBM WebSphere MQ 7.1 muss auf demselben System installiert sein wie der Anwendungsserver. • WebSphere Application Server muss so konfiguriert sein, dass die mit IBM WebSphere MQ 7.1 bereitgestellten nativen Bibliotheken geladen werden. • z/OS Wenn Sie unter z/OS eine WebSphere Application Server-Verbindungsfactory mit einem IBM WebSphere MQ-Warteschlangenmanager im Bindungsmodus verbinden möchten, müssen Sie die richtigen IBM WebSphere MQ-Bibliotheken in der WebSphere Application Server-STEPLIB-Verkettung angeben.	Unterstützt

In der folgenden Tabelle sind die Versionen von WebSphere Application Server aufgeführt, in denen der IBM MQ-Ressourcenadapter ausgeführt werden kann.

Version des IBM MQ-Ressourcenadapters	In welcher Version von WebSphere Application Server kann diese Version des Ressourcenadapters ausgeführt werden?
IBM MQ 9.0	Der Ressourcenadapter kann in folgenden Versionen ausgeführt werden: • In jeder Java EE 7-kompatiblen Version von WebSphere Application Server Liberty. • WebSphere Application Server traditional 9.0

Version des IBM MQ-Ressourcenadapters	In welcher Version von WebSphere Application Server kann diese Version des Ressourcenadapters ausgeführt werden?
IBM MQ 8.0	Der Ressourcenadapter kann in jeder Java EE 7-kompatiblen Version von WebSphere Application Server Liberty ausgeführt werden. Die Ausführung des IBM MQ 8.0-Ressourcenadapters in WebSphere Application Server traditional wird nicht unterstützt. Für die Verbindung zu IBM MQ 8.0-Warteschlangenmanagern sollte der bereits in WebSphere Application Server traditional installierte Ressourcenadapter verwendet werden.
IBM WebSphere MQ 7.5	Der Ressourcenadapter kann in Anwendungsservern verwendet werden, die mit J2EE 1.4 oder einer höheren Version kompatibel sind. Die Version des in WebSphere Application Server 8.0 und 7.0 enthaltenen IBM WebSphere MQ -Ressourcenadapters sollte in diesen Umgebungen verwendet werden. IBM WebSphere MQ 7.5.0 Fix Pack 2 und APAR IC92914 bieten Unterstützung für die Bereitstellung des Ressourcenadapters in WebSphere Application Server Liberty.
IBM WebSphere MQ 7.1	Der Ressourcenadapter kann in Anwendungsservern verwendet werden, die mit J2EE 1.4 oder einer höheren Version kompatibel sind. Die Version des in WebSphere Application Server 8.0 und 7.0 enthaltenen IBM WebSphere MQ -Ressourcenadapters sollte in diesen Umgebungen verwendet werden.

Zugehörige Konzepte

„Unterstützungserklärung für IBM MQ-Ressourcenadapter” auf Seite 436
Der mit IBM MQ 8.0 oder höher ausgelieferte Ressourcenadapter implementiert die JMS 2.0-Spezifikation. 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.

Zugehörige Informationen

Systemvoraussetzungen für IBM MQ

Anzahl der zwischen WebSphere Application Server und IBM MQ erstellten TCP/IP-Verbindungen bestimmen

In IBM WebSphere MQ 7.0 wurde als neue Funktion die gemeinsame Dialognutzung eingeführt. Dank dieser Funktion 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 8 ausgeführt werden und den normalen Modus des IBM MQ -Messaging-Providers verwenden, verwenden dieses Feature 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.

Anhand der Anzahl der Dialoge, die von WebSphere Application Server 7 und 8 erstellt werden, kann die Anzahl der erstellten Kanalinstanzen ermittelt werden.

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 309
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 an Verbindungen, die aus einer Verbindungsfactory erstellt werden kann, wird durch die Verbindungspooleigenschaft **Maximum connections** (Maximale Anzahl an Verbindungen) der Verbindungsfactory festgelegt. 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 bis zum Schließen der Verbindung beim Überschreiten des Zeitlimits für nicht verwendete Verbindungen (**Unused timeout**) geöffnet.

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 an JMS-Sitzungen, die aus einer JMS-Verbindung erstellt werden kann, wird durch die Sitzungspooleigenschaft **Maximum connections** (Maximale Anzahl an Verbindungen) der Verbindungsfactory festgelegt. Der Standardwert dieser Eigenschaft ist 10.

Bei der Erstellung einer JMS-Sitzung wird ein Dialog gestartet. Dieser Dialog bleibt so lange aktiv, bis die JMS-Sitzung geschlossen wird, weil sie die durch die Eigenschaft **Unused timeout** (Zeitlimit für nicht verwendete Sitzungen) des Sitzungspools festgelegte Verweildauer im freien Pool überschritten hat.

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 sowohl für den Verbindungs- als auch für den Sitzungspool die Standardwerte der Eigenschaft **Max. Verbindungen** verwendet, wird die maximale Anzahl an Dialogen zwischen WebSphere Application Server und IBM MQ wie folgt berechnet:

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Beispiel:

```
        = 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
```

Beispiel:

```
        = 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** (Maximale Anzahl an Serversitzungen) der Aktivierungsspezifikation gibt die maximale Anzahl an Serversitzungen an, die für die Aktivierungsspezifikation zur selben Zeit 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.

Für eine einfache Aktivierungsspezifikation, die den Standardwert von **Maximum server sessions** (Maximale Anzahl an Serversitzungen) verwendet, wird die maximale Anzahl an Dialogen zwischen WebSphere Application Server und IBM MQ wie folgt berechnet:

```
Maximum number of conversations = Maximum server sessions + 1
```

Beispiel:

```
    = 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
```

Beispiel:

```
    = 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 Verbindungsfactory 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 505.

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 505.

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 Listenerport, der den Standardwert von **Maximum sessions** (Maximale Anzahl an Sitzungen) verwendet, wird die maximale Anzahl an Dialogen zwischen WebSphere Application Server und IBM MQ wie folgt berechnet:

```
Maximum number of conversations = Maximum sessions + 1
```

Beispiel:

```
    = 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
```

Beispiel:

```
    = 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 505.

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
```

Beispiel:

```
     = 10
```

Wenn dieser Listener-Port eine Verbindung zu IBM MQ über einen Kanal herstellt, dessen Eigenschaft **SHARECNV** auf 10 gesetzt ist, wird die Anzahl der erstellten Kanalinstanzen wie folgt berechnet:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Beispiel:

```
     = 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
```

Beispiel:

```
     = 10
```

Wenn dieser Listener-Port eine Verbindung zu IBM MQ über einen Kanal herstellt, dessen Eigenschaft **SHARECNV** auf 10 gesetzt ist, wird die Anzahl der erstellten Kanalinstanzen wie folgt berechnet:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Beispiel:

```
     = 10 / 10
     = 1
```

Authentifizierungsaliase zum Schutz von WebSphere Application Server-Verbindungen mit IBM MQ verwenden

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 in WebSphere Application Server ausgeführte Unternehmensanwendung versucht, eine JMS-Verbindung mit IBM MQ herzustellen, sucht die Anwendung im Java Naming Directory Interface-Repository (JNDI) des Anwendungsservers nach der Verbindungsfactory-Definition eines IBM MQ-Messaging-Providers.

Wird im JNDI-Repository des Anwendungsservers die Verbindungsfactory-Definition eines IBM MQ-Messaging-Providers gefunden, so 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 Identifizierung den JNDI-Namen `jms/myCF`.

Verbindungsfactorys können auch mit Authentifizierungsaliasen konfiguriert werden. Authentifizierungsaliase sind einer Benutzernamens- und Kennwortkombination zugeordnet. Je nach Verwendung der Verbindungsfactory kann die Benutzernamens- und Kennwortkombination des Authentifizierungsalias bei der Herstellung der JMS-Verbindung an IBM MQ übergeben oder nicht zurückgehalten werden.

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. Nähere Informationen hierzu finden Sie im Abschnitt Kanalsicherheitsexit-Programme.

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 514
- „Verbindungsfactory über ein indirektes Lookup verwenden” auf Seite 515

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 70. 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 71. 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 eine Ressourcenreferenz für die Verbindungsfactory, deren Eigenschaft **`res-auth`** auf *Application* 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 Implementierungsdeskriptor der Anwendung enthält eine Ressourcenreferenz für die Verbindungsfactory, deren Eigenschaft **`res-auth`** auf *Application* 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 Implementierungsdeskriptor der Anwendung enthält eine Ressourcenreferenz für die Verbindungsfactory, für die die Eigenschaft **`res-auth`** auf "*Container* " gesetzt ist, und die Verbindungsfactory verwendet einen IBM MQ -Kanal, für den die 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 71. CLIENT-Modus (Forts.)		
Konfiguration	**Anwendung ruft `ConnectionFactory.createConnection()` auf**	**Anwendung ruft `ConnectionFactory.createConnection(String username, String password)` auf**
Der Implementierungsdeskriptor der Anwendung enthält eine Ressourcenreferenz für die Verbindungsfactory, deren 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 direkte Suche nach einer Verbindungsfactory durchgeführt und dann diese Methode aufgerufen hat, werden der Benutzername und das Kennwort, die an die Methode `createConnection()` übergeben wurden, an IBM MQübergeben.

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. Nähere Informationen hierzu finden Sie im Abschnitt Kanalsicherheitsexit-Programme.

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");
```

Sie benötigen das Präfix `java:comp/env` für den lokalen JNDI -Namen, um dem Anwendungsserver mitzuteilen, dass die Unternehmensanwendung eine indirekte Suche durchführt.

Wenn die Anwendung implementiert wird, ordnet der Benutzer den JNDI Namen der Ressourcenreferenz `jms/myResourceReferenceCF` dem JNDI-Namen der Verbindungsfactory zu, die die Anwendung bereits erstellt hat: `jms/myCF`.

Wenn die Anwendung ausgeführt wird, sucht sie eine JMS -Verbindungsfactory unter Verwendung des lokalen JNDI -Namens, den der Anwendungsserver 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 *Anwendung*gesetzt ist, wird die Standardbenutzeridentität an IBM MQweitergegeben. 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. Wenn eine Anwendung eine JMS -Verbindung erstellt, werden die Authentifizierungsaliasdetails verwendet, auch wenn der Aufruf `createConnection` einen Benutzernamen und ein Kennwort angibt.

Authentifizierungsalias bei Verwendung eines indirekten Lookups überschreiben

Wenn eine Anwendung eine Ressourcenreferenz verwendet, deren Eigenschaft **`res-auth`** auf *Container*gesetzt ist, können Sie den Authentifizierungsalias überschreiben, der verwendet wird, wenn JMS -Verbindungen erstellt werden.

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 zu WebSphere Application Server 8.5.5

Ressourcenverweise

Referenzinformationen zu WebSphere Application Server 8.0

Ressourcenverweise

Referenzinformationen zu WebSphere Application Server 7.0

Ressourcenverweise

Lastausgleich für Message-driven Beans bei Verwendung von WebSphere Application Server-Clustern

Wenn Sie MDB-Anwendungen verwenden, die in einem Cluster mit WebSphere Application Server 7.0 und 8.0 implementiert und für die Ausführung im normalen Modus des IBM WebSphere MQ -Messaging-Providers konfiguriert sind, verarbeitet eines der Cluster-Member die meisten Nachrichten. Sie können die Arbeitslast der Cluster-Member jedoch ausgleichen, um die Nachrichtenverarbeitung auf mehrere Member zu verteilen.

IBM WebSphere MQ 7.0 hat eine neue Funktion namens **Asynchronous consume**eingeführt, die es Anwendungen ermöglicht, Nachrichten asynchron aus einer Warteschlange unter Verwendung von APIs mit den Namen **MQCB** und **MQCTL**zu konsumieren.

Message-driven Bean-Anwendungen, die in WebSphere Application Server 7.0 und 8.0ausgeführt werden und den normalen Modus des IBM WebSphere MQ -Messaging-Providers verwenden, verwenden dieses Feature automatisch. Beim Start einer solchen Anwendung richtet diese mithilfe eines **MQCB**-Aufrufs für die von ihr überwachte JMS-Warteschlange einen asynchronen Konsumenten ein. 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 WebSphere MQ 7.0-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.

Vor IBM WebSphere MQ 7.0.1 Fix Pack 6 bevorzugten es die Warteschlangenmanager, wenn das erste Cluster-Member einen asynchronen Konsumenten für die JMS-Warteschlange einrichtete. Dieses Cluster-Member war dann das erste Member, das beim Eintreffen einer zutreffenden Nachricht in der JMS-Warteschlange benachrichtigt wurde. Das bedeutete, dass das erste Cluster-Member, das die Message-driven Bean-Anwendung startete, den größten Teil der geeigneten Nachrichten aus der JMS-Warteschlange verarbeitete.

Bei einer Verbindung von WebSphere Application Server mit einem IBM WebSphere MQ 7.0.1 Fix Pack 6-Warteschlangenmanager (oder höher) werden die in der JMS-Warteschlange eintreffenden Nachrichten nun gleichmäßiger auf alle für diese JMS-Warteschlange registrierten asynchronen Konsumenten verteilt. Für Message-driven Bean-Anwendungen, die in einem WebSphere Application Server 7.0 -und 8.0 -Cluster implementiert sind, bedeutet dies, dass die Nachrichten gleichmäßiger auf die Cluster-Member verteilt werden.

Zugehörige Informationen

Eigenschaft 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.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 518.
- 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 519.

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 344).

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 372
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 378
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 518).

Vorgehensweise

1. Erstellen Sie ein JMS-Warteschlangenziel, das die SYSTEM.ADMIN.COMMAND.QUEUE darstellt.

   IBM MQ JMS-Anwendungen senden die PCF-Nachrichten an die SYSTEM.ADMIN.COMMAND.QUEUE und 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 133
JMS-Nachrichten setzen sich aus Header, Eigenschaften und Hauptteil zusammen. JMS definiert fünf Typen für den Nachrichtenhauptteil.

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` Bestandteil des JDK.

CL-Befehle verwenden

Der von Ihnen festgelegte Klassenpfad (CLASSPATH) dient für Tests mit MQ-Basis-Java, JMS mit JNDI und JMS ohne JNDI.

Wenn Sie keine `.profile`-Datei unter Ihrem `/home/Userprofile`-Verzeichnis verwenden, müssen Sie die unten angegebenen Umgebungsvariablen auf Systemebene setzen. 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 gesetzt ist, gehen Sie wie folgt vor:

```
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.mqjms.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 `.profile`-Datei 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.

```
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.mqjms.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, 2023. 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, 2023. 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, 2023.
All Rights Reserved.
WebSphere MQ classes for Java(tm) 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, 2023.
All Rights Reserved.
WebSphere MQ classes for Java(tm) 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. Führen Sie die geeigneten Änderungen in der Datei JMSAdmin.config durch. Zum Bearbeiten dieser Datei verwenden Sie den Befehl **EDTF** (Datei bearbeiten) über eine IBM i -Befehlszeile.

   ```
   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, 2023. All Rights Reserved.
Starting WebSphere MQ classes for Java(tm) Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java(tm) 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, 2023. All Rights Reserved.
MQSeries classes for Java(tm) 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
$
```

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.

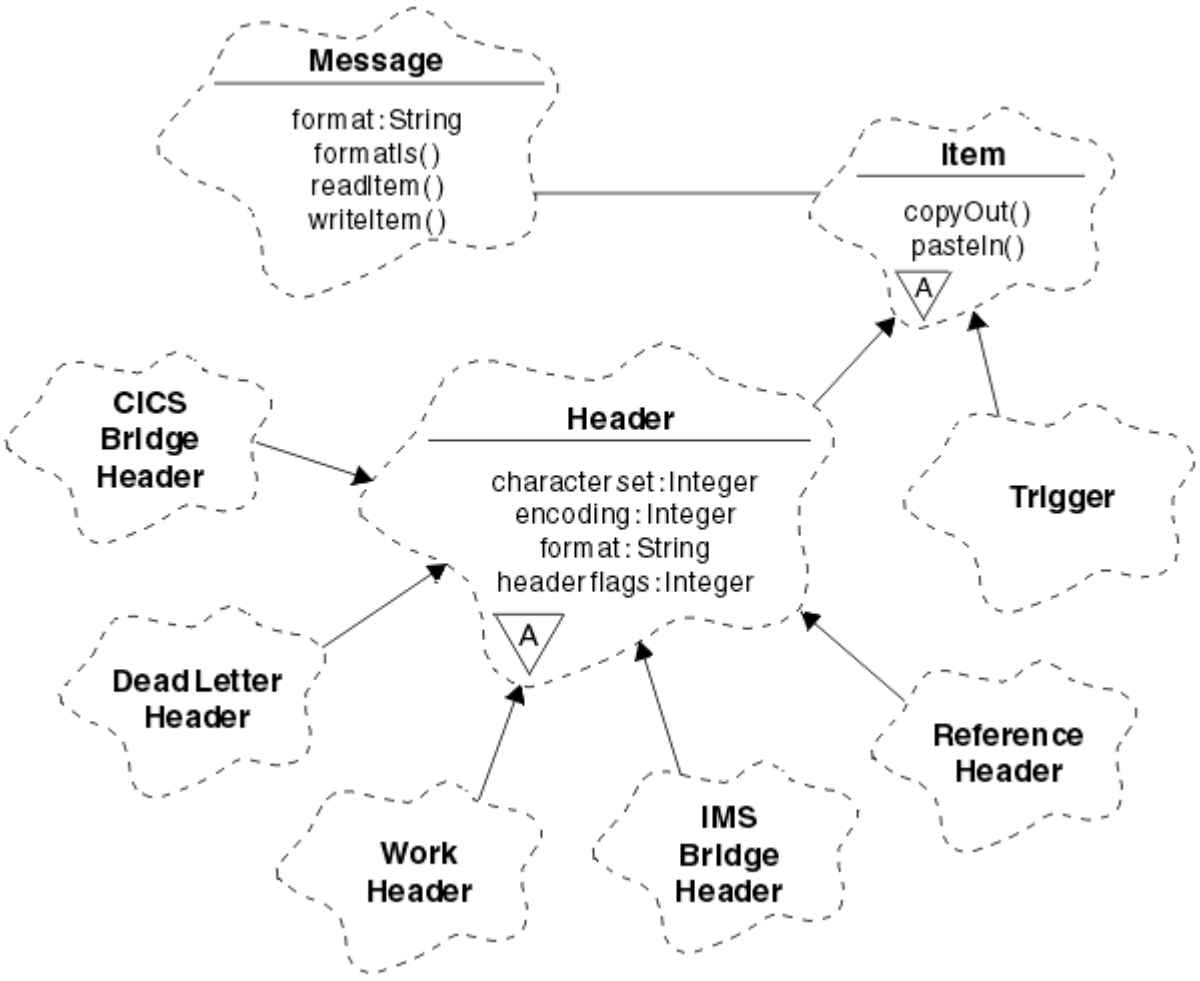


Abbildung 56. IBM MQ C++-Klassen (Behandlung von Elementen)

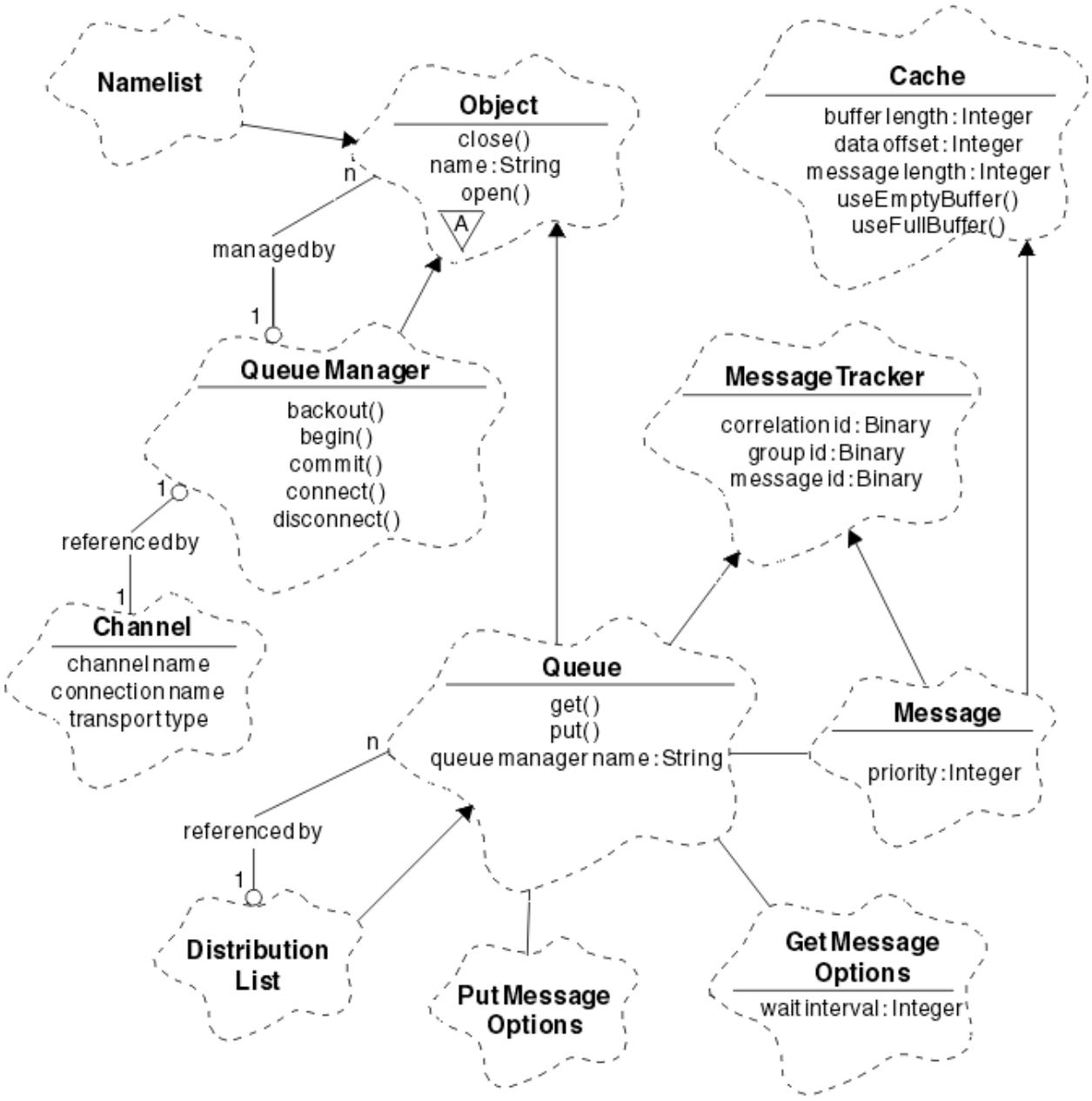


Abbildung 57. IBM MQ C++-Klassen (Warteschlangenmanagement)

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 57 auf Seite 529) und der Klassen zur Elementverarbeitung (siehe Abbildung 56 auf Seite 528) 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 540), 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

„Beispielprogramme in C++” auf Seite 530
In vier bereitgestellten Beispielprogrammen wird das Abrufen und Einreihen von Nachrichten veranschaulicht.

„Überlegungen zur Programmiersprache C++” auf Seite 535
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 539
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 545
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.

Zugehörige Informationen

Technische Übersicht
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 72 auf Seite 531 gezeigten Verzeichnissen.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Tabelle 72. Position der Beispielprogramme

Umgebung	Verzeichnis mit Quelle	Verzeichnis mit erstellten Programme
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
IBM i IBM i IBM i	/QIBM/ProdData/mqm/samp/	(siehe Hinweis „1” auf Seite 531)
HP-UX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/ samp/bin/ah (siehe Hinweis „2” auf Seite 531)
z/OS z/OS z/OS	thlqual.SCSQCPPS	--
Solaris	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/ samp/bin/as
Linux	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/
Windows	*MQ_INSTALLATI-ON_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\ samples\bin\vn (siehe Hinweis „3” auf Seite 531)

Anmerkungen:

1. Programme, die mithilfe des ILE C++-Compilers für IBM i erstellt wurden, befinden sich in der Bibliothek QMQM. Die einzuschließenden Dateien befinden sich in /QIBM/ProdData/mqm/inc.
2. Programme, die mithilfe des HP ANSI C++-Compilers erstellt wurden, befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*/samp/bin/ah. Weitere Informationen finden Sie unter „C++-Programme unter HP-UX erstellen” auf Seite 547.
3. Programme, die mithilfe von Microsoft Visual Studio erstellt wurden, befinden sich in *MQ_INSTALLA-TION_PATH*\tools\cplus\samples\bin\vn. Weitere Informationen zu diesen Compilern finden Sie unter „C++-Programme unter Windows erstellen” auf Seite 553.

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 unter MQSERVER) und führen Sie **imqwrldc** aus, oder
- Führen Sie **imqwrldc** aus und übergeben Sie den **Warteschlangennamen**, **Warteschlangenmanagernamen** und die **Kanaldefinition** als Parameter; eine typische **Kanaldefinition** ist beispielsweise 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 „C++-Programme auf z/OS Batch, RRS Batch und CICS erstellen” auf Seite 555.

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 73. 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.

C++-Programme unter AIX erstellen

IBM MQ C++-Programme auf AIX mit dem Compiler der XL C Enterprise Edition erstellen.

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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

C++-Programme unter HP-UX erstellen

IBM MQ C++-Programme auf HP-UX mit den aC++- oder aCC-Compilern erstellen.

Bei HP-UX Itanium unterstützt IBM MQ nur die Standard-Laufzeit. Verwenden Sie den aCC-Compiler.

- libimqi23bh.sl stellt die IBM MQ C++-Klassen für die Standard-Laufzeit bereit.
- Für Kompatibilität mit früheren Versionen wird eine symbolische Verbindung von libimqi23ah.sl mit libimqi23bh.sl angegeben.

IA64 (IPF)

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client: IA64 (IPF)

32-Bit Unthread-Anwendung

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqic
```

32-Bit Thread-Anwendung

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqic_r -lpthread
```

64-Bit Unthread-Anwendung

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh
-lmqic
```

64-Bit Thread-Anwendung

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqic_r
-lpthread
```

Server: IA64 (IPF)

32-Bit Unthread-Anwendung

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqm
```

32-Bit Thread-Anwendung

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqm_r -lpthread
```

64-Bit Unthread-Anwendung

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqm
```

64-Bit Thread-Anwendung

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
```

```
-lmqm_r
-lpthread
```

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)
      ```

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
```

System x (32-Bit)

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client: System x (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: System x (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
```

C++-Programme unter Solaris erstellen

IBM MQ C++-Programme auf Solaris mit dem Sun ONE-Compiler erstellen.

SPARC

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client: SPARC

32-Bit-Anwendung

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

64-Bit-Anwendung

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Server: SPARC

32-Bit-Anwendung

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

64-Bit-Anwendung

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Client: x86-64

32-Bit-Anwendung

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

64-Bit-Anwendung

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Server: x86-64

32-Bit-Anwendung

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

64-Bit-Anwendung

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
```

```
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

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.

Ab IBM MQ 8.0.0 Fix Pack 4werden weiterverteilbare Clients bereitgestellt, 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 für die Verwendung mit 32-Bit-Anwendungen werden in *MQ_INSTALLATION_PATH*`/Tools/Lib`installiert, Dateien für die Verwendung mit 64-Bit-Anwendungen in *MQ_INSTALLATION_PATH*`/Tools/Lib64`. *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
```

C++-Clientbibliotheken, die mithilfe des Microsoft Visual Studio 2015-Compilers erstellt wurden

V 9.0.1

V 9.0.1 Ab IBM MQ 9.0.1stellt das Produkt C + + -Clientbibliotheken bereit, die mit dem Microsoft Visual Studio 2015 C + + -Compiler erstellt werden. Anwendungen, die mit einem Release von IBM MQ 9.0.1 oder höher erstellt wurden, können diese Bibliotheken nutzen. Diese Bibliotheken werden zusätzlich zu den vorhandenen IBM MQ 9.0.1-C++-Bibliotheken bereitgestellt, die mit dem C++-Compiler von Microsoft Visual Studio 2012 erstellt werden.

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 in den Ordnern unter `bin64\vs2015`.

IBM MQ ist standardmäßig für die Verwendung der Bibliotheken von Microsoft Visual Studio 2012 konfiguriert. Wenn Sie die Microsoft Visual Studio 2015 -Bibliotheken verwenden möchten, müssen Sie die Umgebungsvariable `MQ_PREFIX_VS_LIBRARIES` mit dem Befehl **setmqenv**oder **setmqinst** festlegen.

Sie können eigene Messaging-Anwendungen mit Microsoft Visual Studio 2017 kompilieren und mit den in IBM MQ bereitgestellten C/C++-Bibliotheken von Microsoft Visual Studio 2015 verlinken.

IBM MQ mit dem C++-Compiler von Microsoft Visual Studio 2015 verwenden

In Microsoft Visual Studio 2015 enthalten die Microsoft-Mergemodule, die mit IBM MQ installiert werden, nicht mehr den gesamten C-Laufzeitcode.

Um den C++-Compiler von Microsoft Visual Studio 2015 verwenden zu können, müssen Sie die Microsoft Knowledge Base-Aktualisierung KB3118401 von der Webseite Windows 10 Universal C Runtime installieren, wenn Sie eine Version von Windows vor Windows 10 verwenden.

Diese Seite enthält Systemvoraussetzungen und Installationsanweisungen.

Wenn Sie KB3118401 nicht installieren, werden C++-Programme, die gegen Microsoft Visual Studio 2015 erstellt werden (sowohl Programme von IBM als auch die für Ihr eigenes Unternehmen), nicht gestartet, wobei in der Regel folgende Nachricht angezeigt wird:

```
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.
```

Anders benannte IBM MQ-C++-Bibliotheken verwenden

Ab IBM MQ 8.0.0 Fix Pack 4stellt das Produkt einige zusätzliche C + + -Clientbibliotheken bereit, die anders benannt sind. Diese Bibliotheken werden mit dem C++-Compiler von Microsoft Visual Studio 2012 erstellt. Bibliotheken von Microsoft Visual Studio 2015 sind ebenfalls verfügbar. Diese Bibliotheken werden zusätzlich zu den vorhandenen C++-Bibliotheken bereitgestellt, die auch mit dem C++-Compiler von Microsoft Visual Studio 2012 oder Microsoft Visual Studio 2015 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 2012 und früheren Versionen des Produkts kompiliert wurden, auf derselben Maschine ausführen.

Die zusätzlichen Microsoft Visual Studio 2012-Bibliotheken sind wie folgt benannt:

- `imqb23vnvs2012.dll`
- `imqc23vnvs2012.dll`
- `imqs23vnvs2012.dll`
- `imqx23vnvs2012.dll`

Die zusätzlichen Microsoft Visual Studio 2015-Bibliotheken sind wie folgt benannt:

- `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 im Ordner `bin` und die 64-Bit-Bibliotheken werden im Ordner `bin64` installiert. Entsprechende Importbibliotheken werden in den Verzeichnissen `Tools\lib` und `Tools\lib64` installiert.

Wenn Ihre Anwendung `imq*vs2012.lib`-Dateien verwendet, müssen Sie diese mit dem Microsoft Visual Studio 2012-Compiler kompilieren. Um IBM MQ-C++-Anwendungen, die mit Microsoft Visual Studio 2012 kompiliert werden, und Anwendungen, die mit einer früheren Version des Produkts kompiliert werden, auf derselben Maschine 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\vs2008;%PATH%
```

- Für 64-Bit-Anwendungen:

```
SET PATH=installation folder\bin64\vs2008;%PATH%
```

Zugehörige Informationen

Windows: Änderungen für IBM MQ 8.0

C++-Programme auf z/OS Batch, RRS Batch und CICS erstellen

IBM MQ C++-Programme auf z/OS für die Batch-, RRS Batch- oder CICS-Umgebung erstellen und die Beispielprogramme ausführen.

Sie können für drei der Umgebungen, die IBM MQ for z/OS unterstützt, C++-Programme schreiben:

- Batch
- RRS-Batch
- CICS

Kompilieren, Verbindungsvorbereitung und Verbindung

Erstellen Sie eine z/OS-Anwendung durch Kompilieren, Verbindungsvorbereitung und Programmverbindung des C++-Quellcodes.

IBM MQ C++ für z/OS ist in Form von z/OS-DLLs für die IBM C++ für z/OS-Sprache implementiert. Mithilfe von DLLs verketten Sie die bereitgestellten Definitions-Sidedecks mit der Compiler-Ausgabe zum Zeitpunkt der Verbindungsvorbereitung. Dies ermöglicht dem Linker, Ihre Aufrufe bei den IBM MQ C++-Mitgliedsfunktionen zu überprüfen.

Anmerkung: Es gibt drei Gruppen von Sidedecks für jede der drei Umgebungen.

Um eine IBM MQ for z/OS C++-Anwendung zu erstellen, erstellen Sie eine JCL-Datei und führen Sie sie aus. Gehen Sie dabei wie folgt vor:

1. Wenn Ihre Anwendung unter CICS ausgeführt wird, verwenden Sie die von CICS bereitgestellte Prozedur, um CICS-Befehle in Ihrem Programm zu übersetzen.

   Zudem müssen Sie für CICS-Anwendungen Folgendes tun:

   a. Fügen Sie die SCSQLOAD-Bibliothek der DFHRPL-Verkettung hinzu.
   b. Definieren Sie die CSQCAT1 CEDA-Gruppe mit dem Element IMQ4B100 in der SCSQPROC-Bibliothek.
   c. Installieren Sie CSQCAT1.
2. Kompilieren Sie das Programm, um Objektcode zu erzeugen. Die JCL für Ihre Kompilierung muss Anweisungen enthalten, die die Produktdatendefinitionsdateien für den Compiler verfügbar machen. Die Datendefinitionen werden in den folgenden IBM MQ for z/OS-Bibliotheken bereitgestellt:

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   Geben Sie unbedingt die Compileroption `/cxx` an.

   Anmerkung: Der Name **thlqual** ist das übergeordnete Qualifikationsmerkmal der IBM MQ-Installationsbibliothek auf z/OS.
3. Bereiten Sie die Verbindung des Objektcodes vor, der in Schritt „2" auf Seite 555 erstellt wurde, einschließlich der folgenden Definitions-Sidedecks, die in **thlqual**.SCSQDEFS bereitgestellt werden:

   a. imqs23dm und imqb23dm für Batch
   b. imqs23dr und imqb23dr für RRS batch
   c. imqs23dc und imqb23dc für CICS

   Dies sind die entsprechenden DLLs.

   a. imqs23im und imqb23im für Batch
   b. imqs23ir und imqb23ir für RRS batch
   c. imqs23ic und imqb23ic für CICS
4. Stellen Sie eine Programmverbindung des Objektcodes her, der in Schritt „3" auf Seite 555 erstellt wurde, um ein Lademodul zu erzeugen, und speichern Sie es in Ihrer Anwendungsladebibliothek.

Um Batch oder RRS-Batch-Programme auszuführen, schließen Sie die Bibliotheken **thlqual**.SCSQAUTH und **thlqual**.SCSQLOAD in der STEPLIB- oder JOBLIB-Dataset-Verkettung ein.

Um ein CICS-Programm auszuführen, muss zuerst Ihr Systemadministrator es für CICS als IBM MQ-Programm und Transaktion definieren. Danach können Sie es wie gewohnt ausführen.

Beispielprogramme ausführen

Die Programme werden in „Beispielprogramme in C++” auf Seite 530 beschrieben.

Die Beispielanwendungen werden nur in Form einer Quelle bereitgestellt. Dies sind die Dateien:

Tabelle 74. z/OS-Beispielprogrammdateien

Beispiel	Quellenprogramm (in Bibliothek thlqual.SCSQCPPS)	JCL (in Bibliothek thlqual.SCSQPROC)
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

Um die Beispiele auszuführen, kompilieren Sie sie und stellen Sie eine Programmverbindung her. Sie können hierfür genauso vorgehen wie bei jedem anderen C++-Programm (weitere Informationen hierzu erhalten Sie unter „C++-Programme auf z/OS Batch, RRS Batch und CICS erstellen” auf Seite 555). Verwenden Sie die bereitgestellte JCL, um einen Batch-Job zu erstellen und auszuführen. Sie müssen anfänglich die JCL anpassen, indem Sie die darin enthaltenen Erläuterungen befolgen.

C++-Programme auf z/OS UNIX System Services erstellen

IBM MQ C++-Programme auf z/OS für UNIX System Services erstellen.

Um eine Anwendung unter der UNIX System Services-Shell zu erstellen, müssen Sie dem Compiler Zugriff auf die IBM MQ-Include-Dateien (in `thlqual.SCSQC370` und `hlqual.SCSQHPPS`) gewähren und eine Verbindung zu zwei der DLL-Sidedecks (in `thlqual.SCSQDEFS`) herstellen. Während der Laufzeit benötigt die Anwendung Zugriff auf die IBM MQ-Datasets `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH` und eines der sprachspezifischen Datasets, wie z. B. `thlqual.SCSQANLE`.[6].

Kompilieren

1. Kopieren Sie das Beispiel mit dem TSO-Befehl **oput** in das HFS oder nutzen Sie FTP. Der Rest dieses Beispiels setzt voraus, dass Sie das Beispiel in ein Verzeichnis namens `/u/fred/sample` kopiert und es `imqwrld.cpp` benannt haben.
2. Melden Sie sich bei der UNIX System Services-Shell an und wechseln Sie in das Verzeichnis, in dem Sie das Beispiel untergebracht haben.
3. Richten Sie den C++-Compiler so ein, dass er das DLL-Sidedeck und CPP-Dateien als Eingabe akzeptieren kann:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. Kompilieren und verbinden Sie das Beispielprogramm. Beim folgenden Befehl wird das Programm mit den Batch-Sidedecks verbunden; die RRS batch-Sidedecks können stattdessen verwendet werden. Mit

[6] Sie können einen Link zu einem beliebigen der unter " Vorverlinken des Objektcodes aufgelisteten Seitendecks erstellen, um Ihren UNIX -Systemservice in einer der drei Umgebungen auszuführen: „C++-Programme auf z/OS Batch, RRS Batch und CICS erstellen” auf Seite 555

dem Zeichen \ wird der Befehl auf mehrere Zeilen aufgeteilt. Geben Sie dieses Zeichen nicht ein; geben Sie den Befehl als eine einzelne Zeile ein:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

Weitere Informationen zum TSO-Befehl **oput** erhalten Sie in der *z/OS UNIX System Services-Befehlsreferenz*.

Sie können auch das Dienstprogramm 'make' verwenden, um das Erstellen von C++-Programmen zu vereinfachen. Hier ist eine Beispiel-Makefile zum Erstellen des C++-Beispielprogramms HELLO WORLD. Es trennt die Kompilierungs- und Verbindungsphase voneinander. Richten Sie die Umgebung wie in Schritt „3” auf Seite 556 beschrieben ein, bevor Sie 'make' ausführen.

```
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
```

Unter *z/OS UNIX System Services-Programmiertools* erhalten Sie weitere Informationen zur Verwendung von 'make'.

Aktiv

1. Melden Sie sich bei der UNIX System Services-Shell an und wechseln Sie in das Verzeichnis, in dem Sie das Beispiel erstellt haben.
2. Konfigurieren Sie die STEPLIB-Umgebungsvariable so, dass IBM MQ-Datasets einbezogen werden:

```
/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. Führen Sie das Beispiel aus:

```
/u/fred/sample:> ./imqwrld
```

.NET-Anwendungen entwickeln

Mit IBM MQ classes for .NET kann ein Programm, das mit einem Framework zur .NET-Programmierung geschrieben wurde, eine Verbindung zu IBM MQ als IBM MQ MQI client oder eine direkte Verbindung zu einem IBM MQ-Server herstellen.

Wenn Sie über Anwendungen verfügen, die das Microsoft .NET-Framework verwenden, und Sie die Vorteile der IBM MQ-Funktionen nutzen möchten, müssen Sie IBM MQ classes for .NET verwenden.

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();
```

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. Einzelheiten zu diesen Klassen finden Sie unter IBM MQ .NET-Klassen und -Schnittstellen.

Bevor Sie die von Ihnen geschriebenen Anwendungen kompilieren können, muss ein .NET-Framework installiert sein. Weitere Anweisungen zum Installieren der IBM MQ classes for .NET und des .NET-Frameworks finden Sie unter „IBM MQ classes for .NET installieren” auf Seite 559.

Zugehörige Konzepte

„Optionen für die Verbindung zu einem Warteschlangenmanager” auf Seite 558
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.

„IBM MQ .NET-Programme schreiben und implementieren” auf Seite 573
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.

„Microsoft Windows Communication Foundation-Anwendungen mit IBM MQ entwickeln” auf Seite 1326
Der angepasste WCF-Kanal (WCF = Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

„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 Informationen

Technische Übersicht

Fehlerbehebung bei Problemen in IBM MQ .NET

Erste Schritte mit IBM MQ classes for .NET

Mit IBM MQ classes for .NET kann ein Programm, das mit einem Framework zur .NET-Programmierung geschrieben wurde, eine Verbindung zu IBM MQ als IBM MQ MQI client oder eine direkte Verbindung zu einem IBM MQ-Server herstellen.

Optionen für die Verbindung 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 dazu finden Sie unter „Verwaltete Clientverbindungen” auf Seite 574.

IBM MQ classes for .NET installieren

IBM MQ classes for .NET ist mit integrierten Beispielen in IBM MQ installiert. Microsoft.NET Framework ist als Voraussetzung erforderlich.

Die aktuellste Version von IBM MQ classes for .NET 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.

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, wenn das Support-Pack deinstalliert wurde. Das Feature IBM MQ classes for .NET , das mit IBM MQ installiert wird, enthält dieselbe Funktionalität wie das Support-Pack.

Es werden auch Beispielanwendungen bereitgestellt, einschließlich Quellendateien; siehe „Beispielanwendungen” auf Seite 560.

Zur Ausführung von IBM MQ classes for .NET auf 32-Bit- oder 64-Bit-Plattformen muss Microsoft.NET Framework V3.5 oder höher installiert sein.

Anmerkung: Wenn Microsoft.NET Framework V3.5 oder höher nicht vor der Installation von IBM MQ 8.0 installiert ist, wird die Installation des IBM MQ-Produkts ohne Fehler fortgesetzt, aber die IBM MQ classes for .NET sind nicht verfügbar. Wenn .NET Framework nach der Installation von IBM MQ 8.0installiert wird, müssen die IBM MQ.NET -Assemblys registriert werden, indem das Script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` ausgeführt wird. Dabei ist *WMQInstallDir* das Verzeichnis, in dem IBM MQ 8.0 installiert ist. Mit diesem Script werden die erforderlichen Assemblys im Global Assembly Cache (GAC) installiert. Im Verzeichnis %TEMP% wird eine Gruppe von `amqi*.log`-Dateien erstellt, in denen die ausgeführten Aktionen dokumentiert werden.

Informationen zur Verwendung des angepassten IBM MQ-Kanals für Microsoft WCF mit .NET 3 finden Sie im Abschnitt „Microsoft Windows Communication Foundation-Anwendungen mit IBM MQ entwickeln” auf Seite 1326.

Beispielanwendungen

Um Ihre eigenen .NET-Anwendungen auszuführen, verwenden Sie die Anweisungen für die Prüfprogramme, mit dem die Namen der Beispielanwendungen durch Ihren Anwendungsnamen ersetzt werden.

Es werden fünf Beispielanwendungen 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.

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

Zum Erstellen der Beispielanwendungen wurde für jede Sprache eine Batchdatei 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
```

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 im Abschnitt 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” auf Seite 564
- 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 neuen 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 568

Dezentrale Transaktionsverarbeitung im verwalteten Modus

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

IBM MQ.NET-Klassen unterstützten nicht verwaltete Verbindungen (Client) über den erweiterten Transaktionsclient und COM+/MTS als Transaktionskoordinator. Dabei wird das Programmiermodell für die implizite 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 Assembly im 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 `WebSphere MQ\tools\dotnet\samples\cs\base`

Vorgehensweise

Die Anwendungen können mit den Befehlszeilenparameter 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 den Parameter für eine Warteschlange als `Warteschlange://Warteschlangenname` und für ein Thema als `Thema://Themenname` an.

`-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
```

Wiederherstellung von Transaktionen

In diesem Abschnitt wird der Prozess zum Wiederherstellen von Transaktionen in IBM MQ .NET XA im verwalteten Modus beschrieben.

Übersicht

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 oder 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.

Wiederherstellungsmodell

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 Transaktionsmanager 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 wieder-

herzustellen. 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 an.

1. Protokollierung der Wiederherstellungsinformationen für die Transaktion.
   - 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.
2. Wiederherstellung von Transaktionen mit der Überwachungsanwendung 'WmqDotnetXAMonitor'.
   - Bei 'WmqDotnetXAMonitor' handelt es sich um eine verwaltete .NET-Anwendung, die Nachrichten in der Warteschlange SYSTEM.DOTNET.XARECOVERY.QUEUE verarbeitet und unvollständige Transaktionen wiederherstellt.

   Wenn der MCA die Nachricht nicht in die Zielwarteschlange einlegen kann, generiert er einen Ausnahmebericht, der die ursprüngliche Nachricht enthält, und stellt ihn in eine Übertragungswarteschlange, die an die in der ursprünglichen Nachricht angegebene Warteschlange für Antwortnachrichten gesendet werden soll. (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.)

SYSTEM.DOTNET.XARECOVERY.QUEUE

Hierbei handelt es sich um eine Systemwarteschlange mit Informationen zur Wiederherstellung unvollständiger Transaktionen. Diese Warteschlange wird erstellt, wenn ein Warteschlangenmanager erstellt wird.

Anmerkung: Die Warteschlange SYSTEM.DOTNET.XARECOVERY.QUEUE sollte nicht gelöscht werden.

WMQDotnetXAMonitor Application

Die XA-Überwachungsanwendung für IBM MQ .NET überwacht einen gegebenen Warteschlangenmanager und führt eine Wiederherstellung von unvollständigen Transaktionen aus, falls diese vorhanden sind. Die folgenden Transaktionen gelten als unvollständig und werden wiederhergestellt:

Unvollständige Transaktionen

- 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 Überwachungsanwendung muss aus dem gleichen System ausgeführt werden, auf dem Ihre IBM MQ .NET-Clientanwendung aktiv ist. Wenn Anwendungen, die auf verschiedenen System ausgeführt werden, eine Verbindung zum gleichen Warteschlangenmanager herstellen, muss die Überwachungsanwendung auf allen Systemen ausgeführt werden. Obwohl auf jedem Clientsystem eine Überwachungsanwendung zum Wiederherstellen der Anwendung aktiv ist, sollte jede Überwachungsanwendung in der Lage sein, die Nachricht zu ermitteln, die einer Transaktion zugeordnet ist, die vom lokalen MS-DTC der aktuellen Überwachungsanwendung koordiniert wird, damit diese wieder eingetragen und abgeschlossen werden kann.

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 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 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 WMQ-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 Überwachungsanwendung wiederhergestellt und abgeschlossen.

  In diesem Anwendungsfall wird eine einzelne Überwachungsanwendung ausgeführt, da in den Transaktionen ein Warteschlangenmanager verwendet wird.
- **Mehrere IBM MQ-Anwendungen, mehrere DTC-Dienste und unterschiedliche Warteschlangenmanagerinstanzen:** In diesem Anwendungsfall sind mehrere WMQ-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 Überwachungsanwendung ausgeführt, da jeder Client über einen eigenen Warteschlangenmanager verfügt, der in Transaktionen verwendet wird.
- **Mehrere IBM MQ-Anwendungen, mehrere DTC-Dienste, mehrere Warteschlangenmanagerinstanzen:** In diesem Anwendungsfall sind mehrere WMQ-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 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 WMQ-Anwendungen in einem einzelnen DTC-Dienst vorhanden (d. h. auf einem Computer werden mehrere WMQ-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 Überwachungsanwendung nicht im Hintergrund ausgeführt wird, können Sie diese starten.

Anwendung 'WMQDotnetXAMonitor' verwenden
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.

Starten Sie die Überwachungsanwendung mit folgendem Befehl:

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
```

Dabei gilt:

- **-m *Warteschlangenmanagername***
  Warteschlangenmanagername.

  Optional

  -n *Verbindungsname*
  Verbindungsname im Host- (Port-)Format. Es können mehrere Verbindungsnamen enthalten sein. Mehrere Verbindungsnamen müssen in einer durch Kommas getrennten Liste angegeben werden, z. B. `localhost (1414), localhost (1415), localhost (1416)`. Die Überwachungsanwendung führt die Wiederherstellung für jeden Verbindungsnamen aus, der in der durch Kommas getrennten Liste angegeben ist.

  -c *ChannelName*
  Der Kanalname.

  -i
  Abschluss von heuristischen Verzweigungen.

  Optional

Die Überwachungsanwendung 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 die XA-Überwachung die Warteschlange auf Nachrichten und prüft, ob die Nachricht die Kriterien der unvollständigen Transaktion erfüllt.
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 den lokalen MS-DTC beziehen. Wenn dies zutrifft, wird mit dem Wiederherstellen der Transaktion fortgefahren. Andernfalls wird die nächste Nachricht durchsucht.
5. Die Überwachungsanwendung löst anschließend Aufrufe an den Warteschlangenmanager aus, um die unvollständige Transaktion wiederherzustellen.

Einstellungen in der Anwendungskonfigurationsdatei für WmqDotNETXAMonitor
Zur Überwachung der Anwendung können mithilfe der Anwendungskonfigurationsdatei Eingaben bereitgestellt werden. Ein Beispiel für eine Anwendungskonfigurationsdatei wird mit IBM MQ .NET geliefert. Diese Datei kann für Ihre Anforderungen geändert werden.

Bei der Berücksichtigung der Eingabewerte hat die Anwendungskonfigurationsdatei die höchste Priorität. Wenn Eingabewerte in der Befehlszeile und in der Anwendungskonfigurationsdatei bereitgestellt werden, werden die Werte aus der Anwendungskonfigurationsdatei berücksichtigt.

Konfigurationsdatei der Beispielanwendung

```
<?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>
```

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.

In der IBM MQ-Dokumentation sind nur Informationen zu den Programmiersprachen C#, C++ und Visual Basic enthalten.

In dieser Themengruppe finden Sie Informationen, die Ihnen beim Schreiben von Anwendungen für die Interaktion mit IBM MQ-Systemen helfen. 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.

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 Client mithilfe einer Konfigurationsdatei konfigurieren.

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 75. 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
MessageBuffer	UpdatePercentage
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

Sie können diese Attribute mithilfe der entsprechenden Umgebungsvariable überschreiben.

Anwendungskonfigurationsdatei

Wenn Sie die Anwendung über eine verwaltete Verbindung ausführen, können Sie mit der .NET-Anwendungskonfigurationsdatei auch die IBM MQ-Clientkonfigurationsdatei und die entsprechenden Umgebungsvariablen 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. Das

folgende Beispiel zeigt die Abschnitte, Schlüssel und Werte, die in einer .NET -Anwendungskonfigurationsdatei zur Steuerung der **TCP/IP KeepAlive** -Eigenschaft 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, 2023
// ============================================================================
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

Umfassende Informationen zur Klasse 'MQEnvironment' enthält der Abschnitt .NET-Klasse `MQEnvironment`.

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 Berechtigung zum Abfragen (inq) im 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. Beispiel:

```
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 (mit ReadUTF()), ist dies die einfachste Methode, um 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
- Int
- Int2
- Int4
- Int8
- Long
- Object
- 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 und dabei gleichzeitig 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 Definitionstabelle für den Clientkanal (Client Channel Definition Table, CCDT) mit den .NETKlassen für IBM MQ verwenden. 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 das Attribut 'ChannelDefinitionDirectory', um das Verzeichnis anzugeben, in dem sich die Tabelle befindet, und das Attribut 'ChannelDefinitionFile' zur Angabe des Dateinamens.

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 den Schlüssel 'ChannelDefinitionDirectory', um das Verzeichnis anzugeben, in dem sich die Tabelle befindet, und den Schlüssel 'ChannelDefinitionFile' zur Angabe des Dateinamens.
- 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 das Attribut 'ChannelDefinitionDirectory', um das Verzeichnis anzugeben, in dem sich die Tabelle befindet, und das Attribut 'ChannelDefinitionFile' zur Angabe des Dateinamens.

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.

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 1023).

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 585.

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 585.

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 Warteschlangenmanagern, 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 Informationen

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 auf GSKit. Die MQI in der Programmiersprache 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_SHA";
MQEnvironment.SSLKeyRepository = "C:\mqm\key";
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 vorgewie-

sene 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";
```

Dadurch ist die Verbindung nur erfolgreich, wenn der Warteschlangenmanager ein Zertifikat mit einem allgemeinen Namen bereitstellt, der mit 'QMGR.' beginnt, und wenn das Zertifikat mindestens zwei Namen der Organisationseinheit enthält, wobei der erste Name 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.

TLS-Unterstützung für den verwalteten .NET-Client

Der verwaltete .NET-Client verwendet die Microsoft.NET-Frameworkbibliotheken für die Implementierung von TLS-Secure-Socket-Protokollen. 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-Frameworkversion ist .NET Framework V3.5. Die Version der Verschlüsselungsalgorithmusunterstützung basiert auf der von der Anwendung verwendeten .NET-Frameworkversion.

- Für Anwendungen, die auf .NET-Frameworkversion 3.5 und V4.0 basieren, sind die Secure Socket-Protokolle SSL V3.0 und TSL V1.0 verfügbar.
- Für Anwendungen, die auf .NET-Frameworkversion 4.5 basieren, sind die Secure Socket-Protokolle SSL V3.0, TLS V1.1 und TLS V1.2 verfügbar.

Möglicherweise müssen Sie Anwendungen, die eine höhere TLS-Protokollunterstützung erwarten, auf eine höhere Frameworkversion aktualisieren, wie für die Microsoft-Sicherheitsunterstützung im .NET-Framework definiert.

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 591.

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 592 und „CipherSpec-Zuordnungen für den verwalteten .NET-Client" auf Seite 593.

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 595.

Zertifikate
Zur Implementierung der gegenseitigen Authentifizierung von Client und Warteschlangenmanager können Sie selbst signierte TLS-Zertifikate verwenden. Weitere Informationen finden Sie im Abschnitt „Zertifikate für den verwalteten .NET-Client verwenden" auf Seite 595.

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 596.

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
Zwar wird das Zurücksetzen bzw. die Neuvereinbarung des geheimen Schlüssels von der SSLStream-Klasse nicht unterstützt, aus Gründen der Konsistenz mit anderen IBM MQ-Clients lässt der verwaltete .NET-Client die Festlegung von SSLKeyResetCount durch die Anwendungen zu. Weitere Informationen finden Sie im Abschnitt „Zurücksetzen oder Neuvereinbarung des geheimen Schlüssels" auf Seite 596.

Widerrufsprüfung
Die SSLStream-Klasse unterstützt die Überprüfung des Zertifikatswiderrufs, die automatisch von der Zertifikatsverkettungsengine durchgeführt wird. Weitere Informationen finden Sie im Abschnitt „Widerrufsprüfung" auf Seite 597.

Unterstützung für IBM MQ-Sicherheitsexits
Die SSLStream-Klasse bietet eingeschränkte Unterstützung für IBM MQ-Sicherheitsexits. Die Abfrage lokaler und ferner Zertifikate zum Abrufen von SSLPeerNamePtr (registrierter Name des Zertifikatsinhabers) und SSLRemCertIssNamePtr (registrierter Name des Zertifikatsausstellers) ist möglich, da dies in Microsoft.NET unterstützt 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.

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 im Abschnitt „TLS-Unterstützung für den verwalteten .NET-Client" auf Seite 590.

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 593.

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 die vom IBM MQ-Client festgelegte CipherSpec keine aus den Protokollen SSL 3.0, TLS 1.0 und TLS 1.2 ist, würde der von IBM MQ verwaltete .NET-Client standardmäßig mit einer der 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 CipherSpec in der Windows-Gruppenrichtlinie definiert ist, muss dieselbe CipherSpec für den Eigenschaftswert 'SSLCipherSpec' im Kanal SVRCONN sowie in der Anwendung festgelegt werden. Wenn für die Windows-Gruppenrichtlinie der Standardwert eingestellt ist, die Gruppenrichtlinie also nicht zum Zwecke der CipherSpec-Einstellung aktiviert bzw. bearbeitet wurde, müssen die Anwendungen denselben Standardwert der CipherSpec aus der TLS-Konfiguration der Windows-Gruppenrichtlinie auch in der MQEnvironment-Klasse oder in den Hashtabelleneigenschaften des MQQueueManager-Konstruktors festlegen.

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 580
Damit Sie über die Clientverbindung eine Verbindung zu einem Warteschlangenmanager herstellen können, müssen Sie zuerst die IBM MQ-Umgebung einrichten.

Zugehörige Informationen

CipherSpecs angeben

.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 in Abhängigkeit der verwendeten Microsoft.NET-Frameworkversion TLS V1.0 oder TLS V1.2 sein.

Stellen Sie sicher, dass Sie den richtigen SSLCipherSpec-Wert angeben, denn die Angabe eines falschen Werts kann dazu führen, dass die Protokolle SSL 3.0 oder TLS 1.0 verwendet werden.

Tabelle 76. 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
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_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

Anmerkungen:

1. 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.

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 auf das lokale Computerkonto zu, um die Zertifikate abzurufen.

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 Informationen
.NET-MQEnvironment-Klasse
Client sicher mit einem WS-Manager verbinden

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, Wert)`
3. Verwendung des Warteschlangenmanager-Konstruktors `MQQueueManager (String queueManagerName, Hashtable properties)`. Angabe von SSLPEERNAME in den `Hashtabelleneigenschaften` wie bei Option 2.

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 Informationen
.NET-MQEnvironment-Klasse

Zurücksetzen oder Neuvereinbarung des geheimen Schlüssels
Das Zurücksetzen/die Neuvereinbarung des geheimen Schlüssels wird von der SSLStream-Klasse nicht unterstützt. Aus Gründen der Konsistenz mit anderen IBM MQ-Clients lässt der verwaltete IBM MQ.NET-Client die Festlegung von SSLKeyResetCount durch die Anwendungen zu.

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 Verbindungen 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 Informationen
Geheime TLS-Schlüssel zurücksetzen

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) Verwenden Sie zum Erstellen selbst signierter Zertifikate eines der folgenden mit IBM MQ bereitgestellten Tools:

      Verwenden Sie entweder die grafische Benutzerschnittstelle **`strmqikm`** oder **`runmqckm`** bzw. **`runmqakm`** über die Befehlszeile. Weitere Informationen zur Verwendung dieser Tools finden Sie im Abschnitt **`runmqckm`**, **`runmqakm`** und **`strmqikm`** zum Verwalten digitaler Zertifikate verwenden.
   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 mit dem Global Security Kit auf dem Warteschlangenmanager, wie im Abschnitt Sichere Verbindung zwischen Client und Warteschlangenmanager herstellen beschrieben.
   b) Legen Sie die TLS-Attribute des SVRCONN-Kanals fest:
      - Setzen Sie **`SSLCAUTH`** auf `"REQUIRED/OPTIONAL"`.

- Legen Sie für **SSLCIPH** eine entsprechende CipherSpec fest.

Weitere Informationen finden Sie im Abschnitt „TLS für den nicht verwalteten .NET-Client aktivieren” auf Seite 589.

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, abhängig von der verwendeten Microsoft .NET-Frameworkversion, TLS V1.0 oder TLS V1.2 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 einführende Informationen zur Entwicklung von .NET-TLS-Anwendungen, die durch IBM MQ verwaltet werden, finden Sie im Abschnitt „Einfache Anwendung schreiben” auf Seite 599.

Zugehörige Informationen

.NET-MQEnvironment-Klasse

KeyResetCount (MQLONG)

Federal Information Processing Standards (FIPS) für UNIX, 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 597 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

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.

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 77. 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: See Features that can be used only with the primary installation on Windows for important information.

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, 2023.                               */
/*********************************************************************/
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, 2023.                                */
/**********************************************************************/

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, 2023.                                */
/**********************************************************************/

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

Ab IBM MQ 8.0.0 Fix Pack 2 bietet der IBM MQ .NET-Client die Möglichkeit, eine IBM MQ .NET-Assembly zu packen und zu implementieren, ohne dass zur Ausführung Ihrer Anwendungen die vollständige IBM MQ-Clientinstallation auf Produktionssystemen verwendet werden muss.

Informationen zu diesem Vorgang

Ab IBM MQ 8.0.0 Fix Pack 2 können Sie Ihre IBM MQ-.NET-Anwendungen auf einem System erstellen, auf dem der vollständige IBM MQ-Client installiert ist, und die IBM MQ-.NET-Assembly, also `amqmdnet.dll`, später zusammen mit Ihrer Anwendung packen und auf Produktionssystemen implementieren.

Bei den erstellten und implementierten Anwendungen kann es sich um traditionelle Windows .NET-Anwendungen, -Services oder Microsoft Azure Web/Worker-Anwendungen 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 Versionen auf die Assembly `amqmdnet.dll` in Ihrer Anwendung.

Setzen sie die Eigenschaft **CopyLocal** der Assembly `amqmdnet` auf `True`, damit die Assembly `amqmdnet` 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

- Verwenden Sie zum Starten und Stoppen des Trace die Anwendungskonfigurationsdatei sowie eine IBM MQ-spezifische Tracekonfigurationsdatei.

  Anmerkung: Die folgenden Anweisungen zur Generierung eines Trace gelten sowohl für den weiterverteilbaren verwalteten .NET-Client als auch für den .NET-Standalone-Client.

  Sie müssen die Anwendungskonfigurationsdatei und eine IBM MQ -spezifische Tracekonfigurationsdatei verwenden, da es keine vollständige IBM MQ -Clientinstallation gibt, sind die Standardtools zum Starten und Stoppen des Trace **strmqtrc** und **endmqtrc**nicht verfügbar.

  Anwendungskonfigurationsdatei (`app.config` oder `web.config`)

  Anwendungen müssen die Eigenschaft **MQTRACECONFIGFILEPATH** im Abschnitt <appSettings> der Anwendungskonfigurationsdatei, d. h. der Datei `app.config` oder `web.config`, definieren. (Der tatsächliche Name der Anwendungskonfigurationsdatei hängt von dem Namen Ihrer Anwendung ab.) Der Wert der Eigenschaft **MQTRACECONFIGFILEPATH** gibt den Pfad zum Speicherort der IBM MQ-spezifischen Tracekonfigurationsdatei `mqtrace.config` an, wie im folgenden Beispiel dargestellt:

  ```
  <appSettings>
  <add key="MQTRACECONFIGFILEPATH" value="C:\MQTRACECONFIG"/>
  </appSettings>
  ```

  Die Traceerstellung wird deaktiviert, wenn die Datei `mqtrace.config` nicht in dem Pfad gefunden wird, der in der Anwendungskonfigurationsdatei angegeben ist. Es werden allerdings First Failure Support Technology (FFST) und Fehlerprotokolle im Anwendungsverzeichnis erstellt, wenn die Anwendung berechtigt ist, in das aktuelle Verzeichnis zu schreiben.

  IBM MQ-spezifische Tracekonfigurationsdatei (`mqtrace.config`)

  Die Datei `mqtrace.config` ist eine XML-Datei, in der die Eigenschaften für das Starten und Stoppen des Trace, für den Pfad zu den Tracedateien sowie für den Pfad zu den Fehlerprotokollen definiert sind. Diese Eigenschaften sind in der folgenden Tabelle beschrieben.

  Tabelle 78. In der Datei `mqtrace.config` definierte Eigenschaften

Attribut	Beschreibung
MQTRACELEVEL	0: Stoppt die Traceerstellung - dies ist der Standardwert. 1: Die Traceerstellung wird mit weniger Details gestartet. 2: Die Traceerstellung wird mit vollständigen Details gestartet (empfohlen).

Tabelle 78. In der Datei mqtrace.config definierte Eigenschaften (Forts.)	
Attribut	**Beschreibung**
MQTRACEPATH	Verweist auf einen Ordner, in dem die Tracedateien erstellt werden. Wenn der Pfad leer oder das Attribut **MQTRACEPATH** nicht definiert ist, wird das aktuelle Verzeichnis der Anwendung verwendet.
MQERRORPATH	Verweist auf einen Ordner, in dem die Fehlerprotokolldateien erstellt werden. Wenn der Pfad leer oder das Attribut **MQERRORPATH** nicht definiert ist, wird das aktuelle Verzeichnis der Anwendung verwendet.

Im Folgenden sehen Sie ein Beispiel der Datei `mqtrace.config`:

```
<?xml version="1.0" encoding="utf-8"?>
<traceSettings>
  <MQTRACELEVEL>2</MQTRACELEVEL>
  <MQTRACEPATH>C:\MQTRACEPATH</MQTRACEPATH>
  <MQERRORPATH>C:\MQERRORLOGPATH</MQERRORPATH>
</traceSettings>
```

Die Traceerstellung kann dynamisch gestartet und gestoppt werden, wenn eine Anwendung ausgeführt wird, indem der Wert des Attributs **MQTRACELEVEL** in der Datei `mqtrace.config` geändert wird.

Die aktive Anwendung muss für den über das Attribut **MQTRACELEVEL** angegebenen Ordner über die Berechtigung zum Erstellen und Schreiben verfügen, um Tracedateien generieren zu können. Anwendungen, die in einer Microsoft Azure PaaS-Umgebung ausgeführt werden, müssen die gleichen Zugriffsberechtigungen aufweisen, da Webanwendungen, die eine in Microsoft Azure PaaS ausgeführte IBM MQ .NET-Assembly verwenden, möglicherweise nicht über die Berechtigung zum Erstellen und Schreiben verfügen. Die Generierung des Trace, der Erfassung von Fehlerdaten beim ersten Auftreten und der Fehlerprotokolle schlägt fehl, wenn die Anwendung nicht über die erforderliche Berechtigung zum Erstellen und Schreiben für den angegebenen Ordner verfügt.

Bindungsumleitung 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.

  Der folgende Beispielausschnitt in 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, aber später wurde ein Fixpack IBM MQ 8.0.0 Fix Pack 3angewendet, das die IBM MQ.NET -Assembly auf `8.0.0.3`aktualisiert hat.

```
<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

„Anwendung 'WMQDotnetXAMonitor' verwenden” auf Seite 572
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.

Zugehörige Informationen

Komponenten und Funktionen von IBM MQ

Weiterverteilbare Clients

.NET-Anwendungslaufzeit - nur Windows

Component Object Model-Schnittstelle verwenden (IBM MQ Automation Classes for ActiveX)

Bei IBM MQ Automation Classes for ActiveX (MQAX) handelt es sich um ActiveX-Komponenten, mit denen Klassen bereitgestellt werden, die Sie in Ihrer Anwendung für den Zugriff auf IBM MQ verwenden können.

V 9.0.0 Ab IBM MQ 9.0 wird die Unterstützung für MicrosoftActive X nicht mehr verwendet. IBM MQ-Klassen für .NET sind die empfohlene Ersatztechnologie. Weitere Informationen finden Sie unter .NET-Anwendungen entwickeln.

MQAX erfordert eine IBM MQ-Umgebung und eine entsprechende IBM MQ-Anwendung, mit der sie kommuniziert.

Dadurch kann Ihre ActiveX-Anwendung Transaktionen ausführen und auf Daten in einem Ihrer Unternehmenssysteme zugreifen, auf die Sie über IBM MQ zugreifen können.

IBM MQ Automatisierungsklassen für ActiveX:

- Erteilt Ihnen Zugriff auf die Funktionen und Komponenten der IBM MQ-API und berechtigt zur vollen Interkonnektivität mit andern IBM MQ-Plattformen.
- Entspricht den normalen Konvention, die von einer ActiveX-Komponente erwartet werden.
- Entspricht dem IBM MQ-Objektmodell, das auch für .NET, C++, Java und LotusScript verfügbar ist.

Beispiele für den MQAX-Starter werden bereitgestellt. Mit diesen Beispielen können Sie am Anfang überprüfen, ob Ihre Installation von MQAX erfolgreich war und die IBM MQ-Basisumgebung vorhanden ist. Die Beispiele veranschaulichen außerdem, wie MQAX verwendet werden kann.

Scripterstellung für COM und ActiveX

Component Object Model (COM) ist ein objektbasiertes Programmiermodell, das von Microsoft definiert ist. Darin wird angegeben, wie Softwarekomponenten so bereitgestellt werden können, dass sie einander finden und miteinander kommunizieren können, und zwar unabhängig von der Maschinensprache, in der sie geschrieben wurden, oder ihrer Position.

Bei ActiveX handelt es sich um eine Gruppe von Technologien, die auf COM basieren und mit denen Anwendungsentwicklung, wiederverwendbare Komponenten und Internettechnologien auf den Microsoft Windows-Plattformen integriert werden. ActiveX-Komponenten stellen Schnittstellen bereit, auf die Anwendungen dynamisch zugreifen können. Bei einem ActiveX-Scripting-Client handelt es sich um eine Anwendung (z. B. einen Compiler), die ein Programm oder Script erstellen oder ausführen kann, das die von ActiveX- (oder COM-)Komponenten bereitgestellten Schnittstellen verwendet.

Unterstützung der IBM MQ-Umgebung

IBM MQ Automation Classes for ActiveX kann nur mit der **32-Bit**-Version von ActiveX-Scripting-Clients verwendet werden.

Die COM-Komponente kann nur für **32-Bit**-Anwendungen verwendet werden. Wenn Sie eine 64-Bit-Version der COM-Anwendung schreiben möchten, können Sie die .NET-Schnittstelle verwenden.

Für die Ausführung von MQAX in einer IBM MQ-Serverumgebung muss auf Ihrem System Windows 2000 oder höher installiert sein.

Für die Ausführung von MQAX in einer IBM MQ MQI client-Umgebung muss auf Ihrem System IBM MQ MQI client on Windows 2000 oder höher installiert sein:

Der IBM MQ MQI client erfordert Zugriff auf mindestens einen IBM MQ-Server. Wenn der IBM MQ MQI client und der IBM MQ-Server auf Ihrem System installiert sind, werden MQAX-Anwendungen immer auf dem Server ausgeführt. Die ActiveX-Schnittstelle für MQAI ist nur in IBM MQ-Serverumgebungen verfügbar.

Design und Programmierung mithilfe von IBM MQ Automation-Klassen für ActiveX

MQAX-Anwendungen entwerfen, die auf Nicht-ActiveX-Anwendungen zugreifen

In IBM MQ Automation Classes wird der Zugriff auf die Funktionen der IBM MQ-API bereitgestellt. Sie können deshalb alle Vorteile von IBM MQ für Ihre Windows-Anwendung nutzen.

Die allgemeine Konzeption Ihrer Anwendung entspricht der Konzeption jeder IBM MQ-Anwendung, deshalb können Sie alle Konzeptionsaspekte berücksichtigen, die im Abschnitt „Konzepte für den Entwurf von IBM MQ-Anwendungen” auf Seite 48 beschrieben werden.

Zur Verwendung von IBM MQ Automation Classes codieren Sie die Windows-Programme in Ihrer Anwendung in einer Sprache, die die Erstellung und Verwendung von COM-Objekten unterstützt. Dies sind beispielsweise Visual Basic, Java und andere ActiveX-Scripting-Clients. Die Klassen können anschließend ohne großen Aufwand in Ihre Anwendung integriert werden, da die von Ihnen benötigten IBM MQ-Objekte mithilfe der nativen Syntax der Implementierungssprache codiert werden können.

IBM MQ Automation Classes for ActiveX verwenden

Bei der Gestaltung einer ActiveX-Anwendung, die IBM MQ Automation Classes for ActiveX verwendet, handelt es sich bei der Nachricht, die vom fernen IBM MQ-System gesendet oder empfangen wird, um die wichtigste Information. Deshalb müssen Sie das Format der Elemente kennen, die in die Nachricht integriert werden. Damit ein MQAX-Script funktioniert, müssen das Script und die IBM MQ-Anwendung, die die Nachricht aufnimmt oder sendet, die Nachrichtenstruktur kennen.

Wenn Sie eine Nachricht mit einer MQAX-Anwendung senden und die Datenkonvertierung auf der MQAX-Seite ausführen möchten, müssen Sie außerdem die folgenden Informationen kennen:

- Codepage, die vom fernen System verwendet wird
- Codierung, die vom fernen System verwendet wird

Damit Ihr Code übertragbar bleibt, wird empfohlen, die Codepage und die Codierung festzulegen, selbst wenn sie aktuell in sendenden und empfangenden Systemen identisch sind.

Beachten Sie bei der Auswahl der Struktur für die Implementierung des von Ihnen gestalteten Systems, dass Ihre MQAX-Scripts auf der Maschine ausgeführt werden, auf der der IBM MQ-Warteschlangenmanager oder der IBM MQ-Client installiert sind.

Hinweise und Tipps zur Programmierung

Die folgenden Hinweise und Tipps sind in keiner bezeichnenden Reihenfolge. Wenn Sie für Ihre Arbeit relevant sind, können Sie dadurch möglicherweise Zeit sparen.

Eigenschaften des Nachrichtendeskriptors

Bei der Bearbeitung der Eigenschaften des Nachrichtendeskriptors in einem Programm wird empfohlen, die hexadezimalen Entsprechungen der Felder zu verwenden.

Die Informationen in diesem Abschnitt beziehen sich auf die folgenden Eigenschaften:

- AccountingToken
- CorrelationId
- GroupId
- MessageId

Wenn eine Nachricht von einer IBM MQ-Anwendung erstellt wurde und IBM MQ diese Eigenschaften generiert, wird empfohlen, die Eigenschaften AccountingTokenHex, CorrelationIdHex, GroupIdHex und MessageIdHex zu verwenden, um die zugehörige Werte anzuzeigen oder diese zu bearbeiten, einschließlich dem Zurücksenden in einer Nachricht an IBM MQ. Dies liegt daran, dass die von IBM MQ generierten Werte Byte-Zeichenfolgen mit einem Wert von 0 bis einschließlich 255 sind und keine Zeichenfolgen aus druckbaren Zeichen.

Wenn eine Nachricht von Ihrem MQAX-Script erstellt wurde, können Sie die Eigenschaften AccountingToken, CorrelationId, GroupId und MessageId oder deren hexadezimale Entsprechung verwenden.

IBM MQ-Konstanten

IBM MQ-Konstanten werden als Elemente der IBM MQ-Aufzählung in Bibliothek MQAX200 bereitgestellt.

IBM MQ-Zeichenfolgekonstanten

IBM MQ-Zeichenfolgekonstanten sind bei der Verwendung von IBM MQ Automation Classes for ActiveX nicht verfügbar. Für die Konstanten in der folgenden Liste und für andere Konstanten, die Sie möglicherweise benötigen, müssen Sie die explizite Zeichenfolge verwenden. Die Befehle müssen mithilfe von Leerzeichen auf acht Zeichen aufgefüllt werden:

Tabelle 79. IBM MQ-Zeichenfolgekonstanten und die entsprechenden Zeichenfolgen.

Zeichenfolgekonstante	**Entsprechende Zeichenfolge**
MQFMT_NONE	" "
MQFMT_ADMIN	"MQADMIN "
MQFMT_CHANNEL_COMPLETED	"MQCHCOM "
MQFMT_CICS	"MQCICS "
MQFMT_COMMAND_1	"MQCMD1 "
MQFMT_COMMAND_2	"MQCMD2 "
MQFMT_DEAD_LETTER_HEADER	"MQDEAD "
MQFMT_DIST_HEADER	"MQHDIST "
MQFMT_EVENT	"MQEVENT "
MQFMT_IMS	"MQIMS "
MQFMT_IMS_VAR_STRING	"MQIMSVS "
MQFMT_MD_EXTENSION	"MQHMDE "
MQFMT_PCF	"MQPCF "
MQFMT_REF_MSG_HEADER	"MQHREF "

Tabelle 79. IBM MQ-Zeichenfolgekonstanten und die entsprechenden Zeichenfolgen. (Forts.)	
Zeichenfolgekonstante	**Entsprechende Zeichenfolge**
MQFMT_RF_HEADER	"MQHRF "
MQFMT_STRING	"MQSTR "
MQFMT_TRIGGER	"MQTRIG "
MQFMT_WORK_INFO_HEADER	"MQHWIH "
MQFMT_XMIT_Q_HEADER	"MQXMIT "

Konstanten für Nullzeichenfolgen

Die IBM MQ-Konstanten, die für die Initialisierung der vier MQMessage-Eigenschaften MQMI_NONE (24 Nullzeichen), MQCI_NONE (24 Nullzeichen), MQGI_NONE (24 Nullzeichen) und MQACT_NONE (32 Nullzeichen) verwendet werden, werden von IBM MQ Automation Classes for ActiveX nicht unterstützt. Das Festlegen der Konstanten auf eine leere Zeichenfolge hat die gleiche Auswirkung.

So werden beispielsweise die verschiedenen IDs von MQMessage auf diese Werte gesetzt: *mymessage*. **MessageId** = "" *mymessage*. **CorrelationId** = "" *mymessage*. **AccountingToken** = ""

Nachricht aus IBM MQ empfangen

Es gibt mehrere Möglichkeiten, um eine Nachricht aus IBM MQ zu empfangen:

- Abfrage durch einen GET-Aufruf gefolgt von einem WAIT mithilfe der Visual Basic-Funktion TIMER.
- GET-Aufruf mit der WAIT-Option; Sie geben die Wartedauer an, indem Sie die Eigenschaft 'WaitInterval' festlegen. Berücksichtigen Sie dies, wenn die Software, die zu diesem Zeitpunkt ausgeführt wird, nur als Einzelthread ausgeführt wird, selbst wenn Sie Ihr System für die Ausführung in einer Multithread-Umgebung eingerichtet haben. Dadurch wird verhindert, dass Ihr System unendlich gesperrt ist.

  Andere Threads sind davon nicht betroffen. Wenn für Ihre anderen Threads allerdings der Zugriff auf IBM MQ erforderlich ist, benötigen sie eine zweite Verbindung zu IBM MQ mithilfe eines zusätzlichen MQAX-Warteschlangenmanagers und -Warteschlangenobjekts.

Durch das Ausgeben eines GET-Aufrufs mit der WAIT-Option und das Festlegen der Eigenschaft 'WaitInterval' auf MQWI_UNLIMITED ist Ihr System gesperrt, bis der GET-Aufruf abgeschlossen ist, falls es sich bei dem Prozess um einen Einzelthread handelt.

Verwendung der Datenkonvertierung

Von IBM MQ Automation Classes for ActiveX werden zwei Formen der Datenkonvertierung unterstützt: Die numerische Codierung und die Zeichensatzkonvertierung.

Numerische Codierung

Wenn Sie die MQMessage-Codierungseigenschaft festlegen, werden mit den folgenden Methoden Konvertierungen zwischen unterschiedlichen numerischen Codierungssystemen ausgeführt:

- ReadDecimal2
- ReadDecimal4
- ReadDouble, Methode
- Methode 'ReadDouble4'
- Methode 'ReadFloat'
- ReadInt2
- ReadInt4
- Methode 'ReadLong'

- ReadShort, Methode
- Methode 'ReadUInt2'
- WriteDecimal2
- WriteDecimal4
- WriteDouble, Methode
- Methode 'WriteDouble4'
- Methode 'WriteFloat'
- WriteInt2
- WriteInt4
- Methode 'WriteLong'
- WriteShort, Methode
- WriteUInt2, Methode

Die Codierungseigenschaft kann mit den bereitgestellten IBM MQ-Konstanten festgelegt und interpretiert werden. In Abbildung 58 auf Seite 612 wird ein Beispiel dieser Konstanten gezeigt:

```
/* Encodings for Binary Integers */
MQENC_INTEGER_UNDEFINED
MQENC_INTEGER_NORMAL
MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
MQENC_DECIMAL_UNDEFINED
MQENC_DECIMAL_NORMAL
MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
MQENC_FLOAT_UNDEFINED
MQENC_FLOAT_IEEE_NORMAL
MQENC_FLOAT_IEEE_REVERSED
MQENC_FLOAT_S390
```

Abbildung 58. Bereitgestellte IBM MQ-Konstanten zur Codierung

So Senden Sie beispielsweise eine ganze Zahl aus einem Intel-System an ein System/390-Betriebssystem in System/390-Codierung:

```
Dim msg As New MQMessage  'Define an IBM MQ message for our use..
Print msg. Encoding   'Currently 546 (or X'222')
                                          'Set the encoding property
                                           to 785 (or X'311')
msg. Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                        OR MQENC_FLOAT_S390
Print msg. Encoding   'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                          'Set it
msg. WriteLong (local_num)       'Write the number into the message
```

Konvertierung von Zeichensätzen

Die Konvertierung von Zeichensätzen ist erforderlich, wenn Sie eine Nachricht aus einem System an ein anderes System senden, in dem andere Codepages verwendet werden. Die Codepagekonvertierung wird von folgenden Methoden verwendet:

- Methode 'ReadString'
- Methode 'ReadNullTerminatedString'
- WriteString, Methode
- Methode 'WriteNullTerminatedString'
- MessageData-Eigenschaft

Sie müssen die MQMessage-Eigenschaft 'CharacterSet' auf den Wert eines unterstützten Zeichensatzes setzen (CCSID).

In IBM MQ Automation Classes for ActiveX werden Konvertierungstabellen für die Ausführung von Zeichensatzkonvertierungen verwendet.

So werden beispielsweise Zeichenfolgen automatisch in Codepage 437 konvertiert:

```
Dim msg As New MQMessage                'Define an IBM MQ message
msg.CharacterSet = 437                 'Set code page required
msg.WriteString "A character string"'Put character string in message
```

Die WriteString-Methode empfängt die Zeichenfolgedaten (`A character string` im Beispiel) als Unicode-Zeichenfolge. Anschließend werden diese Daten aus Unicode mithilfe der Konvertierungstabelle 34B001B5.TBL in Codepage 437 konvertiert.

Zeichen in der Unicode-Zeichenfolge, die nicht von Codepage 437 unterstützt werden, werden durch das standardmäßige Substitutionszeichen aus Codepage 437 ersetzt.

Bei der Verwendung der ReadString-Methode enthält die eingehende Nachricht entsprechend einen Zeichensatz, der aus dem Wert des IBM MQ-Nachrichtendeskriptors (MQMD) erstellt wurde; es wird eine Konvertierung aus dieser Codepage in Unicode durchgeführt, bevor die Nachricht in Ihre Scripting-Sprache zurückgegeben wird.

Threading

Mit IBM MQ Automation Classes for ActiveX wird ein Free-Threading-Modell implementiert, in dem Objekte zwischen Threads verwendet werden können.

MQAX ermöglicht die Verwendung von MQQueue- und MQQueueManager-Objekten, während in IBM MQ die gemeinsame Nutzung von Handles zwischen verschiedenen Threads derzeit nicht zulässig ist.

Der Versuch, diese in einem anderen Thread zu verwenden, führt zu einem Fehler und IBM MQ gibt den Rückgabecode MQRC_HCONN_ERROR zurück.

Anmerkung: Es gibt nur ein MQSession-Objekt für jeden Prozess. Die Verwendung des MQSession-Beendigungscodes und -Ursachencodes wird in Multithread-Umgebungen nicht empfohlen. Die MQSession-Fehlerwerte werden möglicherweise nach einem Fehler, der im ersten Thread aufgetreten ist und geprüft wurde, von einem zweiten Thread überschrieben. Threads werden für die Dauer jedes Methodenaufrufs oder Eigenschaftenzugriffs serialisiert. Deshalb wird bei der Ausgabe eines GET-Aufrufs mit der WAIT-Option der Zugriff von Threads auf MQAX-Objekte ausgesetzt, bis der Vorgang abgeschlossen ist.

Fehlerbehandlung

In diesen Informationen werden MQAX-Objekteigenschaften, die Bearbeitung von Fehlern, Regeln zur Verarbeitung von ausgelösten Ausnahmen und der Abruf einer Eigenschaft beschrieben.

Jedes MQAX-Objekt enthält Eigenschaften, die Fehlerinformationen und eine Methode enthalten, mit der Fehler zurückgesetzt oder behoben werden können. Dies sind die folgenden Eigenschaften:

- CompletionCode
- ReasonCode
- ReasonName

Folgende Methode wird verwendet:

- ClearErrorCodes

Funktionsweise der Fehlerbehandlung

Ihr MQAX-Script oder Ihre Anwendung ruft eine Methode für ein MQAX-Objekt auf oder greift auf eine Eigenschaft des MQAX-Objekts zu bzw. aktualisiert diese:

1. Der Ursachencode und der Beendigungscode im betroffenen Objekt werden aktualisiert.

2. Der Ursachencode und der Beendigungscode im MQSession-Objekt werden ebenfalls mit den gleichen Informationen aktualisiert.

   Anmerkung: Weitere Informationen zu Einschränkungen bei der Verwendung von MQSession-Fehlercodes in Thread-Anwendungen finden Sie unter „Threading" auf Seite 613.

   Wenn der Beendigungscode größer-gleich der MQSession-Eigenschaft 'ExceptionThreshold' ist, löst MQAX eine Ausnahme aus (Nummer 32000). Verwenden Sie diese innerhalb Ihres Script mit der On Error-Anweisung (oder einer entsprechenden Anweisung), um sie zu verarbeiten.
3. Rufen Sie über die Fehlerfunktion die zugehörige Fehlerzeichenfolge mit folgendem Format ab:

   ```
   MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
   ```

   Weitere Informationen zur Verwendung der On Error-Anweisungen finden Sie in der Dokumentation zu Ihrer ActiveX-Scripting-Sprache.

   Die Verwendung des Beendigungscodes und des Ursachencodes eignet sich für einfache Fehlerbehandlungsroutinen im MQSession-Objekt.

   Die Eigenschaft 'ReasonName' gibt den symbolischen IBM MQ-Namen für den aktuellen Wert des Ursachencodes zurück.

Ausnahmen auslösen

In den folgenden Regeln wird beschrieben, wie das Auslösen von Ausnahmen verarbeitet wird:

- Sobald eine Eigenschaft oder eine Methode den Beendigungscode auf einen Wert größer-gleich dem Ausnahmebedingungsschwellenwert (normalerweise 2) setzt, wird eine Ausnahme ausgelöst.
- Der Beendigungscode wird von allen Methodenaufrufen und Eigenschaftengruppen festgelegt.

Eigenschaft abrufen

Hierbei handelt es sich um einen Sonderfall, da der Beendigungscode und der Ursachencode nicht immer aktualisiert werden:

- Wenn das Abrufen einer Eigenschaft erfolgreich ausgeführt wurde, bleiben der Ursachencode und der Beendigungscode des Objekts und des MQSession-Objekts unverändert.
- Wenn das Abrufen einer Eigenschaft mit dem Beendigungscode für eine Warnung fehlschlägt, bleiben der Ursachencode und der Beendigungscode unverändert.
- Wenn das Abrufen einer Eigenschaft mit dem Beendigungscode für einen Fehler fehlschlägt, werden der Ursachencode und der Beendigungscode aktualisiert, um die tatsächlichen Werte abzubilden, und die Fehlerbehandlung wird wie beschrieben fortgesetzt.

Die MQSession-Klasse enthält die Methode *ReasonCodeName*, die zum Ersetzen eines IBM MQ-Ursachencodes durch einen symbolischen Namen verwendet werden kann. Dies ist insbesondere beim Entwickeln von Programmen hilfreich, in denen unerwartete Fehler auftreten können. Der Name ist allerdings nicht optimal für die Darstellung für Benutzer.

Jede Klasse enthält außerdem die Eigenschaft *ReasonName*, die den symbolischen Namen des aktuellen Ursachencodes für diese Klasse zurückgibt.

Verweis auf IBM MQ Automation Classes for ActiveX

In diesem Abschnitt werden die Klassen von IBM MQ Automation Classes for ActiveX (MQAX) beschrieben, die für ActiveX entwickelt wurden. Mit den Klassen können Sie ActiveX-Anwendungen schreiben, die mit IBM MQ auf andere Anwendungen zugreifen können, welche in Ihrer Nicht-ActiveX-Umgebung ausgeführt werden.

IBM MQ Automation Classes for ActiveX-Schnittstelle

In IBM MQ Automation Classes for ActiveX werden vordefinierte numerische ActiveX-Konstanten (z. B. MQMT_REQUEST) bereitgestellt, die zur Verwendung der Klassen erforderlich sind.

Die ActiveX-Automatisierungsklassen setzen sich aus den folgenden Klassen zusammen:

- „MQSession-Klasse” auf Seite 616
- „MQQueueManager-Klasse” auf Seite 619
- „MQQueue-Klasse” auf Seite 631
- „Klasse MQMessage” auf Seite 647
- „MQPutMessageOptions-Klasse” auf Seite 670
- „Klasse 'MQGetMessageOptions'” auf Seite 672
- „Klasse MQDistributionList” auf Seite 674
- „Klasse MQDistributionListItem” auf Seite 679

Zusätzlich werden in IBM MQ Automation Classes for ActiveX vordefinierte numerische ActiveX-Konstanten (z. B. MQMT_REQUEST) bereitgestellt, die zur Verwendung der Klassen erforderlich sind. Diese werden in der MQ-Aufzählung in der Bibliothek MQAX200 bereitgestellt. Die Konstanten sind eine Untergruppe der in den IBM MQ-C-Headerdateien (cmqc*.h) beschriebenen Konstanten mit einigen zusätzlichen Ursachencodes für IBM MQ Automation Classes for ActiveX.

Informationen zu IBM MQ-Automatisierungsklassen für ActiveX-Klassen

Lesen Sie diese Informationen zusätzlich zu den Referenzabschnitten unter Anwendungsreferenz entwickeln.

See Features that can be used only with the primary installation on Windows for important information.

Die Klasse 'MQSession' stellt ein Stammobjekt zur Verfügung, das den Status der Aktion enthält, die zuletzt für ein MQAX-Objekt durchgeführt wurde. Weitere Informationen finden Sie im Abschnitt „Fehlerbehandlung” auf Seite 613.

Die Klassen 'MQQueueManager' und 'MQQueue' ermöglichen den Zugriff auf die zugrunde liegenden IBM MQ-Objekte. Die Methoden oder Eigenschaftszugriffe für diese Klassen resultieren aus Aufrufen, die über die IBM MQ-MQI ausgeführt werden.

Die Klassen 'MQMessage', 'MQPutMessageOptions' und 'MQGetMessageOptions' binden die Datenstrukturen MQMD, MQPMO und MQGMO ein und unterstützen Sie beim Senden von Nachrichten an bzw. beim Abrufen von Nachrichten aus Warteschlangen.

Die Klasse 'MQDistributionList' bindet mehrere Warteschlangen ein - lokale, ferne oder Aliaswarteschlangen für die Ausgabe. Die Klasse 'MQDistributionListItem' bindet die Strukturen MQOR, MQRR und MQPMR ein und verknüpft sie mit einer übergeordneten Verteilerliste.

Parameterübergabe

Die Parameter in Methodenaufrufen werden alle nach Wert übergeben, ausgenommen, der Parameter ist ein Objekt. In diesem Fall wird er als Referenz übergeben.

Die bereitgestellten Klassendefinitionen führen den Datentyp für jeden Parameter bzw. jede Eigenschaft auf. Wenn die verwendete Variable nicht den erforderlichen Typ hat, wird bei vielen ActiveX-Clients wie z. B. Visual Basic der Wert automatisch in oder aus dem erforderlich Typ konvertiert, falls eine solche Konvertierung möglich ist. Dies hängt von den Standardregeln des Clients ab; MQAX bietet keine derartige Konvertierung an.

Viele der Methoden verwenden Zeichenfolgeparameter mit fester Länge oder geben eine Zeichenfolge mit fester Länge zurück. Die Konvertierungsregeln lauten wie folgt:

- Gibt der Benutzer als Eingabeparameter oder Rückgabewert eine Zeichenfolge fester Länge an, deren Länge jedoch falsch ist, wird der Wert bei Bedarf abgeschnitten bzw. mit abschließenden Leerzeichen aufgefüllt.
- Gibt der Benutzer als Eingabeparameter eine Zeichenfolge variabler Länge an, deren Länge jedoch falsch ist, wird der Wert bei Bedarf abgeschnitten bzw. mit abschließenden Leerzeichen aufgefüllt.
- Gibt der Benutzer als Rückgabewert eine Zeichenfolge variabler Länge an, deren Länge jedoch falsch ist, wird die Zeichenfolge an die erforderliche Länge angepasst (da bei Rückgabe eines Werts der vorherige Wert in der Zeichenfolge immer gelöscht wird).
- Zeichenfolgen, die als Eingabeparameter angegeben werden, können eingebettete Nullen enthalten.

Diese Klassen finden Sie in der MQAX200-Bibliothek.

Objektzugriffsmethoden

Diese Methoden stehen nicht in direktem Zusammenhang mit einem einzelnen IBM MQ-Aufruf. Jede dieser Methoden erstellt ein Objekt mit Referenzinformationen, gefolgt von der Verbindungsherstellung zu einem IBM MQ-Objekt bzw. dem Öffnen eines solchen Objekts:

Wenn eine Verbindung zu einem Warteschlangenmanager hergestellt wird, enthält sie das von IBM MQ generierte Verbindungskennungsattribut ('connection handle').

Wenn eine Warteschlange geöffnet wird, enthält sie das von IBM MQ generierte Objektkennungsattribut ('object handle').

Diese IBM MQ-Attribute stehen dem Programm MQAX nicht direkt zur Verfügung.

Fehler

Syntaxfehler bei der Parameterübergabe können beim Kompilieren und Ausführen vom ActiveX-Client erkannt werden. Fehler können mit 'On Error' in Visual Basic abgefangen werden.

Alle ActiveX-Klassen von IBM MQ enthalten zwei spezielle schreibgeschützte Eigenschaften: 'Reason-Code' (Ursachencode) und 'CompletionCode' (Beendigungscode). Diese Eigenschaften können jederzeit gelesen werden.

Beim Versuch, auf eine andere Eigenschaft zuzugreifen oder einen Methodenaufruf auszugeben, wird unter Umständen ein IBM MQ-Fehler generiert.

Bei einem erfolgreichen Eigenschaftenzugriff oder einem erfolgreichem Methodenaufruf wird der Ursachencode des übergeordneten Objekts auf MQRC_NONE gesetzt, der Beendigungscode wird auf MQCC_OK gesetzt.

Schlägt der Eigenschaftenzugriff oder der Methodenaufruf fehl, werden Ursachen- und Beendigungscode in diesen Feldern festgelegt.

MQSession-Klasse

Dies ist die Stammklasse für IBM MQ Automation Classes for ActiveX.

Pro ActiveX-Clientprozess gibt es nur ein MQSession-Objekt. Beim Versuch, ein zweites Objekt zu erstellen, wird ein zweiter Verweis auf das ursprüngliche Objekt erstellt.

Erstellung

Mit **New** (Neu) wird ein neues MQSession-Objekt erstellt.

Syntax

Dim *mqsess* **As New MQSession Set** *mqsess* = **New MQSession**

Eigenschaften

- „CompletionCode, Eigenschaft” auf Seite 617.
- „Eigenschaft ExceptionThreshold” auf Seite 617.
- „Eigenschaft 'ReasonCode'” auf Seite 618.
- „Eigenschaft ReasonName” auf Seite 618.

Methode

- „AccessGetMessageOptions-Methode” auf Seite 618.
- „AccessMessage-Methode” auf Seite 618.
- „AccessPutMessageOptions-Methode” auf Seite 618.
- „AccessQueueManager-Methode” auf Seite 619.
- „ClearErrorCodes-Methode” auf Seite 619.
- „ReasonCodeName-Methode” auf Seite 619.

CompletionCode, Eigenschaft

Schreibgeschützt. Gibt den IBM MQ-Beendigungscode zurück, der von der aktuellsten Methode oder Eigenschaftengruppe festgelegt wurde, die von einem IBM MQ-Objekt ausgegeben wurde.

Die Eigenschaft wird auf MQCC_OK zurückgesetzt, wenn eine Methode oder Eigenschaftengruppe erfolgreich von einem MQAX-Objekt aufgerufen wurde.

Ein Fehlerereignishandler kann diese Eigenschaft zur Ermittlung des Fehlers untersuchen, ohne zu wissen, welches Objekt beteiligt war.

Die Verwendung des Beendigungscodes und des Ursachencodes eignet gut sich für einfache Fehlerbehandlungsroutinen im MQSession-Objekt.

Anmerkung: Weitere Informationen zu Einschränkungen bei der Verwendung von MQSession-Fehlercodes in Thread-Anwendungen finden Sie unter „Threading” auf Seite 613.

Definiert in:
MQSession-Klasse

Datentyp:
Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax:
Zum Abrufen: *completioncode & = MQSession*.CompletionCode

Eigenschaft ExceptionThreshold

Lese-/Schreibzugriff. Definiert die IBM MQ-Fehlerstufe, für die MQAX eine Ausnahmebedingung auslösen wird. Der Standardwert ist MQCC_FAILED. Ein höherer Wert als MQCC_FAILED verhindert effektiv eine Ausnahmebehandlung und überlässt es dem Programmierer, Prüfungen zum Beendigungscode und zum Ursachencode auszuführen.

Definiert in: MQSession-Klasse

Datentyp: Lang

Werte:

- Alle, aber prüfen Sie vor allem MQCC_WARNING, MQCC_FAILED oder höher.

Syntax:

Zum Abrufen: *ExceptionThreshold& = MQSession*. **ExceptionThreshold**

To set: *MQSession.* **ExceptionThreshold** = *ExceptionThreshold$*

Eigenschaft 'ReasonCode'

Schreibgeschützt. Gibt den Ursachencode zurück, der von der aktuellsten Methode oder Eigenschaftengruppe festgelegt wurde, die von einem IBM MQ-Objekt ausgegeben wurde.

Ein Fehlerereignishandler kann diese Eigenschaft zur Ermittlung des Fehlers untersuchen, ohne zu wissen, welches Objekt beteiligt war.

Die Verwendung des Beendigungscodes und des Ursachencodes eignet gut sich für einfache Fehlerbehandlungsroutinen im MQSession-Objekt.

Anmerkung: Weitere Informationen zu Einschränkungen bei der Verwendung von MQSession-Fehlercodes in Thread-Anwendungen finden Sie unter „Threading" auf Seite 613.

Definiert in: MQSession-Klasse

Datentyp: Lang

Werte:

- Informationen hierzu finden Sie unter Ursache (MQLONG) und in den zusätzlichen MQAX-Werten, die unter „Ursachencodes für IBM MQ Automation Classes for ActiveX" auf Seite 687 aufgeführt sind.

Syntax: Zum Abrufen: *reasoncode & = MQSession*. **ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Gibt den symbolischen Namen des neusten Ursachencodes zurück. beispielsweise MQRC_QMGR_NOT_AVAILABLE.

Anmerkung: Weitere Informationen zu Einschränkungen bei der Verwendung von MQSession-Fehlercodes in Thread-Anwendungen finden Sie unter „Threading" auf Seite 613.

Definiert in: MQSession-Klasse

Datentyp: String

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasonname$ = MQSession* **.ReasonName**

AccessGetMessageOptions-Methode

Erstellt ein neues MQGetMessageOptions-Objekt.

Definiert in:
MQSession-Klasse

Syntax:
gmo = MQSession **.AccessGetMessageOptions()**

AccessMessage-Methode

Erstellt ein neues MQMessage-Objekt.

Definiert in:
MQSession-Klasse

Syntax:
msg = MQSession **.AccessMessage()**

AccessPutMessageOptions-Methode

Erstellt ein neues MQPutMessageOptions-Objekt.

Definiert in:
MQSession-Klasse

Syntax:
pmo = MQSession **.AccessPutMessageOptions()**

AccessQueueManager-Methode

Erstellt ein neues MQQueueManager-Objekt und stellt über den IBM MQ MQI client oder den IBM MQ-Server eine Verbindung zu einem echten Warteschlangenmanager her. Neben der Verbindungsherstellung führt diese Methode auch das Öffnen des Warteschlangenmanagerobjekts aus.

Wenn der IBM MQ MQI client und der IBM MQ-Server auf Ihrem System installiert sind, werden MQAX-Anwendungen standardmäßig auf dem Server ausgeführt. Zur Ausführung der MQAX-Anwendung auf dem Client muss die Clientbindungsbibliothek in der Umgebungsvariable `GMQ_MQ_LIB` angegeben sein, z. B. durch Festlegen von `GMQ_MQ_LIB=mqic.dll`.

Wenn in der Installation nur ein Client verwendet wird, ist das Festlegen der Umgebungsvariable `GMQ_MQ_LIB` nicht erforderlich. Wenn diese Variable nicht festgelegt ist, versucht IBM MQ, die Bibliothek 'amqzst.dll' zu laden. Ist diese DLL nicht vorhanden (wie bei einer Installation, in der nur der Client verwendet wird), versucht IBM MQ, die Bibliothek 'mqic.dll' zu laden.

Wenn der Vorgang erfolgreich war, wird die MQQueueManager-Eigenschaft 'ConnectionStatus' auf TRUE gesetzt.

Ein Warteschlangenmanager kann mit höchstens einem MQQueueManager-Objekt pro ActiveX-Instanz verbunden werden.

Wenn die Verbindung zum Warteschlangenmanager fehlschlägt, wird ein Fehlerereignis ausgelöst und der Ursachencode und Beendigungscode des MQSession-Objekts werden festgelegt.

Definiert in: MQSession-Klasse

Syntax: *set qm = MQSession* **.AccessQueueManager** (*Name$*)

Parameter: *Name$* String. Name des Warteschlangenmanagers, zu dem eine Verbindung hergestellt werden soll.

ClearErrorCodes-Methode

Setzt den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE.

Definiert in: MQSession-Klasse

Syntax:

```
Call MQSession.ClearErrorCodes()
```

ReasonCodeName-Methode

Gibt den Namen des Ursachencodes mit dem angegebenen numerischen Wert zurück. Es ist hilfreich, den Benutzern eindeutigere Meldungen für Fehlerbedingungen zu übergeben. Der Name ist weiterhin eher kryptisch (z. B. ReasonCodeName(2059) ist **MQRC_Q_MGR_NOT_AVAILABLE**), deshalb sollten Fehler möglichst erfasst und durch einen beschreibenden Text ersetzt werden, der für die Anwendung geeignet ist.

Definiert in: MQSession-Klasse

Syntax: *errname $= MQSession* **.ReasonCodeName** (*reasonCode&*)

Parameter: *ursachencode &* Long. Der Ursachencode, für den der symbolische Name erforderlich ist.

MQQueueManager-Klasse

Diese Klasse stellt eine Verbindung zu einem Warteschlangenmanager dar. Der Warteschlangenmanager kann lokal (IBM MQ-Server) oder fern ausgeführt werden, wobei der Zugriff über den IBM MQ-Client bereitgestellt wird. Eine Anwendung muss ein Objekt dieser Klasse erstellen und es mit einem Warteschlangenmanager verbinden. Wird ein Objekt dieser Klasse gelöscht, wird die Verbindung zum zugehörigen Warteschlangenmanager automatisch getrennt.

Einschlussbeziehung

MQQueue-Klassenobjekte sind dieser Klasse zugeordnet.

Durch 'New' wird ein neues MQQueueManager-Objekt erstellt und für alle Eigenschaften werden die Anfangswerte festgelegt. Alternativ können Sie die Methode 'AccessQueueManager' der MQSession-Klasse verwenden.

Erstellung

Durch 'New' wird ein **neues** MQQueueManager-Objekt erstellt und für alle Eigenschaften werden die Anfangswerte festgelegt. Alternativ können Sie die Methode 'AccessQueueManager' der MQSession-Klasse verwenden.

Syntax

Dim *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

Eigenschaften

- „Eigenschaft AlternateUserId” auf Seite 621.
- „AuthorityEvent, Eigenschaft” auf Seite 622.
- „Eigenschaft BeginOptions” auf Seite 622.
- „Eigenschaft ChannelAutoDefinition” auf Seite 622.
- „Eigenschaft ChannelAutoDefinitionEvent” auf Seite 622.
- „ChannelAutoDefinitionExit, Eigenschaft” auf Seite 623.
- „Eigenschaft 'CharacterSet'” auf Seite 623.
- „Eigenschaft 'CloseOptions'” auf Seite 623.
- „CommandInputQueueName, Eigenschaft” auf Seite 623.
- „Eigenschaft CommandLevel” auf Seite 623.
- „CompletionCode, Eigenschaft” auf Seite 624.
- „Eigenschaft ConnectionHandle” auf Seite 624.
- „Eigenschaft ConnectionStatus” auf Seite 624.
- „Eigenschaft ConnectOptions” auf Seite 624.
- „Eigenschaft DeadLetterQueueName” auf Seite 624.
- „DefaultTransmissionQueueName-Eigenschaft” auf Seite 625.
- „Eigenschaft Description” auf Seite 625.
- „DistributionLists, Eigenschaft” auf Seite 625.
- „Eigenschaft InhibitEvent” auf Seite 625.
- „Eigenschaft IsConnected” auf Seite 625.
- „IsOpen, Eigenschaft” auf Seite 626.
- „Eigenschaft LocalEvent” auf Seite 626.
- „Eigenschaft MaximumHandles” auf Seite 626.
- „Eigenschaft MaximumMessageLength” auf Seite 626.

- „MaximumPriority, Eigenschaft” auf Seite 626.
- „MaximumUncommittedMessages, Eigenschaft” auf Seite 627.
- „Name-Eigenschaft” auf Seite 627.
- „ObjectHandle, Eigenschaft” auf Seite 627.
- „Eigenschaft PerformanceEvent” auf Seite 627.
- „Eigenschaft Platform” auf Seite 627.
- „Eigenschaft 'ReasonCode'” auf Seite 627.
- „Eigenschaft ReasonName” auf Seite 628.
- „Eigenschaft RemoteEvent” auf Seite 628.
- „StartStopEvent, Eigenschaft” auf Seite 628.
- „Eigenschaft SyncPointAvailability” auf Seite 628.
- „Eigenschaft TriggerInterval” auf Seite 629.

Methoden

- „AccessQueue-Methode” auf Seite 629.
- „AddDistributionList-Methode” auf Seite 630.
- „Backout-Methode” auf Seite 630.
- „Begin-Methode” auf Seite 630.
- „ClearErrorCodes-Methode” auf Seite 630.
- „Commit-Methode” auf Seite 630.
- „Connect-Methode” auf Seite 630.
- „Disconnect-Methode” auf Seite 631.

Zugriff auf Eigenschaften

Auf die folgenden Eigenschaften kann zu jedem Zeitpunkt zugegriffen werden.

- „Eigenschaft AlternateUserId” auf Seite 621.
- „CompletionCode, Eigenschaft” auf Seite 624.
- „Eigenschaft ConnectionStatus” auf Seite 624.
- „Eigenschaft 'ReasonCode'” auf Seite 627.

Auf die übrigen Eigenschaften kann nur zugegriffen werden, wenn das Objekt mit einem Warteschlangenmanager verbunden und die Benutzer-ID für die Abfrage dieses Warteschlangenmanagers autorisiert ist. Wenn eine alternative Benutzer-ID festgelegt ist und die aktuelle Benutzer-ID für deren Verwendung autorisiert ist, wird stattdessen die alternative Benutzer-ID auf die Berechtigung zur Abfrage überprüft.

Wenn diese Bedingungen nicht gelten, versucht IBM MQ Automation Classes for ActiveX, eine Verbindung zum Warteschlangenmanager herzustellen und öffnet diese automatisch für die Abfrage. Wenn dieser Vorgang nicht erfolgreich ist, gibt der Aufruf den Beendigungscode MQCC_FAILED und einen der folgenden Ursachencode aus:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

Eigenschaft AlternateUserId

Lese-/Schreibzugriff. Die alternative Benutzer-ID, die zum Prüfen des Zugriffs auf die Warteschlangenmanagerattribute verwendet wird.

Diese Eigenschaft darf nicht festlegt werden, wenn 'IsConnected' auf TRUE gesetzt ist.

Diese Eigenschaft kann nicht festgelegt werden, solange das Objekt geöffnet ist.

Defined in: MQQueueManager-Klasse

Data Type: Zeichenfolge von 12 Zeichen

Syntax: To get: *altuser$ = MQQueueManager* **.AlternateUserId** To set: *MQQueueManager* **.AlternateUserId** = *altuser$*

AuthorityEvent, Eigenschaft

Schreibgeschützt. Das MQI-AuthorityEvent-Attribut.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: To get: *authevent = MQQueueManager* **.AuthorityEvent**

Eigenschaft BeginOptions

Lese-/Schreibzugriff. Dies sind die Optionen, die für die Begin-Methode gelten. Ursprünglich MQBO_NONE.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Lang

Werte:

- MQBO_NONE

Syntax: Zum Abrufen: *beginoptions & =MQQueueManager*. **BeginOptions**

Zur Festlegung: *MQQueueManager* **.BeginOptions** = *beginoptions &*

Eigenschaft ChannelAutoDefinition

Schreibgeschützt. Hier wird überprüft, ob automatische Kanaldefinition zulässig ist.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Lang

Werte:

- MQCHAD_DISABLED
- MQCHAD_ENABLED

Syntax: Zum Abrufen: *channelautodef & = MQQueueManager*. **ChannelAutoDefinition**

Eigenschaft ChannelAutoDefinitionEvent

Schreibgeschützt. Hier wird überprüft, ob Ereignisse der automatischen Kanaldefinition erstellt werden.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *channelautodefevent & =MQQueueManager*. **ChannelAutoDefinitionEvent**

ChannelAutoDefinitionExit, Eigenschaft

Schreibgeschützt. Der Name des Benutzerexits, der für automatische Kanaldefinition verwendet wird.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Zeichenfolge

Syntax: To get: *channelautodefexit$= MQQueueManager*. **ChannelAutoDefinitionExit**

Eigenschaft 'CharacterSet'

Schreibgeschützt. Das MQI-CodedCharSetId-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *characterset & = MQQueueManager* **.CharacterSet**

Eigenschaft 'CloseOptions'

Lese-/Schreibzugriff. Optionen, die verwendet werden, um zu steuern, was passiert, wenn der Warteschlangenmanager geschlossen ist. Der Anfangswert ist MQCO_NONE.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Lang

Werte:

- MQCO_NONE

Syntax: Zum Abrufen: *closeopt & = MQQueueManager* **.CloseOptions**

Zum Festlegen: *MQQueueManager* **.CloseOptions** = *closeopt &*

CommandInputQueueName, Eigenschaft

Schreibgeschützt. Das MQI-CommandInputQName-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *commandinputqname$ = MQQueueManager* **.CommandInputQueueName**

Eigenschaft CommandLevel

Schreibgeschützt. Gibt die Version und die Stufe der IBM MQ-Warteschlangenmanager-Implementierung zurück (MQI-CommandLevel-Attribut)

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *level & = MQQueueManager* **.CommandLevel**

CompletionCode, Eigenschaft

Schreibgeschützt. Gibt den Beendigungscode zurück, der beim letzten Zugriff auf die Methode oder die Eigenschaft, der bei diesem Objekt ausgeführt wurde, eingestellt wurde.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *completioncode & = MQQueueManager* **.CompletionCode**

Eigenschaft ConnectionHandle

Schreibgeschützt. Die Verbindungskennung für das IBM MQ-Warteschlangenmanager-Objekt.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Lang

Syntax: Zum Abrufen: *hconn & = MQQueueManager.* **ConnectionHandle**

Eigenschaft ConnectionStatus

Schreibgeschützt. Gibt an, ob das Objekt mit seinem Warteschlangenmanager verbunden ist oder nicht.

Definiert in: MQQueueManager-Klasse

Datentyp: Boolesch

Werte:

- Wahr/TRUE (-1)
- Falsch/FALSE (0)

Syntax: To get: *status = MQQueueManager* **.ConnectionStatus**

Eigenschaft ConnectOptions

Lese-/Schreibzugriff möglich. Diese Optionen finden bei der Connect-Methode Anwendung. Anfangswert: MQCNO_NONE.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Lang

Werte:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE

Syntax: Zum Abrufen: *connectoptions & =MQQueueManager*. **ConnectOptions**

Zum Festlegen: *MQQueueManager* **.ConnectOptions** = *connectoptions &*

Eigenschaft DeadLetterQueueName

Schreibgeschützt. Das MQI-DeadLetterQName-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: to get: *dlqname$ = MQQueueManager* **.DeadLetterQueueName**

DefaultTransmissionQueueName-Eigenschaft

Schreibgeschützt. Das MQI-DefXmitQName-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *defxmitqname$ = MQQueueManager* **.DefaultTransmissionQueueName**

Eigenschaft Description

Schreibgeschützt. Das MQI-QMgrDesc-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Zeichenfolge von 64 Zeichen

Syntax: To get: *description$ = MQQueueManager* **.Description**

DistributionLists, Eigenschaft

Schreibgeschützt. Hierbei handelt es sich um die Fähigkeit des Warteschlangenmanagers, Verteilerlisten zu unterstützen.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Boolesch

Werte:

- Wahr/TRUE (-1)
- Falsch/FALSE (0)

Syntax: Zum Abrufen: *distributionlists= MQQueueManager*. **DistributionLists**

Eigenschaft InhibitEvent

Schreibgeschützt. Das MQI-InhibitEvent-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *inhibevent & = MQQueueManager* **.InhibitEvent**

Eigenschaft IsConnected

Ein Wert, der angibt, ob der Warteschlangenmanager aktuell verbunden ist.

Schreibgeschützt.

Definiert in: MQQueueManager-Klasse

Datentyp: Boolesch

Werte:

- Wahr/TRUE (-1)
- Falsch/FALSE (0)

Syntax: To get: *isconnected = MQQueueManager* **.IsConnected**

IsOpen, Eigenschaft

Ein Wert, der angibt, ob der Warteschlangenmanager aktuell für Abfragen bereit ist.

Schreibgeschützt.

Definiert in:
MQQueueManager-Klasse

Datentyp:
Boolesch

Werte:

- Wahr/TRUE (-1)
- Falsch/FALSE (0)

Syntax: Zum Abrufen: *IsOpen = MQQueueManager*. **IsOpen**

Eigenschaft LocalEvent

Schreibgeschützt. Das MQI-LocalEvent-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *localevent & = MQQueueManager* **.LocalEvent**

Eigenschaft MaximumHandles

Schreibgeschützt. Das MQI-MaxHandles-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *maxhandles & = MQQueueManager* **.MaximumHandles**

Eigenschaft MaximumMessageLength

Schreibgeschützt. Das MQI-MaxMsgLength Queue Manager-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *maxmessagelength & = MQQueueManager* **.MaximumMessageLänge**

MaximumPriority, Eigenschaft

Schreibgeschützt. Das MQI-MaxPriority-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *maxpriority & = MQQueueManager* **.MaximumPriority**

MaximumUncommittedMessages, Eigenschaft

Schreibgeschützt. Das MQI-MaxUncommittedMsgs-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *maxuncommitted & = MQQueueManager* **.MaximumUncommittedNachrichten**

Name-Eigenschaft

Lese-/Schreibzugriff. Das MQI-Attribut 'QMgrName'. Diese Eigenschaft kann nicht geschrieben werden, wenn eine Verbindung mit MQQueueManager hergestellt wurde.

Definiert in: MQQueueManager-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *name$ = MQQueueManager* **.name**

To set: *MQQueueManager* **.name** = *name$*

Anmerkung: In Visual Basic ist die Eigenschaft "Name" für die Verwendung in der grafischen Schnittstelle reserviert. Verwenden Sie deshalb in Visual Basic die Kleinschreibung, also "name".

ObjectHandle, Eigenschaft

Schreibgeschützt. Die Objektkennung für das IBM MQ-Warteschlangenmanager-Objekt.

Definiert in:
MQQueueManager-Klasse

Datentyp
Lang

Syntax: Zum Abrufen: *hobj & = MQQueueManager.* **ObjectHandle**

Eigenschaft PerformanceEvent

Schreibgeschützt. Das MQI-PerformanceEvent-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *perfevent & = MQQueueManager*.PerformanceEvent

Eigenschaft Platform

Schreibgeschützt. Das MQI-Platform-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQPL_WINDOWS_NT
- MQPL_WINDOWS

Syntax: Zum Abrufen: *platform & = MQQueueManager* **Plattform**

Eigenschaft 'ReasonCode'

Schreibgeschützt. Gibt den Ursachencode zurück, der von dem zuletzt für das Objekt ausgegebenen Methodenaufruf bzw. Eigenschaftenzugriff gesetzt wurde.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasoncode & = MQQueueManager* **.ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Gibt den symbolischen Namen des neusten Ursachencodes zurück. beispielsweise MQRC_QMGR_NOT_AVAILABLE.

Definiert in: MQQueueManager-Klasse

Datentyp: String

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: To get: *reasonname$ = MQQueueManager* **.ReasonName**

Eigenschaft RemoteEvent

Schreibgeschützt. Das MQI-RemoteEvent-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *remoteevent & = MQQueueManager* **.RemoteEvent**

StartStopEvent, Eigenschaft

Schreibgeschützt. Das MQI-StartStopEvent-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *strstpevent & = MQQueueManager* **.StartStop-Ereignis**

Eigenschaft SyncPointAvailability

Schreibgeschützt. Das MQI-SyncPoint-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Werte:

- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

Syntax: Zum Abrufen: *syncpointavailability & = MQQueueManager* **.SyncPointVerfügbarkeit**

Eigenschaft TriggerInterval

Schreibgeschützt. Das MQI-TriggerInterval-Attribut.

Definiert in: MQQueueManager-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *trigint & = MQQueueManager* **.TriggerInterval**

AccessQueue-Methode

Erstellt ein MQQueue-Objekt und ordnet es diesem MQQueueManager-Objekt zu, indem die Referenzeigenschaft für die Verbindung der Warteschlange festgelegt wird. Dabei werden die Eigenschaften 'Name', 'OpenOptions', 'DynamicQueueName' und 'AlternateUserId' des MQQueue-Objekts auf die bereitgestellten Werte gesetzt und es wird versucht, das Objekt zu öffnen.

Wenn das Öffnen nicht erfolgreich ist, schlägt der Aufruf fehl. Es wird ein Fehlerereignis für das Objekt ausgegeben. Der Ursachencode und der Beendigungscode sowe der MQSession-Ursachencode und -Beendigungscode des Objekts werden festgelegt.

Die Parameter DynamicQueueName, QueueManagerName und AlternateUserId sind optional und standardmäßig auf "" gesetzt.

Die Option MQOO_INQUIRE zum Öffnen sollte zusätzlich zu anderen Optionen angegeben sein, wenn die Eigenschaften der Warteschlange gelesen werden sollen.

Legen Sie den Parameter QueueManagerName nicht fest oder setzen Sie ihn auf "", wenn eine lokale Warteschlange geöffnet werden soll. Legen Sie andernfalls den Namen des fernen Warteschlangenmanagers fest, der Eigner der Warteschlange ist. Anschließend wird versucht, eine lokale Definition der fernen Warteschlange zu öffnen. Weitere Informationen zur Namensauflösung für ferne Warteschlangen und zu Aliasnamen für Warteschlangenmanager finden Sie unter Was sind Aliasnamen?.

Wenn für die Eigenschaft 'Name' der Name einer Modellwarteschlange festgelegt ist, geben Sie im Parameter 'DynamicQueueName$' den Namen der dynamischen Warteschlange an, die erstellt werden soll. Wenn im Parameter 'DynamicQueueName$' der Wert "" bereitgestellt wird, ist im Warteschlangenobjekt der Wert "AMQ.*" festgelegt, der auch im Aufruf zum Öffnen verwendet wird. Weitere Informationen zur Benennung von dynamischen Warteschlangen finden Sie unter „Dynamische Warteschlangen erstellen” auf Seite 779.

Definition

Definiert in: MQQueueManager-Klasse.

Syntax

Syntax: set queue = MQQueueManager. **AccessQueue** (Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

Parameter

Name$ Zeichenfolge. Name der IBM MQ-Warteschlange.

OpenOptions: Long. Option, die verwendet wird, wenn Warteschlange geöffnet ist. Siehe OpenOptions (MQLONG).

QueueManagerName$ Zeichenfolge. Name des Warteschlangenmanagers, der Eigner der Warteschlange ist, die geöffnet werden soll. Durch den Wert "" wird angezeigt, dass der Warteschlangenmanager lokal ist.

*DynamicQueueName$* Zeichenfolge. Der Name, der der dynamischen Warteschlange beim Öffnen der Warteschlange zugeordnet wurde, wenn der Parameter 'Name$' eine Modellwarteschlange angibt.

AlternateUserId$ Zeichenfolge. Die alternative Benutzer-ID, mit der der Zugriff beim Öffnen der Warteschlange geprüft wird.

AddDistributionList-Methode

Erstellt ein neues MQDistributionList-Objekt und stellt seine Verbindungsreferenz auf den Warteschlangenmanager ein, der Eigner ist.

Definiert in:
MQQueueManager-Klasse

Syntax: *set distributionlist* = **MQQueueManager.** *AddDistributionList*

Backout-Methode

Setzt alle nicht festgeschriebenen Nachrichten-Put- und -Get-Aufrufe zurück, die seit dem letzten Synchronisationspunkt als Teil einer Arbeitseinheit aufgetreten sind.

Definiert in: MQQueueManager-Klasse

Syntax:

```
Call MQQueueManager.Backout()
```

Begin-Methode

Beginnt eine Arbeitseinheit, die vom Warteschlangenmanager koordiniert wird. Die Startoptionen wirken sich auf das Verhalten dieser Methode aus.

Definiert in:
MQQueueManager-Klasse

Syntax:

```
Call MQQueueManager.Begin()
```

ClearErrorCodes-Methode

Setzt für die MQQueueManager-Klasse und die MQSession-Klasse den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE zurück.

Definiert in:
MQQueueManager-Klasse

Syntax:
Call *MQQueueManager* **.ClearErrorCodes()**

Commit-Methode

Schreibt alle Nachrichten-Put- und -Get-Aufrufe fest, die seit dem letzten Synchronisationspunkt als Teil einer Arbeitseinheit aufgetreten sind.

Definiert in: MQQueueManager-Klasse

Syntax:

```
Call MQQueueManager.Commit()
```

Connect-Methode

Stellt eine Verbindung vom MQQueueManager-Objekt zum echten Warteschlangenmanager über den IBM MQ MQI client oder den Server her. Neben dem Herstellen der Verbindung wird mit dieser Methode auch das Warteschlangenmanagerobjekt zur Abfrage geöffnet.

Setzt 'IsConnected' auf TRUE.

Für die Verbindung zu einem Warteschlangenmanager ist maximal ein MQQueueManager-Objekt pro ActiveX-Instanz zulässig.

Definiert in: MQQueueManager-Klasse

Syntax:

```
Call MQQueueManager.Connect()
```

Disconnect-Methode

Trennt das MQQueueManager-Objekt vom Warteschlangenmanager.

Setzt IsConnected auf FALSE (falsch).

Alle Warteschlangenobjekte, die dem MQQueueManager-Objekt zugeordnet sind, werden unbrauchbar gemacht und können nicht erneut geöffnet werden.

Alle nicht festgeschriebenen Änderungen (Nachrichten-Put- und -Get-Aufrufe) werden festgeschrieben.

Definiert in: MQQueueManager-Klasse

Syntax:

```
Call MQQueueManager.Disconnect()
```

MQQueue-Klasse

Diese Klasse stellt den Zugriff auf eine IBM MQ-Warteschlange dar. Diese Verbindung wird durch ein zugeordnetes MQQueueManager-Objekt bereitgestellt. Wird ein Objekt dieser Klasse gelöscht, wird es automatisch geschlossen.

Einschlussbeziehung

Die MQQueue-Klasse ist in der MQQueueManager-Klasse enthalten.

Erstellung

Durch New wird ein neues MQQueue-Objekt erstellt und für alle Eigenschaften werden die Anfangswerte festgelegt. Alternativ können Sie die Methode AccessQueue der MQQueueManager-Klasse verwenden.

Syntax

```
Dim que As New MQQueue Set que = New MQQueue
```

Eigenschaften

- „Eigenschaft AlternateUserId” auf Seite 634.
- „Eigenschaft BackoutRequeueName” auf Seite 634.
- „BackoutThreshold, Eigenschaft” auf Seite 634.
- „Eigenschaft BaseQueueName” auf Seite 634.
- „Eigenschaft 'CloseOptions'” auf Seite 635.

- „CompletionCode, Eigenschaft” auf Seite 635.
- „Eigenschaft ConnectionReference” auf Seite 635.
- „CreationDateTime, Eigenschaft” auf Seite 635.
- „Eigenschaft CurrentDepth” auf Seite 636.
- „DefaultInputOpenOption, Eigenschaft” auf Seite 636.
- „Eigenschaft DefaultPersistence” auf Seite 636.
- „Eigenschaft DefaultPriority” auf Seite 636.
- „DefinitionType, Eigenschaft” auf Seite 636.
- „Eigenschaft DepthHighEvent” auf Seite 636.
- „DepthHighLimit, Eigenschaft” auf Seite 637.
- „Eigenschaft DepthLowEvent” auf Seite 637.
- „Eigenschaft DepthLowLimit” auf Seite 637.
- „DepthMaximumEvent, Eigenschaft” auf Seite 637.
- „Eigenschaft DepthHighEvent” auf Seite 636.
- „DepthHighLimit, Eigenschaft” auf Seite 637.
- „Eigenschaft DepthLowEvent” auf Seite 637.
- „Eigenschaft DepthLowLimit” auf Seite 637.
- „DepthMaximumEvent, Eigenschaft” auf Seite 637.
- „Eigenschaft Description” auf Seite 637.
- „Eigenschaft DynamicQueueName” auf Seite 637.
- „Eigenschaft HardenGetBackout” auf Seite 638.
- „InhibitGet-Eigenschaft” auf Seite 638.
- „InhibitPut-Eigenschaft” auf Seite 638.
- „InitiationQueueName, Eigenschaft” auf Seite 639.
- „IsOpen, Eigenschaft” auf Seite 639.
- „MaximumDepth, Eigenschaft” auf Seite 639.
- „Eigenschaft MaximumMessageLength” auf Seite 639.
- „MessageDeliverySequence, Eigenschaft” auf Seite 639.
- „ObjectHandle, Eigenschaft” auf Seite 640.
- „Eigenschaft OpenInputCount” auf Seite 640.
- „OpenOptions, Eigenschaft” auf Seite 640.
- „Eigenschaft OpenOutputCount” auf Seite 640.
- „Eigenschaft OpenStatus” auf Seite 640.
- „ProcessName, Eigenschaft” auf Seite 641.
- „Eigenschaft 'QueueManagerName'” auf Seite 641.
- „Eigenschaft QueueType” auf Seite 641.
- „Eigenschaft 'ReasonCode'” auf Seite 641.
- „Eigenschaft ReasonName” auf Seite 641.
- „Eigenschaft RemoteQueueManagerName” auf Seite 642.
- „Eigenschaft RemoteQueueName” auf Seite 642.
- „Eigenschaft 'ResolvedQueueManagerName'” auf Seite 642.
- „Eigenschaft ResolvedQueueName” auf Seite 642.
- „Eigenschaft RetentionInterval” auf Seite 642.

- „Eigenschaft Scope” auf Seite 642.
- „Eigenschaft ServiceInterval” auf Seite 643.
- „Eigenschaft ServiceIntervalEvent” auf Seite 643.
- „Eigenschaft Shareability” auf Seite 643.
- „Eigenschaft TransmissionQueueName” auf Seite 643.
- „Eigenschaft TriggerControl” auf Seite 643.
- „Eigenschaft TriggerData” auf Seite 644.
- „Eigenschaft TriggerDepth” auf Seite 644.
- „TriggerMessagePriority, Eigenschaft” auf Seite 644.
- „Eigenschaft TriggerType” auf Seite 644.
- „Eigenschaft Usage” auf Seite 644.

Methoden

- „ClearErrorCodes-Methode” auf Seite 645
- „Close-Methode” auf Seite 645
- „Get-Methode” auf Seite 645
- „Methode Open” auf Seite 646
- „Methode 'Put'” auf Seite 646

Zugriff auf Eigenschaften

Wenn das Warteschlangenobjekt nicht mit einem Warteschlangenmanager verbunden ist, können Sie die folgenden Eigenschaften lesen:

- „CompletionCode, Eigenschaft” auf Seite 635
- „Eigenschaft OpenStatus” auf Seite 640
- „Eigenschaft 'ReasonCode'” auf Seite 641

Sie können außerdem folgende Eigenschaften lesen und schreiben:

- „Eigenschaft AlternateUserId” auf Seite 634
- „Eigenschaft 'CloseOptions'” auf Seite 635
- „Eigenschaft ConnectionReference” auf Seite 635
- „Name-Eigenschaft” auf Seite 639
- „OpenOptions, Eigenschaft” auf Seite 640

Wenn das Warteschlangenobjekt mit einem Warteschlangenmanager verbunden wird, können Sie alle Eigenschaften lesen.

Eigenschaften von Warteschlangenattributen

Eigenschaften, die nicht im vorigen Abschnitt aufgeführt sind, sind Attribute der zugrundeliegenden IBM MQ-Warteschlange. Auf sie kann nur zugegriffen werden, wenn das Objekt mit einem Warteschlangenmanager verbunden ist und die Benutzer-ID des Benutzers für die Abfrage oder das Festlegen dieser Warteschlange autorisiert ist. Wenn eine alternative Benutzer-ID festgelegt ist und die aktuelle Benutzer-ID für deren Verwendung autorisiert ist, wird stattdessen die alternative Benutzer-ID auf die Berechtigung überprüft.

Bei der Eigenschaft muss es sich um eine geeignete Eigenschaft für den angegebenen Warteschlangentyp handeln. Weitere Informationen finden Sie im Abschnitt Attribute für Warteschlangen.

Wenn diese Bedingungen nicht gelten, wird durch den Zugriff auf die Eigenschaft der Beendigungscode MQCC_FAILED und einer der folgenden Ursachencodes ausgegeben:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode ist MQCC_WARNING)

Warteschlange öffnen

Die einzige Möglichkeit zum Erstellen eines MQQueue-Objekts ist die Verwendung der MQQueueManager-Methode 'AccessQueue' oder über 'New'. Ein geöffnetes MQQueue-Objekt bleibt geöffnet (OpenStatus=TRUE), bis es geschlossen oder gelöscht wird oder bis das erstellende Warteschlangenmanagerobjekt gelöscht wird oder die Verbindung zum Warteschlangenmanager nicht mehr vorhanden ist. Durch den Wert der MQQueue-Eigenschaft 'CloseOptions' wird das Verhalten der Schließoperation beim Löschen des MQQueue-Objekts gesteuert.

Die MQQueueManager-Methode 'AccessQueue' öffnet die Warteschlange mit dem Parameter 'OpenOptions'. Die MQQueue.Open-Methode öffnet die Warteschlange mit der Eigenschaft 'OpenOptions'. IBM MQ prüft den Parameter 'OpenOptions' auf die Benutzerberechtigung als Teil des Prozesses zum Öffnen der Warteschlange.

Eigenschaft AlternateUserId

Lese-/Schreibzugriff. Die alternative Benutzer-ID, die verwendet wird, um den Zugriff auf die Warteschlange zu überprüfen, wenn sie geöffnet wird.

Diese Eigenschaft kann nicht eingestellt werden, während das Objekt geöffnet ist (d. h. wenn IsOpen auf TRUE (wahr) steht).

Definiert in: MQQueue-Klasse

Datentyp: Zeichenfolge von 12 Zeichen

Syntax: To get: *altuser$ = MQQueue* **.AlternateUserId**

Zum Festlegen: *MQQueue.* **AlternateUserId** = *altuser$*

Eigenschaft BackoutRequeueName

Schreibgeschützt. Das MQI-BackOutRequeueQName-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *backoutrequeuename$ = MQQueue* **.BackoutRequeueName**

BackoutThreshold, Eigenschaft

Schreibgeschützt. Das MQI-BackoutThreshold-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- Siehe BackoutThreshold (MQLONG).

Syntax: Zum Abrufen: *backoutthreshold & = MQQueue.* **BackoutThreshold**

Eigenschaft BaseQueueName

Schreibgeschützt. Der Name der Warteschlange, in den der Aliasname aufgelöst wird.

Nur gültig für Aliaswarteschlangen.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *baseqname$ = MQQueue* **.BaseQueueName**

Eigenschaft 'CloseOptions'

Lese-/Schreibzugriff möglich. Optionen, die verwendet werden, um zu steuern, was passiert, wenn die Warteschlange geschlossen ist.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

MQCO_DELETE und MQCO_DELETE_PURGE sind nur für dynamische Warteschlangen gültig.

Syntax: Zum Abrufen: *closeopt & = MQQueue* **.CloseOptions**

Zum Festlegen: *MQQueue* **.CloseOptions** = *closeopt &*

CompletionCode, Eigenschaft

Schreibgeschützt. Gibt den Beendigungscode zurück, der beim letzten Zugriff auf die Methode oder die Eigenschaft, der bei diesem Objekt ausgeführt wurde, eingestellt wurde.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *completioncode & = MQQueue* **.CompletionCode**

Eigenschaft ConnectionReference

Lese-/Schreibzugriff. Definiert das Warteschlangenmanagerobjekt, dem ein Warteschlangenobjekt zugeordnet ist. Die Verbindungsreferenz kann nicht geschrieben werden, während eine Warteschlange geöffnet ist.

Definiert in: MQQueue-Klasse

Datentyp: MQQueueManager

Werte:

- Ein Verweis auf ein aktives IBM MQ-Warteschlangenmanagerobjekt

Syntax: To set: *set MQQueue* **.ConnectionReference** = *ConnectionReference*

To get: *set ConnectionReference = MQQueue* **.ConnectionReference**

CreationDateTime, Eigenschaft

Schreibgeschützt. Datum und Uhrzeit der Erstellung dieser Warteschlange.

Definiert in: MQQueue-Klasse

Datentyp: Variante von Typ 7 (Datum/Uhrzeit) oder EMPTY (leer)

Syntax: To get: *datetime = MQQueue* **.CreationDateTime**

Eigenschaft CurrentDepth

Schreibgeschützt. Die Anzahl der Nachrichten, die sich derzeit in der Warteschlange befinden.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *currentdepth & = MQQueue* **.CurrentDepth**

DefaultInputOpenOption, Eigenschaft

Schreibgeschützt. Steuert, in welchem Modus die Warteschlange geöffnet wird, wenn in den Open-Optionen MQOO_INPUT_AS_Q_DEF angegeben wird.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_SHARED

Syntax: Zum Abrufen: *defaultinop & = MQQueue* **.DefaultInputOpenOption**

Eigenschaft DefaultPersistence

Schreibgeschützt. Die Standardpersistenz für Nachrichten in einer Warteschlange.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *defpersistence & = MQQueue* **.DefaultPersistence**

Eigenschaft DefaultPriority

Schreibgeschützt. Die Standardpriorität für Nachrichten in einer Warteschlange.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *defpriority & = MQQueue* **.DefaultPriority**

DefinitionType, Eigenschaft

Schreibgeschützt. Typ der Warteschlangendefinition

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQQDT_PREDEFINED
- MQQDT_PERMANENT_DYNAMIC
- MQQDT_TEMPORARY_DYNAMIC

Syntax: Zum Abrufen: *deftype & = MQQueue* **.DefinitionType**

Eigenschaft DepthHighEvent

Schreibgeschützt. Das MQI-QDepthHighEvent-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *depthhighevent & = MQQueue.* **DepthHighEvent**

DepthHighLimit, Eigenschaft

Schreibgeschützt. Das MQI-QDepthHighLimit-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *depthhighlimit & = MQQueue.* **DepthHighLimit**

Eigenschaft DepthLowEvent

Schreibgeschützt. Das MQI-QDepthLowEvent-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *depthlowevent & = MQQueue.* **DepthLowEvent**

Eigenschaft DepthLowLimit

Schreibgeschützt. Das MQI-QDepthLowLimit-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *depthlowlimit & = MQQueue.* **DepthLowLimit**

DepthMaximumEvent, Eigenschaft

Schreibgeschützt. Das MQI-QDepthMaxEvent-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntax: Zum Abrufen: *depthmaximumevent & = MQQueue.* **DepthMaximumEvent**

Eigenschaft Description

Schreibgeschützt. Eine Beschreibung der Warteschlange.

Definiert in: MQQueue-Klasse

Datentyp: Zeichenfolge von 64 Zeichen

Syntax: To get: *description$ = MQQueue* **.Description**

Eigenschaft DynamicQueueName

Schreib- und Lesezugriff; schreibgeschützt bei geöffneter Warteschlange.

Mit dieser Eigenschaft wird der dynamische Warteschlangenname gesteuert, der verwendet wird, wenn eine Modellwarteschlange geöffnet ist. Er kann mit einem Platzhalterzeichen vom Benutzer als Eigenschaftengruppe (nur wenn die Warteschlange geschlossen ist) oder als Parameter für MQQueueManager.AccessQueue() festgelegt werden.

Der tatsächliche Name der dynamischen Warteschlange wird durch Abfrage von 'QueueName' abgerufen.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Werte:

- Ein beliebiger gültiger IBM MQ-Warteschlangenname.

Syntax: To set: *MQQueue* **.DynamicQueueName** = *dynamicqueuename$*

To get: *dynamicqueuename$* = *MQQueue* **.DynamicQueueName**

Eigenschaft HardenGetBackout

Schreibgeschützt. Wird ein präziser Rücksetzungszähler gepflegt?

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQQA_BACKOUT_HARDENED
- MQQA_BACKOUT_NOT HARDENED

Syntax: Zum Abrufen: *hardengetback & = MQQueue* **.HardenGetBackout**

InhibitGet-Eigenschaft

Lese-/Schreibzugriff. Das MQI-InhibitGet-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQQA_GET_INHIBITED
- MQQA_GET_ALLOWED

Syntax: Zum Abrufen: *getstatus & = MQQueue* **.InhibitGet**

Zum Festlegen: *MQQueue* **.InhibitGet** = *getstatus &*

InhibitPut-Eigenschaft

Lese-/Schreibzugriff. Das MQI-InhibitPut-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

Syntax: Zum Abrufen: *putstatus & = MQQueue* **.InhibitPut**

Zum Festlegen: *MQQueue* **.InhibitPut** = *putstatus &*

InitiationQueueName, Eigenschaft

Schreibgeschützt. Name der Initialisierungswarteschlange.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *initqname$ = MQQueue* **.InitiationQueueName**

IsOpen, Eigenschaft

Informiert, ob die Warteschlange offen ist.

Schreibgeschützt.

Definiert in: MQQueue-Klasse

Datentyp: Boolesch

Werte:

- Wahr/TRUE (-1)
- Falsch/FALSE (0)

Syntax: To get: *open = MQQueue* **.IsOpen**

MaximumDepth, Eigenschaft

Schreibgeschützt. Gibt die maximale Warteschlangenlänge an.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *maxdepth & = MQQueue* **.MaximumDepth**

Eigenschaft MaximumMessageLength

Schreibgeschützt. Maximale zulässige Nachrichtenlänge in Bytes für diese Warteschlange.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *maxmlength & = MQQueue* **.MaximumMessageLänge**

MessageDeliverySequence, Eigenschaft

Schreibgeschützt. Reihenfolge bei der Nachrichtenübertragung

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQMDS_PRIORITY
- MQMDS_FIFO

Syntax: Zum Abrufen: *messdelseq & = MQQueue* **.MessageDeliveryFolge**

Name-Eigenschaft

Lese-/Schreibzugriff. Das MQI-Warteschlangenattribut. Diese Eigenschaft kann nicht geschrieben werden, nachdem die MQQueue geöffnet wurde.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *name$ = MQQueue* **.name**

To set: *MQQueue* **.name** = *name$*

Anmerkung: In Visual Basic ist die Eigenschaft "Name" für die Verwendung in der grafischen Schnittstelle reserviert. Verwenden Sie deshalb in Visual Basic die Kleinschreibung, also "name".

ObjectHandle, Eigenschaft

Schreibgeschützt. Die Objektkennung für das IBM MQ-Warteschlangenobjekt.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *hobj & = MQQueue.* **ObjectHandle**

Eigenschaft OpenInputCount

Schreibgeschützt. Gibt die Anzahl der Öffnungen zur Eingabe an.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Für Abruf:

```
openincount& = MQQueue.OpenInputCount
```

OpenOptions, Eigenschaft

Lese-/Schreibzugriff. Optionen, die zum Öffnen der Warteschlange verwendet werden.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- Siehe OpenOptions (MQLONG).

Syntax: Für Abruf:

```
openopt& = MQQueue.OpenOptions
```

Zum Festlegen: *MQQueue.* **OpenOptions** = *openopt &*

Eigenschaft OpenOutputCount

Schreibgeschützt. Gibt die Anzahl der Öffnungen zur Ausgabe an

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Für Abruf:

```
openoutcount& = MQQueue.OpenOutputCount
```

Eigenschaft OpenStatus

Schreibgeschützt. Gibt an, ob die Warteschlange geöffnet ist oder nicht. Der Anfangswert ist nach der AccessQueue-Methode TRUE oder nach Neukonfiguration FALSE.

Definiert in: MQQueue-Klasse

Datentyp: Boolesch

Werte:

- Wahr/TRUE (-1)
- Falsch/FALSE (0)

Syntax: Für Abruf:

```
status& = MQQueue.OpenStatus
```

ProcessName, Eigenschaft

Schreibgeschützt. Das MQI-ProcessName-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *procname$* **=** *MQQueue* **.ProcessName**

Eigenschaft 'QueueManagerName'

Lese-/Schreibzugriff. Der Name des IBM MQ-Warteschlangenmanagers.

Definiert in: MQQueue-Klasse

Datentyp: String

Syntax: To get: *QueueManagerName$* = *MQQueue* **.QueueManagerName**

To set: *MQQueue* **.QueueManagerName** = *QueueManagerName$*

Eigenschaft QueueType

Schreibgeschützt. Das MQI-QType-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQQT_ALIAS
- MQQT_LOCAL
- MQQT_MODEL
- MQQT_REMOTE

Syntax: Zum Abrufen: *queuetype & = MQQueue* **.QueueType**

Eigenschaft 'ReasonCode'

Schreibgeschützt. Gibt den Ursachencode zurück, der von dem zuletzt für das Objekt ausgegebenen Methodenaufruf bzw. Eigenschaftenzugriff gesetzt wurde.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasoncode & = MQQueue* **.ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Gibt den symbolischen Namen des neusten Ursachencodes zurück. beispielsweise MQRC_QMGR_NOT_AVAILABLE.

Definiert in: MQQueue-Klasse

Datentyp: String

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: To get: *reasonname$ = MQQueue* **.ReasonName**

Eigenschaft RemoteQueueManagerName

Schreibgeschützt. Gibt den Namen des fernen Warteschlangenmanagers an. Nur gültig für ferne Warteschlangen.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *remqmanname$ = MQQueue* **.RemoteQueueManagerName**

Eigenschaft RemoteQueueName

Schreibgeschützt. Der Name der Warteschlange, wie auf dem fernen Warteschlangenmanager bekannt. Nur gültig für ferne Warteschlangen.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *remqname$ = MQQueue* **.RemoteQueueName**

Eigenschaft 'ResolvedQueueManagerName'

Schreibgeschützt. Der Name des Warteschlangenmanagers am Zielort, wie auf dem lokalen Warteschlangenmanager bekannt.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *resqmanname$ = MQQueue* **.ResolvedQueueManagerName**

Eigenschaft ResolvedQueueName

Schreibgeschützt. Der Name der Warteschlange am Zielort, wie auf dem lokalen Warteschlangenmanager bekannt.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *resqname$ = MQQueue* **.ResolvedQueueName**

Eigenschaft RetentionInterval

Schreibgeschützt. Die Frist, für die die Warteschlange beibehalten werden sollte.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *retinterval & = MQQueue* **.RetentionInterval**

Eigenschaft Scope

Schreibgeschützt. Gibt an, ob ein Eintrag für diese Warteschlange auch in einem Zellenverzeichnis steht.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQSCO_Q_MGR
- MQSCO_CELL

Syntax: Zum Abrufen: *scope & = MQQueue* **.Scope**

Eigenschaft ServiceInterval

Schreibgeschützt. Das MQI-QServiceInterval-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *serviceinterval & = MQQueue.* **ServiceInterval**

Eigenschaft ServiceIntervalEvent

Schreibgeschützt. Das MQI-QServiceIntervalEvent-Attribut.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQQSIE_HIGH
- MQQSIE_OK
- MQQSIE_NONE

Syntax: Zum Abrufen: *serviceintervalevent & = MQQueue.* **ServiceIntervalEvent**

Eigenschaft Shareability

Schreibgeschützt. Ist die Warteschlange gemeinsam nutzbar?

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

Syntax: Zum Abrufen: *shareability & = MQQueue* **.Shareability**

Eigenschaft TransmissionQueueName

Schreibgeschützt. Name der Übertragungswarteschlange. Nur gültig für ferne Warteschlangen.

Definiert in: MQQueue-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *transqname$ = MQQueue* **.TransmissionQueueName**

Eigenschaft TriggerControl

Lese-/Schreibzugriff. Auslösesteuerung.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQTC_OFF

- MQTC_ON

Syntax: Zum Abrufen: *trigcontrol & = MQQueue* **.TriggerControl**

Zum Festlegen: *MQQueue* **.TriggerControl** = *trigcontrol &*

Eigenschaft TriggerData

Lese-/Schreibzugriff. Auslösedaten

Definiert in: MQQueue-Klasse

Datentyp: Zeichenfolge von 64 Zeichen

Syntax: To get: *trigdata$ = MQQueue* **.TriggerData**

To set: *MQQueue* **.TriggerData** = *trigdata$*

Eigenschaft TriggerDepth

Lese-/Schreibzugriff. Die Anzahl von Nachrichten, die sich auf der Warteschlange befinden müssen, bevor eine Auslösenachricht geschrieben wird.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *trigdepth & = MQQueue* **.TriggerDepth**

Zum Festlegen: *MQQueue* **.TriggerDepth** = *trigdepth &*

TriggerMessagePriority, Eigenschaft

Lese-/Schreibzugriff. Nachrichtenprioritätsschwelle für Auslöser

Definiert in: MQQueue-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *trigmesspriority & = MQQueue* **.TriggerMessagePriorität**

Zum Festlegen: *MQQueue* **.TriggerMessagePriorität** = *trigmesspriority &*

Eigenschaft TriggerType

Lese-/Schreibzugriff. Auslösertyp

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQTT_NONE
- MQTT_FIRST
- MQTT_EVERY
- MQTT_DEPTH

Syntax: Zum Abrufen: *trigtype & = MQQueue* **.TriggerType**

Zum Festlegen: *MQQueue* **.TriggerType** = *Trigtype &*

Eigenschaft Usage

Schreibgeschützt. Gibt an, wofür die Warteschlange verwendet wird.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- MQUS_NORMAL
- MQUS_TRANSMISSION

Syntax: Zum Abrufen: *usage & =MQQueue* **.Verwendung**

ClearErrorCodes-Methode

Setzt für die MQQueue-Klasse und die MQSession-Klasse den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE zurück.

Definiert in: MQQueue-Klasse

Syntax:

```
Call MQQueue.ClearErrorCodes()
```

Close-Methode

Schließt eine Warteschlange mithilfe der aktuellen Werte von CloseOptions.

Definiert in: MQQueue-Klasse

Syntax:

```
Call MQQueue.Close()
```

Get-Methode

Ruft eine Nachricht aus der Warteschlange ab.

In dieser Methode werden ein MQMessage-Objekt als Parameter und einige der Felder im Nachrichtendeskriptor MQMD des Objekts als Eingabeparameter verwendet. Insbesondere werden die Felder 'MessageId' und 'CorrelId' verwendet, deshalb müssen Sie unbedingt sicherstellen, dass diese Felder wie erforderlich festgelegt sind. Weitere Informationen zu diesen Feldern finden Sie unter MsgId (MQBYTE24) und CorrelId (MQBYTE24).

Wenn die Methode fehlschlägt, bleibt das MQMessage-Objekt unverändert. Wenn sie erfolgreich ist, werden die MQMD-Teile und Nachrichtendatenteile des MQMessage-Objekts durch den MQMD und die Nachrichtendaten aus der eingehenden Nachricht ersetzt. Die Eigenschaften zur MQMessage-Steuerung werden folgendermaßen festgelegt:

- **MessageLength** wird auf die Länge der IBM MQ-Nachricht gesetzt
- **DataLength** wird auf die Länge der IBM MQ-Nachricht gesetzt
- **DataOffset** wird auf null gesetzt

Definiert in:
MQQueue-Klasse

Syntax:

```
Call MQQueue.Get(Message, GetMsgOptions, GetMsgLength)
```

Parameter
Nachricht:

MQMessage-Objekt, das die Nachricht darstellt, die abgerufen werden soll.

GetMsgOptions:

Optionales MQGetMessageOptions-Objekt zum Steuern der GET-Operation. Ist dieser Parameter nicht angegeben, werden standardmäßige MQGetMessageOptions verwendet.

GetMsgLength:

Optionaler Wert mit einer Länge von 2 oder 4 Byte, um die maximale Länge der IBM MQ-Nachricht zu steuern, die aus der Warteschlange abgerufen wird.

Wenn die MQGMO_ACCEPT_TRUNCATED_MSG-Option angegeben ist, ist der GET-Aufruf erfolgreich und gibt den Beendigungscode MQCC_WARNING und den Ursachencode MQRC_TRUNCATED_MSG_ACCEPTED zurück, wenn die Nachrichtengröße die angegebene Länge überschreitet.

In MessageData sind die ersten GetMsgLength-Byte der Daten enthalten.

Wenn MQGMO_ACCEPT_TRUNCATED_MSG ***nicht*** angegeben ist und die Nachrichtengröße die angegebene Länge überschreitet, wird der Beendigungscode MQCC_FAILED zusammen mit dem Ursachencode MQRC_TRUNCATED_MESSAGE_FAILED zurückgegeben.

Wenn die Inhalte des Nachrichtenpuffers nicht definiert sind, wird die gesamte Nachrichtenlänge auf die vollständige Länge der Nachricht gesetzt, die abgerufen worden wäre.

Wenn der Parameter für die Nachrichtenlänge nicht angegeben ist, wird die Länge des Nachrichtenpuffers automatisch an die Größe der eingehenden Nachricht angepasst.

Methode Open

Öffnet eine Warteschlange mithilfe der aktuellen Werte von:

1. QueueName
2. QueueManagerName
3. AlternateUserId
4. DynamicQueueName

Definiert in:
MQQueue-Klasse

Syntax:

```
Call MQQueue.Open()
```

Methode 'Put'

Reiht eine Nachricht in die Warteschlange ein.

Für diese Methode wird ein MQMessage-Objekt als Parameter angegeben. Die Eigenschaften des Nachrichtendeskriptors (MQMD) für dieses Objekt werden als Ergebnis dieser Methode möglicherweise geändert. Die Werte, die direkt nach Ausführung dieser Methode angegeben sind, sind die Werte, die in IBM MQ eingereiht wurden.

Änderungen am MQMessage-Objekt nach Abschluss der Put-Methode haben keinen Einfluss auf die tatsächliche Nachricht in der IBM MQ-Warteschlange.

Definiert in:
MQQueue-Klasse

Syntax:

```
Call MQQueue.Put(Message, PutMsgOptions)
```

Parameter
Nachricht

MQMessage-Objekt, das die Nachricht darstellt, die eingereiht werden soll.

PutMsgOptions

MQPutMessageOptions-Objekt mit den Optionen zum Steuern der Put-Operation. Wenn diese nicht angegeben sind, werden standardmäßige PutMessageOptions-Objekte verwendet.

Klasse MQMessage

Diese Klasse stellt eine IBM MQ-Nachricht dar. Sie enthält Eigenschaften zum Kapseln des IBM MQ-Nachrichtendeskriptors (MQMD) und stellt einen Puffer für die von der Anwendung definierten Nachrichtendaten bereit.

Die Klasse enthält Write-Methoden, mit denen Daten aus einer ActiveX-Anwendung in ein MQMessage-Objekt kopiert werden können. Ebenso enthält die Klasse Read-Methoden, um Daten aus einem MQMessage-Objekt in eine ActiveX-Anwendung zu kopieren. Die Klasse verwaltet die Zuordnung und die Freigabe von Speicher für den Puffer automatisch. Die Anwendung muss bei der Erstellung eines MQMessage-Objekts die Größe des Puffers nicht deklarieren, da sich der Puffer dem Umfang der Daten, die in ihn geschrieben werden, anpasst.

Eine Nachricht kann nicht in eine IBM MQ-Warteschlange eingereiht werden, wenn die Puffergröße den Wert der Eigenschaft 'MaximumMessageLength' dieser Warteschlange überschreitet.

Nach seiner Erstellung kann ein MQMessage-Objekt mithilfe der Methode 'MQQueue.Put' in eine IBM MQ-Warteschlange eingereiht werden. Diese Methode erstellt eine Kopie der Teile des Objekts mit den MQMD- und Nachrichtendaten und stellt diese Kopie in die Warteschlange. Die Anwendung kann daher ein MQMessage-Objekt nach der Put-Operation löschen, ohne dass sich dies auf die Nachricht in der IBM MQ-Warteschlange auswirkt. Der Warteschlangenmanager kann einige Felder in der MQMD-Struktur beim Kopieren der Nachricht in die IBM MQ-Warteschlange anpassen.

Eine eingehende Nachricht kann mit der Methode 'MQQueue.Get' in ein MQMessage-Objekt gelesen werden. Dadurch werden alle im MQMessage-Objekt eventuell bereits vorhandenen MQMD- oder Nachrichtendaten durch die Werte aus der eingehenden Nachricht ersetzt. Die Größe des Datenpuffers des MQMessage-Objekts wird dabei an den Umfang der eingehenden Nachrichtdaten angepasst.

Einschlussbeziehung

Nachrichten sind in der Klasse 'MQSession' enthalten.

Erstellung

MQMessage-Objekte werden mit **New** erstellt. Die Eigenschaften des Nachrichtendeskriptors eines Objekts sind zunächst auf Standardwerte gesetzt und der Nachrichtendatenpuffer ist leer.

Syntax

```
Dim msg As New MQMessage
```

oder

```
Set msg = New MQMessage
```

Eigenschaften

Bei den Steuerungseigenschaften handelt es sich um folgende:

- „CompletionCode, Eigenschaft” auf Seite 650
- „DataLength, Eigenschaft” auf Seite 650
- „Eigenschaft 'DataOffset'” auf Seite 650
- „Eigenschaft 'MessageLength'” auf Seite 651
- „Eigenschaft 'ReasonCode'” auf Seite 651

- „Eigenschaft ReasonName” auf Seite 651

Bei den Eigenschaften des Nachrichtendeskriptors handelt es sich um folgende:

- „Eigenschaft 'AccountingToken'” auf Seite 651
- „Eigenschaft 'AccountingTokenHex'” auf Seite 652
- „ApplicationIdData, Eigenschaft” auf Seite 652
- „ApplicationOriginData, Eigenschaft” auf Seite 652
- „Eigenschaft BackoutCount” auf Seite 652
- „Eigenschaft 'CharacterSet'” auf Seite 652
- „Eigenschaft CorrelationId” auf Seite 653
- „Eigenschaft 'CorrelationIdHex'” auf Seite 653
- „Encoding, Eigenschaft” auf Seite 654
- „Expiry, Eigenschaft” auf Seite 654
- „Eigenschaft 'Feedback'” auf Seite 655
- „Format, Eigenschaft” auf Seite 655
- „Eigenschaft GroupId” auf Seite 655
- „Eigenschaft 'GroupIdHex'” auf Seite 655
- „Eigenschaft 'MessageData'” auf Seite 656
- „Eigenschaft 'MessageFlags'” auf Seite 656
- „Eigenschaft MessageId” auf Seite 656
- „Eigenschaft 'MessageIdHex'” auf Seite 657
- „Eigenschaft 'MessageSequenceNumber'” auf Seite 657
- „MessageType, Eigenschaft” auf Seite 657
- „Eigenschaft 'Offset'” auf Seite 657
- „Eigenschaft 'OriginalLength'” auf Seite 658
- „Persistence, Eigenschaft” auf Seite 658
- „Eigenschaft Priority” auf Seite 658
- „Eigenschaft PutApplicationName” auf Seite 658
- „Eigenschaft 'PutApplicationType'” auf Seite 658
- „PutDateTime, Eigenschaft” auf Seite 659
- „Eigenschaft ReplyToQueueManagerName” auf Seite 659
- „ReplyToQueueName, Eigenschaft” auf Seite 659
- „Report, Eigenschaft” auf Seite 659
- „Eigenschaft TotalMessageLength” auf Seite 660
- „UserId, Eigenschaft” auf Seite 660

Methoden

- „ClearErrorCodes-Methode” auf Seite 660
- „Methode 'ClearMessage'” auf Seite 660
- „Read-Methode” auf Seite 660
- „ReadBoolean-Methode” auf Seite 661
- „ReadByte, Methode” auf Seite 661
- „ReadDecimal2” auf Seite 661
- „ReadDecimal4” auf Seite 661

- „ReadDouble, Methode” auf Seite 661
- „Methode 'ReadDouble4'” auf Seite 662
- „Methode 'ReadFloat'” auf Seite 662
- „ReadInt2” auf Seite 662
- „ReadInt4” auf Seite 663
- „Methode 'ReadLong'” auf Seite 663
- „Methode 'ReadNullTerminatedString'” auf Seite 663
- „ReadShort, Methode” auf Seite 663
- „Methode 'ReadString'” auf Seite 663
- „Methode 'ReadUInt2'” auf Seite 664
- „ReadUnsignedByte, Methode” auf Seite 664
- „Methode ReadUTF” auf Seite 664
- „Methode 'ResizeBuffer'” auf Seite 665
- „Write, Methode” auf Seite 665
- „WriteBoolean-Methode” auf Seite 665
- „Methode 'WriteByte'” auf Seite 666
- „WriteDecimal2” auf Seite 666
- „WriteDecimal4” auf Seite 666
- „WriteDouble, Methode” auf Seite 666
- „Methode 'WriteDouble4'” auf Seite 667
- „Methode 'WriteFloat'” auf Seite 667
- „WriteInt2” auf Seite 667
- „WriteInt4” auf Seite 668
- „Methode 'WriteLong'” auf Seite 668
- „Methode 'WriteNullTerminatedString'” auf Seite 668
- „WriteShort, Methode” auf Seite 668
- „WriteString, Methode” auf Seite 669
- „WriteUInt2, Methode” auf Seite 669
- „WriteUnsignedByte, Methode” auf Seite 669
- „WriteUTF-Methode” auf Seite 670

Zugriff auf Eigenschaften

Auf alle Eigenschaften ist jederzeit ein Lesezugriff möglich.

Die Steuerungseigenschaften sind schreibgeschützt, mit Ausnahme der Eigenschaft 'DataOffset', auf die ein Lese- und Schreibzugriff möglich ist. Für alle Eigenschaften des Nachrichtendeskriptors ist ein Lese- und Schreibzugriff möglich, ausgenommen der Eigenschaften 'BackoutCount' und 'TotalMessageLength', die schreibgeschützt sind.

Einige der MQMD-Eigenschaften werden jedoch beim Einreihen der Nachricht in eine IBM MQ-Warteschlangen unter Umständen vom Warteschlangenmanager geändert. In den unter MQMD aufgeführten Informationen zu den Feldern finden Sie Hinweise, wie diese Felder unter Umständen geändert werden.

Datenkonvertierung

Sie können Binärdaten an eine IBM MQ-Nachricht übergeben, indem Sie die Eigenschaft **`CharacterSet`** auf die ID des codierten Zeichensatzes des Warteschlangenmanagers (MQCCSI_Q_MGR) setzen und die Daten als Zeichenfolge an die Nachricht übergeben. Wenn die Zeichenfolge Unicode- oder ASCII-Coden-

ummern einschließen muss, können Sie diese mit der Funktion 'chr$' ins Zeichenfolgeformat konvertieren.

Die Read- und Write-Methoden führen Datenkonvertierungen durch. Sie nehmen Konvertierungen zwischen den internen ActiveX-Formaten und den über die Eigenschaften 'Encoding' und 'CharacterSet' des Nachrichtendeskriptors vorgegebenen IBM MQ-Nachrichtenformaten vor. Beim Schreiben einer Nachricht sollten vor dem Absetzen einer Write-Methode Werte für die Eigenschaften 'Encoding' und 'CharacterSet' angegeben werden, die den Merkmalen des Empfängers der Nachricht entsprechen. Beim Lesen einer Nachricht ist dieser Schritt in der Regel nicht erforderlich, da diese Werte über die Werte aus der eingehenden MQMD-Struktur gesetzt wurden.

Dies ist ein zusätzlicher Schritt bei der Datenkonvertierung, der im Anschluss an die von der Methode 'MQQueue.Get' durchgeführte Konvertierung ausgeführt wird.

CompletionCode, Eigenschaft

Schreibgeschützt. Gibt den IBM MQ-Beendigungscode zurück, der über den für dieses Objekt zuletzt ausgegebenen Methodenaufruf bzw. Eigenschaftenzugriff gesetzt wurde.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *completioncode & = MQMessage* **.CompletionCode**

DataLength, Eigenschaft

Schreibgeschützt. Diese Eigenschaft gibt den folgenden Wert zurück:

```
MQMessage.MessageLength - MQMessage.DataOffset
```

Mit ihr kann vor dem Aufruf einer Read-Methode festgestellt werden, ob die erwartete Anzahl an Zeichen auch tatsächlich im Puffer enthalten ist.

Der Anfangswert ist null.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *bytesleft & = MQMessage* **.DataLength**

Eigenschaft 'DataOffset'

Lese-/Schreibzugriff. Gibt die aktuelle Position im Nachrichtendatenteil des Nachrichtenobjekts an.

Der Wert wird als relative Byteadresse ab Beginn des Nachrichtendatenpuffers angegeben; das erste Zeichen im Puffer entspricht einem DataOffset-Wert von null.

Eine Read- oder Write-Methode beginnt die Ausführung ab dem Zeichen, auf das über 'DataOffset' verwiesen wird. Diese Methoden verarbeiten ab dieser Position nacheinander die Daten im Puffer und aktualisieren 'DataOffset' so, dass auf das Byte (sofern vorhanden) verwiesen wird, das unmittelbar auf das zuletzt verarbeitete Byte folgt.

'DataOffset' kann nur auf Werte im Bereich zwischen null und dem Wert von 'MessageLength' (inklusive) gesetzt werden. Bei 'DataOffset = MessageLength' verweist der Wert auf das Ende, d. h. das erste ungültige Zeichen im Puffer. Write-Methoden sind in diesem Fall zulässig; sie erweitern die Daten im Puffer und erhöhen den Wert von 'MessageLength' um die Anzahl der hinzugefügten Bytes. Read-Methoden über das Ende des Puffers hinaus sind hingegen nicht zulässig.

Der Anfangswert ist null.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *currpos & = MQMessage* **.DataOffset**

Festlegung: *MQMessage* **.DataOffset** = *currpos &*

Eigenschaft 'MessageLength'

Schreibgeschützt. Gibt die Gesamtlänge des Nachrichtendatenteils des Nachrichtenobjekts in Zeichen zurück, unabhängig vom Wert von 'DataOffset'.

Der Anfangswert ist null. Sie wird nach dem Aufruf einer Get-Methode, die auf dieses Nachrichtenobjekt verwiesen hat, auf die Länge der eingehenden Nachricht gesetzt. Wenn die Anwendung dem Objekt mit einer Write-Methode Daten hinzufügt, wird der Wert erhöht. Read-Methoden haben keine Auswirkung auf diese Eigenschaft.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *msglength & = MQMessage* **.MessageLength**

Eigenschaft 'ReasonCode'

Schreibgeschützt. Gibt den Ursachencode zurück, der über den für dieses Objekt zuletzt ausgegebenen Methodenaufruf bzw. Eigenschaftenzugriff gesetzt wurde.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasoncode & = MQMessage* **.ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Gibt den symbolischen Namen des neusten Ursachencodes zurück. beispielsweise MQRC_QMGR_NOT_AVAILABLE. **Definiert in:** MQMessage-Klasse

Datentyp: String

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: To get: *reasonname$ = MQMessage* **.ReasonName**

Eigenschaft 'AccountingToken'

Lese-/Schreibzugriff. Das Feld mit dem Abrechnungstoken (AccountingToken) der MQMD-Struktur; Teil des Identitätskontextes der Nachricht.

Ihr Anfangswert besteht aus lauter Nullen.

Definiert in: MQMessage-Klasse

Datentyp: Zeichenfolge von 32 Zeichen

Syntax: To get: *actoken$ = MQMessage* **.AccountingToken**

Zum Setzen: *MQMessage.* **AccountingToken** = *actoken$*

Weitere Informationen dazu, wann die Eigenschaft 'AccountingTokenHex' anstelle der Eigenschaft 'AccountingToken' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

Eigenschaft 'AccountingTokenHex'

Lese-/Schreibzugriff. Das Feld mit dem Abrechnungstoken (AccountingToken) der MQMD-Struktur; Teil des Identitätskontextes der Nachricht.

Je zwei Zeichen entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 64 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0...0'.

Definiert in: MQMessage-Klasse

Datentyp: Eine Zeichenfolge mit 64 Hexadezimalzeichen, die 32 ASCII-Zeichen darstellen

Syntax: To get: *actokenh$ = MQMessage* **.AccountingTokenHex**

Zum Setzen: *MQMessage.* **AccountingTokenHex** = *actokenh$*

Weitere Informationen dazu, wann die Eigenschaft 'AccountingTokenHex' anstelle der Eigenschaft 'AccountingToken' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

ApplicationIdData, Eigenschaft

Lese-/Schreibzugriff. Das MQMD-ApplIdentityData-Attribut - Teil des Nachrichten-Identitätskontexts.

Der Anfangswert besteht aus lauter Leerzeichen.

Definiert in: MQMessage-Klasse

Datentyp: Zeichenfolge von 32 Zeichen

Syntax: To get: *applid$ = MQMessage* **.ApplicationIdData**

To set: *MQMessage* **.ApplicationIdData** = *applid$*

ApplicationOriginData, Eigenschaft

Lese-/Schreibzugriff. Das MQMD-ApplOriginData-Attribut - Teil des Nachrichten-Ursprungskontexts.

Der Anfangswert besteht aus lauter Leerzeichen.

Definiert in: MQMessage-Klasse

Datentyp: Zeichenfolge von 4 Zeichen

Syntax: To get: *applor$ = MQMessage* **.ApplicationOriginData**

To set: *MQMessage* **.ApplicationOriginData** = *applor$*

Eigenschaft BackoutCount

Schreibgeschützt. Der MQMD-Rücksetzungszähler.

Der Anfangswert ist 0.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *backoutct & = MQMessage* **.BackoutCount**

Eigenschaft 'CharacterSet'

Lese-/Schreibzugriff. Das Feld mit der ID des codierten Zeichensatzes (CodedCharSetId) der MQMD-Struktur.

Anfangswert ist der Sonderwert **MQCCSI_Q_MGR**.

Ist 'CharacterSet' auf **MQCCSI_Q_MGR** gesetzt, wird für die Zeichenkonvertierung in der Methode 'WriteString' die Codepage der aktuellen Ländereinstellung verwendet. Bei Serveranwendungen wird die Codepage des Warteschlangenmanagers verwendet, bei Clientanwendungn die Standardcodepage der aktuellen Ländereinstellung.

Beispiel:

```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

Dabei ist 'n' größer-gleich null und kleiner-gleich 255; das Ergebnis ist ein einzelnes Byte mit dem Wert 'n', das in den Puffer geschrieben wird.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *:30ccid& = MQMessage* **.CharacterSet**

Zum Festlegen: *MQMessage* **.CharacterSet** = *ccid &*

Beispiel

Soll die Zeichenfolge im Format der Codepage 437 ausgegeben werden, müssen Sie Folgendes absetzen:

```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

Setzen Sie vor einem WriteString-Aufruf den gewünschten Wert in 'CharacterSet'.

Eigenschaft CorrelationId

Lese-/Schreibzugriff. Die Korrelations-ID, die in die MQMD-Struktur einer Nachricht eingefügt werden soll, wenn die Nachricht in eine Warteschlange eingereiht wird. Außerdem wird beim Abrufen einer Nachricht aus einer Warteschlange die Nachricht anhand dieser ID ermittelt.

Sein Anfangswert ist null.

Definiert in: MQMessage-Klasse

Datentyp: Eine Zeichenfolge mit 24 Zeichen

Syntax: To get: *correlid$ = MQMessage* **.CorrelationId** To set: *MQMessage* **.CorrelationId** = *correlid$*

Weitere Informationen dazu, wann die Eigenschaft 'CorrelationIdHex' anstelle der Eigenschaft 'CorrelationId' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

Eigenschaft 'CorrelationIdHex'

Lese-/Schreibzugriff. Die Korrelations-ID, die in die MQMD-Struktur einer Nachricht eingefügt werden soll, wenn die Nachricht in eine Warteschlange eingereiht wird. Außerdem wird beim Abrufen einer Nachricht aus einer Warteschlange die Nachricht anhand dieser Korrelations-ID ermittelt.

Je zwei Zeichen in der Zeichenfolge entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 48 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0...0'.

Definiert in: MQMessage-Klasse

Datentyp: Eine Zeichenfolge mit 48 Hexadezimalzeichen, die 24 ASCII-Zeichen darstellen

Syntax: To get: *correlidh$ = MQMessage* **.CorrelationIdHex**

To set: *MQMessage* **.CorrelationIdHex** = *correlidh$*

Weitere Informationen dazu, wann die Eigenschaft 'CorrelationIdHex' anstelle der Eigenschaft 'CorrelationId' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

Encoding, Eigenschaft

Lese-/Schreibzugriff. Das MQMD-Feld, das die Darstellung angibt, die in den Anwendungsnachrichtendaten für numerische Werte verwendet wird.

Anfangswert ist der Sonderwert MQENC_NATIVE, der je nach Plattform variiert.

Diese Eigenschaft wird von den folgenden Methoden verwendet:

- ReadDecimal2
- ReadDecimal4
- ReadDouble, Methode
- Methode 'ReadDouble4'
- Methode 'ReadFloat'
- ReadInt2
- ReadInt4
- Methode 'ReadLong'
- ReadShort, Methode
- Methode 'ReadUInt2'
- WriteDecimal2
- WriteDecimal4
- WriteDouble, Methode
- Methode 'WriteDouble4'
- Methode 'WriteFloat'
- WriteInt2
- WriteInt4
- Methode 'WriteLong'
- WriteShort, Methode
- WriteUInt2, Methode

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *encoding & = MQMessage* **.Codierung** Zum Festlegen: *MQMessage* **.Codierung** = *encoding &*

Wenn Sie dabei sind, Daten in den Nachrichtenpuffer zu schreiben, sollten Sie dieses Feld so setzen, dass es den Merkmalen der Plattform des empfangenden Warteschlangenmanagers entspricht, falls der empfangende Warteschlangenmanager selbst keine Datenkonvertierung vornehmen kann.

Expiry, Eigenschaft

Lese-/Schreibzugriff. Das MQMD-Ablaufzeitfeld, erwartet in Zehntelsekunden.

Ihr Anfangswert ist der spezielle Wert MQEI_UNLIMITED.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *expiry & = MQMessage* **Ablauf**

Zum Festlegen: *MQMessage* **.Ablauf** = *Ablauf &*

Eigenschaft 'Feedback'

Lese-/Schreibzugriff. Das Feld 'Feedback" der MQMD-Struktur.

Der Anfangswert ist der Sonderwert MQFB_NONE.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Werte:

- Siehe unter Feedback.

Syntax: Zum Abrufen: *feedback & = MQMessage* **.Feedback**

Zum Festlegen: *MQMessage* **.Feedback** = *Feedback &*

Format, Eigenschaft

Lese-/Schreibzugriff. Das Feld 'MQMD format'. Gibt den Namen eines integrierten oder benutzerdefinierten Formats an, das die Art der Nachrichtendaten beschreibt.

Der Anfangswert ist der spezielle Wert MQFMT_NONE.

Definiert in: MQMessage-Klasse

Datentyp: Zeichenfolge von 4 Zeichen

Syntax: To get: *format$ = MQMessage* **.Format**

To set: *MQMessage* **.Format** = *format$*

Eigenschaft GroupId

Lese-/Schreibzugriff. Die GroupId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird. Außerdem wird beim Abrufen einer Nachricht aus einer Warteschlange die Nachricht anhand dieser ID ermittelt. Ihr Anfangswert besteht aus lauter Nullen.

Definiert in:
Klasse MQMessage

Datentyp:
Zeichenfolge von 24 Zeichen

Syntax: Zum Abrufen: *groupid$ = MQMessage*. **GroupId**

To set: *MQMessage*. **GroupId** = *groupid$*

Weitere Informationen dazu, wann die Eigenschaft 'GroupIdHex' anstelle der Eigenschaft 'GroupId' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

Eigenschaft 'GroupIdHex'

Lese-/Schreibzugriff. Die GroupId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird. Außerdem wird beim Abrufen einer Nachricht aus einer Warteschlange die Nachricht anhand dieser ID ermittelt.

Je zwei Zeichen in der Zeichenfolge entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 48 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0...0'.

Definiert in:
Klasse MQMessage

Datentyp:
Eine Zeichenfolge mit 48 Hexadezimalzeichen, die 24 ASCII-Zeichen darstellen.

Syntax: To get: *groupidh$ = MQMessage*. **GroupIdHex**

Zum Setzen: *MQMessage*. **GroupIdHex** = *groupidh$*

Weitere Informationen dazu, wann die Eigenschaft 'GroupIdHex' anstelle der Eigenschaft 'GroupId' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

Eigenschaft 'MessageData'

Lese-/Schreibzugriff. Über diese Eigenschaft wird der gesamte Inhalt einer Nachricht als Zeichenfolge abgerufen bzw. gesetzt.

Definiert in: MQMessage-Klasse

Datentyp: Variant

Anmerkung: Für diese Eigenschaft wird der Datentyp 'Variant' verwendet, von MQAX wird jedoch als Variant-Typ eine Zeichenfolge (String) erwartet. Wird ein anderer Variant-Typ übergeben, wird der Fehler MQRC_OBJECT_TYPE_ERROR zurückgegeben.

Syntax: To get: *String$ = MQMessage* **.MessageData**

To set: *MQMessage* **.MessageData** = *String$*

Eigenschaft 'MessageFlags'

Lese-/Schreibzugriff möglich. Nachrichtenflags, die Informationen zur Segmentierungssteuerung angeben. Der Anfangswert ist 0.

Definiert in:
Klasse MQMessage

Datentyp:
Lang

Werte:
Siehe unter MsgFlags (MQLONG).

Syntax: Zum Abrufen: *messageflags & = MQMessage*. **MessageFlags**

To set: *MQMessage*. **MessageFlags** = *Nachrichtenmarkierungen &*

Eigenschaft MessageId

Lese-/Schreibzugriff. Die Nachrichten-ID (MessageId), die in die MQMD-Struktur einer Nachricht eingefügt werden soll, wenn die Nachricht in eine Warteschlange eingereiht wird. Außerdem wird beim Abrufen einer Nachricht aus einer Warteschlange die Nachricht anhand dieser ID ermittelt.

Ihr Anfangswert besteht aus lauter Nullen.

Definiert in: MQMessage-Klasse

Datentyp: Eine Zeichenfolge mit 24 Zeichen

Syntax: To get: *messageid$ = MQMessage* **.MessageId**

To set: *MQMessage* **.MessageId** = *messageid$*

Weitere Informationen dazu, wann die Eigenschaft 'MessageIdHex' anstelle der Eigenschaft 'MessageId' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

Eigenschaft 'MessageIdHex'

Lese-/Schreibzugriff. Die Nachrichten-ID (MessageId), die in die MQMD-Struktur einer Nachricht eingefügt werden soll, wenn die Nachricht in eine Warteschlange eingereiht wird. Außerdem wird beim Abrufen einer Nachricht aus einer Warteschlange die Nachricht anhand dieser Nachrichten-ID ermittelt.

Je zwei Zeichen in der Zeichenfolge entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 48 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0...0'.

Definiert in: MQMessage-Klasse

Datentyp: Eine Zeichenfolge mit 48 Hexadezimalzeichen, die 24 ASCII-Zeichen darstellen

Syntax: To get: *messageidh$ = MQMessage* **.MessageIdHex**

To set: *MQMessage* **.MessageIdHex** = *messageidh$*

Weitere Informationen dazu, wann die Eigenschaft 'MessageIdHex' anstelle der Eigenschaft 'MessageId' verwendet werden muss, finden Sie unter „Eigenschaften des Nachrichtendeskriptors” auf Seite 610.

Eigenschaft 'MessageSequenceNumber'

Lese-/Schreibzugriff möglich. Angaben zur Reihenfolge für die Ermittlung einer Nachricht innerhalb einer Gruppe. Der Anfangswert lautet 1.

Definiert in:
Klasse MQMessage

Datentyp:
Lang

Werte:
Siehe unter MsgSeqNumber (MQLONG).

Syntax: Zum Abrufen: *sequencenumber & = MQMessage*. **SequenceNumber**

To set: *MQMessage*. **SequenceNumber** = *Sequenznummer &*

MessageType, Eigenschaft

Lese-/Schreibzugriff. Das Feld 'MsgType' der MQMD-Struktur.

Der Anfangswert ist MQMT_DATAGRAM.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Werte:

- Siehe unter MsgType (MQLONG).

Syntax: Zum Abrufen: *msgtype & = MQMessage* **.MessageType**

Zum Festlegen: *MQMessage* **.MessageType** = *Nachrichtentyp &*

Eigenschaft 'Offset'

Lese-/Schreibzugriff möglich. Die relative Adresse in einer segmentierten Nachricht. Der Anfangswert ist 0.

Definiert in:
Klasse MQMessage

Datentyp:
Lang

Werte:
Siehe unter Offset (MQLONG).

Syntax: Zum Abrufen: *offset & = MQMessage*. **Offset**

To set: *MQMessage*. **Offset** = *offset &*

Eigenschaft 'OriginalLength'

Lese-/Schreibzugriff möglich. Die ursprüngliche Länge einer segmentierten Nachricht. Der Anfangswert ist MQOL_UNDEFINED.

Definiert in:
Klasse MQMessage

Datentyp:
Lang

Werte:
Siehe unter OriginalLength (MQLONG).

Syntax: Zum Abrufen: *originallength & = MQMessage*. **OriginalLength**

To set: *MQMessage*. **OriginalLength** = *Originallänge &*

Persistence, Eigenschaft

Lese-/Schreibzugriff. Die Persistenzeinstellung der Nachricht.

Der Anfangswert ist MQPER_PERSISTENCE_AS_Q_DEF.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *persist & = MQMessage* **.Persistenz**

Zum Festlegen: *MQMessage* **.Persistenz** = *persist &*

Eigenschaft Priority

Lese-/Schreibzugriff. Die Priorität der Nachricht.

Der Anfangswert ist der spezielle Wert MQPRI_PRIORITY_AS_Q_DEF.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *priority & = MQMessage* **.Priorität**

Zum Festlegen: *MQMessage* **.Priorität** = *priority &*

Eigenschaft PutApplicationName

Lese-/Schreibzugriff. Das MQMD PutApplName-Attribut - Teil des Nachrichten-Ursprungs-Kontextes.

Der Anfangswert besteht aus lauter Leerzeichen.

Definiert in: MQMessage-Klasse

Datentyp: Zeichenfolge von 28 Zeichen

Syntax: To get: *putapplnm$ = MQMessage* **.PutApplicationName**

To set: *MQMessage* **.PutApplicationName** = *putapplnm$*

Eigenschaft 'PutApplicationType'

Lese-/Schreibzugriff. Das Feld 'PutApplType' der MQMD-Struktur, Teil des Nachrichtenursprungskontextes.

Der Anfangswert ist MQAT_NO_CONTEXT.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Werte:

- Siehe unter PutApplType (MQLONG).

Syntax: Zum Abrufen: *putappltp & =MQMessage* **.PutApplicationTyp**

Zum Festlegen: *MQMessage* **.PutApplicationTyp** = *putappltp &*

PutDateTime, Eigenschaft

Lese-/Schreibzugriff möglich. Diese Eigenschaft ist eine Kombination der MQMD-Felder 'PutDate' und 'PutTime'. Diese sind Teil des Nachrichtenursprungskontextes und geben an, wann die Nachricht eingereiht wurde.

Die ActiveX-Erweiterung führt die Konvertierung zwischen dem ActiveX-Datums-/Zeitformat und den in einem IBM MQ-Nachrichtendeskriptor (MQMD) verwendeten Datums-/Zeitformaten durch. Beim Empfang einer Nachricht mit einem ungültigen Wert für 'PutDate' (Datum des Einreihens) bzw. 'PutTime' (Uhrzeit des Einreihens) wird die Eigenschaft 'PutDateTime' im Anschluss an die Get-Methode auf EMPTY gesetzt.

Der Anfangswert ist EMPTY.

Definiert in: MQMessage-Klasse

Datentyp: Variant-Typ 7 (Datum/Uhrzeit) oder EMPTY.

Syntax: To get: *datetime = MQMessage* **.PutDateTime**

To set: *MQMessage* **.PutDateTime** = *datetime*

Eigenschaft ReplyToQueueManagerName

Lese-/Schreibzugriff. Das Feld 'MQMD ReplyToQMgr'.

Der Anfangswert besteht aus lauter Leerzeichen.

Definiert in: MQMessage-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *replytoqmgr$ = MQMessage* **.ReplyToQueueManagerName**

To set: *MQMessage* **.ReplyToQueueManagerName** = *replytoqmgr$*

ReplyToQueueName, Eigenschaft

Lese-/Schreibzugriff. Das Feld 'MQMD ReplyToQ'.

Der Anfangswert besteht aus lauter Leerzeichen.

Definiert in: MQMessage-Klasse

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *replytoq$ = MQMessage* **.ReplyToQueueName**

To set: *MQMessage* **.ReplyToQueueName** = *replytoq$*

Report, Eigenschaft

Lese-/Schreibzugriff. Die Berichtsoptionen der Nachricht.

Der Anfangswert ist MQRO_NONE.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Werte:

- Siehe unter Report.

Syntax: Zum Abrufen: *report & = MQMessage* **.Bericht**

Zum Festlegen: *MQMessage* **.Report** = *bericht &*

Eigenschaft TotalMessageLength

Schreibgeschützt. Ruft die Länge der letzten von MQGET erhaltenen Nachricht ab. Wenn die Nachricht nicht abgeschnitten wurde, ist dieser Wert gleich dem Wert der Eigenschaft MessageLength.

Definiert in: MQMessage-Klasse

Datentyp: Lang

Syntax: Zum Abrufen: *totalmessagelength & = MQMessage* **.TotalMessageLänge**

UserId, Eigenschaft

Lese-/Schreibzugriff. Das MQMD-UserIdentifier-Attribut - Teil des Nachrichten-Identitätskontexts.

Der Anfangswert besteht aus lauter Leerzeichen.

Definiert in: MQMessage-Klasse

Datentyp: Zeichenfolge von 12 Zeichen

Syntax: To get: *userid$ = MQMessage* **.UserId**

To set: *MQMessage* **.UserId** = *userid$*

ClearErrorCodes-Methode

Setzt für die MQMessage-Klasse und die MQSession-Klasse den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE zurück.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.ClearErrorCodes()
```

Methode 'ClearMessage'

Diese Methode löscht den Datenpufferteil des MQMessage-Objekts. Alle Nachrichtendaten im Datenpuffer gehen verloren, da 'MessageLength' (Nachrichtenlänge), 'DataLength' (Datenlänge) und 'DataOffset' (relative Datenadresse) auf null gesetzt werden.

Der MQMD-Teil (der Nachrichtendeskriptor) bleibt unberührt davon; unter Umständen muss eine Anwendung einige der MQMD-Felder ändern, bevor das MQMessage-Objekt erneut verwendet wird. Die MQMD-Felder können mit 'New' zurückgesetzt werden, um so das Objekt durch eine neue Instanz zu ersetzen.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.ClearMessage()
```

Read-Methode

Liest eine Folge von Bytes vom Nachrichtenpuffer in ein Byte-Array ein. Die relative Datenadresse wird um die Anzahl an gelesenen Bytes erhöht und die Datenlänge wird um dieselbe Zahl verringert.

Definiert in:
Klasse MQMessage

Syntax: Data = MQMessage. **Lesen** (len &)

Parameter:
len &: Lang. Datenlänge in zu lesenden Bytes.

ReadBoolean-Methode

Liest einen booleschen 1-Byte-Wert aus der aktuellen Position im Nachrichtenpuffer und gibt einen booleschen 2-Byte-Wert TRUE(-1)/FALSE(0) zurück. Die relative Datenadresse wird um eins erhöht und die Datenlänge wird um eins verringert.

Definiert in:
Klasse MQMessage

Syntax: *value = MQMessage*. **ReadBoolean**

ReadByte, Methode

Die Methode 'ReadByte' liest 1 Byte aus dem Nachrichtendatenpuffer ab dem Zeichen, auf das mit 'Data-Offset' verwiesen wird, und gibt es als einen 2-Byte-Ganzzahlwert des Typs 'Integer' (mit Vorzeichen) im Bereich zwischen -128 und 127 zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 1 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 1 erhöht und 'DataLength' um 1 verringert.

Bei dem Nachrichtendatenbyte wird davon ausgegangen, dass es sich um eine binäre Ganzzahl mit Vorzeichen handelt.

Definiert in:
Klasse MQMessage

Syntax:
integerv% = MQMessage **.ReadByte**

ReadDecimal2

Liest eine gepackte 2-Byte-Dezimalzahl und gibt sie als 2-Byte-Ganzzahlwert mit Vorzeichen zurück. Die relative Datenadresse wird um zwei erhöht und die Datenlänge wird um zwei verringert.

Definiert in:
Klasse MQMessage

Syntax: *value% = MQMessage*. **ReadDecimal2**

ReadDecimal4

Liest eine gepackte 4-Byte-Dezimalzahl und gibt sie als 4-Byte-Ganzzahlwert mit Vorzeichen zurück. Die relative Datenadresse wird um vier erhöht und die Datenlänge wird um vier verringert.

Definiert in:
Klasse MQMessage

Syntax:

```
Call value& = MQMessage.ReadDecimal4
```

ReadDouble, Methode

Die Methode 'ReadDouble' liest 8 Byte aus dem Nachrichtendatenpuffer ab dem Byte, auf das mit 'DataOffset' verwiesen wird, und gibt sie als 8-Byte-Gleitkommawert des Typs 'Double' (mit Vorzeichen) zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 8 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 8 erhöht und 'DataLength' um 8 verringert.

Bei den 8 Nachrichtendatenzeichen wird davon ausgegangen, dass es sich um eine binäre Gleitkommazahl handelt. Die Codierung wird über die Eigenschaft 'MQMessage.Encoding' angegeben. Es wird keine Konvertierung aus dem System/360-Format unterstützt.

Definiert in:
Klasse MQMessage

Syntax:
doublev# = MQMessage **.ReadDouble**

Methode 'ReadDouble4'

Die Methoden 'ReadDouble4' und 'WriteDouble4' sind Alternativen zu 'ReadFloat' und 'WriteFloat', da sie 4-Byte-System/390-Gleitkommawerte unterstützen, die zu groß für eine Konvertierung in das 4-Byte-IEEE-Gleitkommaformat sind.

Die Methode 'ReadDouble4' liest 4 Byte aus dem Nachrichtendatenpuffer ab dem Byte, auf das mit 'DataOffset' verwiesen wird, und gibt sie als 8-Byte-Gleitkommawert des Typs 'Double' (mit Vorzeichen) zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 4 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 4 erhöht und 'DataLength' um 4 verringert.

Bei den 4 Nachrichtendatenzeichen wird davon ausgegangen, dass es sich um eine binäre Gleitkommazahl handelt. Die Codierung wird über die Eigenschaft 'MQMessage.Encoding' angegeben. Es wird keine Konvertierung aus dem System/360-Format unterstützt.

Definiert in:
Klasse MQMessage

Syntax:
doublev# = MQMessage. **ReadDouble4**

Methode 'ReadFloat'

Die Methode 'ReadFloat' liest 4 Byte aus dem Nachrichtendatenpuffer ab dem Byte, auf das mit 'DataOffset' verwiesen wird, und gibt sie als 4-Byte-Gleitkommawert des Typs 'Single' (mit Vorzeichen) zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 4 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 4 erhöht und 'DataLength' um 4 verringert.

Bei den 4 Nachrichtendatenzeichen wird davon ausgegangen, dass es sich um eine Gleitkommazahl handelt. Die Codierung wird über die Eigenschaft 'MQMessage.Encoding' angegeben. Es wird keine Konvertierung aus dem System/360-Format unterstützt.

Definiert in:
Klasse MQMessage

Syntax:
singlev! = MQMessage **.ReadFloat**

ReadInt2

Diese Methode ist identisch mit der ReadShort-Methode.

Syntax:
integerv% = MQMessage **.ReadInt2**

ReadInt4

Diese Methode ist identisch mit der ReadLong-Methode.

Syntax:
bigint & = MQMessage **.ReadInt4**

Methode 'ReadLong'

Die Methode 'ReadLong' liest 4 Byte aus dem Nachrichtendatenpuffer ab dem Byte, auf das mit 'DataOffset' verwiesen wird, und gibt sie als 4-Byte-Ganzzahlwert des Typs 'Long' (mit Vorzeichen) zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 4 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 4 erhöht und 'DataLength' um 4 verringert.

Bei den 4 Nachrichtendatenzeichen wird davon ausgegangen, dass es sich um eine binäre Ganzzahl handelt. Die Codierung wird über die Eigenschaft 'MQMessage.Encoding' angegeben.

Definiert in:
Klasse MQMessage

Syntax:
bigint & = MQMessage **.ReadLong**

Methode 'ReadNullTerminatedString'

Diese Methode ist als Alternative für 'ReadString' gedacht, wenn die Zeichenfolge unter Umständen eingebettete Nullzeichen enthält.

Diese Methode liest die angegebene Anzahl von Bytes aus dem Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird, und gibt sie als ActiveX-Zeichenfolge zurück. Enthält die Zeichenfolge vor ihrem Ende ein eingebettetes Nullzeichen, wird die Länge der zurückgegebenen Zeichenfolge auf die Anzahl Zeichen vor diesem Nullzeichen gekürzt.

'DataOffset' wird um den angegebenen Wert erhöht, 'DataLength' wird um den angegebenen Wert verringert, unabhängig davon, ob die Zeichenfolge eingebettete Nullzeichen enthält.

Bei den Zeichen in den Nachrichtendaten wird davon ausgegangen, dass es sich um eine Zeichenfolge in der Codepage handelt, die über die Eigenschaft 'MQMessage.CharacterSet' angegeben ist. Die Konvertierung in die ActiveX-Darstellung wird von der Anwendung vorgenommen.

Definiert in:
Klasse MQMessage

Syntax: *string$ = MQMessage*. **ReadNullTerminatedString(Länge &)**

Parameter:
Länge & *Lang.* Länge des Zeichenfolgefelds in Bytes.

ReadShort, Methode

Die Methode 'ReadShort' liest 2 Byte aus dem Nachrichtendatenpuffer ab dem Byte, auf das mit 'DataOffset' verwiesen wird, und gibt sie als 2-Byte-Ganzzahlwert des Typs 'Integer' (mit Vorzeichen) zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 2 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 2 erhöht, 'DataLength' um 2 verringert.

Bei den 2 Nachrichtendatenzeichen wird davon ausgegangen, dass es sich um eine binäre Ganzzahl handelt. Die Codierung wird über die Eigenschaft 'MQMessage.Encoding' angegeben.

Definiert in:
Klasse MQMessage

Syntax:
integerv% = MQMessage **.ReadShort**

Methode 'ReadString'

Diese Methode liest n Bytes aus dem Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird, und gibt sie als ActiveX-Zeichenfolge zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als n ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um n erhöht, 'DataLength' um n verringert.

Bei den n Nachrichtendatenzeichen wird davon ausgegangen, dass es sich um eine Zeichenfolge in der Codepage handelt, die über die Eigenschaft 'MQMessage.CharacterSet' angegeben ist. Die Konvertierung in die ActiveX-Darstellung wird von der Anwendung vorgenommen.

Definiert in: MQMessage-Klasse

Syntax: *stringv $= MQMessage* **.ReadString** *(Länge &*)

Parameter
länge & Lang. Länge des Zeichenfolgefelds in Bytes.

Methode 'ReadUInt2'

Die Methode 'ReadUInt2' liest 2 Byte aus dem Nachrichtendatenpuffer ab dem Byte, auf das mit 'DataOffset' verwiesen wird, und gibt sie als 4-Byte-Ganzzahlwert des Typs 'Long' (mit Vorzeichen) zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 2 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 2 erhöht, 'DataLength' um 2 verringert.

Bei den 2 Nachrichtendatenbytes wird davon ausgegangen, dass es sich um eine binäre Ganzzahl ohne Vorzeichen handelt. Die Codierung wird über die Eigenschaft 'MQMessage.Encoding' angegeben.

Definiert in:
Klasse MQMessage

Syntax:
bigint & = MQMessage **.ReadUInt2**

ReadUnsignedByte, Methode

Die Methode 'ReadUnsignedByte' liest 1 Byte aus dem Nachrichtendatenpuffer ab dem Zeichen, auf das mit 'DataOffset' verwiesen wird, und gibt es als einen 2-Byte-Ganzzahlwert des Typs 'Integer' (mit Vorzeichen) im Bereich zwischen 0 und 255 zurück.

Wenn 'MQMessage.DataLength' beim Aufruf kleiner als 1 ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 1 erhöht und 'DataLength' um 1 verringert.

Bei dem Nachrichtendatenzeichen wird davon ausgegangen, dass es sich um eine binäre Ganzzahl ohne Vorzeichen handelt.

Definiert in:
Klasse MQMessage

Syntax:
integerv% = MQMessage **.ReadUnsignedByte**

Methode ReadUTF

Diese Methode liest ab dem mit **DataOffset** angegebenen Byte eine UTF-Formatierzeichenfolge aus der Nachricht und gibt die UTF-Formatierzeichenfolge als ActiveX-Zeichenfolge zurück. Die gelesene UTF-Formatierzeichenfolge besteht aus 2 Bytes mit Daten, die die Länge der Zeichenfolge angeben, gefolgt von den UTF-Zeichendaten.

Die Methode schlägt fehl, wenn **MQMessage.DataLength** bei der Ausgabe kleiner als die Länge der Zeichenfolge ist.

Ist die Methode erfolgreich, wird **DataOffset** um die Zeichenfolgelänge erhöht und **DataLength** um die Zeichenfolgelänge verringert.

Definiert in:
Klasse MQMessage

Syntax:

```
value$ = MQMessage.ReadUTF
```

Methode 'ResizeBuffer'

Diese Methode ändert die Speicherkapazität, die momentan intern für den Nachrichtendatenpuffer zugeordnet ist. Die Anwendung erhält dadurch ein gewisses Maß an Kontrolle über die automatische Pufferverwaltung, indem sie sicherstellen kann, dass ein Puffer mit ausreichender Größe zugeordnet ist, wenn die Verarbeitung einer großen Nachricht ansteht. Die Anwendung muss diesen Aufruf nicht verwenden; wenn sie diesen Aufruf nicht ausgibt, wird der Puffer über den Code der automatischen Pufferverwaltung entsprechend vergrößert.

Wird die Größe des Puffers so verändert, dass der Puffer kleiner als der Wert von 'MessageLength' ist, verlieren Sie möglicherweise Daten. Bei einem Datenverlust gibt die Methode den Beendigungscode MQCC_WARNING und den Ursachencode MQRC_DATA_TRUNCATED zurück.

Wird die Größe des Puffers so verändert, dass der Puffer kleiner als der Wert der Eigenschaft **DataOffset** ist, gilt Folgendes:

- Die Eigenschaft **DataOffset** wird so geändert, dass sie auf das Ende des neuen Puffers verweist.
- Die Eigenschaft **DataLength** wird auf null gesetzt.
- Die Eigenschaft **MessageLength** wird auf die Größe des neuen Puffers gesetzt.

Definiert in:
Klasse MQMessage

Syntax: *MQMessage* **.ResizeBuffer** (*Länge &*)

Parameter:
Length& Long. Erforderliche Größe in Zeichen.

Write, Methode

Diese Methode schreibt eine Reihe von Bytes aus einem Byte-Array in den Nachrichtenpuffer an die Position, auf die über 'DataOffset' verwiesen wird. Bei Bedarf wird die Länge des Puffers (MQMessage.MQMessageLength) an die vollständige Länge des Bytes-Arrays angepasst. Wenn die Methode erfolgreich ist, wird 'DataOffset' um die Anzahl der geschriebenen Bytes erhöht.

Definiert in:
Klasse MQMessage

Syntax:

```
Call MQMessage.Write(value)
```

Parameter:
data: Ein Byte-Array oder ein Variant-Verweis auf ein Byte-Array

WriteBoolean-Methode

Schreibt aus einem booleschen 2-Byte-Wert einen booleschen 1-Byte-Wert an die aktuelle Position im Nachrichtenpuffer. Die relative Datenadresse wird um eins erhöht.

Definiert in:
Klasse MQMessage

Syntax:

```
Call MQMessage.WriteBoolean(value)
```

Parameter:
value: boolesch (2-Byte). Wert, der geschrieben werden soll.

Methode 'WriteByte'

Diese Methode schreibt einen 2-Byte-Ganzzahlwert mit Vorzeichen als 1-Byte-Binärzahl in den Nachrichtendatenpuffer an die Position, auf die über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle Daten, die an dieser Position im Puffer eventuell vorhanden sind, und erweitert bei Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 1 erhöht.

Der angegebene Wert sollte im Bereich zwischen -128 und 127 liegen. Ist dies nicht der Fall, gibt die Methode den Beendigungscode MQCC_FAILED und den Ursachencode MQRC_WRITE_VALUE_ERROR zurück.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteByte(value%)
```

Parameter: *value%* Integer. Wert, der geschrieben werden soll.

WriteDecimal2

Schreibt eine 2-Byte-Ganzzahl mit Vorzeichen als gepackte 2-Byte-Dezimalzahl. Die relative Datenadresse wird um zwei erhöht.

Definiert in:
Klasse MQMessage

Syntax:

```
Call MQMessage.WriteDecimal2(value%)
```

Parameter:
value% Integer. Wert, der geschrieben werden soll.

WriteDecimal4

Schreibt eine 4-Byte-Ganzzahl mit Vorzeichen als gepackte 4-Byte-Dezimalzahl. Die relative Datenadresse wird um vier erhöht.

Definiert in:
Klasse MQMessage

Syntax:

```
Call MQMessage.WritedDecimal4(value&)
```

Parameter:
wert & Lang. Wert, der geschrieben werden soll.

WriteDouble, Methode

Diese Methode schreibt einen 8-Byte-Gleitkommawert mit Vorzeichen als 8-Byte-Gleitkommazahl in den Nachrichtendatenpuffer ab der Position, auf die über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle

Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 8 erhöht.

Die Methode führt eine Konvertierung in die über die Eigenschaft 'MQMessage.Encoding' angegebene Gleitkommadarstellung durch. *Die Konvertierung in das System/360-Format wird nicht unterstützt.*

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteDouble(value#)
```

Parameter:
value# Double. Wert, der geschrieben werden soll.

Methode 'WriteDouble4'

Hinweise, wann die Methoden 'ReadDouble4' und 'WriteDouble4' anstellt der Methoden 'ReadFloat' bzw. 'WriteFloat' verwendet werden sollten, finden Sie unter „Methode 'ReadDouble4'" auf Seite 662.

Diese Methode schreibt einen 8-Byte-Gleitkommawert mit Vorzeichen als 4-Byte-Gleitkommazahl ab der Position in den Nachrichtendatenpuffer, auf die über 'DataOffset' verwiesen wird.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 4 erhöht.

Dabei ersetzt sie alle Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Die Methode führt eine Konvertierung in die über die Eigenschaft 'MQMessage.Encoding' angegebene Gleitkommadarstellung durch. *Die Konvertierung in das System/360-Format wird nicht unterstützt.*

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteDouble4(value#)
```

Parameter: *value#* Double. Wert, der geschrieben werden soll.

Methode 'WriteFloat'

Diese Methode schreibt einen 4-Byte-Gleitkommawert mit Vorzeichen als 4-Byte-Gleitkommazahl in den Nachrichtendatenpuffer ab dem Zeichen, auf das über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 4 erhöht.

Die Methode führt eine Konvertierung in die über die Eigenschaft 'MQMessage.Encoding' angegebene binäre Darstellung durch. *Die Konvertierung in das System/360-Format wird nicht unterstützt.*

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteFloat(value!)
```

Parameter *value!* Float. Wert, der geschrieben werden soll.

WriteInt2

Diese Methode ist identisch mit der WriteShort-Methode.

Syntax:

```
Call MQMessage.WriteInt2(value%)
```

Parameter: *value%* Integer. Wert, der geschrieben werden soll.

WriteInt4

Diese Methode ist identisch mit der WriteLong-Methode.

Syntax:

```
Call MQMessage.WriteInt4(value&)
```

Parameter *value &* Lang. Wert, der geschrieben werden soll.

Methode 'WriteLong'

Diese Methode schreibt einen 4-Byte-Ganzzahlwert mit Vorzeichen als 4-Byte-Binärzahl in den Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 4 erhöht.

Die Methode führt eine Konvertierung in die über die Eigenschaft 'MQMessage.Encoding' angegebene binäre Darstellung durch.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteLong(value&)
```

Parameter *value &* Lang. Wert, der geschrieben werden soll.

Methode 'WriteNullTerminatedString'

Diese Methode führt eine normale WriteString-Operation aus und füllt die restlichen Bytes bis zur angegebenen Länge mit Nullen auf. Wenn die Anzahl der von der ursprünglichen WriteString-Operation geschriebenen Bytes der angegebenen Länge entspricht, werden keine Nullen geschrieben. Wenn die Anzahl der Bytes die angegebene Länge überschreitet, wird ein Fehler gesetzt (Ursachencode MQRC_WRITE_VALUE_ERROR).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um die angegebene Länge erhöht.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteNullTerminatedString(value$, length&)
```

Parameter:
value$ String. Wert, der geschrieben werden soll.

length& Long. Länge des Zeichenfolgefelds in Bytes.

WriteShort, Methode

Diese Methode schreibt einen 2-Byte-Ganzzahlwert mit Vorzeichen als 2-Byte-Binärzahl in den Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 2 erhöht.

Die Methode führt eine Konvertierung in die über die Eigenschaft 'MQMessage.Encoding' angegebene binäre Darstellung durch.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteShort(value%)
```

Parameter: *value%* Integer. Wert, der geschrieben werden soll.

WriteString, Methode

Diese Methode schreibt eine ActiveX-Zeichenfolge in den Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um die Länge der Zeichenfolge in Bytes erhöht.

Die Methode konvertiert Zeichen in die über die Eigenschaft 'MQMessage.CharacterSet' angegebene Codepage.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteString(value$)
```

Parameter: *value$* String. Wert, der geschrieben werden soll.

WriteUInt2, Methode

Diese Methode schreibt einen 4-Byte-Ganzzahlwert mit Vorzeichen als 2-Byte-Binärzahl ohne Vorzeichen in den Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 2 erhöht.

Die Methode führt eine Konvertierung in die über die Eigenschaft 'MQMessage.Encoding' angegebene binäre Darstellung durch. Der angegebene Werte sollte im Bereich zwischen 0 und 2**16-1 liegen, andernfalls gibt die Methode den Beendigungscode MQCC_FAILED und den Ursachencode MQRC_WRITE_VALUE_ERROR zurück.

Definiert in: MQMessage-Klasse

Syntax:

```
Call MQMessage.WriteUInt2(value&)
```

Parameter *value &* Lang. Wert, der geschrieben werden soll.

WriteUnsignedByte, Methode

Diese Methode schreibt einen 2-Byte-Ganzzahlwert mit Vorzeichen als 1-Byte-Binärzahl ohne Vorzeichen in den Nachrichtendatenpuffer ab dem Zeichen, auf das über 'DataOffset' verwiesen wird. Dabei ersetzt sie alle Daten, die an diesen Positionen im Puffer eventuell vorhanden sind, und erweitert Bedarf die Pufferlänge (MQMessage.MessageLength).

Wenn die Methode erfolgreich ist, wird 'DataOffset' um 1 erhöht.

Der angegebene Werte sollte im Bereich zwischen 0 und 255 liegen, andernfalls gibt die Methode den Beendigungscode MQCC_FAILED und den Ursachencode MQRC_WRITE_VALUE_ERROR zurück.

Definiert in:
Klasse MQMessage

Syntax:

```
Call MQMessage.WriteUnsignedByte(value%)
```

Parameter: *value%* Integer. Wert, der geschrieben werden soll.

WriteUTF-Methode

Diese Methode verwendet eine ActiveX-Zeichenfolge und schreibt sie an der aktuellen Position im UTF-Format in den Nachrichtendatenpuffer. Die geschriebenen Daten bestehen aus einer 2-Byte-Länge, gefolgt von den Zeichendaten. Die relative Datenadresse (DataOffset) wird um die Länge der Zeichenfolge erhöht, wenn die Methode erfolgreich ist.

Definiert in:
Klasse MQMessage

Syntax: Call *MQMessage*. **WriteUTF** (value$)

Parameter:
value$ String. Wert, der geschrieben werden soll.

MQPutMessageOptions-Klasse

Diese Klasse kapselt eine Reihe von Optionen, über die das Einreihen einer Nachricht in eine IBM MQ-Warteschlange gesteuert wird.

Einschlussbeziehung

Die Klasse 'MQPutMessageOptions' ist in der Klasse 'MQSession' enthalten.

Erstellung

Mit **New** wird ein neues MQPutMessageOptions-Objekt erstellt und dessen Eigenschaften alle auf Anfangswerte gesetzt.

Ebenso kann auch die Methode 'AccessPutMessageOptions' der Klasse 'MQSession' verwendet werden.

Syntax

Dim *pmo* **As New MQPutMessageOptions** oder

Set *pmo* = **New MQPutMessageOptions**

Eigenschaften

- „CompletionCode, Eigenschaft” auf Seite 671.
- „Eigenschaft Options” auf Seite 671.
- „Eigenschaft 'ReasonCode'” auf Seite 671.
- „Eigenschaft ReasonName” auf Seite 671.
- „Eigenschaft 'RecordFields'” auf Seite 671.
- „Eigenschaft 'ResolvedQueueManagerName'” auf Seite 672.
- „Eigenschaft ResolvedQueueName” auf Seite 672.

Methoden

- „ClearErrorCodes-Methode” auf Seite 672.

CompletionCode, Eigenschaft

Schreibgeschützt. Gibt den Beendigungscode zurück, der beim letzten Zugriff auf die Methode oder die Eigenschaft, der bei diesem Objekt ausgeführt wurde, eingestellt wurde.

Ist definiert in: Klasse 'MQPutMessageOptions'

Datentyp: Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *completioncode & = PutOpts* **.CompletionCode**

Eigenschaft Options

Lese-/Schreibzugriff. Das Feld 'Options' der MQPMO-Struktur. Der Anfangswert dieses Feldes ist MQPMO_NONE. Weitere Informationen finden Sie unter MQPMO-Optionen.

Ist definiert in: Klasse 'MQPutMessageOptions'.

Datentyp: Lang

Syntax: Zum Abrufen: *options & = PutOpts* **.Optionen**

Zum Festlegen: *PutOpts* **.Optionen** = *Optionen &*

Die Optionen MQPMO_PASS_IDENTITY_CONTEXT und MQPMO_PASS_ALL_CONTEXT werden nicht unterstützt.

Eigenschaft 'ReasonCode'

Schreibgeschützt. Gibt den Ursachencode zurück, der von dem zuletzt für das Objekt ausgegebenen Methodenaufruf bzw. Eigenschaftenzugriff gesetzt wurde.

Ist definiert in: Klasse 'MQPutMessageOptions'

Datentyp: Lang

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasoncode & = PutOpts* **.ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Gibt den symbolischen Namen des neusten Ursachencodes zurück. beispielsweise MQRC_QMGR_NOT_AVAILABLE.

Ist definiert in: Klasse 'MQPutMessageOptions'

Datentyp: String

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: To get: *reasonname$ = PutOpts* **.ReasonName**

Eigenschaft 'RecordFields'

Lese-/Schreibzugriff. Flags, die die Felder angeben, die für jede Warteschlange beim Einreihen einer Nachricht in eine Verteilerliste angepasst werden müssen. Der Anfangswert ist null.

Diese Eigenschaft entspricht den PutMsgRecFields -Flags der MQI-Struktur MQPMO. In der MQI wird über diese Flags gesteuert, welche Felder (in der MQPMR-Struktur) vorhanden sind und vom MQPUT-Aufruf verwendet werden. In einem MQPutMessageOptions-Objekt sind diese Felder immer vorhanden und über die Flags wird daher lediglich vorgegeben, welche Felder beim Put-Vorgang verwendet werden.

Definiert in:
MQPutMessageOptions-Klasse

Datentyp:
Lang

Syntax: Zum Abrufen: *recordfields & = PutOpts* **.RecordFields**

To set: *PutOpts*. **RecordFields** = *Datensatzfelder &*

Eigenschaft 'ResolvedQueueManagerName'

Schreibgeschützt. Das Feld 'ResolvedQMgrName' der MQPMO-Struktur. Ausführliche Informationen finden Sie unter ResolvedQMgrName (MQCHAR48). Der Anfangswert besteht aus Leerzeichen.

Ist definiert in: Klasse 'MQPutMessageOptions'

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *qmgr$ = PutOpts* **.ResolvedQueueManagerName**

Eigenschaft ResolvedQueueName

Schreibgeschützt. Das Feld 'MQPMO ResolvedQName'. Details finden Sie im Abschnitt ResolvedQName (MQCHAR48). Der Anfangswert besteht aus Leerzeichen.

Ist definiert in: Klasse 'MQPutMessageOptions'

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *qname$ = PutOpts* **.ResolvedQueueName**

ClearErrorCodes-Methode

Setzt für die MQPutMessageOptions-Klasse und die MQSession-Klasse den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE zurück.

Definiert in:
MQPutMessageOptions-Klasse

Syntax:
Call *PutOpts* **.ClearErrorCodes()**

Klasse 'MQGetMessageOptions'

Diese Klasse kapselt eine Reihe von Optionen, über die das Abrufen einer Nachricht aus einer IBM MQ-Warteschlange gesteuert wird.

Einschlussbeziehung

Die Klasse 'MQGetMessageOptions' ist in der Klasse 'MQSession' enthalten.

Erstellung

Mit **New** wird ein neues MQGetMessageOptions-Objekt erstellt und dessen Eigenschaften alle auf Anfangswerte gesetzt.

Ebenso kann auch die Methode 'AccessGetMessageOptions' der Klasse 'MQSession' verwendet werden.

Eigenschaften

- „CompletionCode, Eigenschaft” auf Seite 673
- „Eigenschaft 'MatchOptions'” auf Seite 673
- „Eigenschaft Options” auf Seite 673
- „Eigenschaft 'ReasonCode'” auf Seite 674
- „Eigenschaft ReasonName” auf Seite 674
- „Eigenschaft ResolvedQueueName” auf Seite 674
- „WaitInterval, Eigenschaft” auf Seite 674

Methoden

- „ClearErrorCodes-Methode” auf Seite 674

Syntax

Dim *gmo* **As New MQGetMessageOptions** oder

Set *gmo* = **New MQGetMessageOptions**

CompletionCode, Eigenschaft

Schreibgeschützt. Gibt den Beendigungscode zurück, der beim letzten Zugriff auf die Methode oder die Eigenschaft, der bei diesem Objekt ausgeführt wurde, eingestellt wurde.

Definiert in: MQGetMessageOptions-Klasse.

Datentyp: Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *completioncode & = GetOpts* **.CompletionCode**

Eigenschaft 'MatchOptions'

Lese-/Schreibzugriff. Optionen zur Steuerung der für MQGET verwendeten Auswahlkriterien Der Anfangswert ist MQMO_MATCH_MSG_ID und MQMO_MATCH_CORREL_ID.

Definiert in:
Klasse 'MQGetMessageOptions'

Datentyp:
Lang

Werte:
Siehe unter MatchOptions (MQLONG).

Syntax: Zum Abrufen: *matchoptions & = GetOpts*. **MatchOptions**

To set: *GetOpts*. **MatchOptions** = *Abgleichoptionen &*

Eigenschaft Options

Lese-/Schreibzugriff. Das Feld 'MQGMO Options'. Details finden Sie im Abschnitt Options. Anfangswert ist MQGMO_NO_WAIT.

Definiert in: MQGetMessageOptions-Klasse.

Datentyp: Lang

Syntax: Zum Abrufen: *Optionen & = GetOpts* **.Optionen** Zum Festlegen: *GetOpts* **.Optionen** = *Optionen &*

Eigenschaft 'ReasonCode'

Schreibgeschützt. Gibt den Ursachencode zurück, der von dem zuletzt für das Objekt ausgegebenen Methodenaufruf bzw. Eigenschaftenzugriff gesetzt wurde.

Ist definiert in: Klasse 'MQGetMessageOptions'

Datentyp: Lang

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasoncode & = GetOpts* **.ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Gibt den symbolischen Namen des neusten Ursachencodes zurück. beispielsweise MQRC_QMGR_NOT_AVAILABLE. **Ist definiert in:** Klasse 'MQGetMessageOptions'

Datentyp: String

Werte:

- Siehe API-Beendigungs-und Ursachencodes.

Syntax: To get: *reasonname$ = MQGetMessageOptions* **.ReasonName**

Eigenschaft ResolvedQueueName

Schreibgeschützt. Das Feld 'MQGMO ResolvedQName'. Details finden Sie im Abschnitt ResolvedQName (MQCHAR48). Der Anfangswert besteht aus Leerzeichen.

Ist definiert in: Klasse 'MQGetMessageOptions'

Datentyp: Eine Zeichenfolge mit 48 Zeichen

Syntax: To get: *qname$ = GetOpts* **.ResolvedQueueName**

WaitInterval, Eigenschaft

Lese-/Schreibzugriff möglich. Das Feld 'WaitInterval' der MQGMO-Struktur. Der Zeitraum (in Millisekunden), den die Get-Operation maximal auf den Eingang einer geeigneten Nachricht wartet (sofern die Warteaktion über die Eigenschaft 'Options' angefordert wurde). Dieses Feld hat den Anfangswert 0. Details zu MQGMO-Optionen finden Sie im Abschnitt MQGMO.

Ist definiert in: Klasse 'MQGetMessageOptions'

Datentyp: Lang

Syntax: Zum Abrufen: *wait & = GetOpts* **.WaitInterval**

Zum Festlegen: *GetOpts* **.WaitInterval** = wait &

ClearErrorCodes-Methode

Setzt für die MQGetMessageOptions-Klasse und die MQSession-Klasse den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE zurück.

Definiert in:
Klasse 'MQGetMessageOptions'

Syntax:
Call *GetOpts* **.ClearErrorCodes()**

Klasse MQDistributionList

Diese Klasse kapselt eine Reihe von Warteschlangen - lokale Warteschlangen, ferne Warteschlangen oder Aliaswarteschlangen für die Ausgabe.

Erstellung

Mit **New** wird ein neues MQDistributionList-Objekt erstellt.

Ebenso kann auch die Methode 'AddDistributionList' der Klasse 'MQQueueManager' verwendet werden.

Eigenschaften

- „Eigenschaft AlternateUserId" auf Seite 675
- „Eigenschaft 'CloseOptions'" auf Seite 675
- „CompletionCode, Eigenschaft" auf Seite 676
- „Eigenschaft ConnectionReference" auf Seite 676
- „Eigenschaft FirstDistributionListItem" auf Seite 676
- „IsOpen, Eigenschaft" auf Seite 676
- „OpenOptions, Eigenschaft" auf Seite 677
- „Eigenschaft 'ReasonCode'" auf Seite 677
- „Eigenschaft ReasonName" auf Seite 677

Methode

- „AddDistributionListItem, Methode" auf Seite 677
- „ClearErrorCodes-Methode" auf Seite 678
- „Close-Methode" auf Seite 678
- „Methode Open" auf Seite 678
- „Methode 'Put'" auf Seite 678

Syntax

Dim *distlist.* **As New MQDistributionList** oder **Set** *distlist* = **New MQDistributionList**

Eigenschaft AlternateUserId

Lese-/Schreibzugriff. Die alternative Benutzer-ID, die verwendet wird, um den Zugriff auf die Liste von Warteschlangen zu überprüfen, wenn sie geöffnet werden.

Definiert in:
Klasse MQDistributionList

Datentyp:
Zeichenfolge von 12 Zeichen

Syntax: To get: *altuser$ = MQDistributionList*. **AlternateUserId**

Zum Setzen: *MQDistributionList*. **AlternateUserId** = *altuser$*

Eigenschaft 'CloseOptions'

Lese-/Schreibzugriff. Optionen, über die gesteuert wird, was geschehen soll, wenn eine Verteilerliste geschlossen wird. Der Anfangswert ist MQCO_NONE.

Definiert in:
Klasse MQDistributionList

Datentyp:
Lang

Werte:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

Syntax: Zum Abrufen: *closeopt & = MQDistributionList*. **CloseOptions**

Zum Setzen: *MQDistributionList*. **CloseOptions** = *closeopt &*

CompletionCode, Eigenschaft

Schreibgeschützt. Der Beendigungscode, der beim letzten Zugriff auf die Methode oder die Eigenschaft, der bei diesem Objekt ausgeführt wurde, eingestellt wurde.

Definiert in:
Klasse MQDistributionList

Datentyp:
Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *completioncode & = MQDistributionList*. **CompletionCode**

Eigenschaft ConnectionReference

Lese-/Schreibzugriff. Der Warteschlangenmanager, zu dem die Verteilerliste gehört.

Definiert in:
Klasse MQDistributionList

Datentyp:
MQQueueManager

Syntax: To get: *set queuemanager = MQDistributionList*. **ConnectionReference**

To set: *set MQDistributionList*. **ConnectionReference** = *queuemanager*

Eigenschaft FirstDistributionListItem

Schreibgeschützt. Das erste Objekt der Verteilerlistenelemente, das der Verteilerliste zugeordnet ist.

Definiert in:
Klasse MQDistributionList

Datentyp:
MQDistributionListItem

Werte:

Syntax: To get: *set distributionlistitem = MQDistributionList*. **FirstDistributionListItem**

IsOpen, Eigenschaft

Schreibgeschützt.

Definiert in:
Klasse MQDistributionList

Datentyp:
Boolesch

Werte:

- Wahr/TRUE (-1)

- Falsch/FALSE (0)

Syntax: Zum Abrufen: *IsOpen = MQDistributionList*. **IsOpen**

OpenOptions, Eigenschaft

Lese-/Schreibzugriff. Optionen, die verwendet werden, wenn eine Verteilerliste geöffnet wird.

Definiert in:
Klasse MQDistributionList

Datentyp:
Lang

Werte:
Siehe unter MQPMO-Optionen.

Syntax: Zum Abrufen: *openopt & = MQDistributionList* **OpenOptions**

To set: *MQDistributionList.* **OpenOptions** = openopt &

Eigenschaft 'ReasonCode'

Schreibgeschützt. Der Ursachencode, der von dem zuletzt für das Objekt ausgegebenen Methodenaufruf oder Eigenschaftenzugriff gesetzt wurde.

Definiert in:
Klasse MQDistributionList

Datentyp:
Lang

Werte:
Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasoncode & = MQDistributionList*. **ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Der symbolische Name für den Ursachencode, beispielsweise MQRC_QMGR_NOT_AVAILABLE.

Definiert in:
Klasse MQDistributionList

Datentyp:
Zeichenfolge

Werte:
Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasonname$ = MQDistributionList*. **ReasonName**

AddDistributionListItem, Methode

Verwenden Sie diese Methode, um ein neues `MQDistributionListItem`-Objekt zu erstellen und diesem Objekt das Verteilerlistenobjekt zuzuordnen. Der Parameter für den Warteschlangennamen ist obligatorisch.

Mit dieser Methode wird ein neues Verteilerlistenelement als erstes Element in eine vorhandene Liste eingefügt. Insbesondere erstellt diese Methode folgende Konfiguration:

- In der Verteilerliste legt sie die Eigenschaft **`FirstDistributionListItem`** so fest, dass sie auf das neue Verteilerlistenelement verweist.
- Im neuen Verteilerlistenelement legt sie folgende Eigenschaften fest:
  - Die Eigenschaft **`DistributionList`** wird so festgelegt, dass sie auf die Verteilerliste verweist.
  - Die Eigenschaft **`PreviousDistributionListItem`** wird auf null gesetzt.

– Die Eigenschaft **NextDistributionListItem** wird so festgelegt, dass sie auf das Verteilerlistenelement, das zuvor das erste war, oder auf null verweist, wenn keine vorherigen Elemente in der Liste vorhanden waren.

Sie können mit dieser Methode kein neues Element hinzufügen, wenn die Verteilerliste geöffnet ist.

Definiert in:
Klasse MQDistributionList

Syntax: set distributionlistitem = *MQDistributionList* **.AddDistributionListItem** (QName$, QMgrName$)

Parameter:

QName$ String. Name der IBM MQ-Warteschlange.

QMgrName$ String. Name des IBM MQ-Warteschlangenmanagers.

ClearErrorCodes-Methode

Setzt für die MQDistributionList-Klasse und die MQSession-Klasse den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE zurück.

Definiert in:
Klasse MQDistributionList

Syntax:

```
Call MQDistributionList.ClearErrorCodes()
```

Close-Methode

Schließt eine Verteilerliste mithilfe des aktuellen Werts der Close-Optionen.

Definiert in:
Klasse MQDistributionList

Syntax:
Call *MQDistributionList*. **Close** ()

Methode Open

Öffnet unter Verwendung des aktuellen Wertes der Eigenschaft AlternateUserId jede der Warteschlangen, die über die Eigenschaften **QueueName** und (gegebenenfalls) **QueueManagerName** der Verteilerlistenelemente angegeben sind, die dem aktuellen Objekt zugeordnet sind.

Definiert in:
Klasse MQDistributionList

Syntax:

```
Call MQDistributionList.Open()
```

Methode 'Put'

Reiht in eine Nachricht in jede der Warteschlangen ein, die über die der Verteilerliste zugeordneten Verteilerlistenelemente angegeben sind.

Definiert in:

Klasse MQDistributionList

Syntax

Call MQDistributionList. **Put** (Nachricht, PutMsg-Optionen &)

Parameter

Message: MQMessage-Objekt, das die Nachricht darstellt, die eingereiht werden soll.

PutMsgOptions: Das MQPutMessageOptions-Objekt, das die Optionen enthält, über die die Put-Operation gesteuert wird. Erfolgt keine Angabe, werden die Standardoptionen zum Einreihen von Optionen (PutMessageOptions) verwendet.

Für diese Methode wird ein MQMessage-Objekt als Parameter angegeben. Durch diese Methode können die folgenden Eigenschaften der Verteilerlistenelemente geändert werden:

- CompletionCode
- ReasonCode
- ReasonName
- MessageId
- MessageIdHex
- CorrelationId
- CorrelationIdHex
- GroupId
- GroupIdHex
- Feedback
- AccountingToken
- AccountingTokenHex

Klasse MQDistributionListItem

Diese Klasse kapselt die MQOR-, MQRR- und MQPMR-Strukturen und ordnet sie einer übergeordneten Verteilerliste zu.

Erstellung

Die Methode 'AddDistributionListItem' der Klasse 'MQDistributionList' wird verwendet.

Eigenschaften

Methoden

- „Eigenschaft 'AccountingToken'" auf Seite 680.
- „Eigenschaft 'AccountingTokenHex'" auf Seite 680.
- „CompletionCode, Eigenschaft" auf Seite 681.
- „Eigenschaft CorrelationId" auf Seite 681.
- „Eigenschaft 'CorrelationIdHex'" auf Seite 681.
- „Eigenschaft DistributionList" auf Seite 682.
- „Eigenschaft 'Feedback'" auf Seite 682.
- „Eigenschaft GroupId" auf Seite 682.
- „Eigenschaft 'GroupIdHex'" auf Seite 682.
- „Eigenschaft MessageId" auf Seite 683.
- „Eigenschaft 'MessageIdHex'" auf Seite 683.
- „Eigenschaft NextDistributionListItem" auf Seite 683.
- „Eigenschaft PreviousDistributionListItem" auf Seite 683.
- „Eigenschaft 'QueueManagerName'" auf Seite 683.

- „QueueName, Eigenschaft” auf Seite 684.
- „Eigenschaft 'ReasonCode'” auf Seite 684.
- „Eigenschaft ReasonName” auf Seite 684.
- „ClearErrorCodes-Methode” auf Seite 684.

Eigenschaften:

- Eigenschaft 'AccountingToken'
- Eigenschaft 'AccountingTokenHex'
- CompletionCode, Eigenschaft
- Eigenschaft CorrelationId
- Eigenschaft 'CorrelationIdHex'
- Eigenschaft DistributionList
- Eigenschaft 'Feedback'
- Eigenschaft GroupId
- Eigenschaft 'GroupIdHex'
- Eigenschaft MessageId
- Eigenschaft 'MessageIdHex'
- Eigenschaft NextDistributionListItem
- Eigenschaft PreviousDistributionListItem
- Eigenschaft 'QueueManagerName'
- QueueName, Eigenschaft
- Eigenschaft 'ReasonCode'
- Eigenschaft ReasonName

Methoden:

- ClearErrorCodes-Methode

Erstellung:

Die Methode 'AddDistributionListItem' der Klasse 'MQDistributionList' wird verwendet.

Eigenschaft 'AccountingToken'

Lese-/Schreibzugriff. Das Abrechnungstoken (AccountingToken), das in die MQPMR-Struktur einer Nachricht eingefügt werden soll, wenn die Nachricht in eine Warteschlange eingereiht wird. Ihr Anfangswert besteht aus lauter Nullen.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Eine Zeichenfolge mit 32 Zeichen

Syntax: Zum Abrufen: *accountingtoken$ = MQDistributionListItem*. **AccountingToken**

Zum Setzen: *MQDistributionListItem*. **AccountingToken** = *accountingtoken$*

Eigenschaft 'AccountingTokenHex'

Lese-/Schreibzugriff. Das Abrechnungstoken (AccountingToken), das in die MQPMR-Struktur einer Nachricht eingefügt werden soll, wenn die Nachricht in eine Warteschlange eingereiht wird.

Je zwei Zeichen in der Zeichenfolge entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 64 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0...0'.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge mit 64 Hexadezimalzeichen, die 32 ASCII-Zeichen darstellen.

Syntax: To get: *accountingtokenh$ = MQDistributionListItem*. **AccountingTokenHex**

Zum Setzen: *MQDistributionListItem*. **AccountingTokenHex** = *accountingtokenh$*

CompletionCode, Eigenschaft

Schreibgeschützt. Der Beendigungscode, der von der zuletzt für das übergeordnete Verteilerlistenobjekt aufgerufenen Open- oder Put-Anforderung gesetzt wurde.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Lang

Werte:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *completioncode$ = MQDistributionListItem*. **CompletionCode**

Eigenschaft CorrelationId

Lese-/Schreibzugriff. Die CorrelId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird. Ihr Anfangswert besteht aus lauter Nullen.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge von 24 Zeichen

Syntax: To get: *correlid$ = MQDistributionListItem*. **CorrelationId**

Zum Setzen: *MQDistributionListItem*. **CorrelationId** = *correlid$*

Eigenschaft 'CorrelationIdHex'

Lese-/Schreibzugriff. Die CorrelId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird.

Je zwei Zeichen in der Zeichenfolge entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 48 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0..0'.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Eine Zeichenfolge mit 48 Hexadezimalzeichen, die 24 ASCII-Zeichen darstellen.

Syntax: To get: *correlidh$ = MQDistributionListItem*. **CorrelationIdHex**

Zum Setzen: *MQDistributionListItem*. **CorrelationIdHex** = *correlidh$*

Eigenschaft DistributionList

Schreibgeschützt. Die Verteilerliste, der dieser Verteilerlistenartikel zugeordnet ist.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
MQDistributionList

Syntax: To get: *set distributionlist = MQDistributionListItem*. **DistributionList**

Eigenschaft 'Feedback'

Lese-/Schreibzugriff. Der Feedback-Wert, der in die MQPMR-Struktur einer Nachricht eingefügt werden soll, wenn die Nachricht in eine Warteschlange eingereiht wird.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Lang

Werte:
Siehe unter Feedback (MQLONG).

Syntax: Zum Abrufen: *feedback & = MQDistributionListElement*. **Feedback**

Zum Setzen: *MQDistributionListItem*. **Feedback** = *Feedback &*

Eigenschaft GroupId

Lese-/Schreibzugriff. Die GroupId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird. Ihr Anfangswert besteht aus lauter Nullen.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge von 24 Zeichen

Syntax: Zum Abrufen: *groupid$ = MQDistributionListItem*. **GroupId**

Zum Setzen: *MQDistributionListItem*. **GroupId** = *groupid$*

Eigenschaft 'GroupIdHex'

Lese-/Schreibzugriff. Die GroupId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird.

Je zwei Zeichen in der Zeichenfolge entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 48 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0..0'.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Eine Zeichenfolge mit 48 Hexadezimalzeichen, die 24-ASCII-Zeichen darstellen.

Syntax: To get: *groupidh$ = MQDistributionListItem*. **GroupIdHex**

Zum Setzen: *MQDistributionListItem*. **GroupIdHex** = *groupidh$*

Eigenschaft MessageId

Lese-/Schreibzugriff. Die MessageId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird. Ihr Anfangswert besteht aus lauter Nullen.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge von 24 Zeichen

Syntax: To get: *messageid$ = MQDistributionListItem*. **MessageId**

Zum Setzen: *MQDistributionListItem*. **MessageId** = *messageid$*

Eigenschaft 'MessageIdHex'

Lese-/Schreibzugriff. Die MessageId, die in den MQPMR einer Nachricht einbezogen werden soll, wenn sie in eine Warteschlange eingereiht wird.

Je zwei Zeichen in der Zeichenfolge entsprechen der hexadezimalen Darstellung eines einzelnen ASCII-Zeichens. Das Zeichenpaar '6' und '1' beispielsweise stellt das einzelne Zeichen 'A' dar, das Zeichenpaar '6' und '2' das einzelne Zeichen 'B' usw.

Sie müssen 48 gültige Hexadezimalzeichen bereitstellen.

Der Anfangswert ist '0..0'.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Eine Zeichenfolge mit 48 Hexadezimalzeichen, die 24 ASCII-Zeichen darstellen.

Syntax: To get: *messageidh$ = MQDistributionListItem*. **MessageIdHex**

Zum Setzen: *MQDistributionListItem*. **MessageIdHex** = *messageidh$*

Eigenschaft NextDistributionListItem

Schreibgeschützt. Das nächste Objekt der Verteilerlistenelemente, das derselben Verteilerliste zugeordnet ist.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
MQDistributionListItem

Syntax: To get: *set distributionlistitem = MQDistributionListItem*. **NextDistributionListItem**

Eigenschaft PreviousDistributionListItem

Schreibgeschützt. Das vorherige Objekt der Verteilerlistenelemente, das derselben Verteilerliste zugeordnet ist.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
MQDistributionListItem

Syntax: To get: *set distributionlistitem = MQDistributionListItem*. **PreviousDistributionListItem**

Eigenschaft 'QueueManagerName'

Lese-/Schreibzugriff. Der Name des IBM MQ-Warteschlangenmanagers.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge von 48 Zeichen

Syntax: To get: *qmname$ = MQDistributionListItem*. **QueueManagerName**

Zum Setzen: *MQDistributionListItem*. **QueueManagerName** = *qmname$*

QueueName, Eigenschaft

Lese-/Schreibzugriff. Der IBM MQ-Warteschlangenname.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge von 48 Zeichen

Syntax: To get: *qname$ = MQDistributionListItem*. **Name_der_Warteschlange**

Zum Setzen: *MQDistributionListItem*. **QueueName** = *qname$*

Eigenschaft 'ReasonCode'

Schreibgeschützt. Der Ursachencode, der von der zuletzt für das übergeordnete Verteilerlistenobjekt aufgerufenen Open- oder Put-Anforderung gesetzt wurde.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Lang

Werte:
Siehe API-Beendigungs-und Ursachencodes.

Syntax: Für Abruf:

```
reasoncode& = MQDistributionListItem.ReasonCode
```

Eigenschaft ReasonName

Schreibgeschützt. Der symbolische Name für den Ursachencode, beispielsweise MQRC_QMGR_NOT_AVAILABLE.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge

Werte:
Siehe API-Beendigungs-und Ursachencodes.

Syntax: Zum Abrufen: *reasonname$ = MQDistributionListItem*. **ReasonName**

ClearErrorCodes-Methode

Setzt für die MQDistributionListItem-Klasse und die MQSession-Klasse den Beendigungscode auf MQCC_OK und den Ursachencode auf MQRC_NONE zurück.

Definiert in:
Klasse MQDistributionListItem

Syntax:

```
Call MQDistributionListItem.ClearErrorCodes
```

Tracing von IBM MQ Automation Classes for ActiveX

Informationen zur für IBM MQ Automation Classes for ActiveX bereitgestellten Tracefunktion, zu häufig auftretenden Problemen und Hilfe zu ihrer Vermeidung.

V 9.0.0 Ab IBM MQ 9.0 wird die Unterstützung für MicrosoftActive X nicht mehr verwendet. IBM MQ-Klassen für .NET sind die empfohlene Ersatztechnologie. Weitere Informationen finden Sie unter .NET-Anwendungen entwickeln.

Im folgenden Abschnitt wird die bereitgestellte Tracefunktion erklärt; außerdem werden Details zu häufig auftretenden Problemen und Hilfe zu ihrer Vermeidung angeboten:

- „Tracefunktion für IBM MQ Automation Classes for ActiveX steuern” auf Seite 685
- „IBM MQ Automation Classes for ActiveX-Script schlägt fehl” auf Seite 686
- „Ursachencodes für IBM MQ Automation Classes for ActiveX” auf Seite 687
- „Tool zum Ermitteln der Codeversion” auf Seite 691

Tracefunktion für IBM MQ Automation Classes for ActiveX steuern

IBM MQ Automation Classes for ActiveX (MQAX) beinhaltet eine Tracefunktion, mit der die Serviceorganisation bei Auftreten eines Problems feststellen kann, was geschehen ist.

Sie zeigt die Pfade an, die bei der Ausführung Ihres MQAX-Scripts genommen werden. Wenn keine Probleme vorliegen, sollte die Tracefunktion deaktiviert sein, um eine unnötige Belegung der Systemressourcen zu vermeiden.

Die Tracefunktion wird über drei Umgebungsvariablen gesteuert:

- OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

Anmerkung: Durch Angabe des Werts *any* für die Variable **OMQ_TRACE** wird die Tracefunktion eingeschaltet. Selbst wenn Sie die Variable **OMQ_TRACE** auf OFF setzen, ist der Trace weiter aktiv. Wenn Sie die Tracefunktion ausschalten möchten, geben Sie keinen Wert für **OMQ_TRACE** an.

1. Klicken Sie auf **Start**.
2. Klicken Sie auf **Systemsteuerung**.
3. Doppelklicken Sie auf **System**.
4. Klicken Sie auf **Erweitert**
5. Klicken Sie auf **Umgebungsvariablen**.
6. Klicken Sie im Bereich **Benutzervariablen für (Benutzernamen)** auf **Neu**.
7. Geben Sie den Namen der Variablen und einen gültigen Namen in den entsprechenden Feldern ein und klicken Sie auf **OK**.
8. Klicken Sie auf **OK**, um das Fenster 'Umgebungsvariablen' zu schließen.
9. Klicken Sie auf **OK** 'Systemeigenschaften' zu schließen.
10. Schließen Sie das Fenster 'Systemsteuerung'.

Bei der Auswahl des Verzeichnisses, in das die Tracedateien geschrieben werden sollen, müssen Sie sicherstellen, dass Sie sowohl über Lese- als auch Schreibberechtigung für diesen Datenträger verfügen.

Die aktivierte Tracefunktion verlangsamt die Ausführung des MQAX-Scripts, die Leistung der ActiveX- oder IBM MQ-Umgebung wird jedoch nicht beeinträchtigt. Wenn eine Tracedatei nicht mehr benötigt wird, kann sie gelöscht werden.

Der Status der Variable OMQ_TRACE kann nicht geändert werden, solange MQAX aktiv ist.

Name und Verzeichnis der Tracedatei

Der Name der Tracedatei hat das Format `OMQ`*nnnnn*`.trc`, wobei *nnnnn* die ID des aktiven ActiveX-Prozesses angibt, der zu dem Zeitpunkt ausgeführt wird.

Befehl	Aktion
SET OMQ_TRACE_PATH = Laufwerk:\Verzeichnis	Gibt das Traceverzeichnis an, in das die Tracedatei geschrieben wird.
SET OMQ_TRACE_PATH =	Entfernt alle vorhandenen Einstellungen für das Traceverzeichnis. Wenn das Traceverzeichnis nicht festgelegt ist, wird das aktuelle Arbeitsverzeichnis (beim Start von ActiveX) verwendet.
ECHO %OMQ_TRACE_PATH%	Zeigt die aktuelle Einstellung für das Traceverzeichnis unter Windows an.
SET OMQ_TRACE = xxxxxxxx	Schaltet die Tracefunktion ein. Die Tracefunktion wird aktiviert, indem nach dem Gleichheitszeichen (=) mindestens ein Zeichen angegeben wird. Beispiel: `SET OMQ_TRACE=yes` und `SET OMQ_TRACE=no`. In beiden Beispielen ist die Traceerstellung aktiviert. Diese Einstellung ist nur für ein einzelnes Fenster bzw. eine einzelne Sitzung wirksam.
SET OMQ_TRACE=	Schaltet die Tracefunktion aus.
ECHO %OMQ_TRACE%	Zeigt den Inhalt der Umgebungsvariablen unter Windows an.
SET	Zeigt den Inhalt aller Umgebungsvariablen unter Windows an.
SET OMQ_TRACE_LEVEL = 9	Setzt die Tracestufe auf 9. Werte größer als 9 erzeugen keine zusätzlichen Informationen in der Tracedatei.

IBM MQ Automation Classes for ActiveX-Script schlägt fehl

Bei Fehlschlagen Ihres IBM MQ Automation Classes for ActiveX-Scripts finden Sie in der umfangreichen Referenzliteratur Hilfe.

Bericht mit den Symptomen beim ersten Auftreten eines Fehlers

Unabhängig von der Tracefunktion wird bei unerwarteten und internen Fehlern unter Umständen ein Bericht mit Symptomen beim Auftreten eines ersten Fehlers erstellt.

Dieser Bericht wird in der Datei `OMQ`*nnnnn*`.fdc` generiert; dabei steht *nnnnn* für die Nummer des ActiveX-Prozesses, der zum Zeitpunkt des Fehlers aktiv war. Diese Datei befindet sich in dem Arbeitsverzeichnis, von dem aus ActiveX gestartet wurde, oder in dem in der Umgebungsvariablen OMQ_PATH angegebenen Pfad.

Weitere Informationsquellen

IBM MQ stellt abhängig von der Plattform verschiedene Fehlerprotokolle und Traceinformationen bereit. Diese finden Sie im Anwendungsereignisprotokoll von Windows.

Ursachencodes für IBM MQ Automation Classes for ActiveX

Ursachencodes für IBM MQ Automation Classes for ActiveX (MQAX), die zusätzlich zu MQI-Ursachencodes von IBM MQ auftreten können.

Zusätzlich zu den für die MQI von IBM MQ dokumentierten Ursachencodes können noch die folgenden Ursachencodes auftreten. Informationen zu anderen Codes finden sie im IBM MQ-Anwendungsereignisprotokoll.

Ursachencode	Beschreibung
MQRC_LIBRARY_LOAD_ERROR (6000)	Mindestens eine der IBM MQ-Bibliotheken konnte nicht geladen werden. Überprüfen Sie, ob sich alle IBM MQ-Bibliotheken im korrekten Suchpfad auf dem von Ihnen verwendeten System befinden. Stellen Sie beispielsweise sicher, dass sich die Verzeichnisse, die IBM MQ-Bibliotheken enthalten, in PATH befinden.
MQRC_CLASS_LIBRARY_ERROR (6001)	Einer der IBM MQ `classlibrary` -Aufrufe hat einen unerwarteten Wert **ReasonCode** oder **CompletionCode** zurückgegeben. Überprüfen Sie den Bericht mit den Symptomen beim ersten Auftreten eines Fehlers. Notieren Sie sich die zuletzt verwendete Methode/Eigenschaft und Klasse und melden Sie das Problem an den IBM Support.
MQRC_STRING_LENGTH_TOO_BIG (6002)	Es wurde versucht, eine Zeichenfolge im UTF-Format mit einer Länge von mehr als 65.535 Bytes in den Nachrichtenpuffer zu schreiben.
MQRC_WRITE_VALUE_ERROR (6003)	Es wird ein Wert verwendet, der außerhalb des gültigen Bereichs liegt, z. B. `msg.WriteByte (240)`.
MQRC_PACKED_DECIMAL_ERROR (6004)	Es wurde versucht, eine gepackte Dezimalzahl aus dem Nachrichtenpuffer zu lesen, aber die Daten, auf die vom Datenzeiger verwiesen wird, haben kein gültiges Format für gepackte Daten.
MQRC_FLOAT_CONVERSION_ERROR (6005)	Es wurde versucht eine Gleitkommazahl des Typs 'Single' oder 'Double' aus dem Nachrichtenpuffer zu lesen, doch die Daten, auf die vom Datenzeiger verwiesen wird, haben nicht das entsprechende Gleitkommaformat.
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	Ein geöffnetes Objekt verfügt nicht über die richtigen **OpenOptions**-Einstellungen; es ist mindestens eine weitere Option erforderlich. Es ist ein implizites erneutes Öffnen erforderlich. Ein Schließen wurde jedoch verhindert, weil die Warteschlange für eine exklusive Eingabe geöffnet ist und durch ein Schließen andere die Möglichkeit hätten, auf die Warteschlange zuzugreifen. Legen Sie die **OpenOptions**-Werte explizit so fest, dass alle möglichen Fälle abgedeckt sind und kein implizites erneutes Öffnen erforderlich ist.

Ursachencode	Beschreibung
MQRC_REOPEN_INQUIRE_ERROR (6101)	Ein geöffnetes Objekt verfügt nicht über die richtigen **OpenOptions**-Einstellungen; es ist mindestens eine weitere Option erforderlich. Ein implizites erneutes Öffnen ist erforderlich, aber das Schließen wurde verhindert, da mindestens ein Merkmal des Objekts vor dem Schließen dynamisch überprüft werden muss und die Werte für **OpenOptions** nicht bereits die Option **MQOO_INQUIRE** enthalten. Legen Sie die **OpenOptions** -Werte explizit fest, um die Option **MQOO_INQUIRE** einzuschließen.
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	Ein geöffnetes Objekt verfügt nicht über die richtigen **OpenOptions**-Einstellungen; es ist mindestens eine weitere Option erforderlich. Es ist ein implizites erneutes Öffnen erforderlich. Das Schließen wurde jedoch verhindert, weil die Warteschlange mit der Option **MQOO_SAVE_ALL_CONTEXT** geöffnet ist, und es wurde vorher ein destruktiver `Get`-Aufruf ausgeführt. Dadurch wurden Statusinformationen, die beibehalten wurden, der geöffneten Warteschlange zugeordnet, und diese Informationen würden beim Schließen verloren gehen. Legen Sie die **OpenOptions**-Werte explizit so fest, dass alle möglichen Fälle abgedeckt sind und kein implizites erneutes Öffnen erforderlich ist.
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	Ein geöffnetes Objekt verfügt nicht über die richtigen **OpenOptions**-Einstellungen; es ist mindestens eine weitere Option erforderlich. Es ist ein implizites erneutes Öffnen erforderlich. Das Schließen wurde jedoch verhindert, weil die Warteschlange eine lokale Warteschlange des Definitionstyps **MQQDT_TEMPORARY_DYNAMIC** ist, die durch Schließen zerstört würde. Legen Sie die **OpenOptions**-Werte explizit so fest, dass alle möglichen Fälle abgedeckt sind und kein implizites erneutes Öffnen erforderlich ist.
MQRC_ATTRIBUTE_LOCKED (6104)	Es wurde versucht, den Wert oder das Attribut eines Objekt zu ändern, während dieses Objekt geöffnet war. Bestimmte Attribute wie beispielsweise **AlternateUserId** dürfen nicht geändert werden, solange das Objekt geöffnet ist.
MQRC_CURSOR_NOT_VALID (6105)	Der Anzeigecursor für eine geöffnete Warteschlange wurde seit seiner letzten Verwendung durch ein implizites erneutes Öffnen inaktiviert. Legen Sie die **OpenOptions**-Werte explizit so fest, dass alle möglichen Fälle abgedeckt sind und kein implizites erneutes Öffnen erforderlich ist.
MQRC_ENCODING_ERROR (6106)	Bei der Codierung des nächsten Nachrichtenelements muss es sich um **MQENC_NATIVE** handeln, damit ein Lesevorgang möglich ist.

Ursachencode	Beschreibung
MQRC_STRUCID_ERROR (6107)	Die Struktur der ID für das nächste Nachrichtenelement, die von den vier Zeichen ab dem Datenzeiger abgeleitet wird, fehlt oder ist nicht mit dem Variablentyp kompatibel, in den das Element gelesen werden soll.
MQRC_NULL_POINTER (6108)	Es wurde ein Nullzeiger bereitgestellt, es ist jedoch ein Nicht-Nullzeiger erforderlich bzw. ein Nicht-Nullzeiger wurde impliziert. Der Grund dafür kann die Verwendung expliziter Deklarationen für IBM MQ-Objekte sein, die von Visual Basic oder Excel als Parameter für Aufrufe verwendet werden. Beispiel: • In Visual Basic funktioniert `dim msg as Object` ordnungsgemäß, während `dim msg as MqMessage` möglicherweise nicht ordnungsgemäß funktioniert. • Aus Visual Basic mit einer definierten und festgelegten Warteschlange funktioniert `dim msg as MqMessageq.put msg` ordnungsgemäß, während dieser Befehl aus Excel eine `MQRC_NULL_POINTER`-Ausnahme generiert.
MQRC_NO_CONNECTION_REFERENCE (6109)	Das Objekt `MQQueue` hat seine Verbindung zum Objekt `MQQueueManager` verloren. Dieser Fall tritt ein, wenn die Verbindung zum Warteschlangenmanager getrennt wird. Löschen Sie das Objekt `MQQueue`.
MQRC_NO_BUFFER (6110)	Es ist kein Puffer verfügbar. Für ein `MQMessage`-Objekt kann kein Puffer zugeordnet werden, weil im Objektstatus eine interne Inkonsistenz vorhanden ist.
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	Die Länge der Binärdaten stimmt mit der Länge des Zielattributs nicht überein. Null ist eine richtige Länge für alle Attribute. 24 ist eine korrekte Länge für ein **`CorrelationId`**-Attribut und für ein Attribut **`MessageId`**-Attribut. 32 ist eine korrekte Länge für ein **`AccountingToken`**-Attribut.
MQRC_BUFFER_NOT_AUTOMATIC (6112)	Die Größe eines benutzerdefinierten und verwalteten Puffers kann nicht geändert werden. Da Nachrichtenpuffer vom System verwaltet werden, deutet dies auf eine interne Inkonsistenz hin.
MQRC_INSUFFICIENT_BUFFER (6113)	Nach dem Datenzeiger ist nicht genügend Pufferspeicher für die Anforderung verfügbar. Dies liegt unter Umständen daran, dass die Größe des Puffers nicht geändert werden kann.
MQRC_INSUFFICIENT_DATA (6114)	Nach dem Datenzeiger ist kein ausreichender Pufferspeicher für die Leseanforderung verfügbar. Reduzieren Sie den Puffer auf die korrekte Größe und lesen Sie die Daten erneut.

Ursachencode	Beschreibung
MQRC_DATA_TRUNCATED (6115)	Daten wurden beim Kopieren von einem Puffer zu einem anderen abgeschnitten. Mögliche Ursachen sind, dass die Größe des Zielpuffers nicht geändert werden kann, dass ein Fehler bei der Adressierung einer der Puffer aufgetreten ist oder dass ein Puffer durch einen kleineren Puffer ersetzt wurde.
MQRC_ZERO_LENGTH (6116)	Für die Länge wurde null angegeben, obwohl ein positiver Wert angegeben werden muss bzw. vorausgesetzt wird.
MQRC_NEGATIVE_LENGTH (6117)	Für die Länge wurde ein negativer Wert angegeben, obwohl null oder ein positiver Wert angegeben werden muss.
MQRC_NEGATIVE_OFFSET (6118)	Für die relative Position wurde ein negativer Wert angegeben, obwohl null oder ein positiver Wert angegeben werden muss.
MQRC_INCONSISTENT_FORMAT (6119)	Das Format des nächsten Nachrichtenelements ist nicht mit dem Variablentyp kompatibel, in den das Element gelesen werden soll.
MQRC_INCONSISTENT_OBJECT_STATE (6120)	Es besteht eine Inkonsistenz zwischen diesem Objekt, das geöffnet ist, und dem referenzierten `MQQueueManager`-Objekt, das nicht verbunden ist.
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	Die `MQPutMessageOptions`-Kontextreferenz referenziert kein gültiges `MQQueue`-Objekt. Das Objekt wurde zuvor gelöscht.
MQRC_CONTEXT_OPEN_ERROR (6122)	Die `MQPutMessageOptions`-Kontextreferenz verweist auf ein `MQQueue`-Objekt, das nicht geöffnet werden konnte, um einen Kontext herzustellen. Eine mögliche Ursache ist, dass für das `MQQueue`-Objekt ungeeignete Optionen für das Öffnen definiert sind. Überprüfen Sie den Ursachencode für das Objekt, auf das verwiesen wird, und ermitteln Sie die Ursache.
MQRC_STRUC_LENGTH_ERROR (6123)	Die Länge einer internen Datenstruktur ist nicht konsistent mit ihrem Inhalt. Bei einem `MQRMH`-Header reicht die Länge nicht aus, um die festen Felder und alle Offsetdaten aufzunehmen.
MQRC_NOT_CONNECTED (6124)	Eine Methode ist fehlgeschlagen, weil eine erforderliche Verbindung zu einem Warteschlangenmanager nicht verfügbar ist und eine Verbindung implizit nicht hergestellt werden kann.
MQRC_NOT_OPEN (6125)	Eine Methode ist fehlgeschlagen, weil ein IBM MQ-Objekt nicht geöffnet ist und ein Öffnen implizit nicht durchführbar ist.
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	Eine `MQDistributionList` konnte nicht geöffnet werden, weil in der Verteilerliste keine `MQDistributionListItem`-Objekte enthalten sind. Problembehebung: Fügen Sie mindestens ein `MQDistributionListItem`-Objekt zur Verteilerliste hinzu.

Ursachencode	Beschreibung
MQRC_INCONSISTENT_OPEN_OPTIONS (6127)	Eine Methode ist fehlgeschlagen, da das Objekt geöffnet ist und die Optionen für das Öffnen nicht mit der erforderlichen Operation kompatibel sind. Problembehebung: Öffnen Sie das Objekt mit geeigneten Optionen für das Öffnen und wiederholen Sie den Vorgang.
MQRC_WRONG_VERSION (6128)	Bei einer Methode ist ein Fehler aufgetreten, weil eine angegebene oder vorgefundene Versionsnummer falsch ist oder nicht unterstützt wird.

Tool zum Ermitteln der Codeversion

Unter Umständen möchte das Service-Team bei IBM wissen, welche Codeversion Sie installiert haben.

Sie können die Codeversion mithilfe des Dienstprogramms 'MQAXLEV' ermitteln.

Wechseln Sie in einer Eingabeaufforderung in das Verzeichnis, das die Datei 'MQAX200.dll' enthält, oder geben Sie die vollständige Pfadlänge an und geben Sie Folgendes ein:

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

Dabei ist MQAXLEV.OUT der Name der Ausgabedatei.

Wenn Sie keine Ausgabedatei angeben, werden die Informationen am Bildschirm angezeigt.

Im Folgenden ist ein Beispiel für eine Ausgabedatei zu sehen, die vom Tool zur Ermittlung der Codeversion ausgegeben wird:

Beispielausgabedatei des Tools zur Ermittlung der Codeversion

```
5639-B43 (C) Copyright IBM Corp. 1996, 2023. ALL RIGHTS RESERVED.
     ***** Code Level is 5.1 *****
lib/mqole/mqole.cpp, mqole, p000, p000 L981119       1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212      1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212   1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216        1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212       1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212       1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212      1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212      1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212      1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219        1.11 99/02/18 12:12:59
```

ActiveX-Schnittstelle zur MQAI

Eine kurze Übersicht über die COM-Schnittstellen und ihre Verwendung in der MQAI finden Sie unter „Component Object Model-Schnittstelle verwenden (IBM MQ Automation Classes for ActiveX)” auf Seite 608.

Mithilfe der MQAI können Anwendungen PCF-Befehle (Programmable Command Format) erstellen, ohne die für PCF erforderlichen Puffer variabler Länge direkt abzurufen und zu formatieren. Weitere Informationen zur MQAI finden Sie im Abschnitt IBM MQ Administration Interface (MQAI). Die MQAI-ActiveX-Klasse 'MQBag' kapselt die von der MQAI unterstützten Datenbehälter so, dass sie in jeder Programmiersprache verwendet werden können, die die Erstellung von COM-Objekten unterstützt (beispielsweise Visual Basic, C++, Java und andere ActiveX-Scripting-Clients).

Die MQAI-ActiveX-Schnittstelle ist für die Verwendung zusammen mit MQAX-Klassen gedacht, die eine COM- Schnittstelle zur MQI bereitstellen. Weitere Informationen zu den MQAX-Klassen finden Sie unter „MQAX-Anwendungen entwerfen, die auf Nicht-ActiveX-Anwendungen zugreifen” auf Seite 609.

Die ActiveX-Schnittstelle stellt eine einzige Klasse (MQBag) bereit. Mit dieser Klasse werden MQAI-Datenbehälter erstellt und mit ihren Eigenschaften und Methoden werden Datenelemente in den einzelnen Datenbehältern erstellt und bearbeitet. Die MQBag-Methode 'Execute' sendet die Daten in den Datenbehältern als PCF-Nachricht an einen IBM MQ-Warteschlangenmanager und sammelt die Antworten.

Weitere Informationen zur Klasse 'MQBag' und ihren Eigenschaften und Methoden finden Sie unter „Klasse 'MQBag'” auf Seite 692.

Die PDF-Nachricht wird an das angegebene Warteschlangenmanagerobjekt gesendet, optional unter Angabe von Anforderungs- und Antwortwarteschlangen. Die Antworten werden in einem neuen MQBag-Objekt zurückgegeben. Eine Beschreibung aller Befehle und Antworten finden Sie unter Definitions of the Programmable Command Formats. Befehle können über die Auswahl der entsprechenden Anforderungs- und Antwortwarteschlangen an alle Warteschlangenmanager im IBM MQ-Netz gesendet werden.

Klasse 'MQBag'

Mit der Klasse 'MQBag' werden MQBag-Objekte nach Bedarf erstellt. Wird die Klasse 'MQBag' instanziiert, gibt sie einen neuen MQBag-Objektverweis zurück.

So erstellen Sie ein MQBag-Objekt in Visual Basic:

```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

Eigenschaften des MQBag-Objekts

Informationen zu den Eigenschaften der MQBag-Objekte finden Sie unter den folgenden Links:

- „Item, Eigenschaft” auf Seite 693.
- „Eigenschaft 'Count'” auf Seite 694.
- „Eigenschaft Options” auf Seite 694.

MQBag-Methoden

Informationen zu den Methoden der MQBag-Objekte finden Sie unter den folgenden Links:

- „Methode 'Add'” auf Seite 695.
- „AddInquiry, Methode” auf Seite 696.
- „Methode 'Clear'” auf Seite 696.
- „Methode 'Execute'” auf Seite 696.
- „Methode 'FromMessage'” auf Seite 697.
- „Methode 'ItemType'” auf Seite 697.
- „Remove, Methode” auf Seite 698.
- „Selector, Methode” auf Seite 699.
- „Methode 'ToMessage'” auf Seite 699.
- „Truncate, Methode” auf Seite 700.

Fehlerbehandlung

Wird bei der Ausführung einer Operation für ein MQBag-Objekt ein Fehler festgestellt (einschließlich der Fehler, die von einem zugrunde liegenden MQAX- oder MQAI-Objekt an den Datenbehälter zurückgegeben werden), wird eine Fehlerausnahmebedingung ausgelöst. Die Klasse 'MQBag' unterstützt die

COM-Schnittstelle 'ISupportErrorInfo', daher stehen Ihrer Fehlerbehandlungsroutine die folgenden Informationen zur Verfügung:

- Fehlernummer: Sie setzt sich aus dem IBM MQ-Ursachencode für den Fehler, der festgestellt wurde, und einem COM-Facility-Code zusammen. Das Facility-Feld gibt bei COM standardmäßig den Bereich an, aus dem der Fehler stammt. Für Fehler, die von IBM MQ erkannt werden, ist dies immer FACILITY_ITF.
- Fehlerquelle: Gibt Typ und Version des Objekts an, von dem der Fehler festgestellt wurde. Bei Fehlern, die während einer MQBag-Operation erkannt wurden, ist die Fehlerquelle immer 'MQBag.MQBag1'.
- Fehlerbeschreibung: Die Zeichenfolge mit dem symbolischen Namen des IBM MQ-Ursachencodes.

Der Zugriff auf die Fehlerinformationen hängt von der von Ihnen verwendeten Scriptsprache ab; in Visual Basic beispielsweise werden die Informationen im Err-Objekt zurückgegeben und den IBM MQ-Ursachencode erhält man durch Subtraktion der Konstante 'vbObjectError' von 'Err.Number'.

ReasonCode = Err.Number - vbObjectError

Wenn die MQBag-Methode 'Execute' eine PCF-Nachricht sendet und es wird eine Antwort empfangen, wird die Operation als erfolgreich angesehen, auch wenn der gesendete Befehl unter Umständen fehlgeschlagen ist. In diesem Fall enthält der Antwortbehälter selbst den Beendigungs- und Ursachencode (siehe Definitions of the Programmable Command Formats).

Item, Eigenschaft

Verwendungszweck

Die Eigenschaft 'Item' stellt ein Element in einem Datenbehälter dar. Mit dieser Eigenschaft wird der Wert eines Elements gesetzt bzw. abgefragt. Die Verwendung dieser Eigenschaft entspricht den folgenden MQAI-Aufrufen:

- mqSetString
- mqSetInteger
- mqInquireInteger
- mqInquireString
- mqInquireBag

(siehe Programmierbare Befehlsformate-Referenz).

Format

Item (Selector, ItemIndex, Value)

Parameter

Selector (VARIANT) - Eingabe
Der Selektor des Elements, das gesetzt bzw. abgefragt werden soll.

Bei einer Abfrage des Elements ist MQSEL_ANY_USER_SELECTOR der Standardwert. Soll ein Element gesetzt werden, ist MQIA_LIST oder MQCA_LIST der Standardwert.

Wenn es sich bei `Selector` nicht um den Typ 'long' handelt, wird MQRC_SELECTOR_TYPE_ERROR zurückgegeben.

Dieser Parameter ist optional.

ItemIndex (LONG) - Eingabe
Dieser Wert gibt das Vorkommen des Elements mit dem angegebenen Selektor an, das gesetzt bzw. abgefragt werden soll. Der Standardwert ist MQIND_NONE.

Dieser Parameter ist optional.

Value (VARIANT) - Ein-/Ausgabe
Der Wert, der zurückgegeben wird, bzw. der Wert, der gesetzt wird. Bei der Abfrage eines Elements wird ein Wert des Typs 'Long', 'String' oder 'MQBag' zurückgegeben; soll ein Element gesetzt werden, muss es sich um einen Wert des Typs 'Long' oder 'String' handeln, andernfalls wird MQRC_ITEM_VALUE_ERROR zurückgegeben.

Aufruf in Visual Basic

Wert eines Elements in einem Datenbehälter abfragen:

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

MQBag-Verweise:

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

Wert eines Elements in einem Datenbehälter setzen:

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

Eigenschaft 'Count'

Verwendungszweck

Die Eigenschaft 'Count' stellt die Anzahl der Datenelemente in einem Datenbehälter dar. Diese Eigenschaft entspricht dem MQAI-Aufruf 'mqCountItems' in Programmierbare Befehlsformate-Referenz.

Format

Count (*Selector, Value*)

Parameter

Selector (VARIANT) - Eingabe
Selektor der Datenelemente, die in der Anzahl berücksichtigt werden sollen.

Der Standardwert ist MQSEL_ALL_USER_SELECTORS.

Wenn es sich bei `Selector` nicht um den Typ 'long' handelt, wird MQRC_SELECTOR_TYPE_ERROR zurückgegeben.

Value (LONG) - Ausgabe
Die Anzahl der Elemente im Datenbehälter, die vom *`Selector`* einbezogen wird.

Aufruf in Visual Basic

Anzahl der Elemente in einem Datenbehälter zurückgeben:

```
ItemCount = mqbag.Count([Selector])
```

Eigenschaft Options

Verwendungszweck

Mit der Eigenschaft 'Options' werden die Optionen für die Verwendung eines Datenbehälters gesetzt. Diese Eigenschaft entspricht dem Parameter **Options** des MQAI-Aufrufs 'mqCreateBag' in den Programmierbare Befehlsformate-Referenz.

Format

Options (*Options*)

Parameter

Options (LONG) - Ein-/Ausgabe
Die Datenbehälteroptionen.

Anmerkung: Diese Optionen müssen gesetzt werden, ***bevor*** Datenelemente einem Datenbehälter hinzugefügt bzw. in einem Datenbehälter gesetzt werden. Werden diese Optionen geändert, während der Datenbehälter noch Elemente enthält, wird MQRC_OPTIONS_ERROR zurückgegeben. Dies gilt auch für den Fall, dass der Datenbehälter anschließend gelöscht wird.

Aufruf in Visual Basic

Optionen eines Elements in einem Datenbehälter abfragen:

```
Options = mqbag.Options
```

Option eines Elements in einem Datenbehälter setzen:

```
mqbag.Options = Options
```

MQBag-Methoden

Die Methoden der MQBag-Objekte werden auf den folgenden Seiten erklärt.

Methode 'Add'

Verwendungszweck

Die Methode 'Add' fügt einem Datenbehälter Datenelemente hinzu. Diese Methode entspricht den MQAI-Aufrufen 'mqAddInteger' und 'mqAddString' in Programmierbare Befehlsformate-Referenz.

Format

Add (*Value, Selector*)

Parameter

Value (VARIANT) - Eingabe
Ganzzahl- oder Zeichenfolgewert des Datenelements.

Selector (VARIANT) - Eingabe
Selektor, der das Element angibt, das hinzugefügt werden soll.

Der Standardwert ist MQIA_LIST oder MQCA_LIST, abhängig vom Typ von `Value`. Wenn es sich bei **`Selector`** nicht um den Typ 'long' handelt, wird MQRC_SELECTOR_TYPE_ERROR zurückgegeben.

Aufruf in Visual Basic

Element einem Datenbehälter hinzufügen:

```
mqbag.Add(Value,[Selector])
```

AddInquiry, Methode

Verwendungszweck

Die Methode 'AddInquiry' fügt einen Selektor zur Angabe des Attributs hinzu, das zurückgegeben werden soll, wenn ein Verwaltungsbehälter zur Ausführung eines INQUIRE-Befehls gesendet wird. Diese Methode entspricht dem MQAI-Aufruf 'mqAddInquiry' in Programmierbare Befehlsformate-Referenz.

Format

AddInquiry *(Inquiry)*

Parameter

Inquiry (LONG) - Eingabe
Selektor für das IBM MQ-Attribut, das vom INQUIRE-Verwaltungsbefehl zurückgegeben werden soll.

Aufruf in Visual Basic

Verwendung der Methode 'AddInquiry':

```
mqbag.AddInquiry(Inquiry)
```

Methode 'Clear'

Verwendungszweck

Die Methode 'Clear' löscht sämtliche Datenelemente aus einem Datenbehälter. Diese Methode entspricht dem MQAI-Aufruf 'mqClearBag' in Programmierbare Befehlsformate-Referenz.

Format

Inhalt löschen

Aufruf in Visual Basic

Alle Datenelemente aus einem Datenbehälter löschen:

```
mqbag.Clear
```

Methode 'Execute'

Verwendungszweck

Die Methode 'Execute' sendet eine Verwaltungsbefehlsnachricht an den Befehlsserver und wartet anschließend auf Antwortnachrichten. Diese Methode entspricht dem MQAI-Aufruf 'mqExecute' in Programmierbare Befehlsformate-Referenz.

Format

Execute *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag*)

Parameter

QueueManager (MQQueueManager) - Eingabe
Der Warteschlangenmanager, mit dem die Anwendung verbunden ist.

Command (LONG) - Eingabe
Der Befehl, der ausgeführt werden soll.

OptionsBag (MQBag) - Eingabe
Der Datenbehälter mit den Optionen, die die Verarbeitung des Aufrufs beeinflussen.

RequestQ (MQQueue) - Eingabe
Die Warteschlange, in die die Verwaltungsbefehlsnachricht eingereiht werden wird.

ReplyQ (MQQueue) - Eingabe
Die Warteschlange, in der die Antwortnachrichten empfangen werden.

ReplyBag (MQBag) - Ausgabe
Ein Behälterverweis, der Daten aus Antwortnachrichten enthält.

Aufruf in Visual Basic

Verwaltungsbefehlsnachricht senden und auf Antworten warten:

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

Methode 'FromMessage'

Verwendungszweck

Die Methode 'FromMessage' lädt Daten aus einer Nachricht in einen Datenbehälter. Diese Methode entspricht dem MQAI-Aufruf 'mqBufferToBag' in Programmierbare Befehlsformate-Referenz.

Format

FromMessage *(Message, OptionsBag*)

Parameter

Message (MQMessage) - Eingabe
Die Nachricht mit den Daten, die konvertiert werden sollen.

OptionsBag (MQBag) - Eingabe
Optionen, mit denen die Verarbeitung des Aufrufs gesteuert wird.

Aufruf in Visual Basic

Daten aus einer Nachricht in einen Datenbehälter laden:

```
mqbag.FromMessage(Message,[OptionsBag])
```

Methode 'ItemType'

Verwendungszweck

Die Methode 'ItemType' gibt den Typ des Werts in einem angegebenen Element in einem Datenbehälter zurück. Diese Methode entspricht dem MQAI-Aufruf 'mqInquireItemInfo' in Programmierbare Befehlsformate-Referenz.

Format

ItemType *(Selector, ItemIndex, ItemType)*

Parameter

Selector (VARIANT) - Eingabe
Selektor, der das abzufragende Element kennzeichnet.

Der Standardwert ist MQSEL_ANY_USER_SELECTOR. Wenn es sich bei **Selector** nicht um den Typ 'long' handelt, wird MQRC_SELECTOR_TYPE_ERROR zurückgegeben.

ItemIndex (LONG) - Eingabe
Index der Elemente, die abgefragt werden sollen.

Der Standardwert ist MQIND_NONE.

ItemType (LONG) - Ausgabe
Der Datentyp des angegebenen Elements.

Anmerkung: Die Angabe des Parameters **Selector** und/oder des Parameters **ItemIndex** ist erforderlich. Ist keiner dieser Parameter vorhanden, wird MQRC_PARAMETER_MISSING zurückgegeben.

Aufruf in Visual Basic

Typ eines Werts zurückgeben:

```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

Remove, Methode

Verwendungszweck

Die Methode 'Remove' entfernt ein Element aus einem Datenbehälter. Diese Methode entspricht dem MQAI-Aufruf 'mqDeleteItem' in Programmierbare Befehlsformate-Referenz.

Format

Remove *(Selector, ItemIndex)*

Parameter

Selector (VARIANT) - Eingabe
Selektor, der das Element angibt, das gelöscht werden soll.

Der Standardwert ist MQSEL_ANY_USER_SELECTOR. Wenn es sich bei **Selector** nicht um den Typ 'long' handelt, wird MQRC_SELECTOR_TYPE_ERROR zurückgegeben.

ItemIndex (LONG) - Eingabe
Index des Elements, das gelöscht werden soll.

Der Standardwert ist MQIND_NONE.

Anmerkung: Die Angabe des Parameters **Selector** und/oder des Parameters **ItemIndex** ist erforderlich. Ist keiner dieser Parameter vorhanden, wird MQRC_PARAMETER_MISSING zurückgegeben.

Aufruf in Visual Basic

Element aus einem Datenbehälter löschen:

```
mqbag.Remove([Selector],[ItemIndex])
```

Selector, Methode

Verwendungszweck

Die Methode 'Selector' gibt den Selektor eines angegebenen Elements in einem Datenbehälter zurück. Diese Methode entspricht dem MQAI-Aufruf 'mqInquireItemInfo' in Programmierbare Befehlsformate-Referenz.

Format

Selector ***(Selector, ItemIndex, OutSelector)***

Parameter

Selector (VARIANT) - Eingabe
Selektor, der das abzufragende Element kennzeichnet.

Der Standardwert ist MQSEL_ANY_USER_SELECTOR. Wenn es sich bei **Selector** nicht um den Typ 'long' handelt, wird MQRC_SELECTOR_TYPE_ERROR zurückgegeben.

ItemIndex (LONG) - Eingabe
Index des Elements, das abgefragt werden soll.

Der Standardwert ist MQIND_NONE.

OutSelector (VARIANT) - Ausgabe
Der Selektor des angegebenen Elements.

Anmerkung: Die Angabe des Parameters **Selector** und/oder des Parameters **ItemIndex** ist erforderlich. Ist keiner dieser Parameter vorhanden, wird MQRC_PARAMETER_MISSING zurückgegeben.

Aufruf in Visual Basic

Selektor eines Elements zurückgeben:

```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

Methode 'ToMessage'

Verwendungszweck

Die Methode 'ToMessage' gibt einen Verweis auf ein MQMessage-Objekt zurück. Der Verweis enthält Daten aus einem Datenbehälter. Diese Methode entspricht dem MQAI-Aufruf 'mqBagToBuffer' in Programmierbare Befehlsformate-Referenz.

Format

ToMessage *(OptionsBag, Message)*

Parameter

OptionsBag (MQBag) - Eingabe
Ein Datenbehälter mit Optionen, über die die Verarbeitung der Methode gesteuert wird.

Message (MQMessage) - Ausgabe
Ein MQMessage-Objektverweis, der Daten aus dem Datenbehälter enthält.

Aufruf in Visual Basic

Verwendung der Methode 'ToMessage':

```
Set Message = mqbag.ToMessage([OptionsBag])
```

Truncate, Methode

Verwendungszweck

Die Methode 'Truncate' reduziert die Anzahl der Benutzerelemente in einem Datenbehälter. Diese Methode entspricht dem MQAI-Aufruf 'mqTruncateBag' in Programmierbare Befehlsformate-Referenz.

Format

Truncate *(ItemCount)*

Parameter

ItemCount (LONG) - Eingabe
Die Anzahl der Benutzerelemente, die nach der Reduzierung noch im Datenbehälter vorhanden sein sollen.

Aufruf in Visual Basic

Anzahl der Benutzerelemente in einem Datenbehälter reduzieren:

```
mqbag.Truncate(ItemCount)
```

Beispielprogramme zur Einführung in IBM MQ Automation Classes for ActiveX

Dieser Anhang enthält eine Beschreibung der Beispielprogramme zur Einführung in IBM MQ Automation Classes for ActiveX sowie Hinweise zu ihrer Verwendung.

IBM MQ for Windows stellt die folgenden Beispielprogramme in Visual Basic bereit:

- MQAXTRIV.VBP
- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

Diese Beispiele werden in Visual Basic 4 oder Visual Basic 5 ausgeführt. Sie befinden sich im Verzeichnis ... \tools\mqax\samples\vb.

Dieses Verzeichnis enthält auch Beispielprogramme für Microsoft Excel und HTML. Diese sind:

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

Anmerkung: Bei Verwendung von Visual Basic 5 **müssen** Sie die Visual Basic-Komponente 'grid32.ocx' auswählen und installieren.

Zweck der Beispielprogramme

Die Beispielprogramme zeigen Ihnen, wie Sie mit IBM MQ Automation Classes for ActiveX Folgendes ausführen können:

- Verbindung zu einem Warteschlangenmanager herstellen
- Auf eine Warteschlange zugreifen
- Eine Nachricht in eine Warteschlange einreihen
- Eine Nachricht aus einer Warteschlange abrufen

Der zentrale Teil des Visual Basic-Beispiels wird auf den folgenden Seiten veranschaulicht:

„Ausführung der Beispiele vorbereiten” auf Seite 701

„Fehlerbehandlung in den Beispielen” auf Seite 702

Ausführen, ActiveX-Einführungsbeispiele

Überprüfen Sie vor der Ausführung der Einführungsbeispiele für IBM MQ Automation Classes for ActiveX, ob der Standardwarteschlangenmanager aktiv ist und ob die erforderlichen Warteschlangendefinitionen erstellt wurden. Ausführliche Informationen zum Erstellen und Ausführen eines Warteschlangenmanagers sowie zum Erstellen einer Warteschlange finden Sie unter Verwalten. Das Beispiel verwendet die Warteschlange SYSTEM.DEFAULT.LOCAL.QUEUE , die auf jedem normal eingerichteten IBM MQ -Server definiert werden sollte

Im Folgenden sind die verschiedenen Verwendungsmöglichkeiten von Datenbehältern aufgeführt:

- Verbindung zu einem Warteschlangenmanager herstellen
- Auf eine Warteschlange zugreifen
- Eine Nachricht in eine Warteschlange einreihen
- Eine Nachricht aus einer Warteschlange abrufen

Informationen zu den MQAX-Einstiegsbeispielen für Microsoft Basic 4 oder höher finden Sie im Abschnitt „Beispielprogramm MQAXTRIV ausführen” auf Seite 702 .

Informationen zu einem Beispielprogramm, mit dem Sie Eigenschaften und Methoden von Warteschlangenmanagern und Warteschlangenobjekten durchsuchen können, finden Sie unter „Beispiel MQAXCLSS starten” auf Seite 703.

Informationen zum Beispiel MQAXDLST finden Sie unter „MQAXDLST-Beispiel” auf Seite 704.

Informationen zum Ausführen des MQAX-Einführungsbeispiels für Microsoft Excel 95 oder höher (MQAXTRIV.XLS) finden Sie unter „Beispiel MQAXTRIV.XLS ausführen” auf Seite 704.

Informationen zum Ausführen der Bankdemo mit MQAX.XLS finden Sie unter „Bankdemo mit MQAX.XLS ausführen” auf Seite 704.

Informationen zum Einführungsbeispiel unter Verwendung eines mit ActiveX kompatiblen Web-Browsers finden Sie unter „Beispielprogramm in einem mit ActiveX kompatiblen Web-Browser ausführen” auf Seite 704.

Ausführung der Beispiele vorbereiten

Für die Ausführung der Beispiele benötigen Sie eines der folgenden Programme, je nachdem, welches der Beispiele Sie ausführen möchten.

- Microsoft Visual Basic 4 (oder höher)
- Microsoft Excel 95 (oder neuer)
- Einen Web-Browser

Außerdem benötigen Sie Folgendes:

- Einen aktiven IBM MQ-Warteschlangenmanager.
- Eine bereits definierte IBM MQ-Warteschlange.

Fehlerbehandlung in den Beispielen

Die meisten Beispiele, die im IBM MQ Automation Classes for ActiveX-Paket angegeben sind, geben nur wenige oder keine Hinweise zur Fehlerbehandlung. Weitere Informationen zur Fehlerbehandlung finden Sie unter „Fehlerbehandlung” auf Seite 613.

Beispielprogramm MQAXTRIV ausführen

1. Starten Sie den Warteschlangenmanager.
2. Wählen Sie in Windows Explorer oder Dateimanager das Symbol für das Beispielprogramm (MQAXTRIV.VBP, eine Visual Basic-Projektdatei) aus und öffnen Sie die Datei.

   Visual Basic wird gestartet und die Datei MQAXTRIV.VBP geöffnet.
3. Drücken Sie in Visual Basic die Funktionstaste F5, um das Beispiel auszuführen.
4. Klicken Sie in das Fenster **MQAX trivial tester** (Einfaches MQAX-Prüftool).

Wenn keine Fehler auftreten, sollte der Fensterhintergrund grün werden. Treten Fehler bei der Konfiguration auf, sollte der Fensterhintergrund rot werden und Fehlerinformationen angezeigt werden.

Die folgende Abbildung zeigt den zentralen Teil des Visual Basic-Beispiels.

```
Option Explicit

Private Sub Form_Click()

'*****************************************************************************
'* This simple example illustrates how to put and get an IBM MQ message to
'* and from an IBM MQ message queue. The data from the message returned by the
'* get is read and compared with that from the original message.
'*****************************************************************************
Dim MQSess As MQSession                 '* session object
Dim QMgr As MQQueueManager              '* queue manager object
Dim Queue As MQQueue                    '* queue object
Dim PutMsg As MQMessage                 '* message object for put
Dim GetMsg As MQMessage                 '* message object for get
Dim PutOptions As MQPutMessageOptions   '* put message options
Dim GetOptions As MQGetMessageOptions   '* get message options
Dim PutMsgStr As String                 '* put message data string
Dim GetMsgStr As String                 '* get message data string
'*****************************************************************************
'* Handle errors
'*****************************************************************************
On Error GoTo HandleError

'*****************************************************************************
'* Initialize the current position for the form
'*****************************************************************************
CurrentX = 0
CurrentY = 0

'*****************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*****************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                        MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)

'*****************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*****************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions
```

```
'*************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127) '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:        """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
End If

Exit Sub
'*************************************************************************
'* Handle errors
'*************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer

Cls
BackColor = RGB(255, 0, 0)  '* set to red for error
Print "An error occurred as follows:"
Print ""
If MQSess.CompletionCode <> MQCC_OK Then
    ErrMsg = Err.Description
    StrPos = InStr(ErrMsg, " ")            '* search for first blank
    If StrPos > 0 Then
        Print Left(ErrMsg, StrPos)         '* print offending MQAX object name
    Else
        Print Error(Err)                   '* print complete error object
    End If
    Print ""
    Print "IBM MQ Completion Code = " & MQSess.CompletionCode
    Print "IBM MQ Reason Code = " & MQSess.ReasonCode
    Print "(" & MQSess.ReasonName & ")"
Else
    Print "Visual Basic error: " & Err
    Print Error(Err)
End If

Exit Sub

End Sub
```

Beispiel MQAXCLSS starten

Mit diesem Beispiel können Sie die Eigenschaften und Methoden für Warteschlangenmanager und Warteschlangenobjekte durchsuchen.

1. Starten Sie den Warteschlangenmanager.
2. Öffnen Sie die Datei MQAXCLSS.VBP, indem Sie doppelt auf das Dokumentsymbol in Windows Explorer klicken oder indem Sie in Visual Basic im Dateimenü auf 'File - Open' (Datei - Öffnen) klicken.

3. Starten Sie das Beispiel.
4. Geben Sie den Namen des Warteschlangenmanagers und der Warteschlange ein und klicken Sie auf die entsprechenden Schaltflächen.

MQAXDLST-Beispiel

Das MQAXDLST-Beispiel von Visual Basic demonstriert die Verwendung einer Verteilerliste, um dieselbe Nachricht mit einer Eingabe an zwei Warteschlangen zu senden. Um das Beispiel auszuführen, gehen Sie vor wie beim MQAXCLSS-Beispiel.

MQAX Starter-Beispiel für Microsoft Excel 95 oder neuer

In diesem Abschnitt wird erklärt, wie das MQAX Starter-Beispiel für Microsoft Excel 95 oder neuer, MQAXTRIV.XLS, ausgeführt wird.

Beispiel MQAXTRIV.XLS ausführen

1. Starten Sie den Warteschlangenmanager.
2. Wählen Sie im Explorer oder File Manager das Symbol für das MQAX-Beispiel MQAXTRIV.XLS.
3. Klicken Sie auf die Schaltfläche im Arbeitsblatt.
4. Die Anzeige wird mit einer Erfolgsmeldung (oder Fehlermeldung) aktualisiert.

Bankdemo mit MQAX.XLS ausführen

In diesem Abschnitt wird beschrieben, wie Sie die Bankdemo ausführen.

1. Starten Sie den Warteschlangenmanager.
2. Führen Sie den IBM MQ-MQSC-Befehl BANK.TST aus. Dabei werden die erforderlichen IBM MQ-Warteschlangendefinitionen vorgenommen.

   Hinweise zur Verwendung von MQSC-Befehlsdateien finden Sie unter Script (MQSC) Commands.
3. Führen Sie MQAXBSRV.VBP aus. Bei diesem Beispielprogramm handelt es sich um den Server, der eine Back-End-Anwendung simuliert; es muss mit Microsoft Excel ausgeführt werden.
4. Führen Sie MQAX.XLS aus. Bei diesem Beispielprogramm handelt es sich um die IBM MQ-Clientdemo.
5. Wählen Sie in der Liste einen Kunden aus.
6. Klicken Sie auf **Submit** (Übergeben).

Nach einer kurzen Wartezeit (ca. 3 Sekunden) werden die Felder mit Werten aufgefüllt und ein Balkendiagramm wird angezeigt.

Beispielprogramm in einem mit ActiveX kompatiblen Web-Browser ausführen

Anmerkung: Für die Ausführung dieses Beispielprogramms ist ein mit ActiveX kompatibler Web-Browser erforderlich (beispielsweise Microsoft Internet Explorer (nicht jedoch Netscape Navigator)).

Ausführen, HTML-Beispiel

Dieses Beispiel veranschaulicht den Aufruf von MQAX über VBScript und JavaScript.

1. Starten Sie den Warteschlangenmanager.
2. Öffnen Sie in dem mit ActiveX kompatiblen Web-Browser die Datei MQAXTRIV.HTM.

   Dazu können Sie entweder in Windows Explorer doppelt auf das Dateisymbol klicken oder im Dateimenü des mit ActiveX kompatiblen Web-Browsers auf 'File - Open' (Datei - Öffnen) klicken.
3. Gehen Sie entsprechend den Anweisungen am Bildschirm vor.

ULW V 9.0.0 AMQP-Clientanwendungen entwickeln

Die IBM MQ-Unterstützung für AMQP-APIs, einschließlich der MQ Light-API, 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 unter UNIX, Linux oder Windows installieren; unter IBM i und unter z/OS ist er nicht verfügbar.

Die MQ Light-API basiert auf dem Protokoll Oasis AMQP 1.0. Es gibt Messaging-APIs für Node.js, Java, Ruby und Python.

Eine Anwendung, die für die Verwendung der MQ Light-API entwickelt wird, kann entweder mit einer MQ Light-Runtime, einem IBM MQ-Warteschlangenmanager mit AMQP-Kanal oder einer Instanz eines MQ Light-Service in IBM Cloud (formerly Bluemix) verbunden werden.

AMQP-Clients entwickeln

Die MQ Light-API soll die Erstellung von Prototypen erleichtern und eine schnellere Entwicklung von Geschäftsanwendungen ermöglichen. Es gibt MQ Light-APIs für Node.js, Java, Ruby und Python, die unter https://github.com/mqlight verfügbar sind.

AMQP-Beispielclients herunterladen

Im Lieferumfang von IBM MQ sind keine MQ Light-Clients enthalten, aber Sie können folgende MQ Light-Clients herunterladen und installieren:

Node.js
Installieren Sie die MQ Light-API Node.js mit folgendem npm-Befehl in Ihrem Arbeitsverzeichnis: `npm install mqlight@1.0`

Java
Laden Sie das mqlight-distribution-Paket für die erforderliche Version von Maven Central herunter und extrahieren Sie den Inhalt. Sie finden die verfügbaren mqlight-distribution-Pakete unter Maven Central.

Ruby
Installieren Sie die MQ Light-Ruby-API mit folgendem gem-Befehl in Ihrem Arbeitsverzeichnis: `gem install mqlight --pre`

Python
Installieren Sie die MQ Light-Python-API mit folgendem pip-Befehl in Ihrem Arbeitsverzeichnis: `pip install mqlight --pre`

Alle Downloads des MQ Light-Clients schließen mehrere Beispielprogramme ein, die die verschiedenen Messaging-Funktionen veranschaulichen:

- Beispiel Send
- Beispiel Receive
- Beispiel UI Workout

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.

AMQP-Clients schützen

Informationen zum Sichern von MQ Light -Anwendungen finden Sie unter AMQP-Clients sichern.

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 Unterneh-

mensanwendungen austauschen. Sie können MQ Light-Anwendungen auf einem IBM MQ-Warteschlangenmanager implementieren. Weitere Informationen finden Sie unter „MQ Light-Anwendungen in einer lokalen IBM MQ-Umgebung bereitstellen” auf Seite 721 .

Nachdem Sie einen AMQP-Client implementiert haben, können Sie Nachrichten mit IBM MQ-Anwendungen austauschen. Wenn Sie beispielsweise mit dem MQ Light-Client Node.js eine JavaScript-Zeichenfolgenachricht senden, empfängt die IBM MQ-Anwendung eine MQ-Nachricht, in 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.

Sie können einen AMQP-Kanal beim Start testen, indem Sie mithilfe einer der folgenden Methoden eine Verbindung zu einer MQ Light-Anwendung herstellen:

- Verwendung des IBM MQ Light-Clients für Node.js und Java
- Verwendung des IBM MQ Light-Clients für Ruby und Python.
- Verwendung eines anderen AMQP 1.0-Clients, z. B. Apache Qpid Proton

Zugehörige Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW MQ Light und AMQP (Advanced Message Queuing Protocol)

Die IBM MQ Light -API basiert 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-Messaging-API basiert auf AMQP 1.0. Die API stellt den größten Teil der Messaging-Funktionalität bereit, die für die Mehrzahl der Publish/Subscribe- und Punkt-zu-Punkt-Messaging-Datenflüsse benötigt wird.

Die MQ Light-API verfügt über die folgenden 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 zur MQ Light-API finden Sie auf den folgenden Websites:

- Die Dokumentation zur API 'Node.js' finden Sie unter https://www.npmjs.org/package/mqlight
- Die Dokumentation zur Ruby-API finden Sie unter https://www.rubydoc.info/github/mqlight/ruby-mqlight
- Die Dokumentation zur Python-API finden Sie unter https://python-mqlight.readthedocs.org
- Die Dokumentation zur Java-API finden Sie unter https://mqlight.github.io/java-mqlight.

Zugehörige Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW 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 entweder MQ Light-Clients oder andere 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:

Linknamen

AMQP-Kanäle erwarten, dass der Name eines AMQP-Links eines der folgenden drei Formate hat:

- Einfaches Thema (für Veröffentlichung und Subskription)
  - Veröffentlichung von Nachrichten: Eine einfache Themenzeichenfolge (z. B. ein Linkname "`/sports/football`") bewirkt, dass eine Nachricht im Thema `/sports/football` veröffentlicht wird.
  - Subskription eines Themas, um Nachrichten zu empfangen: Eine einfache Themenzeichenfolge (z. B. ein Linkname "`/sports/football`") bewirkt, dass eine Subskription im 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 für einen bestimmten AMQP-Client 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`").

Weitere Informationen darüber, wie AMQP-Nachrichten IBM MQ-Nachrichten zugeordnet werden und umgekehrt, finden Sie im Abschnitt AMQP-Felder zu IBM MQ-Feldern zuordnen (eingehende Nachrichten).

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 MQ Light-Client-ID enthält. Die maximale Länge einer MQ Light-Client-ID beträgt 256 Zeichen und die ID kann alphanumerische Zeichen, Prozentzeichen (%), Schrägstrich (/), Punkt (.) und Unterstrich (_) 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'.

Zugehörige Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW V 9.0.0 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:

Kopfzeile
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

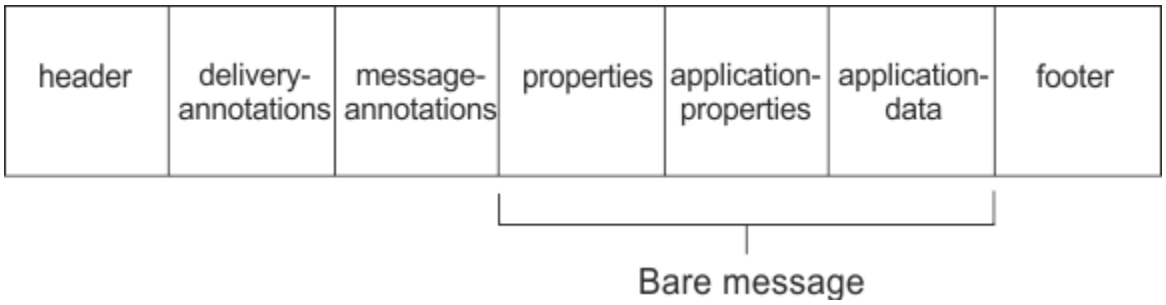
- **to** - Adresse des Knotens, an den die Nachricht gerichtet ist
- **subject** - Betreff der Nachricht
- **reply-to** - Knoten, an den die Antworten gehen
- **correlation-id** - Korrelations-ID der Anwendung
- **content-type** - MIME-Inhaltstyp
- **content-encoding** - MIME-Inhaltstyp. Wird als Modifikator 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 Informationen
AMQP-Kanäle erstellen und verwenden
AMQP-Clients schützen

ULW V 9.0.0 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 80. Headerfeldzuordnungen

Feld	Standardwert	Wert
durable	false	'true', wenn `MQMD.Persistence` auf `MQPER_PERSISTENT` gesetzt ist, andernfalls 'false'.
priority	4	Aus `mq_amqp.Hdr.Pri`, wenn festgelegt, oder andernfalls aus `MQMD.Priority`, wenn festgelegt. Ist keine der beiden festgelegt, wird 4 eingestellt.
Lebensdauer	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 81. 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.
zu	Die Warteschlange, aus der die Nachricht abgerufen wurde, bzw. die Themenzeichenfolge bei einer Veröffentlichung.
Betreff	Nicht festgelegt.
reply-to	Die `MQMD.ReplyToQ`, falls angegeben, andernfalls nicht festgelegt.

Tabelle 81. Eigenschaftsfeldzuordnungen (Forts.)	
Name	**Wert**
correlation-id	Die `MQMD.CorrelId` wird als binär festgelegt, falls angegeben; andernfalls nicht festgelegt.
Inhaltstyp	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.
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.

Fußzeile

Es ist keine Fußzeile enthalten.

Zugehörige Informationen

MQMD - Nachrichtendeskriptor

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW V 9.0.0 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 82. Nachrichtendeskriptor für AMQP-Nachricht

Feld	**Wert**
StrucId	MQMD_STRUC_ID
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.
CorrelId	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	""
ReplyToQMgr	""
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.

Tabelle 82. Nachrichtendeskriptor für AMQP-Nachricht (Forts.)	
Feld	**Wert**
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 83.			
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.

Tabelle 83. (Forts.)

Eigenschaftsname	MQRFH2-Name	Typ	Beschreibung
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.
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.

Tabelle 83. (Forts.)			
Eigenschaftsname	**MQRFH2-Name**	**Typ**	**Beschreibung**
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.
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. Beispiel:
  - Die Länge des Eigenschaftsnamens ist auf MQ_MAX_PROPERTY_NAME_LENGTH begrenzt.
  - Der Eigenschaftsname muss den Regeln entsprechen, die in der Spezifikation der Sprache Java 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.

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 Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW V 9.0.0 Zuverlässigkeit der Nachrichtenübermittlung mit AMQP

Es gibt vier Features der IBM MQ Light-API, mit denen Sie die Zuverlässigkeit der Nachrichtenübermittlung an und von MQ Light- und AMQP-Anwendungen steuern können.

Diese sind:

- „Nachrichtenservicequalität (QOS)” auf Seite 716
- „Automatische Bestätigung des Subskribenten” auf Seite 716

- „Lebensdauer einer Subskription” auf Seite 717
- „Nachrichtenpersistenz” auf Seite 717

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, legen Sie für die Client- oder Subskriptionsoption **qos** den Wert *QOS_AT_MOST_ONCE* oder *QOS_AT_LEAST_ONCE* fest.

Wenn Sie einen anderen AMQP-Client verwenden, setzen Sie das Attribut **settled** des Übertragungsrahmens (für Publisher) oder des Dispositionsrahmens (für Subskribenten) auf *true* oder *false*, je nachdem, welche Servicequalität Sie erreichen möchten.

Die Servicequalität bestimmt, wann eine Nachricht von der `sending`-Seite eines Datenaustausches verworfen wird.

Veröffentlichen

Wenn ein Publisher **QOS** *0* (höchstens einmal) auswählt, wartet der Publisher nicht auf eine Bestätigung des Warteschlangenmanagers, bevor er seine Kopie der Nachricht verwirft.

Wenn die Verbindung zum Warteschlangenmanager fehlschlägt, bevor der Sendevorgang abgeschlossen ist, wird die Nachricht möglicherweise nicht von Subskribenten empfangen.

Wenn ein Publisher **QOS** *1* auswählt (mindestens einmal), wartet der Publisher, bis 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.

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 Warteschlangenmanager seine Kopie verwirft. 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.

Mit der MQ Light-Subskriptionsoption **ttl** wird die Zeit (in Millisekunden) angegeben, die eine Subskription bestehen bleiben soll, nachdem die Anwendung die Verbindung getrennt hat. 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 **QOS** *0* (höchstens einmal) verwendet und eine nicht persistente Subskription erstellt, reiht der AMQP-Kanal unabhängig von den anderen Optionen, die im folgenden Text beschrieben werden, 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 *true* 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 Thema SYSTEM.BASE.TOPIC, für das das Attribut **DEFPSIST** auf *NO* (nicht persistent) gesetzt ist.

Achtung: Spätere Versionen des MQ Light-Clients unterstützen die Festlegung des permanenten AMQP-Headers nicht.

Zugehörige Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW Topologien für AMQP-Clients mit IBM MQ

Beispieltopologien, die Sie bei der Entwicklung Ihrer AMQP-Clients für die Arbeit mit IBM MQ unterstützen.

Zugehörige Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW AMQP-Clients, die über IBM MQ kommunizieren

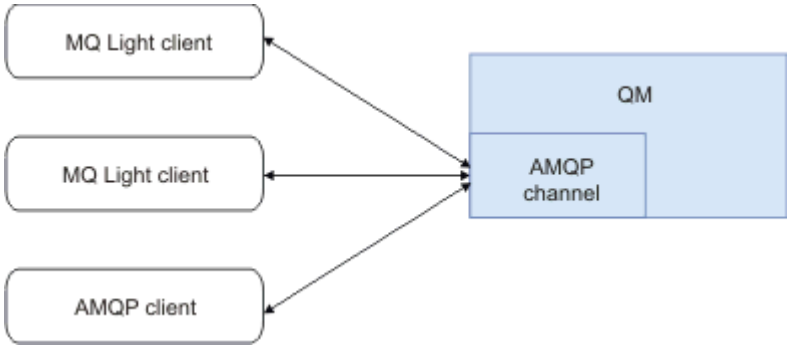
Sie können IBM MQ als Messaging-Provider für IBM MQ Light oder eine Anwendung verwenden, die mit AMQP 1.0 konform ist. Obwohl jeder AMQP 1.0-Client eine Verbindung zu einem AMQP-Kanal herstellen kann, werden einige AMQP-Funktionen nicht unterstützt, z. B. Transaktionen oder mehrere Sitzungen.

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 die einzigen Anwendungen, die Nachrichten senden und empfangen MQ Light- oder AMQP 1.0-Anwendungen.

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 Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW AMQP-Clients, die mit IBM MQ-Anwendungen Nachrichten austauschen

Durch Definition und Start eines AMQP-Kanals können MQ Light- oder 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 im Abschnitt AMQP-Felder zu IBM MQ-Feldern zuordnen (eingehende Nachrichten).

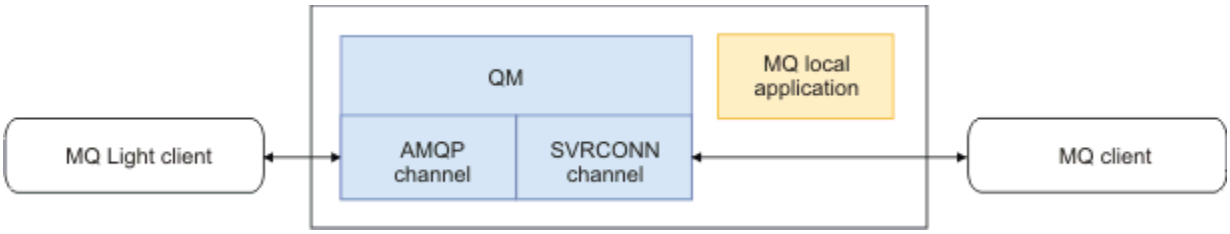
Wenn die MQ-Anwendung eine dauerhafte Subskription erstellt hat, werden Nachrichten, die von der AMQP-Anwendung veröffentlicht werden, in der Warteschlange gespeichert, die die Subskription sichert. Die Nachrichten werden dann von der MQ-Anwendung empfangen, wenn die Anwendung ihre Subskription wieder aufnimmt. Wenn die AMQP-Anwendung eine Nachrichtenlebensdauer angibt und die MQ-An-

wendung die Verbindung nicht innerhalb dieses Zeitraums wiederherstellt, wird die Nachricht aus der Warteschlange entfernt.

MQ Light- oder AMQP 1.0-Anwendungen können auch Nachrichten verarbeiten, 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 MQ-AMQP-Nachrichtenzuordnungen finden Sie im Abschnitt AMQP-Felder zu IBM MQ-Feldern zuordnen (eingehende Nachrichten).



Zugehörige Informationen

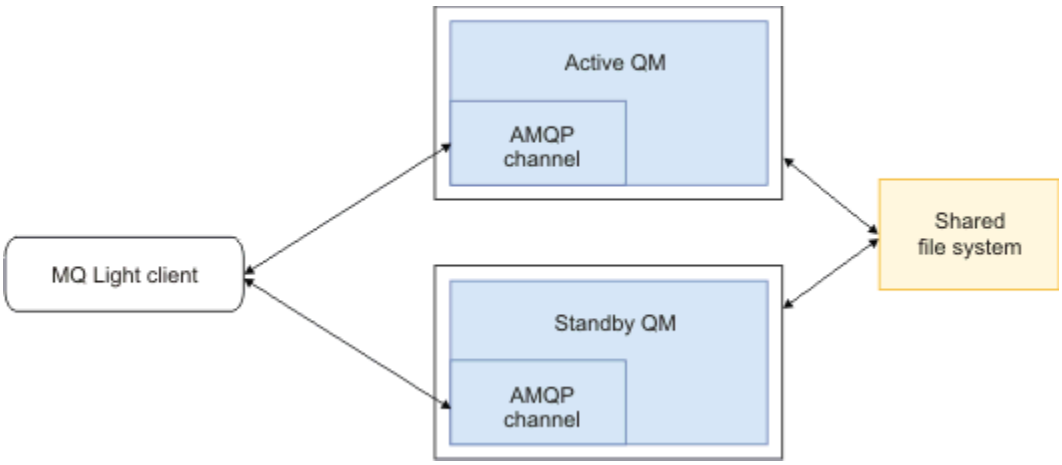
AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW AMQP-Client für hohe Verfügbarkeit konfigurieren

Sie können MQ Light- oder AMQP 1.0-Anwendungen so konfigurieren, dass sie eine Verbindung zu der aktiven Instanz eines IBM MQ-Multi-Instanz-Warteschlangenmanagers herstellen und an die Standby-Instanz des Multi-Instanz-Warteschlangenmanagers in einem Hochverfügbarkeits-(HA-)Paar übergeben. Zu diesem Zweck konfigurieren Sie die AMQP-Anwendung mit zwei IP-Adressen und Portpaaren.

Sie können die MQ Light-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-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 Informationen

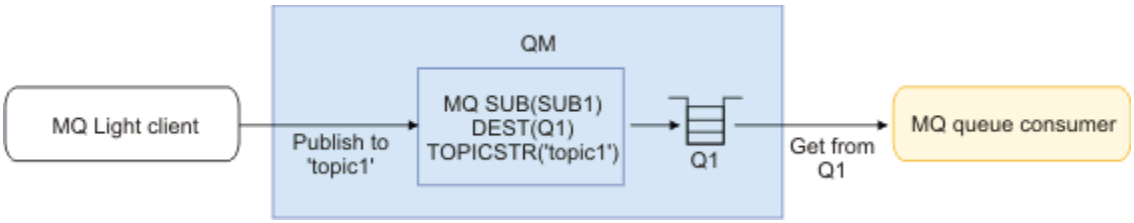
AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW 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 MQ Light- oder AMQP 1.0-Anwendung Nachrichten an eine vorhandene IBM MQ-Anwendung senden soll, die für das Lesen von Nachrichten 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 Informationen

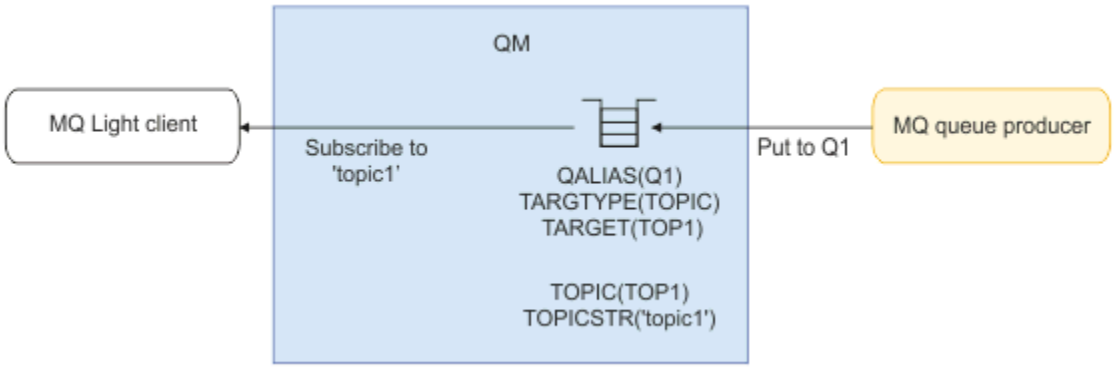
AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW 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 MQ Light- oder 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 Warteschlangenaliasnamen (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 Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW AMQP-Client, der Anforderungen an einen Anwendungsserver übergibt und Antworten von einem Anwendungsserver verarbeitet

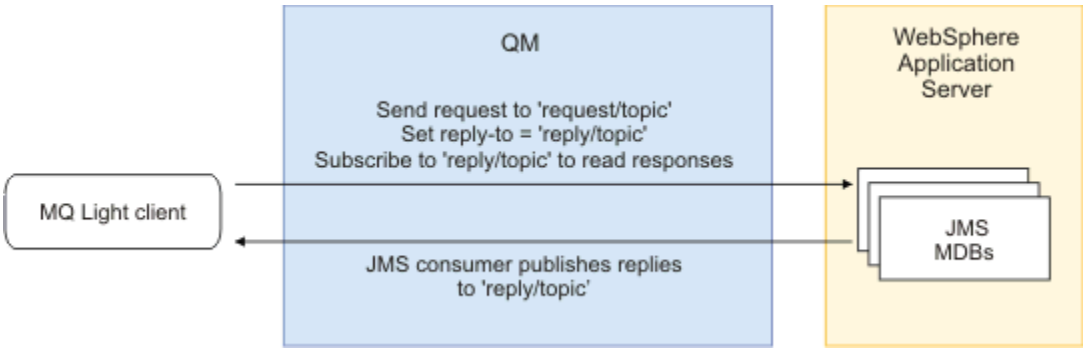
Ein MQ Light- oder ein anderer AMQP-Client kann Anforderungen an eine Message-driven Bean übergeben, die in einem Anwendungsserver ausgeführt wird, und Antworten aus einem Antwortthema verarbeiten. IBM MQ unterstützt AMQP 1.0-Anwendungen, indem in den Nachrichten, die IBM MQ veröffentlicht, ein Antwortthema 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. Zum 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 AMQP-Felder IBM MQ-Feldern zuordnen (eingehende Nachrichten)



Zugehörige Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

ULW MQ Light-Anwendungen in einer lokalen IBM MQ-Umgebung bereitstellen

IBM MQ unterstützt die Messaging-API von IBM MQ Light, d. h., Sie können IBM MQ verwenden, um Ihre MQ Light-Anwendung in einer lokalen IBM MQ-Umgebung bereitzustellen.

Sie können MQ Light-Anwendungen für einen IBM MQ-Warteschlangenmanager bereitstellen, um Ihren MQ Light-Anwendungen die Kommunikation mit vorhandenen Unternehmensanwendungen, die bereits mit IBM MQ verbunden sind, zu ermöglichen, so wie im folgenden Diagramm dargestellt:

MQ Light-Anwendungen können einen Themenbereich mit vorhandenen IBM MQ-Anwendungen gemeinsam nutzen, was sie in die Lage versetzt, mit vorhandenen Unternehmenssystemen zu interagieren.

Zugehörige Informationen

AMQP-Kanäle erstellen und verwenden

AMQP-Clients schützen

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 mithilfe von REST unterstützt IBM MQ verschiedene Optionen. Sie können zwischen folgenden Optionen wählen:

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

V 9.0.4 IBM MQ messaging REST API

messaging REST API wird standardmäßig mit IBM MQ aus IBM MQ 9.0.4 bereitgestellt und ist standardmäßig aktiviert. Sie können mithilfe der messaging REST API IBM MQ-Nachrichten im einfachen Textformat senden und empfangen.

Anwendungen können mit HTTP POST eine Nachricht an IBM MQ senden oder mit HTTP DELETE eine Nachricht von IBM MQ abrufen, wobei es sich um einen Abruf mit Löschen handelt. 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. Benutzer müssen beim MQWeb-Server authentifiziert und Mitglied der Rolle MQWebUser sein.

Weitere Informationen finden Sie im Abschnitt Messaging mit der REST API.

IBM MQ-Brücke für HTTP

Die IBM MQ-Brücke für HTTP ist eine JEE-Anwendung, die in einer geeigneten Laufzeitumgebung installiert werden kann und eine grundlegende REST API sowohl für die Warteschlangen als auch die Themen bereitstellt, die von einem einzelnen MQ-Warteschlangenmanager gehostet werden.

Anwendungen können einen HTTP POST an die Brücke ausgeben, um eine Nachricht an IBM MQ zu senden, einen HTTP GET, um eine Nachricht von IBM MQ anzuzeigen, oder einen HTTP DELETE, um eine Nachricht von IBM MQ abzurufen (Abruf mit Löschen). Es werden verschiedene HTTP-Header unterstützt, mit deren Hilfe Nachrichteneigenschaften festgelegt werden können.

Anmerkung: Ab IBM MQ 8.0 wird IBM MQ bridge for HTTP nicht weiter unterstützt. Als Alternative sollte die IBM messaging REST API, die von IBM MQ 9.0.4 bereitgestellt wird, verwendet werden.

Weitere Informationen finden Sie im Abschnitt Web-Services mit IBM MQ-Brücke für HTTP entwickeln.

IBM z/OS Connect EE

IBM z/OS Connect EE (zCEE) 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 wird vor 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.

Mit zCEE können Sie ohne großen Aufwand eine REST API erstellen, die JSON-Daten sendet und empfängt und diese Daten optional in die traditionellen Sprachstrukturen umwandeln kann, die von vielen Mainframe-Anwendungen erwartet werden. Dies gilt beispielsweise für COBOL-Copybooks.

Mit einem Eclipse-basierten API-Editor können Sie unter Verwendung von Abfrageparametern und URL-Pfadsegmenten eine komplexe REST-konforme API erstellen, um das JSON-Format zu bearbeiten, während es die zCEE-Runtime durchläuft.

Mithilfe von zCEE können IBM MQ-Warteschlangen und -Themen als Services zugänglich gemacht werden. Es werden zwei verschiedene Servicetypen unterstützt:

- Unidirektionale Services: Die bereitgestellte Funktion ähnelt der, die mit der IBM MQ-Brücke für HTTP bereitgestellt wird, bei der ein HTTP POST eine Nachricht sendet oder ein HTTP DELETE eine Nachricht abruft (Abruf mit Löschen). Die Hauptvorteile gegenüber der Brücke sind die integrierte Datenkonvertierungsunterstützung und die Möglichkeit, mit dem API-Editor eine komplexere REST-konforme API erstellen zu können.
- Bidirektionale Services: Es wird eine REST API für ein Warteschlangenpaar bereitgestellt, die von einer Back-End-Anwendung mit Anforderung/Antwort-Funktion verwendet wird. Aufrufende geben einen HTTP POST an den bidirektionalen Service aus und senden JSON-Daten. Diese Daten werden in eine Anforderungswarteschlange eingereiht, wo sie von der Back-End-Anwendung verarbeitet werden, die anschließend eine Antwort in die Antwortwarteschlange stellt. Diese Antwort wird abgerufen und als POST-Antwortteil an den Aufrufenden zurückgesendet.

zCEE wird unter IBM MQ 8 und höher unterstützt. Weitere Informationen finden Sie im Abschnitt IBM MQ for z/OS-Service-Provider für z/OS Connect.

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 service.

V 9.0.4 Messaging mit der REST API

Sie können die messaging REST API zum Senden und Empfangen von IBM MQ-Nachrichten verwenden. Aktuell werden Informationen 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 manuell inaktivieren, um jegliche Nachrichtenübermittlung zu verhindern. 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. Der Aufrufende muss beim MQWeb-Server authentifiziert und ein Mitglied der Rolle MQWebUser sein. Er muss auch eine Zugriffsberechtigung für die angegebene Warteschlange besitzen. Weitere Informationen zur Sicherheit für die REST API finden Sie im Abschnitt Sicherheit der IBM MQ Console und REST API.

Prozedur

- „Erste Schritte mit der messaging REST API” auf Seite 724
- „messaging REST API verwenden” auf Seite 726
- „Einschränkungen für Messaging-REST API” auf Seite 728
- REST API Fehlerbehandlung
- REST API Erkennung
- REST API Unterstützung in der Landessprache

Zugehörige Informationen

Referenz zur Messaging-REST API

V 9.0.4 Erste Schritte mit der messaging REST API

Bevor Sie die messaging REST API starten können, müssen Sie die richtigen Komponenten installieren, die REST API aktivieren, die Sicherheit konfigurieren und den MQWeb-Server starten.

Vorbereitende Schritte

IBM i Unter IBM isollten die Befehle in QSHELL ausgeführt werden.

Informationen zu diesem Vorgang

Die Prozedur für diese Task konzentriert sich auf den schnellen Einstieg in die messaging REST API. Die Schritte zum Konfigurieren der Sicherheit beschreiben, wie eine Basisbenutzerregistry konfiguriert wird, aber andere Optionen für die Konfiguration von Benutzern und Rollen vorhanden sind. Weitere Informationen zur Konfiguration der Sicherheit für die messaging REST API finden Sie im Abschnitt IBM MQ Console und REST API-Sicherheit.

Anmerkung: Sie müssen ein privilegierter Benutzer sein, um auf die Datei `mqwebuser.xml` zugreifen zu können.

Vorgehensweise

1. Installieren Sie die Komponente IBM MQ Console und REST API:
   - AIX Installieren Sie unter AIX die Dateigruppe `mqm.web.rte`.
   - Linux Installieren Sie unter Linux die Komponente `MQSeriesWeb`. Weitere Informationen zur Installation von Komponenten und Features unter Linux finden Sie im Abschnitt Linux-Installationsaufgaben.
   - Windows Installieren Sie unter Windows das Feature `Web Administration`. Weitere Informationen zur Installation von Komponenten und Features unter Windows finden Sie im Abschnitt Windows-Installationsaufgaben.
   - z/OS Installieren Sie unter z/OS das Feature `IBM MQ for z/OS Unix System Services Components`. Weitere Informationen zur Installation von Komponenten und Features unter z/OS finden Sie im Abschnitt z/OS-Installationsaufgaben.

2. Konfigurieren Sie Benutzer und Rollen, damit sie die messaging REST API verwenden können:
   a) Kopieren Sie die Datei `basic_registry.xml` aus dem Verzeichnis *MQ_INSTALLATION_PATH* `/web/mq/samp/configuration`.
   b) Stellen Sie die XML-Beispieldatei in das zutreffende Verzeichnis:
      - ULW Unter UNIX, Linux, and Windows: *MQ_DATA_DIRECTORY*`/web/installations/`*installationName*`/servers/mqweb`
      - z/OS Unter z/OS: *WLP_user_directory*`/servers/mqweb`

      Hierbei steht *WLP-Benutzerverzeichnis* für das Verzeichnis, das angegeben wurde, als das Script **crtmqweb.sh** ausgeführt wurde, um die mqweb-Serverdefinition zu erstellen.
   c) Benennen Sie die XML-Beispieldatei in `mqwebuser.xml` um.

      Anmerkung: Diese umbenannte Datei ersetzt eine vorhandene Datei, die auch für die IBM MQ Console verwendet wird. Kopieren Sie deshalb, falls Sie die Datei `mqwebuser.xml` für die IBM MQ Console geändert haben, Ihre Änderungen in die neue XML-Datei, bevor Sie sie umbenennen.
   d) Optional: Bearbeiten Sie die Datei `mqwebuser.xml`, um Benutzer und Gruppen hinzuzufügen. Ordnen Sie diese Benutzer und Gruppen der Rolle `MQWebUser` zu, damit sie zur Verwendung der messaging REST API berechtigt sind. Die Rollen `MQWebAdmin` und `MQWebAdminRO` gelten nicht für messaging REST API. Sie können auch die Kennwörter für die Benutzer ändern, die standardmäßig definiert sind, und die neuen Kennwörter codieren. Stellen Sie sicher, dass die Benutzer berechtigt sind, auf die angegebene Warteschlange zuzugreifen. Weitere Informationen finden Sie im Abschnitt Benutzer und Rollen konfigurieren.
3. Aktivieren Sie mit dem Befehl **setmqweb** Fernverbindungen zum MQWeb-Server:

   ```
   setmqweb properties -k httpHost -v hostname
   ```

   Dabei gibt *hostname* die IP-Adresse, den Hostnamen des Domänennamensservers (DNS) mit dem Domänennamenssuffix oder den DNS-Hostnamen des Servers an, auf dem IBM MQ installiert ist. Verwenden Sie einen Stern (*), um alle verfügbaren Netzschnittstellen anzugeben.

   Achtung: V 9.0.4 z/OS

   Bevor Sie den Befehl **setmqweb** oder **dspmqweb** unter ausgeben z/OS, müssen Sie die Umgebungsvariable WLP_USER_DIR so setzen, dass sie auf Ihre mqweb-Serverkonfiguration verweist.

   Geben Sie dazu folgenden Befehl aus:

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   Dabei ist *WLP_user_directory* der Name des Verzeichnisses, das an `crtmqweb.sh` übergeben wird. Beispiel:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   Weitere Informationen finden Sie unter Liberty-Serverdefinition erstellen.
4. Starten Sie den MQWeb-Server, der die REST API unterstützt:
   - ULW Geben Sie als privilegierter Benutzer folgenden Befehl in der Befehlszeile ein:

     ```
     strmqweb
     ```
   - z/OS Starten Sie unter z/OS die Prozedur, die Sie in Task 29: Create a procedure for the IBM WLP server erstellt haben.

Nächste Schritte

1. Wählen Sie aus, wie Benutzer der messaging REST API auf dem mqweb-Server authentifiziert werden sollen. Sie müssen nicht für alle Benutzer die gleiche Methode verwenden. Die folgenden Optionen sind verfügbar:
   - Lassen Sie sich Benutzer authentifizieren, indem Sie die HTTP-Basisauthentifizierung verwenden In diesem Fall werden ein Benutzername und ein Kennwort codiert, jedoch nicht verschlüsselt, und mit jeder REST API-Anforderung gesendet, um den Benutzer für diese Anforderung zu authentifizieren und zu autorisieren. Damit diese Authentifizierung sicher ist, müssen Sie eine sichere Verbindung verwenden. Das heißt, Sie müssen HTTPS verwenden. Weitere Informationen finden Sie im Abschnitt HTTP-Basisauthentifizierung mit der REST API verwenden.
   - Lassen Sie sich Benutzer authentifizieren, indem Sie die Tokenauthentifizierung In diesem Fall stellt ein Benutzer der REST API `login`-Ressource mit der HTTP-POST-Methode eine Benutzer-ID und ein Kennwort bereit. Es wird ein LTPA-Token generiert, das es dem Benutzer ermöglicht, angemeldet zu bleiben und für einen festgelegten Zeitraum autorisiert zu sein. Damit diese Authentifizierung sicher ist, müssen Sie eine sichere Verbindung verwenden. Das heißt, Sie müssen HTTPS verwenden. Weitere Informationen finden Sie im Abschnitt Tokenbasierte Authentifizierung mit der REST API verwenden.
   - Lassen Sie sich Benutzer authentifizieren, indem Sie Clientzertifikate verwenden In diesem Fall verwendet der Benutzer keine Benutzer-ID und kein Kennwort für die Anmeldung an der messaging REST API, sondern er verwendet stattdessen das Clientzertifikat. Weitere Informationen finden Sie im Abschnitt Clientzertifikatsauthentifizierung mit der REST API verwenden.
2. Konfigurieren Sie REST API-Einstellungen, einschließlich Aktivierung von HTTP-Verbindungen und Änderung der Portnummer. Weitere Informationen finden Sie im Abschnitt IBM MQ Console und REST API konfigurieren.
3. Konfigurieren Sie optional Cross Origin Resource Sharing für die REST API. Standardmäßig können Sie auf die REST API nicht über Webressourcen zugreifen, die nicht in derselben Domäne gehostet werden wie die REST API. Dies bedeutet, dass Kreuzursprungsanforderungen nicht aktiviert sind. Sie können Cross Origin Resource Sharing (CORS) konfigurieren, um Cross-Origin-Anforderungen von angegebenen URLs zu ermöglichen. Weitere Informationen finden Sie im Abschnitt CORS für den REST API konfigurieren.
4. Verwenden Sie den REST API. Weitere Informationen finden Sie im Abschnitt „messaging REST API verwenden” auf Seite 726 und in der Referenz zur Messaging-REST API.

Anmerkung: Sie können den MQWeb-Server jederzeit mit dem Befehl **endmqweb** stoppen. Wenn der mqweb-Server nicht aktiv ist, können Sie die REST API oder IBM MQ Console allerdings nicht verwenden.

V 9.0.4 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 im Abschnitt IBM MQ Console und REST API-Sicherheit.

- Bei der REST API muss die Groß-/Kleinschreibung beachtet werden. Beispielsweise führt ein HTTP-POST für die folgende URL zu einem Fehler, wenn `qmgr1` als Name des Warteschlangenmanagers angegeben wird.

  ```
  /ibmmq/rest/v1/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- 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 (/) wird durch %2F codiert.
  - Ein Prozentzeichen (%) wird durch %25 codiert.

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 727 und „1.b" auf Seite 727ausführen.

Vorgehensweise

1. Konstrukt die URL:
   a) Beginnen Sie mit der folgenden Präfix-URL:

   ```
   https://host:port/ibmmq/rest/v1/messaging
   ```

   host
   Gibt den Hostnamen oder die IP-Adresse an, unter der die messaging REST API verfügbar ist.

   Der Standardwert ist `localhost`.

   port
   Gibt die HTTPS-Portnummer an, die von der messaging REST API verwendet wird.

   Der Standardwert ist 9443.

   Wenn Sie HTTP-Verbindungen aktivieren, können Sie HTTP anstelle von HTTPS verwenden. Weitere Informationen zur Aktivierung von HTTP finden Sie im Abschnitt HTTP- und HTTPS-Ports konfigurieren.

   Weitere Informationen zum Bestimmen der Präfix-URL finden Sie im Abschnitt URL der REST API bestimmen.

   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 Variablensegmente in der URL durch geschweifte Klammern *{ }*angegeben werden. Weitere Informationen finden Sie unter /messaging/qmgr/{qmgrName}/queue/{queueName}/message.

Fügen Sie beispielsweise für eine Interaktion mit der Warteschlange *Q1*, die dem Warteschlangenmanager *QM1* zugeordnet ist, `/qmgr` und `/queue` zur Präfix-URL hinzu, um folgende URL zu erstellen:

```
https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message
```

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/v1/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. Beispiel:
   - Verwenden Sie die HTTP/REST-Implementierung der von Ihnen gewählten Programmiersprache.
   - Verwenden Sie ein Tool wie beispielsweise ein REST-Client-Browser-Add-on oder cURL.

V 9.0.4 Einschränkungen für Messaging-REST API

Beachten Sie folgende Einschränkungen, bevor Sie die messaging REST API verwenden:

- Die API unterstützt aktuell kein Publish/Subscribe-Messaging. Die messaging REST API unterstützt nur synchrones Punkt-zu-Punkt-Messaging.
- Die API unterstützt aktuell kein Browsing aus Warteschlangen. Nachrichten werden mit der HTTP-Methode DELETE durch einen Abruf mit Löschen aus der IBM MQ-Warteschlange empfangen. Weitere Informationen finden Sie im Abschnitt DELETE.
- Beim Senden einer Nachricht können nur textbasierte UTF-8-Inhalte verwendet werden. Nachrichten werden im IBM MQ-Format MQSTR eingereiht. Weitere Informationen finden Sie im Abschnitt POST.

  Beim Empfang von Nachrichten werden nur Nachrichten im IBM MQ-Format MQSTR unterstützt. Anschließend werden alle Nachrichten unter dem Synchronisationspunkt empfangen und alle unbearbeiteten Nachrichten verbleiben 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 im Abschnitt Falsch formatierte Nachrichten in IBM MQ-Klassen für JMS behandeln.
- Wenn Sie Advanced Message Security (AMS) mit der messaging REST API verwenden, müssen Sie beachten, dass alle Nachrichten mit dem Kontext des Mqweb-Servers verschlüsselt werden und nicht mit dem Kontext des Benutzers, der die Nachricht veröffentlicht.
- Zeilenumbrüche in eingehenden Zeichenfolgen werden bei der HTTP-POST-Operation entfernt. REST-Anwendungen

  Verwenden Sie keine Zeilenumbrüche in Nachrichten, die über die REST-API gesendet oder veröffentlicht werden, da sie verloren gehen.

Web-Services mit IBM MQ bridge for HTTP entwickeln

Bei IBM MQ bridge for HTTP können Clientanwendungen Nachrichten mit IBM MQ austauschen, ohne dass ein IBM MQ MQI client installiert werden muss. Sie können IBM MQ von jeder Plattform oder Sprache mit HTTP-Funktionalität aufrufen.

Informationen zu diesem Vorgang

Anmerkung: Ab IBM MQ 8.0 wird IBM MQ bridge for HTTP nicht weiter unterstützt. Als Alternative sollte die IBM messaging REST API, die von IBM MQ 9.0.4 bereitgestellt wird, verwendet werden.

Einführung in IBM MQ bridge for HTTP

Die IBM MQ bridge for HTTP ist eine Java Enterprise Environment (JEE)-Webanwendung. HTTP-Clients können dieser Anwendung **POST**-, **GET**- und **DELETE**-Anforderungen senden, um Nachrichten in IBM MQ-Warteschlangen einzureihen, anzuzeigen und zu löschen. Die IBM MQ bridge for HTTP eignet sich nicht, wenn eine gesicherte Nachrichtenzustellung erforderlich ist.

Leistungen

Über die IBM MQ bridge for HTTP ist der Austausch von IBM MQ-Nachrichten über HTTP mit den verschiedensten Umgebungen möglich:

- Umgebungen, die HTTP unterstützen, aber nicht IBM MQ.
- Umgebungen mit zu wenig Speicherplatz für die Installation eines IBM MQ MQI clients.
- Umgebungen, die zu groß sind, um den IBM MQ MQI client auf jedem System zu installieren, das Zugriff auf IBM MQ benötigt.
- Webbasierte Anwendungen, mit denen Sie Nachrichten austauschen möchten, ohne eine eigene Bridge für IBM MQ erstellen zu müssen.
- Webbasierte Anwendungen, die Sie mit asynchronen Verfahren wie AJAX erweitern möchten. Die IBM MQ bridge for HTTP stellt IBM MQ-Warteschlangen und -Themen via Representation State Transfer (REST) over HTTP zur Verfügung.

HTTP-Support kann sowohl bei Punkt-zu-Punkt- als auch bei Publish/Subscribe-Messaging-Topologien genutzt werden.

Wie funktioniert HTTP-Support?

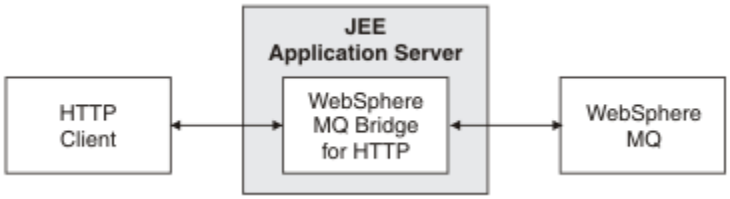


Abbildung 59. IBM MQ bridge for HTTP

Die Webanwendung der IBM MQ bridge for HTTP erhält von einem oder mehreren Clients eine HTTP-Anforderung. Sie interagiert in deren Auftrag mit IBM MQ und gibt die HTTP-Antworten an sie zurück.

Die IBM MQ bridge for HTTP ist ein JEE-Servlet, das über einen Ressourcenadapter mit IBM MQ verbunden ist. Das HTTP-Servlet verarbeitet die HTTP-Anforderungstypen **POST**, **GET** und **DELETE**.

Tabelle 84. IBM MQ bridge for HTTP-Verben

HTTP-Anforderung	**Ergebnis**
VERÖFFENTLICHEN	Reiht eine Nachricht in eine Warteschlange oder in ein Thema ein.
GET	Zeigt die erste Nachricht einer Warteschlange an. Im Rahmen des HTTP-Protokolls wird mit **GET** die Nachricht in der Warteschlange nicht gelöscht. Verwenden Sie **GET** nicht für das Publish/Subscribe-Messaging.

Tabelle 84. IBM MQ bridge for HTTP-Verben (Forts.)	
HTTP-Anforderung	**Ergebnis**
DELETE	Ruft eine Nachricht aus einer Warteschlange oder aus einem Thema ab und löscht sie.

Beispiel für HTTP POST

Mit HTTP **POST** wird eine Nachricht in eine Warteschlange eingereiht oder eine Veröffentlichung zu einem Thema durchgeführt. Das Java-Beispiel **HTTPPOST** ist ein Beispiel für eine HTTP-**POST**-Anforderung einer Nachricht an eine Warteschlange. Anstatt Java zu verwenden, können Sie die HTTP-Anforderung **POST** in einem Browserformular oder mithilfe eines AJAX-Toolkits erstellen.

Die folgende Abbildung zeigt eine HTTP-Anforderung zum Einreihen einer Nachricht in eine Warteschlange namens `myQueue`. Diese Anforderung enthält den HTTP-Header `x-msg-correlId`, mit dem die Korrelations-ID der IBM MQ-Nachricht festgelegt wird.

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

*Abbildung 60. Beispiel für eine HTTP-Anforderung vom Typ **POST** an eine Warteschlange*

Die folgende Abbildung zeigt die an den Client zurückgegebene Antwort. Es ist kein Antwortinhalt vorhanden.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

Abbildung 61. Beispiel für eine HTTP-Antwort vom Typ POST

Beispiel für HTTP DELETE

Mit HTTP **DELETE** wird eine Nachricht aus einer Warteschlange oder eine Veröffentlichung abgerufen und gelöscht. Das **HTTPDELETE** Java-Beispiel ist ein Beispiel für eine HTTP **DELETE** -Anforderung, die eine Nachricht aus einer Warteschlange liest. Anstatt Java zu verwenden, können Sie die HTTP-Anforderung **DELETE** in einem Browserformular oder mithilfe eines AJAX-Toolkits erstellen.

Die folgende Abbildung zeigt eine HTTP-Anforderung zum Löschen der nächsten Nachricht in einer Warteschlange namens `myQueue`. Als Antwort wird der Nachrichtentext an den Client zurückgegeben. In IBM MQ ist HTTP-**DELETE** ein Abruf mit Löschen.

Die Anforderung enthält den HTTP-Anforderungsheader `x-msg-wait`, der IBM MQ Bridge for HTTP darüber informiert, wie lange auf die Ankunft einer Nachricht in der Warteschlange gewartet werden soll. Die Anforderung enthält zudem den Anforderungsheader `x-msg-require-headers`, der angibt, dass der Client in der Antwort die Korrelations-ID der Nachricht erhalten soll.

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Abbildung 62. Beispiel für eine HTTP-Anforderung vom Typ **DELETE***

Die folgende Abbildung zeigt die an den Client zurückgegebene Antwort. Die Korrelations-ID wird an den Client zurückgegeben, wie in `x-msg-require-headers` der Anforderung angefordert.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

*Abbildung 63. Beispiel für eine HTTP-Antwort vom Typ **DELETE***

Beispiel für HTTP GET

Mit der HTTP-Anforderung vom Typ **GET** wird eine Nachricht aus einer Warteschlange abgerufen. Die Nachricht bleibt in der Warteschlange. Bei IBM MQ ist die HTTP-Anforderung vom Typ **GET** eine Suchanforderung. Sie können eine HTTP-Anforderung vom Typ **GET** mithilfe eines Java-Clients, eines Browserformulars oder eines AJAX-Toolkits erstellen.

Die folgende Abbildung zeigt eine HTTP-Anforderung zum Durchsuchen der nächsten Nachricht in einer Warteschlange namens `myQueue`.

Die Anforderung enthält den HTTP-Anforderungsheader `x-msg-wait`, der IBM MQ bridge for HTTP darüber informiert, wie lange auf die Ankunft einer Nachricht in der Warteschlange gewartet werden soll. Die Anforderung enthält zudem den Anforderungsheader `x-msg-require-headers`, der angibt, dass der Client in der Antwort die Korrelations-ID der Nachricht erhalten soll.

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Abbildung 64. Beispiel für eine HTTP-Anforderung vom Typ **GET***

Die folgende Abbildung zeigt die an den Client zurückgegebene Antwort. Die Korrelations-ID wird an den Client zurückgegeben, wie in `x-msg-require-headers` der Anforderung angefordert.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

*Abbildung 65. Beispiel für eine HTTP-Antwort vom Typ **GET***

IBM MQ bridge for HTTP installieren, konfigurieren und prüfen

Die IBM MQ bridge for HTTP erhalten Sie, wenn Sie Java Messaging and Web Services von den Installationsmedien des IBM MQ MQI clients oder -Servers installieren. Implementieren Sie die IBM MQ bridge for HTTP auf einem geeigneten Anwendungsserver.

Vorbereitende Schritte

Überprüfen Sie die vorausgesetzten Produkte im Abschnitt Systemvoraussetzungen für IBM MQ. Bei der Installation werden Vorhandensein und Verfügbarkeit der Softwarevoraussetzungen für die Ausführung von IBM MQ bridge for HTTP nicht überprüft. Sie müssen überprüfen, ob die Voraussetzungen installiert sind.

IBM MQ bridge for HTTP ist eine Java EE 4-Anwendung. Informationen zu unterstützten Anwendungsservern finden Sie im Abschnitt Systemvoraussetzungen für IBM MQ.

Informationen zu diesem Vorgang

IBM MQ bridge for HTTP wird in der `.war`-Datei `WMQHTTP.war` bereitgestellt.

- Unter UNIX und Linux:
  - `WMQHTTP.war` ist Teil der Installationsoption von Java Messaging and Web Services. Diese Option ist im Installationsprogramm des Clients und des Servers enthalten.
  - `WMQHTTP.war` wird in *mqmtop*`/java/http/WMQHTTP.war`installiert. *mqmtop* ist das Verzeichnis, in dem IBM MQ installiert ist.
  - `WMQHTTP.samples` wird in *mqmtop*`/java/http/samples`installiert. *mqmtop* ist das Verzeichnis, in dem IBM MQ installiert ist.
- z/OS Unter z/OS:
  - `WMQHTTP.war` wird mit der Funktion IBM MQ z/OS UNIX System Services Components bereitgestellt.
  - `WMQHTTP.war` wird in *PathPrefix*`/usr/lpp/mqm/V7R0M0/HTTPBridge/`installiert, wobei *PathPrefix* ein optionales vom Kunden definiertes Präfix ist.

Führen Sie zur Installation, Implementierung und Konfiguration der IBM MQ bridge for HTTP sowie zur Prüfung deren Konfiguration die im folgenden Abschnitt beschriebenen Schritte aus. Die einzelnen Schritte der Konfiguration sind je nach Anwendungsserver unterschiedlich. Verwenden Sie „IBM MQ bridge for HTTP unter WebSphere Application Server 6.1.0.9 implementieren und überprüfen” auf Seite 733 als Vorlage für die auf Ihrem Anwendungsserver durchzuführenden Schritte.

Vorgehensweise

1. Rufen Sie die Datei `WMQHTTP.war` ab, indem Sie den IBM MQ MQI client oder -Server installieren.
2. Kopieren Sie die Datei `WMQHTTP.war` auf einen Server, über den Sie sie auf einem Anwendungsserver implementieren können.
3. Implementieren Sie die Datei `WMQHTTP.war` auf einem Anwendungsserver.

4. Installieren Sie IBM MQ bei Bedarf als Ressourcenadapter auf Ihrem Anwendungsserver.

   Überprüfen Sie, ob IBM MQ bereits als Messaging-Provider auf Ihrem Anwendungsserver konfiguriert ist. Suchen Sie mit dem von Ihrem Anwendungsserver bereitgestellten Verwaltungstool nach IBM MQ. IBM MQ befindet sich möglicherweise im folgenden Pfad: **Ressourcen > JMS > Messaging-Provider**.
5. Konfigurieren Sie eine Verbindungsfactory im Anwendungsserver, um eine Verbindung zu einem Warteschlangenmanager herzustellen, der den IBM MQ MQI client -Transport verwendet.[7].
6. Konfigurieren Sie die Webanwendung `WMQHTTP.war` auf dem Anwendungsserver für die Verwendung der Verbindungsfactory.
7. Prüfen Sie die Konfiguration.
   a) Richten Sie den in der Verbindungsfactory genannten Warteschlangenmanager und eine lokale Warteschlange ein.
   b) Stellen Sie eine Nachricht in die lokale Warteschlange.
   c) Erstellen Sie einen in der Verbindungsfactory genannten Serververbindungskanal mit der Berechtigung, in der lokalen Warteschlange zu lesen und zu schreiben.
   d) Starten Sie den Warteschlangenmanager und den Listener.
   e) Starten Sie den Anwendungsserver und die Webanwendung `WMQHTTP.war`, sofern sie nicht bereits ausgeführt werden.
   f) Öffnen Sie einen Browser und geben Sie `http://`*`hostname`*`:` *`web port`*`/`*`Context root`*`/msg/queue/`*`local queue`* ein.

Ergebnisse

Im Browserfenster wird die Nachricht angezeigt, die Sie in die lokale Warteschlange gestellt haben.

Nächste Schritte

1. Probieren Sie das folgende Beispiel aus: „IBM MQ bridge for HTTP unter WebSphere Application Server 6.1.0.9 implementieren und überprüfen" auf Seite 733.
2. Führen Sie die HTTP Java-Beispielanwendungen aus.

IBM MQ bridge for HTTP unter WebSphere Application Server 6.1.0.9 implementieren und überprüfen

Gehen Sie zur Vorbereitung einer IBM MQ bridge for HTTP-Bereitstellung für die Ausführung der HTTP Java-Beispielprogramme nach folgendem Beispiel vor. Die Implementierung erfolgt unter WebSphere Application Server 6.1.0.9.

Vorbereitende Schritte

1. Folgen Sie den Anweisungen im Abschnitt „IBM MQ bridge for HTTP installieren, konfigurieren und prüfen" auf Seite 732, um die Datei `WMQHTTP.war` auf einen Server zu kopieren, der für Ihre Installation von WebSphere Application Server zugänglich ist.
2. Konfigurieren Sie einen Warteschlangenmanager und eine Warteschlange zum Testen der Konfiguration:
   - Im Beispiel wird der Warteschlangenmanager mit den Werten aus Tabelle 85 auf Seite 733 konfiguriert:

Tabelle 85. WS-Manager-Konfiguration

Objekt	**Wert**
Hostname	`itso-01`

[7] Konfigurieren Sie zunächst mindestens den Clienttransport. Einige Anwendungsserver können jedoch auch über direkte Verbindungen oder über den Bindungsmodus Verbindungen mit IBM MQ herstellen.

Tabelle 85. WS-Manager-Konfiguration (Forts.)	
Objekt	**Wert**
Warteschlangenmanager	QM1
Lokale Warteschlange	HTTPTESTQ
Serververbindungskanal	In: MYSVRCON. Konfigurieren Sie eine MCA-Benutzer-ID mit ausreichender Berechtigung zum Lesen und Schreiben in HTTPTESTQ.
Port des Empfangsprogramms	1414

3. Starten Sie den Warteschlangenmanager und das Empfangsprogramm.
4. Stellen Sie eine Testnachricht in die Warteschlange HTTPTESTQ. Beispiel:
   a. Starten Sie IBM MQ Explorer.
   b. Klicken Sie in der Liste der lokalen Warteschlangen für QM1mit der rechten Maustaste auf **HTTPTESTQ > Put test message > type `First Message` > Put message > Close** .
5. Starten Sie den Anwendungsserver und melden Sie sich bei der Integrated Solutions Console an.

Informationen zu diesem Vorgang

Das Beispiel zeigt die Schritte, die auszuführen sind, wenn Sie WebSphere Application Server 6.1.0.9 als Anwendungsserver verwenden. Wenn Sie eine andere WebSphere Application Server-Version oder einen anderen Anwendungsserver verwenden, weicht das Verfahren hiervon ab. WebSphere Application Server 6.1.0.9 ist mit der Installation von IBM MQ als Nachrichtenprovider unter Verwendung der IBM MQ MQI client -Bibliotheken vorkonfiguriert. Falls IBM MQ nicht als Messaging-Provider vorkonfiguriert ist oder wenn Sie IBM MQ-Serverbindungen verwenden möchten, müssen Sie auf Ihrem Anwendungsserver den IBM MQ-Ressourcenadapter für JEE installieren.

Befolgen Sie die Anweisungen zum Implementieren von IBM MQ bridge for HTTP in WebSphere Application Server 6.1.0.9und überprüfen Sie die Implementierung mit einem Browser:

Vorgehensweise

1. Klicken Sie im Navigationsfenster auf **Ressourcen > JMS -Provider > IBM MQ Messaging-Provider**.

   Je nach WebSphere Application Server-Bereitstellung kann die Konfiguration auf Knoten-, Zellen- oder Serverebene stattfinden. Im Beispiel wird die Bereitstellung auf Serverebene verwendet.
2. Klicken Sie unter **Additional properties** (Weitere Eigenschaften) auf **Connection factories > New** (Verbindungsfactorys > Neu).
3. Geben Sie im Formular zu den JMS-Providern die Informationen aus Tabelle 86 auf Seite 734 oder Alternativen Ihrer Wahl ein und klicken Sie auf **Apply > Save** (Anwenden > Speichern).

Tabelle 86. Legen Sie die Werte der folgenden Felder fest	
Feld	**Wert**
Name	WMQHTTPBridge
JNDI Name (JNDI-Name)	jms/WMQHTTPJCAConnectionFactory
Warteschlangenmanager	QM1
Host	itso-01
Port	1414
Kanal	MYSVRCON
Transporttyp	CLIENT

4. Klicken Sie in der Navigationsstruktur auf **Applications > Install New Application** (Anwendungen > Neue Anwendung installieren).
5. Geben Sie den Pfad der Datei `WMQHTTP.war` und das Kontextstammverzeichnis im Formular an und klicken Sie auf **Weiter**.
   a) Das Kontextstammverzeichnis ist optional. In den HTTP-Beispielanwendungen ist `mq` das Standard-Kontextstammverzeichnis.
   b) Das Kontextstammverzeichnis ist ein Teil der URI der IBM MQ bridge for HTTP. Sie können das Kontextstammverzeichnis weglassen oder es auch erst später ändern.
6. Auf der Seite **Select installation options** (Installationsoptionen auswählen) des Installationsassistenten muss nichts geändert werden. Klicken Sie einfach auf **Weiter**.
7. Wählen Sie auf der Seite **Map modules to servers** (Module zu Servern zuordnen) ein Cluster oder einen Server aus, aktivieren Sie das Auswahlfeld und klicken Sie auf **Apply > Next** (Anwenden > Weiter).
8. Klicken Sie auf der Seite **Ressourcenreferenzen zu Ressourcen zuordnen** im Formular **javax.jms.ConnectionFactory** auf **Durchsuchen ...** in der Zeile IBM MQ bridge for HTTP .
9. Wählen Sie auf der Seite **Enterprise Applications > Available resources** (Unternehmensanwendungen > Verfügbare Ressourcen) **WMQHTTPBridge** aus und klicken Sie auf **Apply** (Anwenden).
10. Wählen Sie nun wieder im Formular **javax.jms.ConnectionFactory** die Authentifizierungsmethode aus.
    a) Wählen Sie zum Beispiel **None** (Keine) aus und klicken Sie auf **Apply** (Anwenden). Für alle anderen Optionen sind weitere Einstellungen erforderlich.
11. Aktivieren Sie das Kontrollkästchen **Select** (Auswählen) für die IBM MQ bridge for HTTP und klicken Sie auf **Next > Next > Finish > Save** (Weiter > Weiter > Fertig > Speichern).
12. Klicken Sie in der Navigationsstruktur auf **Applications > Enterprise Applications** (Anwendungen > Unternehmensanwendungen).
13. Aktivieren Sie das Auswahlfeld für `WMQHTTP.war` und klicken Sie auf **Start**.
14. Öffnen Sie ein Browserfenster. Geben Sie `http://itso-01:9080/mq/msg/queue/HTTPTESTQ` mit dem richtigen Hostnamen und Port ein.

Ergebnisse

Im Browserfenster wird `First Message`angezeigt, wenn die Konfiguration erfolgreich war.

Nächste Schritte

Führen Sie die HTTP Java-Beispielanwendungen aus.

Publish/Subscribe mithilfe von IBM MQ bridge for HTTP

IBM MQ bridge for HTTP verwendet die Publish/Subscribe-Schnittstelle von IBM MQ classes for JMS. Mit HTTP **POST** wird eine Veröffentlichung erstellt. Mit HTTP **DELETE** wird eine nicht permanente Subskription erstellt. Sie müssen Publish/Subscribe für JMS konfigurieren, um die Themen-URI verwenden zu können.

Publish/Subscribe ist vollständig in IBM WebSphere MQ 7 vor Version 7 integriert. Ein separater Publish/Subscribe-Broker verarbeitet Veröffentlichungen und Subskriptionen. Es wird als "eingereihtes" Publish/Subscribe bezeichnet, um es von dem vollständig integrierten Publish/Subscribe in Version 7 zu unterscheiden. IBM WebSphere MQ 7 emuliert eingereihtes Publish/Subscribe unter Verwendung von integriertem Publish/Subscribe. Durch die Emulation können vorhandene Publish/Subscribe-Anwendungen neben integrierten Anwendungen ausgeführt werden, die auf dem gleichen Warteschlangenmanager aktiv sind. Eingereihte Publish/Subscribe-Anwendungen können außerdem mit integrierten Anwendungen zusammenarbeiten, wobei die gleichen Themen verwendet werden. In IBM WebSphere MQ 6 wurde der Broker mit IBM MQ geliefert; vor IBM WebSphere MQ 6 war er als Support-Pack verfügbar.

Konfiguration

Die IBM MQ bridge for HTTP verwendet die JMS-Schnittstelle zum Veröffentlichen und Subskribieren. In Version 7 können Sie mit der Eigenschaft PROVIDERVERSION JMS steuern, ob IBM MQ classes for JMS eingereihtes oder integriertes Publish/Subscribe verwendet.

Sie können außerdem entscheiden, ob Sie die IBM MQ MQI client-Bibliotheken mit IBM MQ bridge for HTTP oder Serverbibliotheken verwenden möchten. Clientbibliotheken der IBM WebSphere MQ 6 unterstützen nur das eingereihte Publish/Subscribe, wohingegen Bibliotheken der Version 7 sowohl das eingereihte als auch das integrierte Publish/Subscribe unterstützen. Die meisten Web- oder Anwendungsserver, die IBM MQ als Messaging-Provider verwenden, nutzen dabei Clientbibliotheken. Damit integriertes Publish/Subscribe verwendet werden kann, müssen sowohl die IBM MQ MQI client -als auch die Serverbibliotheken mindestens Version 7 aufweisen. Wenn eine der beiden Versionen eine frühere Version von WebSphere als 7 ausführt, müssen Sie eingereihtes Publish/Subscribe konfigurieren; siehe Tabelle 87 auf Seite 736. Überprüfen Sie, welche Bibliotheken mit dem von Ihnen verwendeten Web-Server oder Anwendungsserver installiert oder konfiguriert sind.

Tabelle 87. Publish/Subscribe-Konfigurationsmodus

	Client V6 oder früher	**Client V7 oder höher**
Server V6 oder früher	1. Führen Sie das Script `\java\bin\MQJMS_PSQ.mqsc` aus.	Nicht unterstützt
Server V7 oder höher	1. Führen Sie das Script `\java\bin\MQJMS_PSQ.mqsc` aus. 2. Setzen Sie den Warteschlangenmanager auf PSMODE=ENABLED.	1. Falls PROVIDERVERSION = 7 a. Setzen Sie den Warteschlangenmanager auf PSMODE=ENABLED oder PSMODE=COMPAT. 2. Falls PROVIDERVERSION = 6 a. Setzen Sie den Warteschlangenmanager auf PSMODE=ENABLED.

Veröffentlichen

Senden Sie eine HTTP **POST**-Anforderung mit folgender URI:

```
http://hostname: port/context_root/msg/topic/topicString
```

Der Nachrichteninhalt wird mit der Themenzeichenfolge *topicString*veröffentlicht.

Abonnieren

Senden Sie eine HTTP **DELETE**-Anforderung mit folgender URI:

```
http://hostname: port/context_root/msg/topic/topicString
```

IBM MQ bridge for HTTP erstellt eine verwaltete nicht permanente Subskription für die Themenzeichenfolge *topicString*. Die Subskription wird gelöscht, sobald eine Veröffentlichung zurückgegeben wird oder wenn das Warteintervall abläuft, das mit dem angepassten Entity-Header `x-msg-wait` festgelegt wurde.

IBM MQ bridge for HTTP-Beispiele ausführen

Die IBM MQ bridge for HTTP-Beispiele sind nur für die Verwendung auf dem Windows-Betriebssystem verfügbar. Die Beispiele zeigen, wie HTTP **POST** -und HTTP **DELETE** -Befehle aus Java -Programmen an IBM MQ bridge for HTTP übergeben werden.

Vorbereitende Schritte

Überprüfen Sie Ihre IBM MQ Bridge for HTTP-Installation, indem Sie Schritt „7” auf Seite 733 unter „IBM MQ bridge for HTTP installieren, konfigurieren und prüfen” auf Seite 732 ausführen.

Die HTTP-Beispiele werden in den Verzeichnissen installiert, die in Tabelle 88 auf Seite 737 gezeigt werden. In jedem Fall wird der Quellcode im Unterverzeichnis `/src` installiert.

Tabelle 88. Position der HTTP-Beispiele

Plattform	Position
Windows	*MQ_INSTALLATION_PATH*`/tools/http/samples`
z/OS	*PathPrefix*`/usr/lpp/mqm/V7R0M0/http/samples`
IBM i	*MQ_INSTALLATION_PATH*`/java/samples/http`
Alle anderen Plattformen	*MQ_INSTALLATION_PATH*`/samp/http`

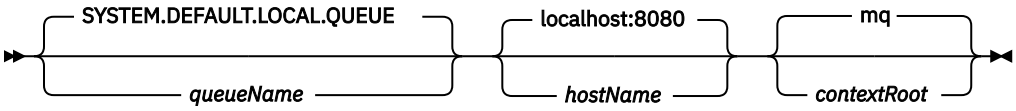
MQ_INSTALLATION_PATH steht für das Verzeichnis, in dem IBM MQ installiert ist.

Informationen zu diesem Vorgang

In den Beispielen werden die IBM MQ-Beispielanwendungen AMQSPUT und AMQSGET simuliert. Sie veranschaulichen die folgenden Funktionen in einer Punkt-zu-Punkt-Messaging-Umgebung:

- **HTTPPOST** -Sendet HTTP-Anforderungen vom Typ **POST** in einer Java -Anwendung zum Einreihen von Nachrichten in eine IBM MQ -Warteschlange unter Verwendung der IBM MQ bridge for HTTP und verarbeitet die Antworten.
- **HTTPDELETE** -Sendet HTTP **DELETE** -Anforderungen in einer Java -Anwendung zum Abrufen von Nachrichten aus einer IBM MQ -Warteschlange unter Verwendung der IBM MQ bridge for HTTP und verarbeitet die Antworten, die die IBM MQ -Nachricht enthalten.

Parameter für HTTPPOST und HTTPDELETE



Gehen Sie zur Ausführung des **HTTPPOST**-Beispiels folgendermaßen vor:

Vorgehensweise

1. Navigieren Sie in einem Befehlsfenster zum Verzeichnis mit den HTTP-Beispielen.
2. Führen Sie das Beispiel **HTTPPOST** aus.

```
java -classpath . HTTPPOST [parameters]
```

Beim Start des **HTTPPOST**-Beispiels wird die folgende Ausgabe angezeigt:

```
HTTP POST Sample start
Target server is ' hostName '
```

```
Target queue is ' queueName '
Target context-root is ' contextRoot '
```

3. Geben Sie in der Eingabeaufforderung den Text für den Nachrichtenhauptteil ein.
4. Drücken Sie die Eingabetaste, um die Nachricht an die IBM MQ-Warteschlange zu übergeben.
   a) Wenn Sie eine weitere Nachricht senden möchten, geben Sie weiteren Text ein.
      Der Text bildet den Hauptteil einer zweiten IBM MQ-Nachricht.
   b) Drücken Sie die Eingabetaste, um die Nachricht an die IBM MQ-Warteschlange zu übergeben.
5. Drücken Sie die Eingabetaste zwei Mal, um **HTTPPOST** zu beenden.
   Die folgende Ausgabe wird angezeigt:

```
HTTP POST Sample end
```

Nächste Schritte

Das **HTTPDELETE**-Beispiel führt einen Abruf mit Löschen aller Nachrichten aus, die Sie in die IBM MQ-Warteschlange eingereiht haben.

Führen Sie das **HTTPDELETE**-Beispiel folgendermaßen aus:

1. Navigieren Sie in einem Befehlsfenster zu *MQ_INSTALLATION_PATH*/tools/samples. *MQ_INSTALLATION_PATH* steht für das Verzeichnis, in dem IBM MQ installiert ist.
2. Führen Sie das Beispiel **HTTPDELETE** aus.

```
java -classpath . HTTPPOST [parameters]
```

   Beim Start des **HTTPDELETE**-Beispiels wird die folgende Ausgabe angezeigt:

```
HTTP DELETE Sample start
Target server is ' host:port '
Target queue is ' your queue name '
Target context-root is ' your context-root '
message
message
...
```

Sicherheitsaspekte bei IBM MQ bridge for HTTP

Die standardmäßigen Aspekte zur Websicherheit gelten für die Authentifizierung eines Web-Browser-Clients. Die Autorisierung für IBM MQ-Ressourcen erfolgt auf der Stufe des Benutzers, der das Servlet IBM MQ bridge for HTTP ausführt, und nicht auf der Stufe des einzelnen Web-Browser-Clients. Aspekte zur Standardsicherheit in IBM MQ gelten für IBM MQ.

Bei der Übertragung von Daten aus einem Web-Browser an eine IBM MQ-Anwendung mithilfe der IBM MQ bridge for HTTP und zurück werden drei Schritte ausgeführt:

Clientverbindung
Aus dem Browser zur IBM MQ bridge for HTTP über eine TCP/IP-Verbindung mithilfe von HTTP.

Ressourcenadapterverbindung zu IBM MQ
Die Verbindung wird von der IBM MQ bridge for HTTP zu einem IBM MQ-Warteschlangenmanager hergestellt. Bei der Verbindung handelt es sich um eine Clientverbindung, eine Verbindung über TCP/IP oder eine lokale IBM MQ-Bindungsverbindung. Sobald die Verbindung hergestellt wurde, wird die HTTP-Anforderung in eine standardmäßige lokale Warteschlange oder eine Übertragungswarteschlange eingereiht.

Aus der lokalen IBM MQ-Warteschlange über einen oder mehrere Kanäle an die Zielwarteschlange.
Wenden Sie Standardverfahren zur Sicherung von Warteschlangen, Themen, Warteschlangenmanagern und Kanälen an.

Bei der Antwort werden die Schritte in umgekehrter Reihenfolge ausgeführt.

Clientverbindung

Sichern Sie Verbindungen zwischen HTTP-Clients und dem Anwendungsserver mithilfe des Web-Containers. Verwenden Sie standardmäßige Verfahren für den HTTP-Server, z. B. HTTPS. Weitere Informationen finden Sie in der Dokumentation zu Ihrem Anwendungsserver.

Ressourcenadapterverbindung zu IBM MQ

Die Verbindung zwischen dem Ressourcenadapter und dem Warteschlangenmanager wird mithilfe einer Einzelbenutzer-ID autorisiert. Ordnen Sie eine Einzelbenutzer-ID zu, um Anforderungen von der IBM MQ bridge for HTTP zu ermitteln. Die Benutzer-ID muss über eingeschränkte IBM MQ-Berechtigungen für ausschließlich die Ressourcen verfügen, auf die externe Benutzer zugreifen müssen. Sie müssen den tatsächlichen Client separat authentifizieren und eine Vertrauensbeziehung für aufeinanderfolgende Interaktionen mit dem Client aufbauen. Verwenden Sie dabei Standardverfahren zur Websicherheit.

Sichern Sie die Verbindung zwischen dem Ressourcenadapter und dem Warteschlangenmanager mit der Einzelbenutzer-ID. Schränken Sie die Berechtigungen für die Benutzer-ID auf das erforderliche Minimum ein, damit Nachrichten in Warteschlangen und Themen gelesen und geschrieben werden können. Bei der IBM MQ bridge for HTTP handelt es sich um einen Angriffspunkt zwischen dem Internet und Ihrem Intranet.

Die Vorgehensweise beim Sichern der Verbindung zwischen Ihrem Ressourcenadapter und IBM MQ ist von Ihrem speziellen Ressourcenadapter abhängig. Weitere Informationen finden Sie in der Dokumentation zum Ressourcenadapter.

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 740.

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 11
IBM MQ unterstützt .NET, ActiveX, C++, Java und JMS. 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.

„Konzepte für die Anwendungsentwicklung" auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

Zugehörige Informationen

Technische Übersicht

Verweis auf die Anwendungsentwicklung

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 978 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:

Zusatz	Sprache
a	Assemblersprache
b	Visual Basic
c	C
l	COBOL (ohne Anfangswerte)
p	PL/I
v	COBOL (mit gesetzten Standardwerten)

Installationsbibliothek

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 741
- „Visual Basic-Moduldateien” auf Seite 741
- „Kopierdateien in COBOL” auf Seite 741
- z/OS „Makros in Assemblersprache System/390” auf Seite 743
- z/OS „PL/I-Kopfdatendateien” auf Seite 743

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
IBM i IBM i	
UNIX	*MQ_INSTALLATION_PATH*/inc/
Systeme mit Windows	*MQ_INSTALLATION_PATH*\Tools\c\include
z/OS	**thlqual**.SCSQC370
z/OS z/OS	

Dabei steht *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Anmerkung: Unter UNIX 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**
Andere UNIX-Plattformen	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i 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 z/OS 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 „Programm für die Ausführung vorbereiten” auf Seite 1088). Schließen Sie nur CMQP mit ein, wenn Sie die IBM MQ-Aufrufe dynamisch verknüpfen möchten (siehe „IBM MQ-Stub dynamisch aufrufen” auf Seite 1094). 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 744
- „Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
- „Objekte öffnen und schließen” auf Seite 769
- „Nachrichten in eine Warteschlange einreihen” auf Seite 781
- „Nachrichten aus einer Warteschlange abrufen” auf Seite 797
- „Publish/Subscribe-Anwendungen schreiben” auf Seite 840
- „Objektattribute abfragen und einstellen” auf Seite 885
- „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
- „IBM MQ-Anwendungen durch Auslöser starten” auf Seite 901
- „Mit MQI und Clustern arbeiten” auf Seite 922
- z/OS „Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
- z/OS „IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

„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 48
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 953
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„Prozedurale Anwendung erstellen” auf Seite 1051
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 1103
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 1124
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

„Web-Services mit IBM MQ entwickeln” auf Seite 1368
Sie können IBM MQ-Anwendungen für Web-Services mithilfe des IBM MQ-Transports für SOAP entwickeln.

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 „Beispielprogramme für z/OS verwenden” auf Seite 1236.

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.

IBM MQ for Windows und IBM MQ auf UNIX and Linux-Systemen stellen darüber hinaus noch Folgendes bereit:

- Aufrufe, über die Systemprogramme von IBM MQ for Windows und IBM MQ unter UNIX and Linux 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 1125.
- Beispielquellcode und ausführbaren Beispielcode für die Verbindung zu externen Transaktionsmanagern

Unter den folgenden Links finden Sie weitere Informationen zur MQI:

- „MQI-Aufrufe” auf Seite 746
- „Synchronisationspunktaufrufe” auf Seite 747
- „Datenkonvertierung, Datentypen, Datendefinitionen und Strukturen” auf Seite 747
- „IBM MQ-Stubprogramme und -Bibliotheksdateien” auf Seite 748
- „Parameter, die in allen Aufrufen verwendet werden” auf Seite 754
- „Puffer angeben” auf Seite 755
- z/OS „Hinweise zu z/OS-Stapelprogrammen” auf Seite 756
- „Signalverarbeitung unter UNIX and Linux” auf Seite 757

Zugehörige Konzepte

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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. 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 „Transaktionsmanagement und Recoverable Resource Manager Services (RRS)" auf Seite 894.)

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 unter Windows, UNIX and Linux
ULW

In den folgenden Produkten werden die Aufrufe MQCMIT und MQBACK bereitgestellt:

- IBM MQ for Windows
- IBM MQ auf UNIX and Linux-Systemen

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. Mit Ausnahme von IBM MQ for z/OS kann dieser Aufruf nur über einen Datenkonvertierungsexit verwendet werden.

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 1033.

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 740.

Strukturen

Strukturen, die zusammen mit den unter „MQI-Aufrufe" auf Seite 746 aufgeführten MQI-Aufrufen verwendet werden, werden in Datendefinitionsdateien für jede unterstützte Programmiersprache bereitgestellt. IBM i z/OS IBM MQ for z/OS und IBM MQ for IBM i stellen Dateien bereit, die Konstanten zum Ausfüllen einiger Felder in diesen Strukturen enthalten. Weitere Informationen hierzu finden Sie im Abschnitt IBM MQ-Datendefinitionen.

Eine Übersicht über die Strukturen finden Sie unter Structure data types summary.

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 1051. Informationen zum Herstellen einer Verknüpfung mit C++-Bibliotheksdateien finden Sie unter C++ verwenden *IBM MQC++ verwenden*.

AIX *IBM MQ for AIX*

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 89. Bibliotheken für AIX-Anwendungen, bei denen es sich nicht um Thread-Anwendungen handelt	
Bibliotheksdatei	**Umgebung**
libmqm.a	Server für C
libmqic.a & **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
libmqicb.a	Client für COBOL
libimqc23ia.a	Client für C++
libimqs23ia.a	Server für C++

Stellen Sie in einer Thread-Anwendung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Tabelle 90. 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 & **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++
libimqs23ia_r.a	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.

IBM MQ for HP-UX

In IBM MQ for HP-UX 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.

IA64-Plattform (IPF)

Stellen Sie in einer Nicht-Thread-Anwendung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Tabelle 91. Bibliotheken für HP-UX-Anwendungen, bei denen es sich nicht um Thread-Anwendungen handelt	
Bibliotheksdatei	**Umgebung**
libmqm.so	Server für C
libmqic.so & **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
libimqi23ah.so	C + +
libmqmcbrt.o	IBM MQ-Laufzeitbibliothek für Micro Focus COBOL-Unterstützung
libmqmcb.so	Server für COBOL
libmqicb.so	Client für COBOL

Stellen Sie in einer Thread-Anwendung eine Verknüpfung zu einer der folgenden Bibliotheken her:

Tabelle 92. Bibliotheksdateien für HP-UX-Thread-Anwendungen	
Bibliotheksdatei	**Umgebung**
libmqm_r.so	Server für C
libmqmzf_r.so & **libmqm_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
libimqi23ah_r.so	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.

IBM i *IBM MQ for IBM i*

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 93. 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 94. 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.

IBM MQ for Linux

In IBM MQ for 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 95. Bibliotheken für Linux-Anwendungen, bei denen es sich nicht um Thread-Anwendungen handelt	
Bibliotheksdatei	**Umgebung**
libmqm.so	Server für C
libmqic.so & **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 96. Bibliotheksdateien für Linux-Thread-Anwendungen	
Bibliotheksdatei	**Umgebung**
libmqm_r.so	Server für C
libmqic_r.so & **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.

Solaris *IBM MQ for Solaris*

In IBM MQ for Solaris 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.

Tabelle 97. Bibliotheksdateien für Solaris-Anwendungen	
Bibliotheksdatei	**Umgebung**
libmqm.so	Server und Client für C
libmqmzse.so	Für C
libmqic.so	Client für C
libmqmcs.so	Allgemeine Services 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
libimqc23as.a	Client für C++
libimqs23as.a	Server für C++

Windows *IBM MQ for Windows*

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 wird:

Tabelle 98. 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)
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_INSTALLATI-ON_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_INSTALLATI-ON_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 605 im Abschnitt „.NET-Anwendungen entwickeln” auf Seite 557.

Die folgenden Dateien werden aus Gründen der Kompatibilität mit früheren Releases bereitgestellt:

```
mqic32.lib
mqic32xa.lib
```

z/OS *IBM MQ for z/OS-Stubprogramme*

Bevor Sie ein mit IBM MQ for z/OS erstelltes Programm ausführen können, müssen Sie für dieses Programm eine Programmverknüpfung mit dem in IBM MQ for z/OS bereitgestellten Stubprogramm für die Umgebung herstellen, in der die Anwendung ausgeführt wird.

Das Stubprogramm stellt die erste Phase in die Verarbeitung Ihrer Aufrufe in Anforderungen, die von IBM MQ for z/OS verarbeitet werden können, bereit.

In IBM MQ for z/OS stehen die folgenden Stubprogramme zur Verfügung:

CSQBSTUB
Stubprogramm für z/OS-Stapelprogramme

CSQBRRSI
Stubprogramm für z/OS-Stapelprogramme unter Verwendung von RRS über die MQI

CSQBRSTB
Stubprogramm für z/OS-Stapelprogramme unter direkter Verwendung von RRS

CSQCSTUB
Stubprogramm für CICS-Programme

CSQQSTUB
Stubprogramm für IMS-Programme

CSQXSTUB
Stubprogramm für Exits (bei denen es sich nicht um CICS-Exits handelt) für die verteilte Steuerung von Warteschlangen

CSQASTUB
Stubprogramm für Datenkonvertierungsexit

Achtung: Wenn Sie für eine Umgebung ein anderes als eines der aufgeführten Stubprogramme verwenden, kann dies ein unvorhersehbares Ergebnis zur Folge haben.

Anmerkung: Bei Verwendung des Stubprogramms CSQBRSTB müssen Sie über SYS1.CSSLIB eine Programmverknüpfung mit ATRSCSS herstellen (SYS1.CSSLIB wird auch als *Bibliothek mit den aufrufbaren Services*) bezeichnet. Weitere Informationen zu RSS finden Sie unter „Transaktionsmanagement und Recoverable Resource Manager Services (RRS)” auf Seite 894.

Ebenso können Sie das Stubprogramm auch dynamisch innerhalb Ihres Programms aufrufen. Eine Beschreibung dieser Vorgehensweise finden Sie unter „IBM MQ-Stub dynamisch aufrufen” auf Seite 1094.

In IMS müssen Sie unter Umständen zusätzlich noch ein bestimmtes Sprachschnittstellenmodul verwenden, das von IBM MQ bereitgestellt wird.

Anwendungen, für die Programmverknüpfungen mit CSQBSTUB und CSQQSTUB hergestellt wurden, dürfen nicht in demselben IMS-MPP-Bereich ausgeführt werden, da dies zu Fehlernachrichten wie beispielsweise DFS3607I oder CSQQ005E führen kann. Der erste MQCONN-Aufruf in einem Adressraum legt die Schnittstelle fest, die verwendet wird, daher müssen die Transaktionen CSQQSTUB und CSQBSTUB in unterschiedlichen IMS-Nachrichtenbereichen ausgeführt werden.

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 for IBM 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 Warte-

schlangenmanager 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 *Hinweise zu z/OS-Stapelprogrammen*

z/OS-Stapelprogramme, die die MQI aufrufen, können den Supervisor- oder den Fehlerstatus haben.

Sie müssen jedoch die folgenden Bedingungen erfüllen:

- Sie müssen im Taskmodus sein, nicht im SRB-Modus (Service Request Block, Serviceanforderungsblock).
- Sie müssen im Primary ASC-Modus (Primary Address Space Control) sein, nicht im Access Register ASC-Modus.
- Sie dürfen nicht im speicherübergreifenden Modus sein. Die Nummer des Primäradressraums muss der Nummer des Sekundäradressraums und der Nummer des Speicherbereichs für die Ausgangsadresse entsprechen.
- Sie dürfen nicht als MPF-Exitprogramme verwendet werden.
- Es können keine z/OS-Sperren bestehen.
- Es kann keine Funktionswiederherstellungsroutinen (FFR, Function Recovery Routines) für den FR-Stack geben.
- Für den MQCONN- oder MQCONNX-Aufruf kann ein beliebiger PSW-Schlüssel (Program Status Word, Programmstatuswort) gelten (sofern dieser Schlüssel mit der Verwendung von Speicher mit dem TCB-Schlüssel kompatibel ist); für nachfolgende Aufrufe allerdings, die die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden, gilt Folgendes:
  - Sie müssen denselben PSW-Schlüssel haben, der auch im MQCONN- oder MQCONNX-Aufruf verwendet wurde.
  - Sie müssen Parameter haben, auf die unter demselben PSW-Schlüssel zugegriffen werden kann (bei Bedarf müssen Schreibzugriffe möglich sein).
  - Sie müssen unter Steuerung derselben Task (TCB) ausgegeben werden, nicht jedoch in einer Subtask dieser Task.
- Sie können im 24-Bit- oder 31-Bit-Adressierungsmodus sein. Im 24-Bit-Adressierungsmodus müssen die Parameteradressen jedoch als gültige 31-Bit-Adressen interpretiert werden.

Wird eine dieser Bedingungen nicht erfüllt, kommt es unter Umständen zu einem Programmfehler. In einigen Fällen wird der Aufruf fehlschlagen und ein Ursachencode zurückgegeben.

Linux UNIX *Überlegungen zu UNIX and Linux*

Hier sind einige einige Punkte aufgeführt, die Sie beachten sollten.

Bei der Entwicklung von Anwendungen für UNIX and Linux sollten Sie Folgendes beachten.

Linux UNIX *Systemaufruf 'fork' unter UNIX 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 dem Absetzen von IBM MQ -Aufrufen einen untergeordneten Prozess erstellen möchte, muss der Anwendungscode ein `fork()` mit `exec()` verwenden, um sicherzustellen, dass das untergeordnete Element eine neue Instanz und keine exakte Kopie des übergeordneten Elements ist.

Wenn Ihre Anwendung `exec()`nicht verwendet, gibt der API-Aufruf IBM MQ innerhalb des untergeordneten Prozesses den Fehler MQRC_ENVIRONMENT_ERROR zurück.

Linux UNIX *Signalverarbeitung unter UNIX and Linux*

Diese Informationen gelten nicht für IBM MQ for z/OS oder IBM MQ for Windows.

Bei UNIX-, Linux- und IBM i-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.

UNIX *Nicht-Thread-Anwendungen*

Diese Informationen gelten nicht für Solaris, da alle Anwendungen als Thread-Anwendungen gesehen werden, auch wenn sie nur einen Einzelthread verwenden.

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 UNIX *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.

UNIX 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)

UNIX *Zusätzliche Informationen bei Verwendung der UNIX-Signalverarbeitung in MQI*
Bei der Verwendung der UNIX-Signalverarbeitung müssen Sie außerdem noch Folgendes beachten.

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 763.

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.

VMS-Exit-Handler

Benutzer können mithilfe des Systemservice **SYS$DCLEXH** Exit-Handler für eine IBM MQ-Anwendung installieren.

Der Exit-Handler erhält bei der Beendigung eines Image die Steuerung. Normalerweise kommt es zur Beendigung eines Image beim Aufruf des Exit-Service ($EXIT) oder des Force Exit-Serivce ($FORCEX). Der $FORCEX-Service unterbricht den Zielprozess im Benutzermodus. Anschließend werden alle Exit-Handler im Benutzermodus (die mit $DCLEXH definiert wurden) in der umgekehrten Reihenfolge ihrer Definition ausgeführt. Ausführlichere Informationen zu Exit-Handlern und dem $FORCEX-Service finden Sie im *VMS Programming Concepts Manual* und im *VMS System Services Manual*.

Wenn Sie eine MQI-Funktion innerhalb eines Exit-Handlers aufrufen, hängt das Verhalten der Funktion davon ab, wie das Image beendet wurde. Wurde es beendet, während eine andere MQI-Funktion noch aktiv war, wird der Fehler MQRC_CALL_IN_PROGRESS zurückgegeben.

Es ist möglich, eine MQI-Funktion innerhalb eines Exit-Handlers aufzurufen, wenn keine andere MQI-Funktion aktiv ist und aufwärts gerichtete Aufrufe (Upcalls) für die IBM MQ-Anwendung inakti-

viert wurden. Sind Upcalls für die IBM MQ-Anwendung aktiviert, schlägt sie mit dem Ursachencode MQRC_HCONN_ERROR fehl.

Der Gültigkeitsbereich eines MQCONN- oder MQCONNX-Aufrufs ist in der Regel der Thread, von dem er ausgegeben wurde. Wenn Upcalls aktiviert sind, wird der Exit-Handler als separater Thread ausgeführt und die Verbindungskennungen können nicht gemeinsam genutzt werden.

Exit-Handler werden innerhalb des unterbrochenen Kontextes des Zielprozesses gestartet. Die Anwendung muss sicherstellen, dass die vom Handler ausgeführten Aktionen für den asynchron unterbrochenen Kontext, aus dem sie ausgerufen wurden, sicher und zuverlässig 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 „Synchronisationspunkte in IMS-Anwendungen” auf Seite 892).

IMS-Programme können nacheinander oder gleichzeitig eine Verbindung zu mehreren Warteschlangenmanagern innerhalb desselben TCB herstellen.

z/OS **CICS Transaction Server for 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. In einer CICS-Umgebung werden diese Aufrufe jedoch *immer* 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 761
- „Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen" auf Seite 763
- „Programme mit MQDISC von einem Warteschlangenmanager trennen" auf Seite 768

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht" auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Objekte öffnen und schließen" auf Seite 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen" auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen" auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen" auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen" auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten" auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben" auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS" auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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:

- Bei IBM MQ for z/OS in der Stapelumgebung wird der Standardwarteschlangenmanager im Modul CSQBDEFV angegeben.
- Bei IBM MQ for Windows-, IBM i-, UNIX- und Linux-Systemen wird der Standardwarteschlangenmanager in der Datei 'mqs.ini' angegeben.

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.

In der IMS-Umgebung muss der Warteschlangenmanager mit der IMS-Steuerregion verbunden sein und mit der abhängigen Region, die vom Programm verwendet wird. Der Standardwarteschlangenmanager wird im CSQQDEFV-Modul angegeben, wenn IBM MQ for z/OS installiert wird.

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 770).

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. Wenn es möglich sein soll, dass von einem einzelnen Thread mehrere IBM MQ-Verbindungen hergestellt werden bzw. dass eine IBM MQ-Verbindung von von jedem Thread verwendet wird, lesen Sie unter „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX” auf Seite 766 nach.[8]

[8] Wenn Sie Multithread-Anwendungen mit IBM MQ auf UNIX and Linux -Systemen verwenden, müssen Sie sicherstellen, dass die Stackgröße der Anwendungen für die Threads ausreicht. Wenn Multithread-Anwendungen MQI-Aufrufe entweder selbst oder über andere Signalhandler (beispielsweise CICS) ausgeben, sollten Sie eine Stackgröße von 256 KB oder mehr verwenden.

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.

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 Ausgabe von MQCONNX enthält folgende Komponenten:

- Eine Verbindungskennung (Hconn)
- Einen Beendigungscode
- Einen Ursachencode

Die Verbindungskennung verwenden Sie für nachfolgende MQI-Aufrufe.

Eine Beschreibung aller MQCONNX-Parameter finden Sie in MQCONNX. Im Feld *Options* (Optionen) können Sie für jede MQCNO-Version folgende Bindungstypen festlegen: STANDARD_BINDING, FASTPATH_BINDING, SHARED_BINDING oder ISOLATED_BINDING. Zudem können Sie mithilfe eines MQCONNX-Aufrufs gemeinsam genutzte (threadunabhängige) Verbindungen herstellen. Weitere Informationen hierzu finden Sie im Abschnitt „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX" auf Seite 766.

MQCNO_STANDARD_BINDING

MQCONNX schließt (wie MQCONN) zwei logische Threads ein, wobei die IBM MQ-Anwendung und der lokale Warteschlangenmanageragent in separaten Prozessen ausgeführt werden. Die IBM MQ-Anwendung fordert die IBM MQ-Operation an, und der lokale Warteschlangenmanageragent verarbeitet die Anforderung. Dies wird durch die Option MQCNO_STANDARD_BINDING im MQCONNX-Aufruf festgelegt.

Wenn Sie MQCNO_STANDARD_BINDING angeben, verwendet der MQCONNX-Aufruf abhängig vom Wert des Attributs **DefaultBindType** des Warteschlangenmanagers, der in qm.inidefiniert ist, entweder MQCNO_SHARED_BINDING oder MQCNO_ISOLATED_BINDING.

Dies ist der Standardwert.

Wenn Sie eine Verbindung zur Bibliothek `mqm` herstellen, wird zunächst versucht, eine standardmäßige Serververbindung unter Verwendung des Standardbindungstyps herzustellen. Kann die entsprechende Serverbibliothek nicht geladen werden, wird versucht, eine Clientverbindung herzustellen.

- Bei Angabe der Umgebungsvariablen MQ_CONNECT_TYPE kann das Verhalten von MQCONN bzw. von MQCONNX, wenn MQCNO_STANDARD_BINDING angegeben ist, mit einer der folgenden Optionen geändert werden (außer wenn MQCNO_FASTPATH_BINDING angegeben wird und MQ_CONNECT_TYPE auf LOCAL oder STANDARD gesetzt ist, damit der Administrator für Fastpath-Verbindungen einen Downgrade durchführen kann, ohne dass Änderungen an der Anwendung vorgenommen werden müssen):

Wert	Bedeutet
CLIENT	Es wird nur versucht, eine Clientverbindung herzustellen.
FASTPATH	Dieser Wert wird in früheren Releases unterstützt, jetzt bei Angabe aber ignoriert.
LOKAL	Es wird versucht, nur eine Serververbindung herzustellen. Für Fastpath-Verbindungen wird ein Downgrade auf eine Standardserververbindung durchgeführt.

Wert	Bedeutet
STANDARD	Wird aus Gründen der Kompatibilität mit früheren Releases unterstützt. Dieser Wert wird nicht wie LOCAL gehandhabt.

- Ist die Umgebungsvariable MQ_CONNECT_TYPE beim Aufruf von MQCONN nicht gesetzt, wird versucht, eine standardmäßige Serververbindung mit dem Standardbindungstyp herzustellen. Kann die Serverbibliothek nicht geladen werden, wird versucht, eine Clientverbindung herzustellen.

MQCNO_FASTPATH_BINDING

Vertrauenswürdige Anwendungen implizieren, dass die IBM MQ-Anwendung und der lokale Warteschlangenmanageragent in einem Prozess ausgeführt werden. Da der Agentenprozess auf den Warteschlangenmanager nicht mehr über eine Schnittstelle zugreifen muss, werden diese Anwendungen zu einer Erweiterung des Warteschlangenmanagers. Dies wird im MQCONNX-Aufruf durch die Option MQCNO_FASTPATH_BINDING festgelegt.

Sie müssen vertrauenswürdige Anwendungen mit den IBM MQ-Threadbibliotheken verknüpfen. Anweisungen zur Definition einer IBM MQ-Anwendung als vertrauenswürdig finden Sie im Abschnitt MQCNO-Optionen.

Diese Option bietet die beste Leistung.

Anmerkung: Diese Option beeinträchtigt die Integrität des Warteschlangenmanagers: es gibt keinen Schutz vor Überschreibung seines Speichers. Das gilt ebenso, wenn die Anwendung Fehler enthält, die für Nachrichten und andere Daten im Warteschlangenmanager zugänglich sein können. Bedenken Sie diese Probleme, bevor Sie diese Option verwenden.

MQCNO_SHARED_BINDING

Geben Sie diese Option an, damit die Anwendung und der lokale Warteschlangenmanageragent in separaten Prozessen ausgeführt werden. Dadurch wird die Integrität des Warteschlangenmanagers aufrechterhalten, d. h., er wird vor fehlgeleiteten Programmen geschützt. Die Anwendung und der lokale Warteschlangenmanageragent nutzen jedoch einige Ressourcen gemeinsam.

Diese Option stellt sowohl hinsichtlich des Schutzes der Integrität des Warteschlangenmanagers als auch hinsichtlich der Leistung von MQI-Aufrufen eine Zwischenstufe zwischen MQCNO_FASTPATH_BINDING und MQCNO_ISOLATED_BINDING dar.

Wenn der Warteschlangenmanager diesen Bindungstyp nicht unterstützt, wird MQCNO_SHARED_BINDING ignoriert. Die Verarbeitung wird so fortgesetzt, als ob diese Option nicht angegeben wurde.

Wenn eine Anwendung über MQCNO_SHARED_BINDING eine Verbindung zum lokalen Warteschlangenmanager hergestellt hat, kann der Warteschlangenmanager gestoppt werden, während die Anwendung aktiv ist. Wenn Sie den Warteschlangenmanager erneut starten, während die Anwendung noch aktiv ist, schlägt der Versuch zum Starten des Warteschlangenmanagers mit dem Fehler AMQ7018 fehl, weil die Anwendung noch Ressourcen belegt, die vom Warteschlangenmanager benötigt werden.

Um den Warteschlangenmanager starten zu können, müssen Sie die Anwendung stoppen.

MQCNO_ISOLATED_BINDING

Geben Sie diese Option an, damit die Anwendung und der lokale Warteschlangenmanageragent in separaten Prozessen ausgeführt werden wie bei MQCNO_SHARED_BINDING. In diesem Fall sind der Anwendungsprozess und der lokale Warteschlangenmanageragent jedoch separat aktiv, d. h., es gibt keine gemeinsame Nutzung von Ressourcen.

Das ist die sicherste Option für den Schutz der Integrität des Warteschlangenmanagers, aber sie führt dazu, dass MQI-Aufrufe am langsamsten verarbeitet werden.

Wenn der Warteschlangenmanager diesen Bindungstyp nicht unterstützt, wird MQCNO_ISOLATED_BINDING ignoriert. Die Verarbeitung wird so fortgesetzt, als ob diese Option nicht angegeben wurde.

MQCNO_CLIENT_BINDING

Geben Sie diese Option an, wenn die Anwendung nur eine Clientverbindung herstellen soll. Für diese Option gelten die folgenden Einschränkungen:

- z/OS MQCNO_CLIENT_BINDING wird unter z/OS ignoriert.
- MQCNO_CLIENT_BINDING wird mit MQRC_OPTIONS_ERROR abgelehnt, wenn eine andere MQCNO-Bindungsoption als MQCNO_STANDARD_BINDING angegeben wird.
- MQCNO_CLIENT_BINDING steht für Java nicht zur Verfügung, da dies über seinen eigenen Mechanismus zur Auswahl des Bindungstyps verfügt.
- Ist die Umgebungsvariable MQ_CONNECT_TYPE beim Aufruf von MQCONN nicht gesetzt, wird versucht, eine standardmäßige Serververbindung mit dem Standardbindungstyp herzustellen. Kann die Serverbibliothek nicht geladen werden, wird versucht, eine Clientverbindung herzustellen.

MQCNO_LOCAL_BINDING

Geben Sie diese Option an, wenn die Anwendung nur eine Serververbindung herstellen soll. Wenn MQCNO_FASTPATH_BINDING, MQCNO_ISOLATED_BINDING oder MQCNO_SHARED_BINDING ebenfalls angegeben ist, wird stattdessen der jeweilige Verbindungstyp verwendet und in diesem Abschnitt dokumentiert. Andernfalls wird versucht, unter Verwendung des standardmäßigen Bindungstyps eine Standardserververbindung herzustellen. Für MQCNO_LOCAL_BINDING gelten die folgenden Einschränkungen:

- z/OS MQCNO_LOCAL_BINDING wird unter z/OS ignoriert.
- MQCNO_LOCAL_BINDING wird mit MQRC_OPTIONS_ERROR abgelehnt, wenn eine andere MQCNO-Option als MQCNO_RECONNECT_AS_DEF für die Verbindungswiederholung angegeben ist.
- MQCNO_LOCAL_BINDING steht für Java nicht zur Verfügung, da dies über seinen eigenen Mechanismus zur Auswahl des Bindungstyps verfügt.
- Ist die Umgebungsvariable MQ_CONNECT_TYPE beim Aufruf von MQCONN nicht gesetzt, wird versucht, eine standardmäßige Serververbindung mit dem Standardbindungstyp herzustellen. Kann die Serverbibliothek nicht geladen werden, wird versucht, eine Clientverbindung herzustellen.

z/OS Unter z/OS werden diese Optionen toleriert, allerdings wird nur eine Verbindung mit Standardbindung hergestellt.

z/OS MQCNO Version 3 bietet für z/OS vier unterschiedliche Optionen:

MQCNO_SERIALIZE_CONN_TAG_QSG

Damit kann eine Anwendung anfordern, dass zu einem Zeitpunkt in einer Gruppe mit gemeinsamer Warteschlange nur eine Instanz einer Anwendung ausgeführt werden kann. Das geschieht, indem die Verwendung des Verbindungstags mit einem Wert registriert wird, der durch die Anwendung angegeben oder von ihr abgeleitet wird. Das Tag ist eine 128-Byte-Zeichenfolge, die in der Version 3 MQCNO angegeben ist.

MQCNO_RESTRICT_CONN_TAG_QSG

Dies wird verwendet, wenn eine Anwendung aus mehr als einem Prozess besteht (oder einem Tasksteuerblock), von dem jeder eine Verbindung zu einem Warteschlangenmanager herstellen kann. Die Verbindung wird nur zugelassen, wenn der Tag aktuell nicht verwendet wird oder wenn sich die anfordernde Anwendung im selben Verarbeitungsbereich befindet. Das ist der MVS-Adressraum innerhalb derselben Gruppe mit gemeinsamer Warteschlange wie der Tageigner.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

Das ist ähnlich zu MQCNO_SERIALIZE_CONN_TAG_QSG, allerdings wird nur der lokale Warteschlangenmanager abgefragt, um herauszufinden, ob der angeforderte Tag bereits verwendet wird.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

Das ist ähnlich zu MQCNO_RESTRICT_CONN_TAG_QSG, allerdings wird nur der lokale Warteschlangenmanager abgefragt, um herauszufinden, ob der angeforderte Tag bereits verwendet wird.

Einschränkungen für vertrauenswürdige Anwendungen

Für vertrauenswürdige Anwendungen gelten folgende Einschränkungen:

- Vertrauenswürdige Anwendungen müssen explizit vom Warteschlangenmanager getrennt werden.
- Vertrauenswürdige Anwendungen müssen gestoppt werden, bevor der Warteschlangenmanager mit dem Befehl 'endmqm' beendet wird.
- 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).
- Unter IBM MQ auf UNIX 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 MQ for 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. Dieses Problem lässt sich zum Beispiel mit den IBM i-API-Funktionen QSYGETPH, QWTSETP und QSYRLSPH lösen. Mit diesen Funktionen kann der aktuelle Benutzer des Jobs während der Laufzeit des MQ-Programms vorübergehend auf QMQM gesetzt werden. Nähere Informationen zu diesen Funktionen sowie ein Anwendungsbeispiel finden Sie im Abschnitt zu den Sicherheits-APIs im Handbuch *IBM i System API Reference*.
  2. Vertrauenswürdige Anwendungen dürfen nicht mit Systemanfrageoption 2 oder durch Beenden des zugehörigen Jobs mit ENDJOB abgebrochen werden.
- Unter IBM MQ for HP-UX reicht für Multithread-Anwendungen mit Direktaufruf die Standard-Stackgröße vermutlich nicht aus. 256 KB werden empfohlen.
- Unter IBM MQ for Windows werden vertrauenswürdige 64-Bit-Anwendungen nicht unterstützt. Beim Versuch, eine vertrauenswürdige 64-Bit-Anwendung auszuführen, wird diese auf eine Standard-Binding-Verbindung herabgestuft.
- Unter IBM MQ auf UNIX and Linux-Systemen werden vertrauenswürdige 32-Bit-Anwendungen nicht unterstützt. Beim Versuch, eine vertrauenswürdige 32-Bit-Anwendung auszuführen, wird diese auf eine Standard-Binding-Verbindung herabgestuft.

Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX
Dieser Abschnitt enthält Informationen zu gemeinsam genutzten Verbindungen mit MQCONNX und einige zu beachtende Hinweise zur Verwendung.

Anmerkung: Nicht unterstützt unter IBM MQ for z/OS.

Auf IBM MQ-Plattformen mit Ausnahme von IBM MQ for z/OS steht eine mit `MQCONN` hergestellte Verbindung nur dem Thread zur Verfügung, 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.

Der Standardwert ist MQCNO_HANDLE_SHARE_NONE, außer in MTS-Umgebungen (Microsoft Transaction Server). 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 767 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.

Umgebungsvariable MQCONNX
In diesem Abschnitt werden die unterschiedlichen MQCONNX-Aufrufoptionen und deren Verwendung in Verbindung mit MQ_CONNECT_TYPE beschrieben. Dabei ist zu beachten, dass MQ_CONNECT_TYPE

sich nur auf Bindungen des Typs STANDARD auswirkt. Bei anderen Bindungen wird MQ_CONNECT_TYPE ignoriert.

Auf Systemen mit IBM MQ for IBM i, IBM MQ for Windows und IBM MQ for UNIX and Linux können Sie die Umgebungsvariable MQ_CONNECT_TYPE in Kombination mit dem Bindungstyp verwenden, der im Feld *Options* (Optionen) der MQCNO-Struktur angegeben ist, die in einem MQCONNX-Aufruf verwendet wird.

Tabelle 99. Umgebungsvariable MQ_CONNECT_TYPE

MQCONNX-Aufrufoption	**MQ_CONNECT_TYPE, Umgebungsvariable**	**Ergebnis**
STANDARD	NICHT DEFINIERT	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	FASTPATH	STANDARD
STANDARD	CLIENT	CLIENT
STANDARD	LOCAL	STANDARD

Wenn MQCNO_STANDARD_BINDING nicht angegeben wird, können Sie MQCNO_NONE verwenden, das standardmäßig den Wert MQCNO_STANDARD_BINDING annimmt.

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:

- 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.
- 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.

Außer unter CICS on z/OS ist die Verbindungskennung (Hconn) nach dem Aufruf von MQDISC nicht mehr gültig, und Sie können ohne erneuten Aufruf von MQCONN oder MQCONNX keine weiteren MQI-Aufrufe mehr durchführen. 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.

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.

Für jede der folgenden Operationen müssen Sie zunächst 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 790).

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 760.

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 792.

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 770
- „Dynamische Warteschlangen erstellen” auf Seite 779
- „Ferne Warteschlangen öffnen” auf Seite 780
- „Objekte mit dem MQCLOSE-Aufruf schließen” auf Seite 780

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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/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 Programme müssen Sie immer die vom MQCONN- oder MQCONNX-Aufruf zurückgegebene Verbindungskennung verwenden.
- 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 762 und „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX" auf Seite 766. 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-Stapel
In IMS- und Stapelumgebungen können Sie die Kennung zwar in der gleichen Task, jedoch nicht in einer untergeordneten Task 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 793 unter „Verteilerlisten" auf Seite 792.

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 100 auf Seite 772 finden Sie eine Übersicht des Namensauflösungsprozesses. Der Name, den Sie im Feld *ObjectQMgrName* angeben, wird *vor* dem im Feld *ObjectName* bereitgestellten Namen aufgelöst.

Tabelle 100 auf Seite 772 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 100. Warteschlangennamen bei Verwendung von MQOPEN auflösen

Eingabe in MQOD	**Eingabe in MQOD**	**Aufgelöste Namen**	**Aufgelöste Namen**	**Aufgelöste Namen**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	`Transmission queue`
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)

Tabelle 100. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)

Eingabe in MQOD	Eingabe in MQOD	Aufgelöste Namen	Aufgelöste Namen	Aufgelöste Namen
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.		
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)

Tabelle 100. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)

Eingabe in MQOD	**Eingabe in MQOD**	**Aufgelöste Namen**	**Aufgelöste Namen**	**Aufgelöste Namen**
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)
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)

Tabelle 100. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)				
Eingabe in MQOD	**Eingabe in MQOD**	**Aufgelöste Namen**	**Aufgelöste Namen**	**Aufgelöste Namen**
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* ist der Name der Basiswarteschlange aus der Definition der Aliaswarteschlange.
2. *RemoteQName* ist der Name der fernen Warteschlange aus der lokalen Definition der fernen Warteschlange.
3. *RemoteQMgrName* ist der Name des fernen Warteschlangenmanagers aus der lokalen Definition der fernen Warteschlange.
4. *XmitQName* ist der Name der Übertragungswarteschlange aus der lokalen Definition der fernen Warteschlange.
5. Bei Verwendung von IBM MQ for z/OS-Warteschlangenmanagern, die zu einer Gruppe mit gemeinsamer Warteschlange gehören, kann in Tabelle 100 auf Seite 772 statt des Namens des lokalen Warteschlangenmanagers auch der Name der Gruppe mit gemeinsamer Warteschlange verwendet 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 auf die Attribute CLUSTER und 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.CLUSTER.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 Übertragungswarte-

schlange 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 923.

Wenn Sie mit einer QALIAS-Definition eine Clusterwarteschlange öffnen, werden einige Warteschlangenattribute durch die Aliaswarteschlange und nicht durch die Basiswarteschlange festgelegt. Cluste-

rattribute 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) TARGQ(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 im Abschnitt „Nachrichten in einer Warteschlange durchsuchen (Browsing)" auf Seite 833.

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 792.
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 101 auf Seite 777 wird erläutert, wie sich die Einstellung dieser Attribute auf die Art des Zugriffs auswirkt, die Sie bei Verwendung dieser Option erhalten.

Tabelle 101. 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 101 auf Seite 777 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 unter „Verteilerlisten” auf Seite 792.

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 885).

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 Kontextinformationen selbst festlegen möchten oder ob der Kontext automatisch aus einer anderen Warteschlangenkennung übernommen werden soll.

Zugehörige Konzepte
„Nachrichtenkontext” auf Seite 45
Nachrichtenkontext-Informationen ermöglichen der Anwendung, die die Nachricht abruft, Informationen über den Absender der Nachricht zu erhalten.

„Nachrichtenkontextinformationen steuern” auf Seite 788
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 zum Stilllegen eines Warteschlangenmanagers*
Wenn Sie den MQOPEN-Aufruf verwenden, während der Warteschlangenmanager stillgelegt ist, schlägt der Aufruf je nach Umgebung eventuell fehl.

Wenn Sie den MQOPEN-Aufruf in der CICS-Umgebung unter z/OS verwenden, während der Warteschlangenmanager stillgelegt ist, schlägt der Aufruf auf jeden Fall fehl.

In anderen z/OS-Umgebungen, auf IBM i- und Windows-Systemen und in UNIX and Linux-Systemumgebungen schlägt der Aufruf bei einem still gelegten Warteschlangenmanager nur bei Verwendung der Option MQOO_FAIL_IF_QUIESCING des MQOPEN-Aufrufs fehl.

MQOPEN-Option zum Auflösen der Namen lokaler Warteschlangen
Wenn Sie eine lokale Alias- oder Modellwarteschlange öffnen, wird die lokale Warteschlange zurückgegeben.

Wenn Sie allerdings eine ferne Warteschlange oder eine ferne Clusterwarteschlange öffnen, werden die Felder *ResolvedQName* und *ResolvedQMgrName* der MQOD-Struktur mit dem Namen der fernen Warteschlange und dem Namen des fernen Warteschlangenmanagers aus der Definition der fernen Warteschlange bzw. mit dem Namen der angegebenen fernen Clusterwarteschlange gefüllt.

Verwenden Sie die Option MQOO_RESOLVE_LOCAL_Q des MQOPEN-Aufrufs, um das Feld *ResolvedQName* der MQOD-Struktur mit dem Namen der geöffneten lokalen Warteschlange zu füllen. In diesem Fall wird das Feld *ResolvedQMgrName* ebenso mit dem Namen des lokalen Warteschlangenmanagers gefüllt, der die lokale Warteschlange bereitstellt. 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 MQOO_RESOLVE_LOCAL_Q beispielsweise beim Öffnen einer fernen Warteschlange angeben, gibt *ResolvedQName* den Namen der Übertragungswarteschlange an, in die Nachrichten eingereiht werden. *ResolvedQMgrName* gibt den Namen des lokalen Warteschlangenmanagers an, 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 783).

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. Im Feld *ObjectName* der MQOD-Struktur; geben Sie dort den Namen der fernen Warteschlange so ein, wie er dem *lokalen* Warteschlangenmanager bekannt ist.

   Anmerkung: Lassen Sie das Feld *ObjectQMgrName* in diesem Fall leer.
2. Im Feld *ObjectName* der MQOD-Struktur; geben Sie dort den Namen der fernen Warteschlange so ein, wie er dem *fernen* Warteschlangenmanager bekannt ist. Geben Sie dann im Feld *ObjectQMgrName* einen der folgenden Werte ein:

   - 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 so ein, wie er dem *fernen* Warteschlangenmanager bekannt ist.

   Anmerkung: Geben Sie im Feld *ObjectQMgrName* den Namen des fernen Warteschlangenmanagers ein (das Feld darf in diesem Fall 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 100 auf Seite 772.

Objekte mit dem MQCLOSE-Aufruf schließen

Mit dem MQCLOSE-Aufruf können Sie ein Objekt schließen.

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.
- 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 die gleiche Verbindungskennung, die Sie auch zum Öffnen verwendet haben; für CICS-Anwendungen unter z/OS können Sie alternativ auch die Konstante MQHC_DEF_HCONN (mit dem Wert null) angeben.
- 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 782
- „Nachrichten in eine ferne Warteschlange einreihen” auf Seite 787
- „Nachrichteneigenschaften festlegen” auf Seite 788
- „Nachrichtenkontextinformationen steuern” auf Seite 788
- „Einzelnachricht mit dem MQPUT1-Aufruf in eine Warteschlange einreihen” auf Seite 790
- „Verteilerlisten” auf Seite 792
- „Fälle, in denen der PUT-Aufruf fehlschlägt” auf Seite 797

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760

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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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 783
- „Nachrichten mit der MQMD-Struktur definieren” auf Seite 783
- „Optionen in der MQPMO-Struktur angeben” auf Seite 783
- „Die Daten der Nachricht” auf Seite 786
- „Nachrichten einreihen; Nachrichtenkennungen verwenden” auf Seite 787

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 14, 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 792). 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 Plattform steuert darüber hinaus, wie der Aufruf mit Synchronisationspunkten umgeht. Unter z/OS ist der Standardwert für die Synchronisationspunktsteuerung 'yes', auf anderen Plattformen ist er 'no'.

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 45. Informationen zur Verwendung der MQPMO-Struktur zur Steuerung der Kontextinformationen einer Nachricht finden Sie im Abschnitt „Nachrichtenkontextinformationen steuern" auf Seite 788.

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 792). 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 796.

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 796).

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 795).

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 780. 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 789

- „Benutzerkontext übergeben” auf Seite 789
- „Gesamten Kontext übergeben” auf Seite 789
- „Identitätskontext einstellen” auf Seite 790
- „Benutzerkontext einstellen” auf Seite 790
- „Gesamten Kontext einstellen” auf Seite 790

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 45
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 778
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 771). 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 783).

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.

Verteilerlisten

Unter IBM MQ for z/OS nicht unterstützt. Mit Verteilerlisten können Sie eine Nachricht mit einem einzigen MQPUT- oder MQPUT1-Aufruf mehreren Zielen zustellen.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.

V 9.0.0.1 V 9.0.1

Achtung: Verteilerlisten unterstützen nicht die Verwendung von Aliaswarteschlangen, die auf Topic-Objekte verweisen. Ab IBM MQ 9.0.1 und IBM MQ 9.0.0 Fix Pack 1gibt IBM MQ MQRC_ALIAS_BASE_Q_TYPE_ERR0Rzurück, wenn eine Aliaswarteschlange auf ein Themenobjekt in einer Verteilerliste verweist.

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 66 auf Seite 793 zeigt die Funktionsweise von Verteilerlisten.

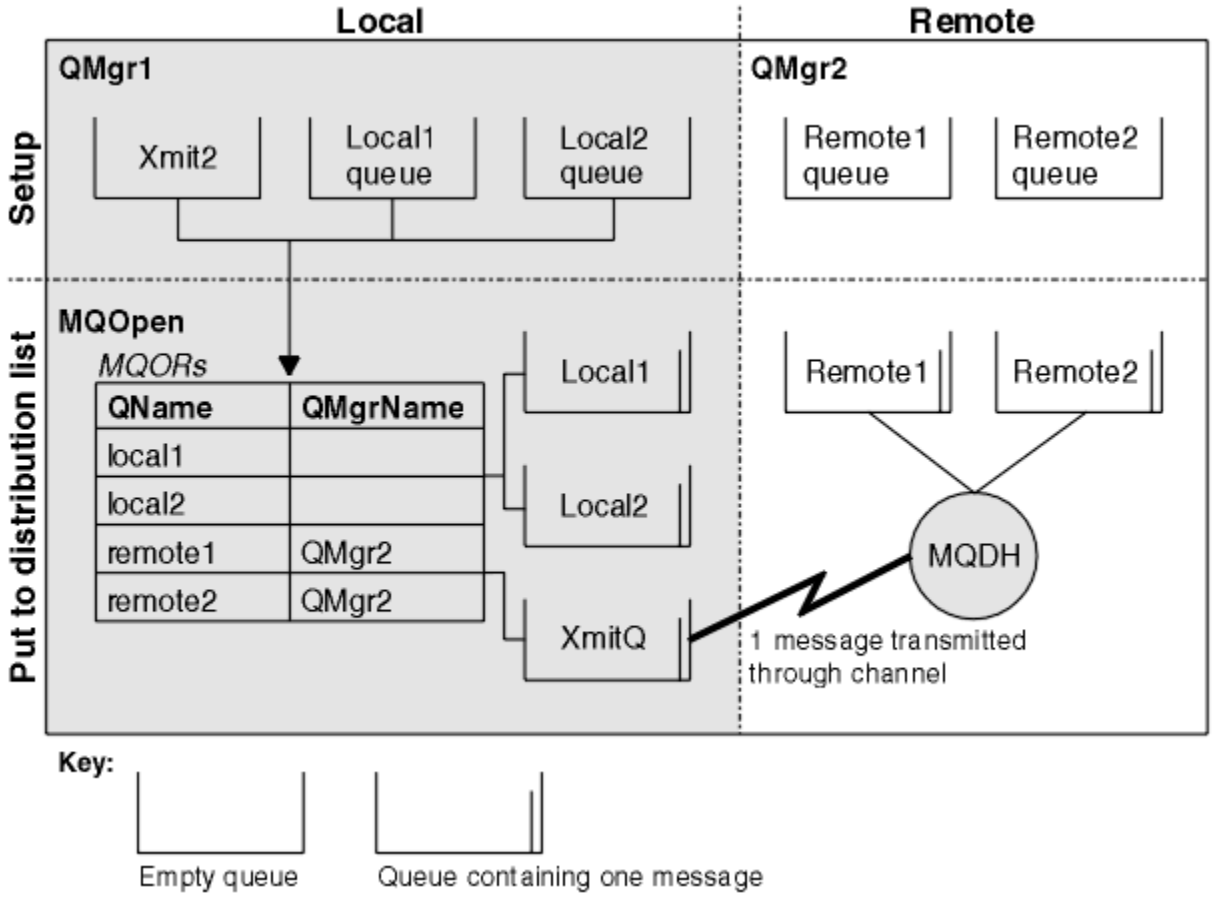


Abbildung 66. So funktionieren Verteilerlisten

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 781)
- 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 67 auf Seite 795 zeigt, wie Sie eine Verteilerliste in C öffnen.

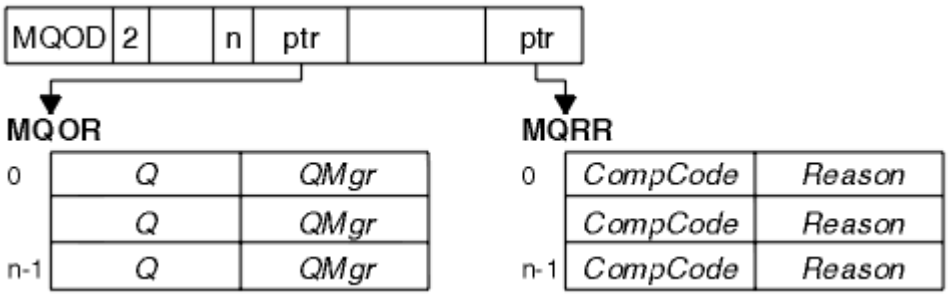


Abbildung 67. Verteilerliste in C öffnen

Abbildung 68 auf Seite 795 zeigt, wie Sie eine Verteilerliste in COBOL öffnen.

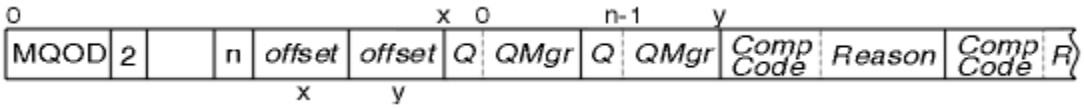


Abbildung 68. Verteilerliste in COBOL öffnen

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 769.

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 781)
- 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 783
- 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 1157), 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 69 auf Seite 796 zeigt, wie Sie in C eine Nachricht in eine Verteilerliste einreihen.

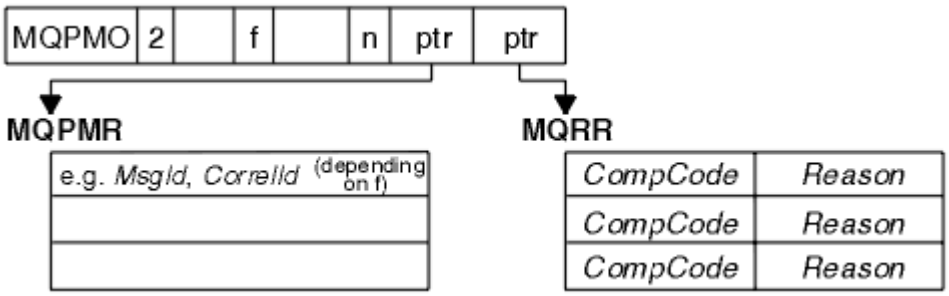


Abbildung 69. Nachricht in C in eine Verteilerliste einreihen

Abbildung 70 auf Seite 796 zeigt, wie Sie in COBOL eine Nachricht in eine Verteilerliste einreihen.

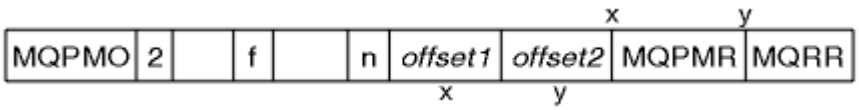


Abbildung 70. Nachricht in COBOL in eine Verteilerliste einreihen

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 760 und „Objekte öffnen und schließen” auf Seite 769.

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 798
- „Abrufreihenfolge der Nachrichten aus einer Warteschlange” auf Seite 803
- „Bestimmte Nachricht abrufen” auf Seite 814
- „Leistung nicht persistenter Nachrichten verbessern” auf Seite 816
- z/OS „Indextyp” auf Seite 820
- „Nachrichten über 4 MB verarbeiten” auf Seite 821
- „Warten auf Nachrichten” auf Seite 827
- z/OS „Signalisierung” auf Seite 828
- „Backout überspringen” auf Seite 829
- „Anwendungsdatenkonvertierung” auf Seite 832
- „Nachrichten in einer Warteschlange durchsuchen (Browsing)” auf Seite 833
- „Fälle, in denen der MQGET-Aufruf fehlschlägt” auf Seite 839

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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 799
- „Nachrichten mit der MQMD-Struktur und dem MQGET-Aufruf beschreiben” auf Seite 799
- „MQGET-Optionen in der MQGMO-Struktur angeben” auf Seite 800
- „Größe des Pufferbereichs angeben” auf Seite 802

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 14, 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 814.

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 803, 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 827)
- 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 829)
- 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 832)
- 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
UNIX and Linux-Systeme	Nein
z/OS z/OS z/OS	Ja
Windows-Systeme	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 der MQGET-Aufruf bei Verwendung der Option MQGMO_WAIT auf den Eingang einer Nachricht in der Warteschlange wartet. Wenn innerhalb der in *WaitInterval* angegebenen Zeit keine Nachricht eintrifft, wird der Aufruf abgeschlossen und gibt einen Ursachencode zurück, der anzeigt, dass keine Nachricht vorhanden war, die Ihren Auswahlkriterien in der Warteschlange entspricht.

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 827 z/OS und „Signalisierung” auf Seite 828 .

Signal1

Signal1 wird nur unter z/OS IBM MQ for z/OSunterstützt.

Wenn Sie die Option MQGMO_SET_SIGNAL verwenden, um anzufordern, dass Ihre Anwendung benachrichtigt wird, wenn eine geeignete Nachricht eingeht, 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 im Abschnitt „Signalisierung” auf Seite 828.

Signal2
Das Feld *Signal2* ist auf allen Plattformen reserviert und sein Wert ohne Bedeutung.

z/OS Weitere Informationen finden Sie im Abschnitt „Signalisierung” auf Seite 828.

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 821.

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 23). 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 814.

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 im Abschnitt „Indextyp" auf Seite 820.

Logische und physische Reihenfolge

Nachrichten in Warteschlangen können (innerhalb jeder Prioritätsstufe) 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 41. 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 71 auf Seite 804 aussehen:

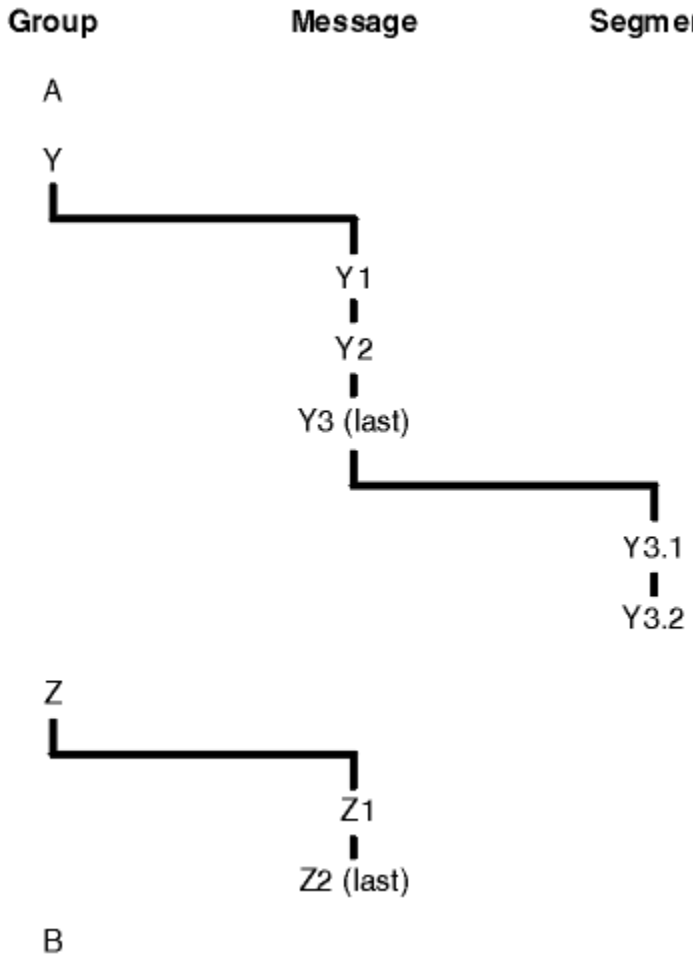


Abbildung 71. Logische Reihenfolge in einer Warteschlange

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 72 auf Seite 805:

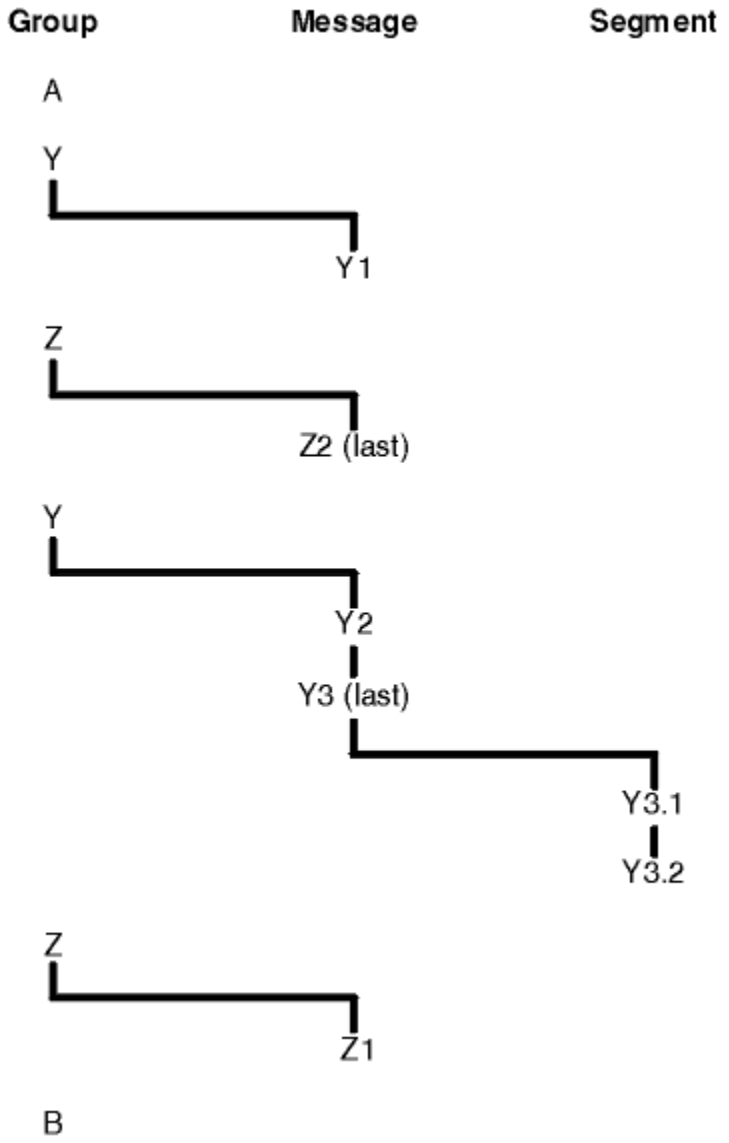


Abbildung 72. Physische Reihenfolge in einer Warteschlange

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: 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. Einreihen der Segmente in jede einzelne logische Nachricht in der Reihenfolge zunehmender Segmentoffsets, beginnend bei 0 und ohne Lücken.
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 72 auf Seite 805 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 102. 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. In der Anwendung muss dafür gesorgt werden, dass die Felder *GroupId*, *MsgSeqNumber*, *Offset* und *MsgFlags* die korrekten Werte enthalten.

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 103 auf Seite 810 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 103. 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 41.

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 824 und „Gruppe in einer Arbeitseinheit einreihen und abrufen” auf Seite 812.

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 Auswahl nach MsgToken. Ebenso können Sie beim Öffnen einer Warteschlange eine Auswahlzeichenfolge verwenden.

Wenn Sie eine bestimmte Nachricht abrufen wollen, verwenden Sie die Felder *MsgId* und *CorrelId* der MQMD-Struktur. 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 104 auf Seite 815 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 104. 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 105 auf Seite 815 zeigt, welche Nachrichten bei den möglichen Einstellungen dieser Felder abgerufen werden.

Tabelle 105. 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 803 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 „Indextyp" auf Seite 820.

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 28.

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:

- Sowohl der Client als auch der ferne Warteschlangenmanager müssen IBM WebSphere MQ 7 oder höher haben.
- 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 eingerichtet 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 819
Wenn Sie AIX 5.3 oder höher verwenden, sollten Sie Ihren Optimierungsparameter so einstellen, dass die vollständige Leistung für nicht persistente Nachrichten verwendet wird.

Zugehörige Tasks

„Vorauslesen aktivieren und inaktivieren” auf Seite 818
Vorauslesen ist standardmäßig inaktiviert. Es kann aber auf Warteschlangen- oder Anwendungsebene aktiviert werden.

Zugehörige Verweise

„MQGET-Optionen und Vorauslesen” auf Seite 817
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:

- Sowohl der Client als auch der ferne Warteschlangenmanager müssen IBM WebSphere MQ 7 oder höher haben.
- 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 106. MQGET-Optionen und Vorauslesen			
	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:

- Sowohl der Client als auch der ferne Warteschlangenmanager müssen IBM WebSphere MQ 7 oder höher haben.
- 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 5.3 oder höher verwenden, sollten Sie Ihren Optimierungsparameter so einstellen, dass die vollständige Leistung für nicht persistente Nachrichten verwendet wird.

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 verwenden, besonders auf Multiprozessormaschinen, empfehlen wir Ihnen zur Leistungsoptimierung und für eine solidere Zeitplanung vor dem Start der Anwendung dringend die Einstellung AIXTHREAD_SCOPE=S in der Datei `.profile` für die ID 'mqm' bzw. die Einstellung AIXTHREAD_SCOPE=S in der Umgebung. Beispiel:

```
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
```

Indextyp

Das Warteschlangenattribut *IndexType* gibt den Indextyp an, den der Warteschlangenmanager verwendet, um die Geschwindigkeit von MQGET-Operationen aus einer Warteschlange zu beschleunigen.

Anmerkung: Nur unter IBM MQ for z/OS unterstützt.

Sie haben fünf Optionen:

Value	Beschreibung
KEINE	Es wird kein Index verwendet. Diese Einstellung empfiehlt sich beim sequenziellen Abruf von Nachrichten (siehe „Priority” auf Seite 803).
GROUPID	Es wird ein Index mit Gruppen-IDs verwendet. Dieser Indextyp muss verwendet werden, wenn die logische Reihenfolge der Nachrichtengruppen erhalten bleiben soll (siehe „Logische und physische Reihenfolge” auf Seite 803).

Value	Beschreibung
MSGID	Es wird ein Index mit Nachrichtenkennungen verwendet. Verwenden Sie diese Option, wenn Sie Nachrichten mit dem Feld *MsgId* als Auswahlkriterium im MQGET-Aufruf abrufen (siehe „Bestimmte Nachricht abrufen" auf Seite 814).
MSGTOKEN	Es wird ein Index mit Nachrichtentoken verwendet.
CORRELID	Es wird ein Index mit Korrelationskennungen verwendet. Verwenden Sie diese Option, wenn Sie Nachrichten mit dem Feld *CorrelId* als Auswahlkriterium im MQGET-Aufruf abrufen (siehe „Bestimmte Nachricht abrufen" auf Seite 814).

Anmerkung:

1. Wenn Sie die Indizierung mit der Option MSGID oder CORRELID vornehmen, setzen Sie im MQMD entsprechend auch den Parameter **MsgId** oder **CorrelId**. Es ist nicht von Vorteil, beide Parameter zu setzen.
2. Beim Durchsuchen von Nachrichten wird der Indizierungsmechanismus verwendet, wenn eine Warteschlange alle der folgenden Bedingungen erfüllt:
   - Sie hat den Indextyp MSGID, CORRELID oder GROUPID
   - Sie wird mit dem gleichen Kennungs- bzw. ID-Typ durchsucht
   - Die darin enthaltenen Nachrichten haben alle die gleiche Priorität
3. Vermeiden Sie Warteschlangen mit Tausenden von Nachrichten, wenn Sie als Indizierung *MsgId* oder *CorrelId* verwenden, da sich dies auf die Neustartzeit auswirkt. (Dies gilt nicht für nicht persistente Nachrichten, da diese beim Neustart gelöscht werden.)
4. MSGTOKEN definiert vom z/OS-Workload-Manager verwaltete Warteschlangen.

Eine vollständige Beschreibung des Attributs **IndexType** finden Sie im Abschnitt IndexType. Darüber hinaus finden Sie auch im Abschnitt „Entwurfs- und Leistungsaspekte für Anwendungen unter z/OS" auf Seite 59 Informationen zum Attribut **IndexType**.

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. Auf allen IBM MQ-Systemen mit Ausnahme von IBM MQ for z/OS haben Sie zudem die folgenden 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.

Diese Attribute können, wenn lange Nachrichten verarbeitet werden müssen, auf anderen Plattformen als z/OS auch unabhängig voneinander geändert werden. Der gültige Bereich des Warteschlangenmanagerattributs liegt zwischen 32768 Byte und 100 MB.

Achtung: Unter IBM MQ for z/OS ist das Attribut **MaxMsgLength** des Warteschlangenmanagers mit 100 MB fest codiert.

Auf allen Plattformen liegt der gültige Bereich des Warteschlangenattributs zwischen 0 und 100 MB.

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. Diese Funktion wird in IBM MQ for z/OS oder von Anwendungen, die IBM MQ classes for JMS verwenden, nicht unterstützt.

Die Erhöhung der maximalen Nachrichtenlänge, wie im Abschnitt „Maximale Nachrichtenlänge heraufsetzen” auf Seite 821 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 41.

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.

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 812 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.

Referenznachrichten
In diesem Abschnitt erhalten Sie Informationen zu Referenznachrichten.

Anmerkung: Unter IBM MQ for z/OS nicht unterstützt.

Mit dieser Methode kann ein großes Objekt zwischen zwei Knoten verschoben werden, ohne dass das Objekt auf dem Quell- oder dem Zielknoten in einer IBM MQ-Warteschlange gespeichert werden muss. Besonders vorteilhaft ist dies, wenn die Daten in einem anderen Format vorliegen, wie dies z. B. bei E-Mail-Anwendungen der Fall ist.

Sie geben hierfür an beiden Enden des Kanals einen Nachrichtenexit an. Weitere Informationen hierzu finden Sie im Abschnitt „Kanalnachrichtenexitprogramme” auf Seite 1029.

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 1125.

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 „Signalisierung" auf Seite 828). Weitere Informationen zu den Bedingungen, unter denen diese Aktionen auftreten, finden Sie im Abschnitt MQGMO.

Signalisierung

Die Signalisierung wird nur unter IBM MQ for z/OS unterstützt.

Signalisierung ist eine Option des MQGET-Aufrufs, die es dem Betriebssystem ermöglicht, einem Programm die Ankunft einer erwarteten Nachricht in einer Warteschlange zu melden (ergo zu *signalisieren*). Dies ist vergleichbar mit der Abruffunktion *MQGET mit Warteintervall*, die im Abschnitt „Warten auf Nachrichten" auf Seite 827 beschrieben wird, weil sie Ihrem Programm erlaubt, mit anderen Aufgaben fortzufahren, während es auf das Signal wartet. Wenn Sie jedoch die Signalisierung verwenden, können Sie den Anwendungsthread freigeben und sich darauf verlassen, dass das Betriebssystem das Programm benachrichtigt, wenn eine Nachricht eintrifft.

Signal einstellen

Wenn Sie ein Signal einstellen möchten, führen Sie folgende MQGMO-Struktur aus, die Sie bei Ihrem MQGET-Aufruf verwenden:

1. Stellen Sie im Feld *Options* die Option MQGMO_SET_SIGNAL ein.
2. Stellen Sie im Feld *WaitInterval* die maximale Lebensdauer für das Signal ein. Damit legen Sie fest, wie lange in Millisekunden IBM MQ die Warteschlange überwachen soll. Verwenden Sie den Wert MQWI_UNLIMITED, um eine unbegrenzte Lebensdauer anzugeben.

   Anmerkung: 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 können IMS-Anwendungen den Status des Ereignissteuerblocks in regelmäßigen Abständen überprüfen (siehe Schritt 3). Ein Programm kann Signale bei mehreren Warteschlangenkennungen gleichzeitig einstellen:
3. Geben Sie im Feld *Signal1* die Adresse des *Ereignissteuerblocks* (ECB) an. Dadurch werden Sie über das Ergebnis Ihres Signals benachrichtigt. Der ECB-Speicher muss verfügbar bleiben, bis die Warteschlange geschlossen wird.

Anmerkung: Sie können die MQGMO_SET_SIGNAL-Option nicht mit der MQGMO_WAIT-Option verwenden.

Wenn die Nachricht eintrifft

Wenn eine geeignete Nachricht eintrifft, wird dem Ereignissteuerblock ein Beendigungscode zurückgegeben.

Der Beendigungscode beschreibt eine der folgenden Bedingungen:

- Die Nachricht, für die Sie das Signal eingestellt haben, ist in der Warteschlange eingetroffen. Die Nachricht ist nicht für das Programm reserviert, dass ein Signal angefordert hat, sodass das Programm erneut einen MQGET-Aufruf ausgegeben hat, um die Nachricht abzurufen.

  Anmerkung: Eine andere Anwendung kann die Nachricht in der Zeit abrufen, die zwischen dem Empfang des Signals und dem Senden eines weiteren MQGET-Aufrufs liegt.
- Das Warteintervall, das Sie eingestellt haben, ist abgelaufen und die Nachricht, für die Sie das Signal festgelegt haben, ist nicht bei der Warteschlange eingegangen. IBM MQ hat das Signal abgebrochen.
- Das Signal wurde abgebrochen. Dies geschieht zum Beispiel, wenn der Warteschlangenmanager gestoppt oder das Attribut der Warteschlange geändert wird, so dass MQGET-Aufrufe nicht mehr zulässig sind.

Wenn eine geeignete Nachricht bereits in der Warteschlange ist, wird der MQGET-Aufruf wie ein MQGET-Aufruf ohne Signalisierung abgeschlossen. Wenn ein Fehler sofort festgestellt wird, wird der Aufruf beendet und die Rückkehrcodes werden eingestellt.

Wenn der Aufruf akzeptiert wird und keine Nachricht sofort verfügbar ist, geht die Steuerung wieder auf das Programm über, damit es mit anderen Aufgaben fortfahren kann. Im Nachrichtendeskriptor ist keines der Ausgabefelder eingestellt, jedoch ist **CompCode** auf MQCC_WARNING und **Reason** auf MQRC_SIGNAL_REQUEST_ACCEPTED gesetzt.

Informationen zu den möglichen Rückgaben von IBM MQ an die Anwendung bei einem MQGET-Aufruf mit Signalisierung finden Sie im Abschnitt MQGET.

Wenn das Programm keine anderen Aufgaben ausführen muss, während es darauf wartet, dass der Ereignissteuerblock übergeben wird, kann es auf den Ereignissteuerblock mit Folgendem warten:

- Bei einem CICS Transaction Server for z/OS-Programm mit dem Befehl EXEC CICS WAIT EXTERNAL
- Bei Stapel- und IMS-Programmen mit dem z/OS WAIT-Makro

Wenn sich der Status der Warteschlange oder des Warteschlangenmanagers ändert, während das Signal eingestellt ist (d. h., dass der Ereignissteuerblock noch nicht übergeben wurde), finden folgende Aktionen statt:

- Wenn der Warteschlangenmanager in den Quiescestatus wechselt und Sie die Option MQGMO_FAIL_IF_QUIESCING verwendet haben, wird das Signal abgebrochen. Der Ereignissteuerblock wird mit dem Beendigungscode MQEC_Q_MGR_QUIESCING übergeben. Ohne diese Option bleibt das Signal eingestellt.
- Wenn der Warteschlangenmanager zum Herunterfahren gezwungen oder abgebrochen wird, erfolgt ein Abbruch des Signals. Das Signal wird mit dem Beendigungscode MQEC_WAIT_CANCELED zugestellt.
- Wenn die Attribute der Warteschlange (oder einer Warteschlange, auf die sich der Warteschlangenname auflöst) geändert werden, so dass die GET-Anforderungen nun unterdrückt werden, erfolgt der Abbruch des Signals. Das Signal wird mit dem Beendigungscode MQEC_WAIT_CANCELED zugestellt.

Anmerkung:

1. Wenn mehrere Programme ein Signal in derselben gemeinsam genutzten Warteschlange zum Entfernen einer Nachricht eingestellt haben, 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. Die Regeln, die der Warteschlangenmanager befolgt, um zu entscheiden, welche Anwendungen aktiviert werden sollen, sind die gleichen wie die für wartende Anwendungen. Weitere Informationen hierzu finden Sie unter der Beschreibung des Felds *Options* der MQGMO-Struktur im Abschnitt MQGMO - Nachrichtenabrufoptionen.
2. Falls mehrere MQGET-Aufrufe, die sowohl Warte- als auch Signaloptionen verwenden, auf dieselbe Nachricht warten, wird jeder wartende Aufruf gleichermaßen berücksichtigt. Weitere Informationen finden Sie unter der Beschreibung des Felds *Options* der MQGMO-Struktur in MQGMO - Nachrichtenabrufoptionen.
3. Unter bestimmten Bedingungen ist es möglich, dass sowohl ein MQGET-Aufruf eine Nachricht abruft als auch dass ein Signal (durch den Eingang derselben Nachricht) zugestellt wird. Das bedeutet, wenn Ihr Programm einen weiteren MQGET-Aufruf ausgibt (weil das Signal zugestellt wurde), ist möglicherweise keine Nachricht verfügbar. Entwickeln Sie Ihr Programm so, dass diese Situation getestet werden kann.

Informationen zur Einstellung eines Signals finden Sie unter der Beschreibung der Option MQGMO_SET_SIGNAL und des *Signal1*-Felds im Abschnitt Signal1.

Backout überspringen

Mit der Option **MQGMO_MARK_SKIP_BACKOUT** des MQGET-Aufrufs verhindern Sie, dass ein Anwendungsprogramm in eine *MQGET-error-backout*-Schleife gerät.

Anmerkung: Nur unter IBM MQ for z/OS unterstützt.

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 gekennzeichnete 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 73 auf Seite 831 zeigt einen typischen Ablauf in einem Anwendungsprogramm, wenn das Backout mit einer MQGET-Anforderung übersprungen werden soll.

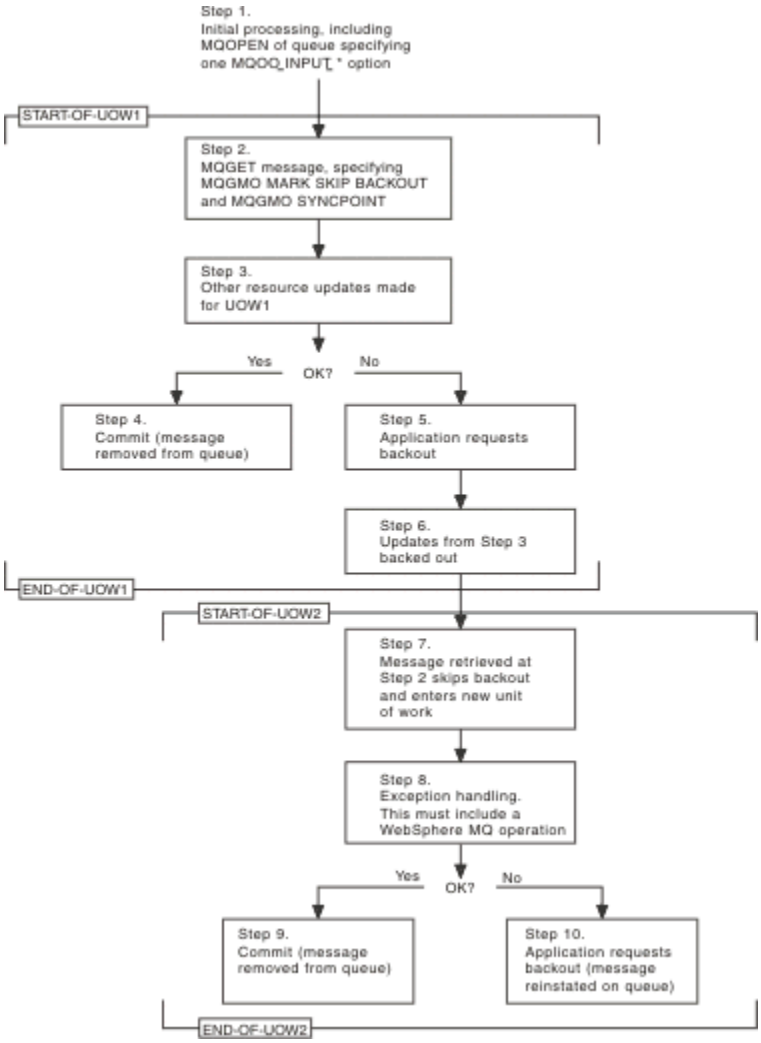


Abbildung 73. Backout mit MQGMO_MARK_SKIP_BACKOUT überspringen

Schritte in Abbildung 73 auf Seite 831:

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 73 auf Seite 831 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 1033. 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 803.)

- „Warteschlangen in FIFO-Reihenfolge (First In First Out)" auf Seite 834
- „Warteschlangen in Prioritätsreihenfolge" auf Seite 834
- „Nicht festgeschriebene Nachrichten" auf Seite 835
- „Änderung an der Warteschlangenreihenfolge" auf Seite 835
- „Verwendung des Warteschlangenindex" auf Seite 835

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.

Verwendung des Warteschlangenindex

Wenn Sie eine indizierte Warteschlange durchsuchen, die nur Nachrichten der gleichen Priorität enthält (persistent oder nicht persistent bzw. beides), verwendet der Warteschlangenmanager bei bestimmten Formen des Browsings den Index.

Anmerkung: Nur unter IBM MQ for z/OS unterstützt.

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.)

Zum Entfernen der Nachricht rufen Sie MQGET erneut auf, geben aber im Feld *Options* der MQGMO-Struktur die Option 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 803 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 836 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 841
- „Subskribentenanwendungen erstellen” auf Seite 848
- „Publish/Subscribe-Lebenszyklen” auf Seite 867
- „Publish/Subscribe-Nachrichteneigenschaften” auf Seite 872
- „Nachrichtenreihenfolge bestimmen” auf Seite 874
- „Veröffentlichungen abfangen” auf Seite 875
- „Veröffentlichungsoptionen” auf Seite 883
- „Subskriptionsoptionen” auf Seite 883

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

„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 48
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 743
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 953
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„Prozedurale Anwendung erstellen” auf Seite 1051
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 1103

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 1124
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

„Web-Services mit IBM MQ entwickeln" auf Seite 1368
Sie können IBM MQ-Anwendungen für Web-Services mithilfe des IBM MQ-Transports für SOAP entwickeln.

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.

Das Schreiben einer einfachen IBM MQ -Publisher-Anwendung entspricht dem Schreiben einer IBM MQ -Punkt-zu-Punkt-Anwendung, die Nachrichten in eine Warteschlange einreiht (Tabelle 107 auf Seite 841). Der Unterschied besteht darin, dass Sie MQPUT-Nachrichten an ein Thema und nicht an eine Warteschlange senden.

Tabelle 107. 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 842), 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 846) 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 848
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 Informationen

TOPIC DEFINI

ANZEIGEN TOPIC

ANZEIGEN TPSTATUS

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 75 auf Seite 843 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 74. 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 75. 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 846 veranschaulicht zudem die gemeinsame Verwendung von Themen und Themenzeichenfolgen.

Zugehörige Konzepte

„Beispiel 2: Veröffentlichungsanwendung für ein variables Thema” auf Seite 846
Ein WebSphere MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Programm definiertes Thema.

„Subskribentenanwendungen erstellen” auf Seite 848
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 77 auf Seite 847 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 76. 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 77. 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.

Hier ein Beispiel: Es gibt die beiden Themenobjekte A und B, wobei A das Thema "a" definiert und B das Thema "a/b" (Abbildung 78 auf Seite 848). Wenn das Veröffentlichungsprogramm das Themenobjekt A referenziert und die Themenzeichenfolge "b" bereitstellt, das Thema also in die Themenzeichenfolge "a/b" aufgelöst wird, übernimmt die Veröffentlichung ihre Eigenschaften aus dem Themenobjekt B, da das Thema mit der für das Themenobjekt B definierten Themenzeichenfolge "a/b" übereinstimmt.

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 77 auf Seite 847 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.

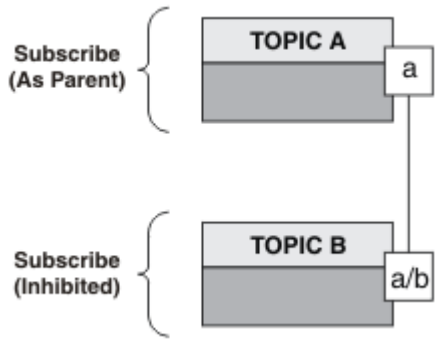


Abbildung 78. Zuordnung der Themenobjekte

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 minimalen 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 842
Ein IBM MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Administrator definiertes Thema.

„Subskribentenanwendungen erstellen” auf Seite 848
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 108 auf Seite 849 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 Subskrip-

tion, 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.
3. Bei der dritten Subskribentenart entscheidet die Subskribentenanwendung, wann die zugrunde liegende Warteschlange, die für Veröffentlichungen verwendet wird, geöffnet und geschlossen wird. Außerdem gibt die Subskribentenanwendung die Subskriptionen aus, durch die die Warteschlange mit Veröffentlichungen gefüllt wird.

Zum besseren Verständnis dieser Subskribentenarten empfiehlt es sich, die in Tabelle 108 auf Seite 849 aufgeführten C-Beispielprogramme zu untersuchen. Die Beispiele wurden passend zu den im Abschnitt „Veröffentlichungsanwendungen erstellen" auf Seite 841 als Beispiele vorgestellten Veröffentlichungsanwendungen entwickelt.

Tabelle 108. Vergleich zwischen Punkt-zu-Punkt- und Subscribe-IBM MQ-Programmmustern

Schritt	**MQ-Nachrichtenkonsument**	**„Beispiel 1: Konsument einer MQ-Veröffentlichung" auf Seite 849**	**„Beispiel 2: Verwalteter MQ-Subskribent" auf Seite 852**	**„Beispiel 3: Nicht verwalteter MQ-Subskribent" auf Seite 857**
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 867 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. Siehe „Beispiel 3: Nicht verwalteter MQ-Subskribent” auf Seite 857

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 80 auf Seite 851 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 79. 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 80. 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 852
Der verwaltete MQ-Subskribent ist das am häufigsten verwendete Subskribentenmuster. Dieses Beispiel setzt *keine* administrative Definition der Warteschlangen, Themen oder Subskriptionen voraus.

„Beispiel 3: Nicht verwalteter MQ-Subskribent" auf Seite 857
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. Dieses Beispiel veranschaulicht verschiedene Arten der Kombination von Subskriptionen und Warteschlangen.

„Veröffentlichungsanwendungen erstellen" auf Seite 841
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. 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 846).

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 Subskriptionswar-

teschlange 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 83 auf Seite 856 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 81. 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 82. 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 83. 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 857 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 849
Der MQ-Veröffentlichungskonsument ist ein IBM MQ-Nachrichtenkonsument, der nicht selbst Themen subskribiert.

„Beispiel 3: Nicht verwalteter MQ-Subskribent” auf Seite 857
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. Dieses Beispiel veranschaulicht verschiedene Arten der Kombination von Subskriptionen und Warteschlangen.

„Veröffentlichungsanwendungen erstellen” auf Seite 841
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. 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. Wenn die Subskription als permanent eingerichtet wird und gemeinsam genutzte Subskriptionen verwendet werden, öffnet jede Anwendung, die die Subskription gemeinsam nutzt, dieselbe verwaltete Warteschlange und ruft die Nachrichten daraus ab.

Eine *verwaltete* Subskription ist eine Subskription, die von IBM MQ verwaltet wird (einschließlich Registrierung und Deregistrierung), während bei einer *nicht verwalteten* Subskription die Anwendung für die Angabe der Warteschlange zuständig ist, in der die Subskriptionen gespeichert werden.

Der Warteschlangenmanager öffnet die permanente verwaltete Subskriptionswarteschlange für einen Subskribenten implizit auf eine Weise, dass eine gemeinsame Verarbeitung der Warteschlange nicht möglich ist. Zudem können Sie pro verwalteter Warteschlange nur eine Subskription erstellen, 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 86 auf Seite 864 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 109 auf Seite 860). Abbildung 92 auf Seite 867 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 85 auf Seite 863 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 109 auf Seite 860 an, die die Ergebnisse zeigt, wenn MQSUB verschiedene Warteschlangenkennungen übergeben werden; sehen Sie sich außerdem die Ergebnisse der Ausführung des Beispielprogramms in Abbildung 87 auf Seite 865 bis Abbildung 92 auf Seite 867 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 109 auf Seite 860wird `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 109 auf Seite 860, 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 109. MQSUB-Fehler bei verschiedenen Warteschlangenkennungen und Subskriptionskombinationen

	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

Tabelle 109. MQSUB-Fehler bei verschiedenen Warteschlangenkennungen und Subskriptionskombinationen (Forts.)

	Warteschlange A **Subscription X** **Warteschlange B** **Keine Subskription**	**Warteschlange A** **Keine Subskription** **Warteschlange B** **Keine Subskription**
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
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 84. 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 85. 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 86. 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 87 auf Seite 865 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 87. Veröffentlichung von 130 unter NYSE/IBM/PRICE

In Abbildung 88 auf Seite 865 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 92 auf Seite 867 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 88. Ständige Veröffentlichung empfangen

In Abbildung 89 auf Seite 866 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 89. Subskription wieder aufnehmen

In Abbildung 90 auf Seite 866 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 90. Ständige Veröffentlichung bei neuer, nicht verwalteter, nicht permanenter Subskription empfangen

Zur Veranschaulichung sich überschneidender Subskriptionen wird in Abbildung 91 auf Seite 866 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 91. Überlappende Subskriptionen

Abbildung 92 auf Seite 867 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 92. Subskriptionsthemen können nicht geändert werden

Zugehörige Konzepte

„Beispiel 1: Konsument einer MQ-Veröffentlichung" auf Seite 849
Der MQ-Veröffentlichungskonsument ist ein IBM MQ-Nachrichtenkonsument, der nicht selbst Themen subskribiert.

„Beispiel 2: Verwalteter MQ-Subskribent" auf Seite 852
Der verwaltete MQ-Subskribent ist das am häufigsten verwendete Subskribentenmuster. Dieses Beispiel setzt *keine* administrative Definition der Warteschlangen, Themen oder Subskriptionen voraus.

„Veröffentlichungsanwendungen erstellen" auf Seite 841
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.[9] 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 93 auf Seite 869 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.

[9] Natürlich muss auch das Veröffentlichungsprogramm die Passagierdatensätze als persistente Nachrichten senden, um anderen möglichen Ausfällen vorzubeugen.

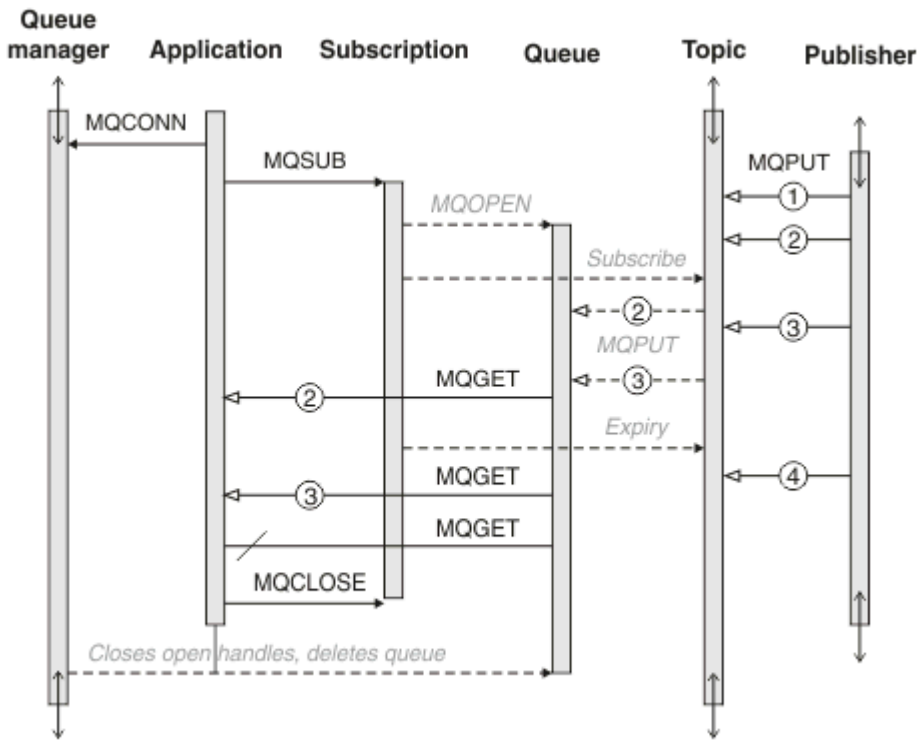


Abbildung 93. Lebenslinien eines verwalteten, nicht permanenten Subskribenten

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.

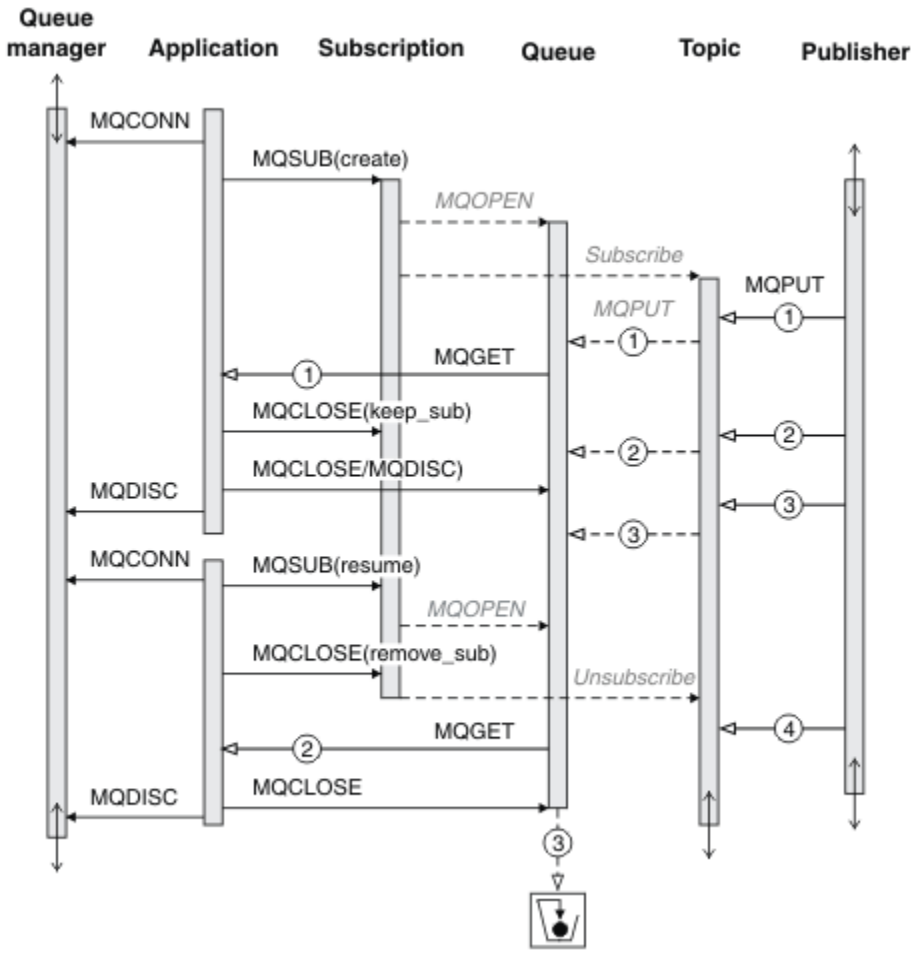


Abbildung 94. Lebenslinien eines verwalteten, permanenten Subskribenten

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.

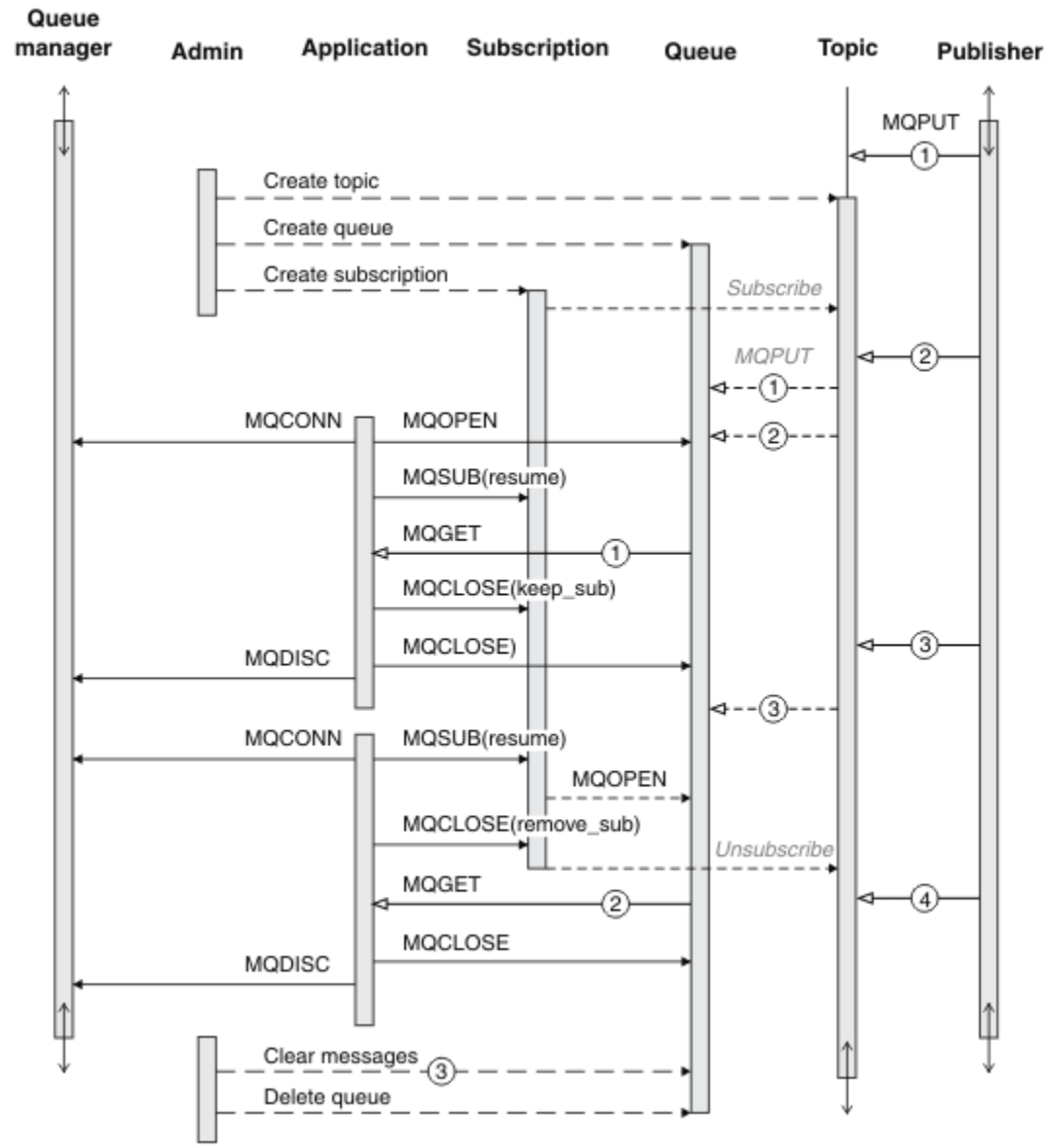


Abbildung 95. Lebenslinien eines nicht verwalteten, permanenten Subskribenten

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 45. 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 45. 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 110 auf Seite 874 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 110. 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 Informationen

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.

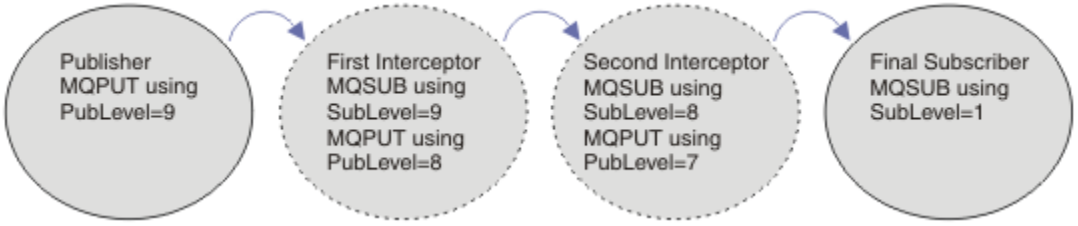


Abbildung 96. Abfolge abfangender Subskribenten

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 111 auf Seite 876 beschriebenen Subskriptionsoptionen.

Tabelle 111. 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 111 auf Seite 876 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 112 auf Seite 877 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 877 an.

Tabelle 112. 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 112 auf Seite 877 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 97 auf Seite 878 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 97. Vorprozessoranweisungen

Abbildung 98 auf Seite 878 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 98. Deklarationen

Initialisierungen, die in den Deklarationen nicht leicht durchführbar sind, werden im Abschnitt Abbildung 99 auf Seite 879 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.

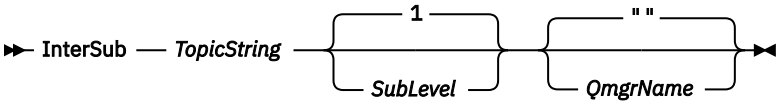
MQPRI_PRIORITY_AS_PUBLISHED
Die Veröffentlichung wird mit ihrer ursprünglichen, bei der Veröffentlichung zugewiesenen Nachrichtenpriorität in die Veröffentlichungswarteschlange eingereiht.

```
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 99. Initialisierungen

Abbildung 100 auf Seite 880 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 100. Abfangen von Veröffentlichungen vorbereiten

Das Hauptcodefragment, Abbildung 101 auf Seite 881 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 101. Veröffentlichung abfangen und erneut veröffentlichen

Das abschließende Codefragment wird im Abschnitt Abbildung 102 auf Seite 881 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 102. 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 103 auf Seite 882 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.

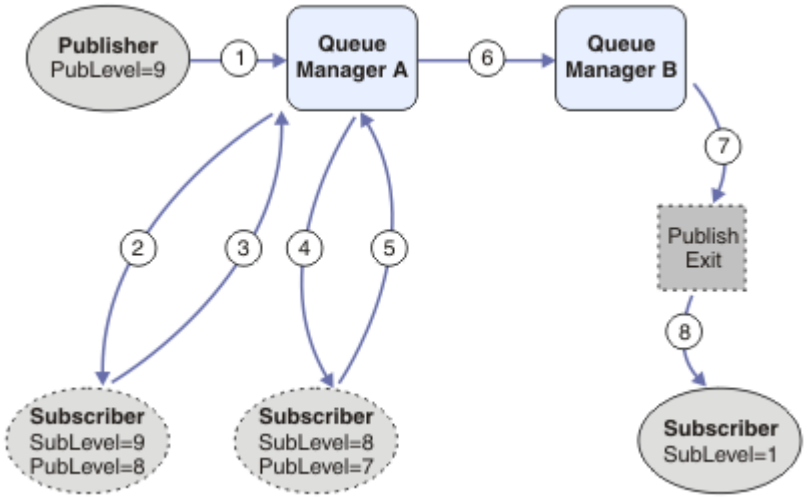


Abbildung 103. Abfangende Subskribenten und Publish-Exit in einem Cluster

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 104 auf Seite 883 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 104 auf Seite 883, dass die Veröffentlichung nicht vom Subskribenten auf `SubLevel` 9 und dem Warteschlangenmanager B abgefangen wird.

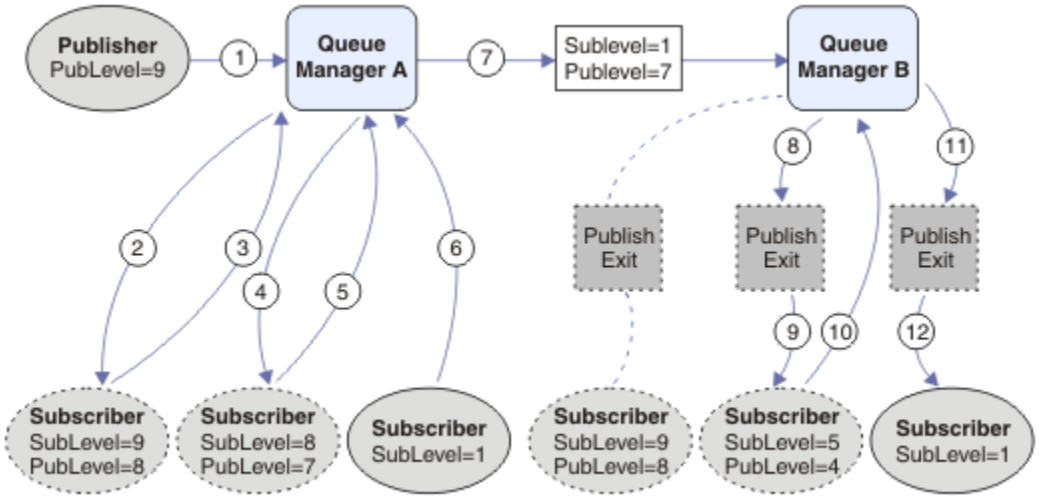


Abbildung 104. Komplexe Implementierung abfangender Subskribenten

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
Permanent

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.

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:

- ULW Die MQSC-Funktion (siehe MQSC-Referenz)
- 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 113. 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 760 und „Objekte öffnen und schließen” auf Seite 769.

Unter den folgenden Links erhalten Sie weitere Informationen zum Abfragen und Einstellen von Objektattributen:

- „Abfragen der Attribute eines Objekts” auf Seite 887
- „Fälle, in denen der MQINQ-Aufruf fehlschlägt” auf Seite 888
- „Warteschlangenattribute einstellen” auf Seite 888

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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** festgelegt. 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 mögli-

chen 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 923 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)
- *DistList* (nicht unter z/OS)
- *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:

- Commit
- 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 891 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 891
- z/OS „Synchronisationspunkte in IBM MQ for z/OS-Anwendungen” auf Seite 892
- IBM i „Synchronisationspunkte in CICS for IBM i-Anwendungen” auf Seite 895
- „Synchronisationspunkte in IBM MQ for Multiplatforms” auf Seite 895
- IBM i „Schnittstellen mit dem externen Synchronisationspunktmanager von IBM i” auf Seite 900

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„IBM MQ-Anwendungen durch Auslöser starten” auf Seite 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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:

-

IBM MQ for AIX
-

IBM MQ for IBM i
-

IBM MQ for HP-UX
-

IBM MQ for Linux
-

IBM MQ for Solaris
-

IBM MQ for Windows
-

CICS Transaction Server for z/OS
-

TXSeries
-

IMS/ESA
-

z/OS-Stapel mit RRS
- Andere externe Koordinatoren, die die X/Open XA-Schnittstelle verwenden

Das einphasige Commit wird in den folgenden Umgebungen unterstützt:

-

IBM MQ for IBM i
-

IBM MQ on UNIX
-

IBM MQ for Windows
-

z/OS-Stapel

Ausführliche Informationen zu externen Schnittstellen finden Sie im Abschnitt „Schnittstellen zu externen Synchronisationspunktmanagern auf Multiplatforms” auf Seite 898 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 1124.

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:

-

Der Standardwert für die Synchronisationspunktsteuerung ist NO.
-

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 zu-

rückgegeben 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. Weitere Informationen zu *BackoutCount* finden Sie im Abschnitt BackoutCount.

Wenn ein Programm (ausgenommen bei einem z/OS-Stapel mit RRS) einen MQDISC-Aufruf ausgibt und einige Anforderungen noch nicht festgeschrieben sind, erfolgt ein impliziter Synchronisationspunkt. 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 finden Sie im Abschnitt Einführung in die Behandlung von Bedingungen in Language Environment.

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 829.

Änderungen an den Warteschlangenattributen (entweder durch den MQSET-Aufruf oder durch Befehle) werden nicht durch Festschreiben oder Zurücksetzen von Arbeitseinheiten beeinträchtigt.

z/OS *Synchronisationspunkte in IBM MQ for z/OS-Anwendungen*

In diesem Abschnitt wird die Verwendung von Synchronisationspunkten in Transaktionsmanageranwendungen (CICS und IMS) und Stapelanwendungen erläutert.

z/OS *Synchronisationspunkte in CICS Transaction Server for z/OS-Anwendungen*

In einer CICS-Anwendung richten Sie einen Synchronisationspunkt mit dem Befehl EXEC CICS SYNCPOINT ein.

Wenn Sie die Änderungen bis zum vorherigen Synchronisationspunkt zurücksetzen möchten, verwenden Sie den Befehl EXEC CICS SYNCPOINT ROLLBACK. Weitere Informationen hierzu finden Sie im Referenzhandbuch *CICS Application Programming Reference*.

Falls auch andere wiederherstellbare Ressourcen an der Arbeitseinheit beteiligt sind, wirkt der Warteschlangenmanager in Verbindung mit dem CICS-Synchronisationspunktmanager an einem zweiphasigen Commit mit. Andernfalls führt er ein einphasiges Commit aus.

Wenn eine CICS-Anwendung einen MQDISC-Aufruf ausgibt, wird kein impliziter Synchronisationspunkt eingerichtet. Bei einer normalen Beendigung der Anwendung werden alle offenen Warteschlangen geschlossen und ein implizites Commit wird ausgeführt. Bei einer abnormalen Beendigung der Anwendung werden alle offenen Warteschlangen geschlossen und ein implizites Backout wird ausgeführt.

z/OS *Synchronisationspunkte in IMS-Anwendungen*

Sie erstellen einen Synchronisationspunkt in einer IMS-Anwendung mithilfe von IMS-Aufrufen wie beispielsweise GU (Get Unique) an den primären E/A-Steuerblock (IOPCB) und CHKP (Prüfpunkt).

Wenn Sie alle Änderungen seit dem letzten Prüfpunkt zurücksetzen möchten, verwenden Sie den IMS-Aufruf ROLB (Rollback). Weitere Informationen finden Sie in der Dokumentation zu IMS.

Falls auch andere wiederherstellbare Ressourcen an der Arbeitseinheit beteiligt sind, wirkt der Warteschlangenmanager in Verbindung mit dem IMS-Synchronisationspunktmanager an einem zweiphasigen Commit mit.

Alle offenen Kennungen werden vom IMS-Adapter an einem Synchronisationspunkt geschlossen (außer in einer Stapel- oder nicht nachrichtengesteuerten BMP-Umgebung). Grund hierfür ist, dass ein anderer Benutzer die nächste Arbeitseinheit initiieren könnte und die Sicherheitsprüfung durch IBM MQ bei einem MQCONN-, MQCONNX- oder MQOPEN-Aufruf durchgeführt wird und nicht bei einem MQPUT- oder MQGET-Aufruf.

In einer Wait-for-Input-Umgebung (WFI) oder einer Pseudo-Wait-for-Input-Umgebung (PWFI) wird IBM MQ jedoch erst dann von IMS angewiesen, die Kennungen zu schließen, wenn die nächste Nachricht eintrifft oder ein QC-Statuscode an die Anwendung zurückgegeben wird. Wenn die Anwendung im IMS-Bereich wartet und solche Kennungen zu ausgelösten Warteschlangen gehören, findet keine Auslösung statt, da die Warteschlangen bereits offen sind. Aus diesem Grund sollten Anwendungen in einer WFI- oder PWFI-Umgebung die Warteschlangenkennungen explizit mit MQCLOSE schließen, bevor sie für die nächste Nachricht ein GU an IOPCB ausgeben.

Wenn eine IMS-Anwendung (BMP oder MPP) einen MQDISC-Aufruf ausgibt, werden offene Warteschlangen geschlossen, jedoch kein impliziter Synchronisationspunkt eingerichtet. Bei einer normalen Beendigung der Anwendung werden alle offenen Warteschlangen geschlossen und ein implizites Commit wird ausgeführt. Bei einer abnormalen Beendigung der Anwendung werden alle offenen Warteschlangen geschlossen und ein implizites Backout wird ausgeführt.

z/OS *Synchronisationspunkte in z/OS-Stapelanwendungen*

Bei Stapelanwendungen können Sie die IBM MQ-Aufrufe MQCMIT und MQBACK für das Synchronisationspunktmanagement verwenden. Aus Kompatibilitätsgründen mit früheren Versionen stehen auch die Aufrufe CSQBCMT und CSQBBAK als Synonyme zur Verfügung.

Anmerkung: Wenn Sie innerhalb einer Arbeitseinheit Änderungen an Ressourcen festschreiben oder zurücksetzen müssen, die von verschiedenen Ressourcenmanagern wie IBM MQ und Db2 verwaltet werden, können Sie RRS verwenden. Weitere Informationen hierzu finden Sie im Abschnitt „Transaktionsmanagement und Recoverable Resource Manager Services (RRS)” auf Seite 894.

Änderungen mit einem MQCMIT-Aufruf festschreiben

Als Eingabe ist die Verbindungskennung (*Hconn*) erforderlich, die durch den MQCONN- oder MQCONNX-Aufruf zurückgegeben wird.

Die Ausgabe von MQCMIT ist ein Beendigungscode und ein Ursachencode. Der Aufruf wird mit einer Warnung beendet, wenn der Synchronisationspunkt ausgeführt wurde, aber der Warteschlangenmanager die Put- und Get-Operationen seit dem letzten Synchronisationspunkt zurückgesetzt hat.

Eine erfolgreiche Ausführung des MQCMIT-Aufrufs zeigt dem Warteschlangenmanager, dass die Anwendung einen Synchronisationspunkt erreicht hat und dass alle Put- und Get-Operationen seit dem letzten Synchronisationspunkt festgeschrieben wurden.

Nicht alle Fehlerantworten bedeuten, dass das MQCMIT nicht ausgeführt wurde. Die Anwendung kann zum Beispiel auch den Fehler MQRC_CONNECTION_BROKEN erhalten.

Eine genaue Beschreibung des MQCMIT-Aufrufs finden Sie im Abschnitt MQCMIT.

Änderungen mit einem MQBACK-Aufruf zurücksetzen

Als Eingabe müssen Sie eine Verbindungskennung (*Hconn*) angeben. Verwenden Sie die Kennung, die vom MQCONN-oder MQCONNX-Aufruf zurückgegeben wird.

Die Ausgabe von MQBACK ist ein Beendigungscode und ein Ursachencode.

Die Ausgabe zeigt dem Warteschlangenmanager, dass die Anwendung einen Synchronisationspunkt erreicht hat und dass alle Put- und Get-Operationen seit dem letzten Synchronisationspunkt zurückgesetzt wurden.

Eine genaue Beschreibung des MQBACK-Aufrufs finden Sie im Abschnitt MQBACK.

Transaktionsmanagement und Recoverable Resource Manager Services (RRS)

Transaktionsmanagement und Recoverable Resource Manager Services (RRS) ist eine z/OS-Funktion zur Unterstützung zweiphasiger Synchronisationspunkte zwischen den beteiligten Ressourcenmanagern.

Eine Anwendung kann die von verschiedenen z/OS-Ressourcenmanagern (wie IBM MQ und Db2) verwalteten wiederherstellbaren Ressourcen aktualisieren und diese Änderungen dann in einer Arbeitseinheit festschreiben oder zurücksetzen. RRS stellt für die Arbeitseinheiten während des normalen Betriebs die erforderliche Statusprotokollierung bereit, koordiniert die Synchronisationspunktverarbeitung und stellt während des Neustarts von Subsystemen die entsprechenden Statusinformationen für die Arbeitseinheiten bereit.

Die IBM MQ for z/OS RRS-Teilnehmerunterstützung ermöglicht IBM MQ-Anwendungen in Umgebungen mit gespeicherten Stapel-, TSO- und Db2-Prozeduren sowohl die Aktualisierung von IBM MQ- als auch die Aktualisierung von Nicht-IBM MQ-Ressourcen (z. B. Db2) innerhalb einer einzigen logischen Arbeitseinheit. Informationen zur RRS-Teilnehmerunterstützung finden Sie unter *z/OS MVS Programming: Resource Recovery*.

Ihre IBM MQ-Anwendung kann MQCMIT und MQBACK oder die funktional entsprechenden RRS-Aufrufe SRRCMIT und SRRBACK verwenden. Weitere Informationen finden Sie im Abschnitt „RRS-Batch-Adapter” auf Seite 930.

Verfügbarkeit von RRS

Wenn auf Ihrem z/OS-System kein RRS verfügbar ist, gibt jeder IBM MQ-Aufruf von einem Programm, das mit einem RRS-Stub (CSQBRSTB oder CSQBRRSI) verknüpft ist, den Fehler MQRC_ENVIRONMENT_ERROR zurück.

Gespeicherte Db2-Prozeduren

Wenn Sie gespeicherte Db2-Prozeduren mit RRS verwenden, berücksichtigen Sie Folgendes:

- Gespeicherte Db2-Prozeduren, die RSS verwenden, müssen WLM-verwaltet sein (d. h. durch einen Workload-Manager verwaltet werden).
- Wenn eine Db2-verwaltete gespeicherte Prozedur IBM MQ-Aufrufe enthält und mit einem RRS-Stub (CSQBRSTB oder CSQBRRSI) verknüpft ist, gibt der MQCONN- oder MQCONNX-Aufruf den Fehler MQRC_ENVIRONMENT_ERROR zurück.
- Wenn eine WLM-verwaltete gespeicherte Prozedur IBM MQ-Aufrufe enthält und mit einem Nicht-RRS-Stub verknüpft ist, gibt der MQCONN- oder MQCONNX-Aufruf den Fehler MQRC_ENVIRONMENT_ERROR zurück, es sei denn, es handelt sich um den ersten IBM MQ-Aufruf seit dem Start des Adressraums der gespeicherten Prozedur.
- Wenn Ihre gespeicherte Db2-Prozedur IBM MQ-Aufrufe enthält und mit einem Nicht-RRS-Stub verknüpft ist, werden die mit dieser gespeicherten Prozedur aktualisierten IBM MQ-Ressourcen erst festgeschrieben, wenn der Adressraum der gespeicherten Prozedur endet oder eine nachfolgende gespeicherte Prozedur einen MQCMIT-Aufruf (mit einem IBM MQ-Stapel-/TSO-Stub) ausführt.
- Mehrere Kopien der gleichen gespeicherten Prozedur können parallel im gleichen Adressraum ausgeführt werden. Wenn Sie möchten, dass Db2 nur eine Kopie Ihrer gespeicherten Prozedur verwendet, müssen Sie sicherstellen, dass Ihr Programm auf eine wiedereintrittsfähige Weise codiert ist. Andernfalls erhalten Sie vermutlich für jeden IBM MQ-Aufruf Ihres Programms den Fehler MQRC_HCONN_ERROR.
- MQCMIT und MQBACK dürfen nicht in einer WLM-verwalteten gespeicherten Db2-Prozedur codiert werden.
- Entwickeln Sie alle Programme so, dass sie in der Language Environment (LE) ausgeführt werden.

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. Beispiel:

- 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. Beispiel:

- TXSeries
- Universal Database
- IBM i Commit-Controller von IBM i

ULW 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

V 9.0.5

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.

Globale Arbeitseinheiten unter Multiplatforms
Verwenden Sie globale Arbeitseinheiten, wenn Sie auch Änderungen an Ressourcen einbeziehen müssen, die zu anderen Ressourcenmanagern gehören.

Hier kann die Koordination innerhalb oder außerhalb des Warteschlangenmanagers erfolgen:

Interne Synchronisationspunktkoordination

Die Koordination globaler Arbeitseinheiten durch den Warteschlangenmanager wird unter IBM MQ for IBM i und IBM MQ for z/OS nicht unterstützt. Auch in einer IBM MQ MQI client-Umgebung wird diese 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 unter UNIX and Linux und IBM MQ for Windows ihr Interesse am Ergebnis der Arbeitseinheit bei dem externen Synchronisationspunktkoordinator, so dass sie nicht festgeschriebene Put- oder Get-Operationen nach Bedarf festschreiben oder zurücksetzen 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.

Auch wenn Sie sich für die Verwendung der XA-Funktionen von CICS entschieden haben, können Sie in CICS-Programmen die API-Aufrufe von IBM MQ verwenden. 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*.

V 9.0.5 Multi *Impliziter Synchronisationspunkt unter Multiplatforms*

IBM MQ 9.0.5 fügt einen impliziten Synchronisationspunkt für persistente Nachrichten hinzu, die außerhalb des Synchronisationspunkts eingereiht werden.

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`. Der Ressourcenmanager für IBM MQ unter UNIX, Linux, and Windows heißt 'MQSeries_XA_RMI'. IBM i Für IBM ilautet der Name des Ressourcenmanagers 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 114. 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 766).
2. IBM i IBM MQ for IBM i unterstützt keine dynamische Registrierung von XA-Ressourcenmanagern.

   Als Transaktionsmanager wird nur WebSphere Application Server unterstützt.
3. 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. Dies muss als *JOB für IBM MQ for IBM idefiniert werden. Beispiel:

```
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. Beispiel:

- 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 Vor-

handensein 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, die einer Warteschlange zugeordnet ist, wird durch das Attribut `ProcessName` der lokalen Warteschlange angegeben. 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 UNIX, 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.

Für eine Übertragungswarteschlange auf einer beliebigen Plattform außer Linuxkann das Attribut `TriggerData` der Übertragungswarteschlange den Namen des zu startenden Kanals angeben. 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 908.

IBM MQ bietet verschiedene Optionen zur Steuerung der Bedingungen, die ein Auslöserereignis bewirken (siehe „Auslöserereignisse steuern” auf Seite 913).

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 920).

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 „Auslösemonitore" auf Seite 916.

Sehen Sie sich zum besseren Verständnis des Auslösemechanismus auch das Beispiel im Abschnitt Abbildung 105 auf Seite 904 an, das den Auslösertyp FIRST (MQTT_FIRST) veranschaulicht.

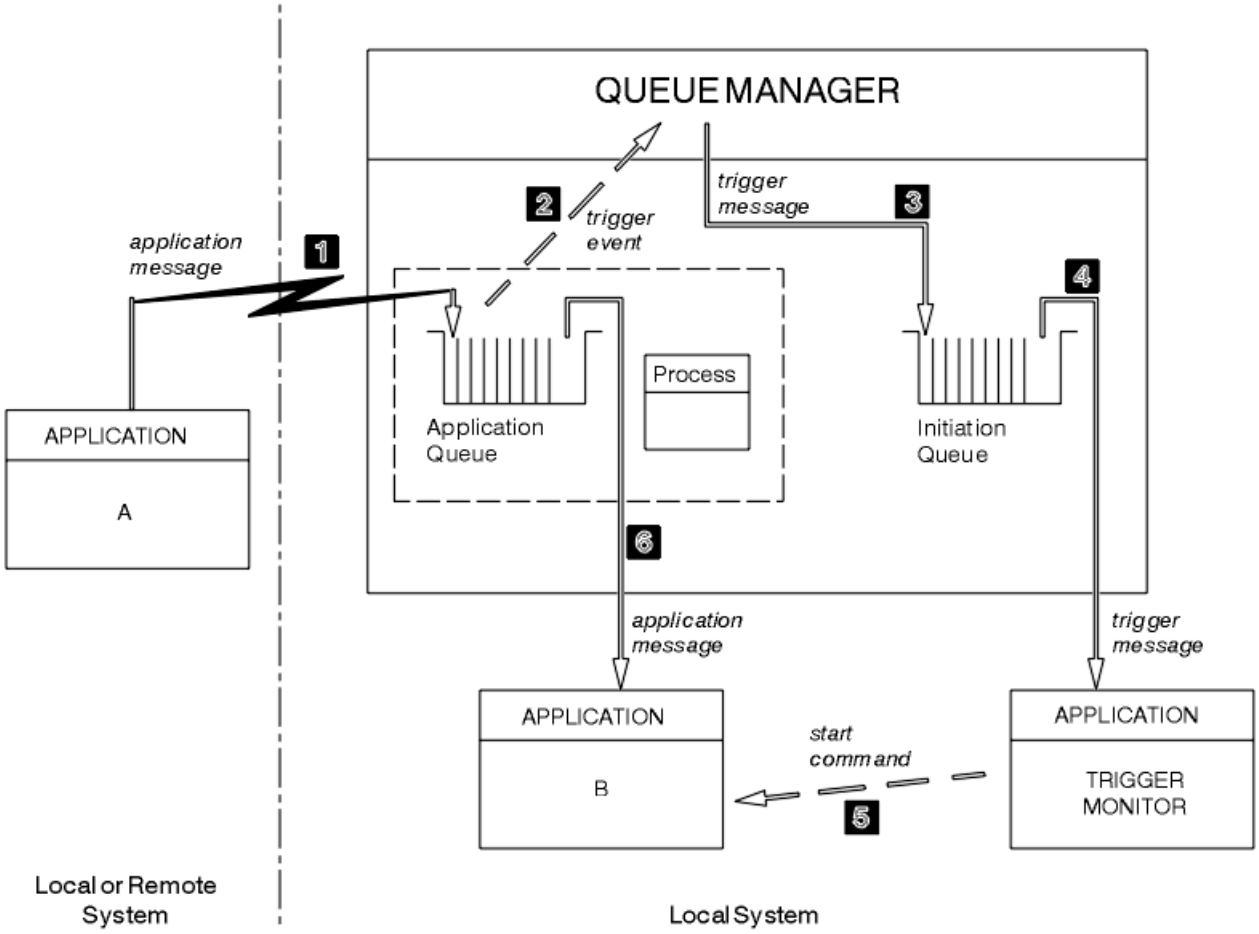


Abbildung 105. Ablauf von Anwendungs- und Auslösenachrichten

In Abbildung 105 auf Seite 904 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 106 auf Seite 905.

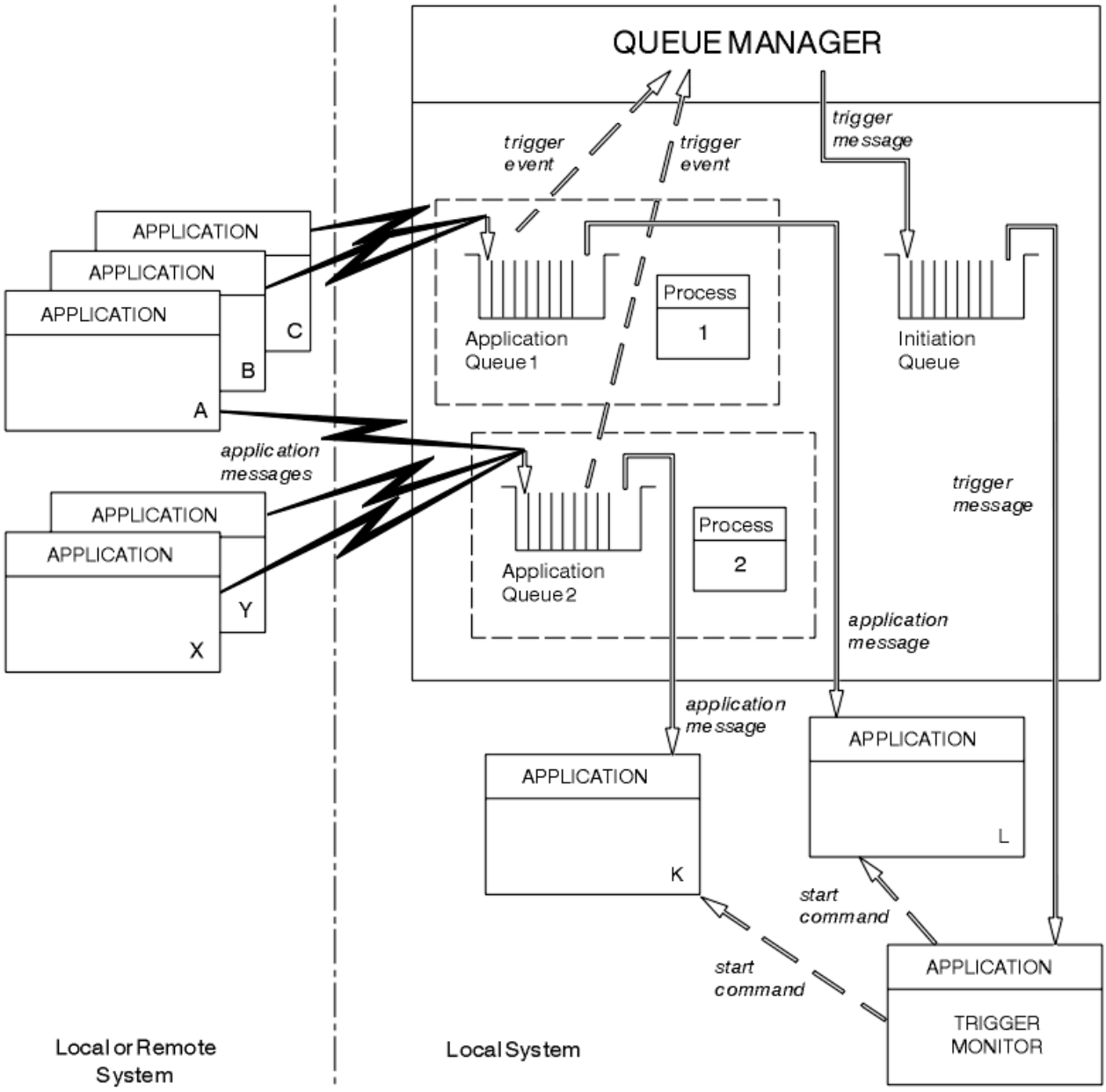


Abbildung 106. Warteschlangenbeziehungen im Zusammenhang mit der Auslösefunktion

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 906
- „Bedingungen für ein Auslöserereignis” auf Seite 908
- „Auslöserereignisse steuern” auf Seite 913
- „Anwendung entwickeln, die ausgelöste Warteschlangen verwendet” auf Seite 915
- „Auslösemonitore” auf Seite 916
- „Eigenschaften von Auslösenachrichten” auf Seite 920
- „Wenn die Auslösung nicht funktioniert” auf Seite 921

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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. Beispiel:

```
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
Systeme mit UNIX, Linux, and Windows	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 907) 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 907 enthält.

4. Optional: Erstellen Sie die Definition einer Übertragungswarteschlange, wobei Sie für das Attribut **ProcessName** Leerzeichen verwenden.

   Das Attribut **TrigData** kann den Namen des auszulösenden Kanals enthalten oder leer bleiben. Wenn es - außer auf IBM MQ for z/OS - leer bleibt, sucht der Kanalinitiator die Kanaldefinitionsdateien, bis er einen Kanal findet, der zur benannten Übertragungswarteschlange gehört. 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
Systeme mit UNIX, Linux, and Windows	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 „Auslösemonitore" auf Seite 916.

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 913.

Bedingungen für ein Auslöserereignis

Der Warteschlangenmanager erzeugt eine Auslösenachricht, wenn die in diesem Abschnitt beschriebenen Bedingungen erfüllt sind.

Wenn in diesem Thema von gemeinsam genutzten Warteschlangen gesprochen wird, sind gemeinsam genutzte Warteschlangen in einer Gruppe mit gemeinsamer Warteschlange gemeint, die nur unter IBM MQ for z/OS zur Verfügung stehen.

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:

   a. 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.

   b. 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:

   a. 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.

   b. 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. 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 909 bis „10" auf Seite 910 (mit Ausnahme von „3" auf Seite 909) 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 909), und die Bedingungen „6" auf Seite 910 bis „10" auf Seite 910 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 44.

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 909), und die Bedingungen „4" auf Seite 909 bis „10" auf Seite 910 (mit Ausnahme von „8" auf Seite 910) 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 909), und die Bedingungen „4" auf Seite 909 bis „10" auf Seite 910 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 909), und die Bedingungen „4" auf Seite 909 bis „10" auf Seite 910, mit Ausnahme von „5" auf Seite 910, 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 909), und die Bedingungen „4" auf Seite 909 bis „10" auf Seite 910 (mit Ausnahme von „8" auf Seite 910) 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 913.

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.

Anmerkung:

1. Ab Schritt „12" auf Seite 911 (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.

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 914.

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 zurückgesetzt 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 908 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.
- 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.)
- Wenn *TRIGINT* unter IBM MQ for z/OS sehr niedrig eingestellt ist und keine Anwendung die Anwendungswarteschlange mit dem Auslösertyp FIRST bedient, generiert der Backstop-Triggering-Prozess bei jedem der regelmäßigen Scans einer Anwendungswarteschlange, die sich auf eine offene Initialisierungswarteschlange bezieht, 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 Warteschlangenmanager 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.

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.

Auslösemonitore auf UNIX- und Windows-Systemen

In diesem Abschnitt erhalten Sie Informationen zu den auf UNIX- und 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 1125.

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.

Um die gestartete Anwendung im Hintergrund auf Windows -Systemen auszuführen, stellen Sie dem Namen Ihrer Anwendung im Feld *ApplId* den Befehl START voran. Beispiel:

```
START ?B AMQSECHA
```

UNIX Um die gestartete Anwendung im Hintergrund unter UNIXauszuführen, fügen Sie ein & am Ende der *EnvData* der PROCESS-Definition ein.

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.

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 Artikel gilt nur für Windows, UNIX und Linux x86-64.

Der Auslösemonitor wird als CICS-Programm bereitgestellt. Sie müssen ihn mit einem vierstelligen Transaktionsnamen definieren. Zum Starten geben Sie einen vierstelligen Namen ein. Der Monitor verwendet den Standardwarteschlangenmanager (wie in der Datei qm.ini bzw. unter IBM MQ for Windows in der Registrierung festgelegt) sowie die Initialisierungswarteschlange 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.

Nachrichten werden aus der Initialisierungswarteschlange gelesen und zum Starten von CICS -Transaktionen mit EXEC CICS START verwendet, vorausgesetzt, der APPL_TYPE in der Auslösenachricht lautet MQAT_CICS. Das Lesen von Nachrichten aus der Initialisierungswarteschlange erfolgt unter CICS -Synchronisationspunktsteuerung.

Beim Starten und Stoppen des Monitors sowie bei Fehlern werden Nachrichten generiert. Diese Nachrichten werden an die CSMT-Warteschlange mit transienten Daten gesendet.

Der Auslösemonitor ist in den folgenden Versionen verfügbar:

Version	Verwenden Sie
amqltmc0	TXSeries 5.1 für AIX, HP-UX, Linux x86-64 -Systeme und Oracle Solaris
amqltmc4	TXSeries 5.1 fürWindows
amqltmcc	Client-gebundene Version des CICS-Auslösemonitors

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.

   IBM i Verwenden Sie DEFINE QLOCAL oder ALTER QLOCAL. 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 920
- „Neustart des Warteschlangenmanagers und Auslösenachrichten” auf Seite 920
- „Auslösenachrichten und Änderungen bei Objektattributen” auf Seite 920
- „Format von Auslösenachrichten” auf Seite 920

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. Dieses Feld kann in jedem IBM MQ-Produkt mit Ausnahme von IBM MQ for z/OS zur Angabe des Namens des Kanals verwendet werden, der ausgelöst werden soll.

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.

Bei Verwendung des mit IBM MQ for z/OS bereitgestellten Auslösemonitors CSQQTRMN ist das Attribut **Anwendungs-ID** 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. Von dem mit CICS bereitgestellten Auslösemonitor CKTI oder dem mit IBM MQ for z/OS bereitgestellten Auslösemonitor CSQQTRMN wird dieses Feld nicht verwendet, andere Auslösemonitore verwenden es jedoch unter Umständen.

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

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 923
- „MQPUT, MQPUT1 und Cluster” auf Seite 924
- „MQINQ und Cluster” auf Seite 924
- „MQSET und Cluster” auf Seite 925
- „Rückkehrcodes” auf Seite 925

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit 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 MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED oder MQBND_BIND_ON_GROUP.Bei der Verwendung von Gruppen mit Clustern muss entweder MQBND_BIND_ON_OPEN oder MQBND_BIND_ON_GROUP 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 776
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.

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')

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 MQOPEN-, MQPUT-, MQPUT1- oder MQSUB-Aufruf für ein Themenobjekt mit der Angabe PUBSCOPE (ALL) oder SUBSCOPE (ALL) wurde ausgegeben. Die Clusterthemendefinition kann nicht korrekt aufgelöst werden, da vom Warteschlangenmanager mit vollständigem Repository eine Antwort benötigt wird, die nicht eintrifft.

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 Anwendungen unter IBM MQ for z/OS verwenden und schreiben

IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

Dieser Abschnitt bietet einen Überblick über die IBM MQ-Funktionen, die den in den unterstützten Umgebungen ausgeführten Programmen zur Verfügung stehen. Folgende Informationen könnten an dieser Stelle ebenso interessant sein:

- Informationen zur Verwendung der IBM MQ-CICS bridge finden Sie im Abschnitt IBM MQ mit CICS verwenden .
- Informationen zur Verwendung von IMS und der IMS-Bridge (siehe „IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63).

Unter den folgenden Links erhalten Sie weitere Informationen zur Verwendung und Entwicklung von Anwendungen unter IBM MQ for z/OS:

- „Umgebungsabhängige IBM MQ for z/OS-Funktionen” auf Seite 928
- „Debugfunktionen und Unterstützung für Synchronisationspunkte und Wiederherstellung” auf Seite 929
- „IBM MQ for z/OS-Schnittstelle für die Anwendungsumgebung” auf Seite 929
- „z/OS UNIX System Services-Anwendungen schreiben” auf Seite 931
- „Anwendungsprogrammierung mit gemeinsam genutzten Warteschlangen” auf Seite 934

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS” auf Seite 63
Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit IBM MQ.

Umgebungsabhängige IBM MQ for z/OS-Funktionen

In diesem Abschnitt erhalten Sie Informationen zu den IBM MQ for z/OS-Funktionen.

Die IBM MQ-Funktionen unterscheiden sich in den Umgebungen, in denen IBM MQ for z/OS ausgeführt wird, im Wesentlich in Folgendem:

- IBM MQ for z/OS stellt folgende Auslösemonitor bereit:
  - CKTI für die CICS-Umgebung
  - CSQQTRMN für die IMS-Umgebung

  Wenn Sie Anwendungen in anderen Umgebungen ausführen möchten, müssen Sie Ihr eigenes Modul entwickeln.
- Synchronisationspunkte mit zweiphasigem Commit werden in CICS- und IMS-Umgebungen unterstützt. Auch in der z/OS-Stapelumgebung mit Transaktionsmanagement und Recoverable Resource Manager Services (RRS) wird diese Funktion unterstützt. Einphasiges Commit wird in der z/OS-Umgebung durch IBM MQ selbst unterstützt.
- Für Stapel- und IMS-Umgebungen stellt MQI Aufrufe für die Verbindung und Trennung von Programmen und Warteschlangenmanagern bereit. Programme können auch mit mehreren Warteschlangenmanagern verbunden werden.
- Ein CICS-System kann nur mit einem Warteschlangenmanager verbunden werden. Diese Verbindung kann bereits bei der Initiierung von CICS erfolgen, wenn der Name des Subsystems im Startjob für das CICS-System angegeben ist. In der CICS-Umgebung werden die Verbindungs- und Trennaufrufe von MQI toleriert, haben jedoch keine Wirkung.
- Der API-Steuerübergabeexit ermöglicht es einem Programm, in allen MQI-Aufrufen zu intervenieren. Dieser Exit ist allerdings nur in der CICS-Umgebung verfügbar.
- CICS auf Multiprozessorsystemen bringt einige Leistungsvorteile, da MQI-Aufrufe auf diesen Systemen unter mehreren z/OS-TCBs ausgeführt werden können. Weitere Informationen finden Sie im Handbuch Planung für z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

Eine Zusammenfassung dieser Funktionen finden Sie im Abschnitt Tabelle 115 auf Seite 928.

Tabelle 115. Funktionen in z/OS-Umgebungen

	CICS	**IMS**	**Stapel/TSO**
Auslösemonitor bereitgestellt	Ja	Ja	No
Zweiphasiges Commit	Ja	Ja	Ja
Einphasige Festschreibung	Ja	No	Ja
MQI-Aufrufe zum Verbinden/Trennen	Toleriert	Ja	Ja
API-Steuerübergabeexit	Ja	No	No

Anmerkung: Zweiphasiges Commit wird in der Stapel/TSO-Umgebung mit RRS unterstützt.

Debugfunktionen und Unterstützung für Synchronisationspunkte und Wiederherstellung

In diesem Abschnitt lernen Sie die Debugfunktionen des Programms sowie seine Unterstützung für Synchronisationspunkte und Wiederherstellung kennen.

Debugfunktionen des Programms

IBM MQ for z/OS beinhaltet eine Tracefunktion, die Sie zum Programmdebugging in allen Umgebungen verwenden können.

In der CICS-Umgebung können Sie darüber hinaus folgende Komponenten verwenden:

- CICS Execution Diagnostic Facility (CEDF)
- CICS Trace Control Transaction (CETR)
- API-Steuerübergabeexit für IBM MQ for z/OS

Auf der z/OS-Plattform können Sie jedes verfügbare interaktive Debug-Tool verwenden, das von der von Ihnen verwendeten Programmiersprache unterstützt wird.

Synchronisationspunktunterstützung

In einer Transaktionsverarbeitungsumgebung müssen Anfang und Ende der Arbeitseinheiten synchronisiert werden. Nur dann ist die Verwendung der Transaktionsverarbeitung sicher.

Dies wird von IBM MQ for z/OS in der CICS- und IMS-Umgebung vollständig unterstützt. Mit vollständiger Unterstützung ist hierbei die Zusammenarbeit der Ressourcenmanager gemeint, so dass die Arbeitseinheiten unter der Kontrolle von CICS oder IMS synchronisiert festgeschrieben (Commit) oder zurückgesetzt (Backout) werden können. Beispiele für Ressourcenmanager sind Db2, CICS File Control, IMS und IBM MQ for z/OS.

z/OS-Stapelanwendungen können als einphasige Commitfunktion IBM MQ for z/OS-Aufrufe verwenden. Das bedeutet, das ein von der Anwendung definierter Satz an Warteschlangenoperationen ohne Referenz auf andere Ressourcenmanager festgeschrieben oder zurückgesetzt werden kann.

Auch zweiphasiges Commit wird in der z/OS-Stapelumgebung mit Transaktionsmanagement und Recoverable Resource Manager Services (RRS) unterstützt. Weitere Informationen hierzu finden Sie im Abschnitt Syncpoints in z/OS-Stapelanwendungen.

Unterstützung für die Wiederherstellung

Wenn die Verbindung zwischen einem Warteschlangenmanager und dem CICS- oder IMS-System während einer Transaktion getrennt wird, werden einige Arbeitseinheiten unter Umständen nicht erfolgreich zurückgesetzt.

Jedoch kann der Warteschlangenmanager dies unter der Steuerung des Synchronisationspunktmanagers nach der Wiederherstellung der Verbindung mit dem CICS- oder IMS-System wieder korrigieren.

IBM MQ for z/OS-Schnittstelle für die Anwendungsumgebung

Damit Anwendungen in verschiedenen Umgebungen Nachrichten über ein Message-Queuing-Netz austauschen können, bietet IBM MQ for z/OS für jede unterstützte Umgebung einen *Adapter*.

Diese Adapter sind die Schnittstelle zwischen den Anwendungsprogrammen und den IBM MQ for z/OS-Subsystemen. Sie ermöglichen den Programmen die Verwendung der MQI.

Batch-Adapter

Dieser Abschnitt enthält Informationen zum Batch-Adapter und dem von ihm unterstützten Commitprotokoll.

Der *Batch-Adapter* ermöglicht Zugriff auf IBM MQ for z/OS-Ressourcen für Programme, die in einem der folgenden Modi ausgeführt werden:

- Taskmodus (Tasksteuerblock)

- Fehlerstatus oder Supervisorstatus
- Im Steuermodus für den Primäradressraum

Das Programm darf nicht im speicherübergreifenden Modus ausgeführt werden.

Die Verbindungen zwischen Anwendungsprogrammen und IBM MQ for z/OS finden auf Taskebene statt. Der Adapter bietet einen einzigen Verbindungsthread vom Tasksteuerblock (TCB) einer Anwendung zu IBM MQ for z/OS.

Der Adapter unterstützt ein einphasiges Commitprotokoll für Änderungen an IBM MQ for z/OS-Ressourcen; er unterstützt keine zweiphasigen Commitprotokolle.

RRS-Batch-Adapter
Dieser Abschnitt enthält allgemeine Informationen zu Batch-Adaptern für Recoverable Resource Manager Services (RRS) sowie Informationen zu den beiden RRS-Batch-Adaptern von IBM MQ.

Transaktionsmanagement und RRS-Adapter (Recoverable Resource Manager Services):

- Verwendet z/OS RRS zur Commitsteuerung.
- Unterstützt gleichzeitige Verbindungen mit mehreren IBM MQ-Subsystemen, die aus der gleichen Task auf einer einzigen z/OS-Instanz ausgeführt werden.
- Bietet für folgende Komponenten z/OS-weite koordinierte Commitsteuerung (mit z/OS RRS) für wiederherstellbare Ressourcen, auf die über z/OS RRS-konforme Manager für wiederherstellbare Ressourcen zugegriffen wird:
  - Anwendungen, die über den RRS-Batch-Adapter eine Verbindung zu IBM MQ herstellen.
  - Gespeicherte Db2-Prozeduren, die in einem Adressraum für gespeicherte Db2-Prozeduren ausgeführt werden, der von einem Workload-Manager (WLM) unter z/OS verwaltet wird.
- Unterstützt den Wechsel eines IBM MQ-Stapelthreads zwischen Tasksteuerblöcken (TCBs).

IBM MQ for z/OS bietet zwei RRS-Batch-Adapter:

CSQBRSTB
Für diesen Adapter müssen Sie jede MQCMIT-Anweisung in Ihrer IBM MQ-Anwendung in eine SRRCMIT-Anweisung und jede MQBACK-Anweisung in eine SRRBACK-Anweisung ändern. (Wenn Sie in einer mit CSQBRSTB verbundenen Anwendung MQCMIT oder MQBACK codieren, erhalten Sie den Fehler MQRC_ENVIRONMENT_ERROR.)

CSQBRRSI
Dieser Adapter ermöglicht in Ihrer IBM MQ-Anwendung sowohl die Verwendung von MQCMIT- und MQBACK-Anweisungen als auch die Verwendung von SRRCMIT- und SRRBACK-Anweisungen.

Anmerkung: CSQBRSTB und CSQBRRSI werden mit den Verbindungsattributen AMODE(31) und RMODE(ANY) ausgeliefert. Wenn Ihre Anwendung irgendeinen Stub unterhalb der 16-MB-Linie lädt, müssen Sie den Stub zuerst mit RMODE(24) erneut verbinden.

Migration

Sie können vorhandene IBM MQ-Stapel/TSO-Anwendungen mit kaum oder gar keinen Anpassungen für die Verwendung von RRS migrieren.

Wenn Sie zwischen Ihrer IBM MQ-Anwendung und dem CSQBRRSI-Adapter eine Programmverknüpfung herstellen, müssen Sie Ihre Arbeitseinheiten mittels MQCMIT und MQBACK über IBM MQ und alle anderen RRS-fähigen Ressourcenmanager synchronisieren. Wenn Sie zwischen Ihrer IBM MQ-Anwendung und dem CSQBRSTB-Adapter eine Programmverknüpfung herstellen, müssen Sie MQCMIT durch SRRCMIT und MQBACK durch SRRBACK ersetzen. Das zweite Verfahren ist vorzuziehen. Es weist eindeutig darauf hin, dass Synchronisationspunkte nicht auf IBM MQ-Ressourcen beschränkt sind.

IMS-Adapter
Wenn Sie den IMS-Adapter eines IBM MQ for z/OS-Systems verwenden, müssen Sie sicherstellen, dass IMS ausreichend Speicher für Nachrichtengrößen bis zu 100 MB zur Verfügung steht.

Hinweis für Benutzer

Der *IMS-Adapter* ermöglicht Zugriff auf IBM MQ for z/OS-Ressourcen für:

- Online-MPPs (Online Message Processing Programs = Online-Nachrichtenverarbeitungsprogramme)
- IFPs (Interactive Fast Path Programs = interaktive Direktaufrufprogramme)
- BMPs (Batch Message Processing Programs = Stapelnachrichtenverarbeitungprogramme)

Um diese Ressourcen verwenden zu können, müssen die Programme im Taskmodus (TCB) und im Fehlerstatus laufen (d. h., sie dürfen nicht im speicherübergreifenden oder im AR-Modus ausgeführt werden.

Der Adapter stellt einen Verbindungsthread von einem Anwendungstasksteuerblock zu IBM MQ bereit. Der Adapter unterstützt ein zweiphasiges Commitprotokoll für Änderungen an IBM MQ for z/OS-Ressourceen, wobei IMS als Synchronisationspunktkoordinator fungiert.

Außerdem stellt der Adapter ein Auslösemonitorprogramm bereit, das Programme automatisch starten kann, wenn in einer Warteschlange bestimmte Auslöserbedingungen vorliegen. Weitere Informationen finden Sie unter „IBM MQ-Anwendungen durch Auslöser starten" auf Seite 901.

Wenn Sie DL/I-Stapelprogramme schreiben, folgen Sie den Anleitungen in diesem Abschnitt für z/OS-Stapelprogramme.

z/OS UNIX System Services-Anwendungen schreiben

Der Batch-Adapter unterstützt Warteschlangenmanagerverbindungen von Stapel- und TSO-Adressräumen:

Bei einem Stapeladressraum unterstützt der Adapter wie folgt Verbindungen von mehreren (Tasksteuerblöcken) TCBs innerhalb dieses Adressraums:

- Jeder TCB kann über einen MQCONN- oder MQCONNX-Aufruf Verbindungen mit mehreren Warteschlangenmanagern herstellen (jedoch kann ein TCB jeweils nur eine Verbindungsinstanz zu einem bestimmten Warteschlangenmanager haben).
- Mit dem gleichen Warteschlangenmanager können auch mehrere TCBs verbunden sein (die von einem MQCONN- oder MQCONNX-Aufruf zurückgegebene Warteschlangenmanagerkennung ist jedoch an den ausgebenden TCB gebunden und kann von keinem anderen TCB verwendet werden).

z/OS UNIX System Services unterstützt zwei Arten von pthread_create-Aufrufen:

1. Heavyweight-Threads, einer pro TCB, die zu Beginn und Ende eines Threads von z/OS mit ATTACH und DETACH angehängt bzw. wieder gelöst werden.
2. Mediumweight-Threads, einer pro TCB, wobei der TCB ein Block aus einem Pool von TCBs mit langer Laufzeit sein kann. Um die erforderlichen Bereinigungstasks muss sich die Anwendung kümmern, da die Standard-Threadbeendigung, die normalerweise bei Verbindung mit einem Server vom Server bei Task-/TCB-Beendigung bereitgestellt wird, **nicht** immer ausgeführt wird.

Lightweight-Threads werden nicht unterstützt. (Wenn eine Anwendung permanente Threads erstellt, die ihre eigenen Arbeitsanforderungen senden, ist die ***Anwendung*** vor Ausgabe der nächsten Arbeitsanforderung für die Bereinigung der Ressourcen verantwortlich.)

IBM MQ for z/OS unterstützt z/OS UNIX System Services-Threads über den Batch-Adapter wie folgt:

1. Heavyweight-Threads werden als Stapelverbindungen vollständig unterstützt. Jeder Thread läuft in seinem eigenen TCB, der zu Beginn und Ende des Threads angehängt bzw. wieder getrennt wird. Falls der Thread vor Ausgabe eines MQDISC-Aufrufs endet, führt IBM MQ for z/OS seine Standard-Taskbereinigung durch, die bei normaler Threadbeendigung auch die Festschreibung noch ausstehender Arbeitseinheiten umfasst, bzw. bei abnormer Beendigung des Threads das Zurücksetzen der Arbeitseinheiten.
2. Mediumweight-Threads werden vollständig unterstützt. Falls aber der TCB von einem anderen Thread benötigt wird, muss die Anwendung sicherstellen, dass vor Beginn des nächstens Threads ein MQDISC-Ausruf (und davor entweder ein MQCMIT oder ein MQBACK) ausgegeben wird. Wenn die Anwendung einen Program Interrupt Handler eingerichtet hat, impliziert dies bei einer abnormen

Beendigung der Anwendung, dass der Interrupt Handler MQCMIT- und MQDISC-Aufrufe ausgeben muss, bevor der TCB für einen anderen Thread wieder verwendet wird.

Anmerkung: Die Threading-Modelle unterstützen **keinen** Zugriff auf gemeinsame IBM MQ-Ressourcen von mehreren Threads.

API-Steuerübergabeexit für z/OS

Dieser Abschnitt enthält Informationen zu produktabhängigen Programmierschnittstellen.

Ein Exit ist eine Stelle im von IBM bereitgestellten Code, an der Sie Ihren eigenen Code ausführen können. IBM MQ for z/OS stellt einen *API-Steuerübergabeexit* bereit, mit dem Sie Aufrufe an die MQI abfangen und die Funktion der MQI-Aufrufe überwachen oder ändern können. In diesem Abschnitt werden die Verwendung des API-Steuerübergabeexits und das mit IBM MQ for z/OS bereitgestellte Beispielexitprogramm beschrieben.

Dieser Abschnitt betrifft nur Benutzer von CICS TS V3.1 und früher. Benutzer ab CICS TS V3.2 finden die entsprechenden Informationen im Abschnitt CICS Integration with IBM MQ in der CICS-Produktdokumentation.

Hinweis

Der API-Steuerübergabeexit kann nur vom CICS-Adapter von IBM MQ for z/OS aufgerufen werden. Das Exitprogramm wird im Adressraum von CICS ausgeführt.

Eigenes Exitprogramm erstellen
Sie können das mit IBM MQ for z/OS bereitgestellte Beispiel-API-Steuerübergabeexitprogramm (CSQCAPX) als Framework für Ihr eigenes Programm verwenden.

Weitere Informationen hierzu finden Sie im Abschnitt „Beispielprogramm CSQCAPX für API-Steuerübergabeexit” auf Seite 933.

Wenn Sie beim Erstellen eines Exitprogramms den Namen eines von einer Anwendung ausgegebenen MQI-Aufrufs benötigen, sehen Sie sich das Feld *ExitCommand* der MQXP-Struktur an. Die Anzahl der Parameter des Aufrufs finden Sie im Feld *ExitParmCount*. Im 16 Byte großen Feld *ExitUserArea* können Sie die Adresse jedes dynamischen Speichers festhalten, den die Anwendung anfordert. Der Wert in diesem Feld bleibt auch über mehrere Aufrufe des Exits hinweg erhalten und hat die gleiche Lebensdauer wie eine CICS-Task.

Wenn Sie CICS Transaction Server V3.2 verwenden, müssen Sie Ihr Exitprogramm so schreiben, dass es threadsicher ist, und Sie müssen es zusätzlich als threadsicher deklarieren. In Vorbereitung auf die Migration auf CICS Transaction Server V3.2 empfiehlt sich dies im übrigen bereits bei Verwendung früherer CICS-Versionen.

Ihr Exitprogramm kann die Ausführung eines MQI-Aufrufs durch Rückgabe von MQXCC_SUPPRESS_FUNCTION oder MQXCC_SKIP_FUNCTION im Feld *ExitResponse* unterdrücken. Damit der Aufruf ausgeführt wird (und das Exitprogramm nach Abschluss des Aufrufs erneut aufgerufen werden kann) muss das Exitprogramm MQXCC_OK zurückgeben.

Ein Exitprogramm, das erst nach der Ausführung eines MQI-Aufrufs aufgerufen wird, kann den Abschluss des Aufrufs untersuchen und die vom Aufruf zurückgegebenen Ursachencodes ändern.

Hinweise zur Verwendung

Die folgenden Punkte sollten Sie beim Schreiben eigener Exitprogramme generell beachten:

- Aus Leistungsgründen sollten Sie das Programm in Assemblersprache schreiben. Wenn Sie es in einer anderen der von IBM MQ for z/OS unterstützten Sprachen schreiben, müssen Sie eine eigene Datendefinitionsdatei bereitstellen.
- Verknüpfen Sie Ihr Programm als AMODE(31) und RMODE(ANY).
- Verwenden Sie für die Definition des Exit-Parameterblocks für Ihr Programm das Makro CMQXPA der Assemblersprache.

- Geben Sie bei der Definition Ihres Exitprogramms und aller Programme, die Ihr Exitprogramm aufruft, CONCURRENCY(THREADSAFE) an.
- Wenn Sie CICS Transaction Server für die Speicherschutzfunktion von z/OS verwenden, muss Ihr Programm mit dem CICS-Ausführungsschlüssel ausgeführt werden. Das heißt, Sie müssen sowohl bei der Definition Ihres Exitprogramms als auch bei der Definition jedes Programms, an das das Exitprogramm die Steuerung übergibt, den Schlüssel EXECKEY(CICS) angeben. Informationen zu den Exitprogrammen von CICS und der Speicherschutzfunktion von CICS finden Sie im *CICS Customization Guide*.
- Ihr Programm kann alle APIs verwenden (z. B. IMS, Db2 und CICS), die auch ein taskbezogenes Benutzerexitprogramm von CICS verwenden kann. Es kann außerdem alle MQI-Aufrufe mit Ausnahme von MQCONN, MQCONNX und MQDISC verwenden. Allerdings rufen MQI-Aufrufe innerhalb des Exitprogramms das Exitprogramm nicht ein zweites Mal auf.
- Ihr Programm kann EXEC CICS SYNCPOINT- und EXEC CICS SYNCPOINT ROLLBACK-Befehle ausgeben. Allerdings schreiben diese Befehle ***alle*** Änderungen fest (bzw. setzen sie zurück), die durch diese Task bis zum Zeitpunkt des Exits vorgenommen wurden, weshalb deren Verwendung nicht empfohlen wird.
- Ihr Programm muss mit der Ausgabe eines EXEC CICS RETURN-Befehls enden. Es darf die Steuerung nicht über einen XCTL-Befehl übergeben.
- Exits werden als Erweiterungen zum IBM MQ for z/OS-Code geschrieben. Stellen Sie sicher, dass Ihr Exit keine IBM MQ for z/OS-Programme oder -Transaktionen unterbricht, die die MQI verwenden. Diese sind in der Regel am Präfix CSQ bzw. CK erkennbar.
- Wenn CSQCAPX für CICS definiert ist, versucht das CICS-System das Exitprogramm zu laden, sobald CICS eine Verbindung zu IBM MQ for z/OS herstellt. Wenn dieser Versuch erfolgreich ist, wird in der CKQC-Anzeige bzw. in der Systemkonsole die Nachricht CSQC301I ausgegeben. Kommt es dabei zu einem Fehler (weil zum Beispiel keine der Bibliotheken der CICS-Programmbibliotheksverkettung ein Lademodul enthält), wird in der CKQC-Anzeige bzw. in der Systemkonsole die Nachricht CSQC315 ausgegeben.
- Da es sich bei den Parametern im Kommunikationsbereich um Adressen handelt, muss das Exitprogramm relativ zum CICS-System lokal definiert werden (d. h. nicht als fernes Programm).

Beispielprogramm CSQCAPX für API-Steuerübergabeexit
Das Beispielexitprogramm wird in der Assemblersprache bereitgestellt. Die Quellendatei (CSQCAPX) wird in der Bibliothek **thlqual**.SCSQASMS bereitgestellt (wobei **thlqual** das von Ihrer Installation verwendete übergeordnete Qualifikationsmerkmal ist. Die Quellenmodule enthalten Pseudocode, der die Programmlogik beschreibt.

Das Beispielprogramm enthält Initialisierungscode und ein Layout, die Sie in eigenen Exitprogrammen übernehmen können.

Das Beispiel zeigt Folgendes:

- Einrichten des Exitparameterblocks
- Adressieren der Aufruf- und Exitparameterblöcke
- Festlegen, für welchen MQI-Aufruf der Exit aufgerufen wird
- Festlegen, ob der Exit vor oder nach der Verarbeitung des MQI-Aufrufs aufgerufen wird
- Einreihen einer Nachricht in eine temporäre CICS-Speicherwarteschlange
- Verwenden des Makros DFHEIENT für die dynamische Speicherzuweisung zum Erhalt der Wiedereintrittsfähigkeit
- Verwenden von DFHEIBLK für den CICS EXEC-Schnittstellensteuerblock
- Abfangen von Fehlerbedingungen
- Rückgabe der Steuerung an das aufrufende Programm

Gestaltung des Beispielexitprogramms

Das Beispielexitprogramm schreibt Nachrichten in eine temporäre CICS-Speicherwarteschlange (CSQ1EXIT), um die Abläufe des Exits nachzuweisen.

Die Nachrichten zeigen, ob der Exit vor oder nach dem MQI-Aufruf aufgerufen wird. Wenn der Exit nach dem Aufruf aufgerufen wird, enthält die Nachricht den vom Aufruf zurückgegebenen Beendigungs- und Ursachencode. Zur Feststellung des Eintrittstyps (vor oder nach dem Aufruf) werden im Beispiel benannte Konstanten des CMQXPA-Makros verwendet.

Das Beispiel hat keine Überwachungsfunktion, sondern fügt einer CICS-Warteschlange lediglich Nachrichten mit Zeitstempel hinzu, die den Typ des verarbeiteten Aufrufs angeben. Damit lässt sich nicht nur die Leistung der MQI überprüfen, sondern auch, ob das Exitprogramm funktioniert.

Anmerkung: Das Beispielexitprogramm gibt für jeden während der Laufzeit des Programms ausgegebenen MQI-Aufruf sechs EXEC CICS-Aufrufe aus. Die Verwendung dieses Exitprogramms wirkt sich nachteilig auf die Leistung von IBM MQ for z/OS aus.

API-Steuerübergabeexit vorbereiten und verwenden
Der Beispielexit wird nur im Quellenformat bereitgestellt.

Zur Verwendung des Beispielexits bzw. eines jeden selbst entwickelten Exitprogramms erstellen Sie wie für jedes andere CICS-Programm eine Ladebibliothek (siehe „CICS-Anwendungen in z/OS erstellen" auf Seite 1092).

- Wenn Sie für CICS Transaction Server für z/OS und für CICS für MVS/ESA den CICS-Systemdefinitions(CSD)-Datensatz aktualisieren, benötigen Sie die Definitionen aus dem Mitglied **thlqual**.SCSQPROC(CSQ4B100).

  Anmerkung: Die Definitionen verwenden das Suffix MQ. Falls dieses Suffix in Ihrem Unternehmen bereits verwendet wird, müssen Sie dieses vor der Assemblierung ändern.

Wenn Sie die bereitgestellten CICS-Standardprogrammdefinitionen verwenden, wird das Exitprogramm CSQCAPX in einem ***inaktivierten*** Zustand installiert. Der Grund hierfür ist die beträchtliche Leistungsdegradierung bei Verwendung dieses Exitprogramms.

So aktivieren Sie den API-Steuerübergabeexit vorübergehend:

1. Geben Sie den Befehl **CEMT S PROGRAM(CSQCAPX) ENABLED** vom CICS-Hauptterminal aus.
2. Führen Sie die Transaktion CKQC aus. Verwenden Sie dabei Option 3 aus dem Pulldown-Menü 'Connection' (Verbindung), um den Status des API-Steuerübergabeexits auf **Enabled** zu setzen.

Wenn IBM MQ for z/OS für CICS Transaction Server für z/OS und für CICS für MVS/ESA immer mit aktiviertem API-Steuerübergabeexit ausgeführt werden soll, führen Sie einen der folgenden Schritte aus:

- Ändern Sie die CSQCAPX-Definition in Mitglied CSQ4B100 von STATUS(DISABLED) in STATUS(ENABLED). Die CICS-CSD-Definition können Sie mit dem von CICS bereitgestellten Stapelprogramm DFHCSDUP aktualisieren.
- Ändern Sie die CSQCAPX-Definition in der CSQCAT1-Gruppe, indem Sie den Status von DISABLED auf ENABLED setzen.

In beiden Fällen müssen Sie die Gruppe neu installieren. Dazu können Sie Ihr CICS-System kalt starten oder, wenn Sie die Installation ausführen möchten, während CICS läuft, die CICS-Transaktion CEDA ausführen.

Anmerkung: Die Transaktion CEDA kann zu einem Fehler führen, wenn ein Eintrag der Gruppe zurzeit verwendet wird.

Ende der Informationen zur produktabhängigen Programmierschnittstelle.

Anwendungsprogrammierung mit gemeinsam genutzten Warteschlangen

In diesem Abschnitt erfahren Sie, was Sie bei der Entwicklung neuer Anwendungen mit gemeinsam genutzten Warteschlangen berücksichtigen sollten bzw. in welchen Fällen Sie vorhandene Anwendungen in eine Umgebung migrieren können, die Gruppen mit gemeinsamer Warteschlange unterstützt.

Anwendungen serialisieren
Bei bestimmten Anwendungstypen müssen die Nachrichten in exakt der Reihenfolge aus einer Warteschlange abgerufen werden, in der sie dort eintreffen.

Wird beispielsweise IBM MQ zur Spiegelung von Datenbankaktualisierungen auf einem fernen System verwendet, darf eine Nachricht, die eine Änderung an einem Datensatz beschreibt, erst nach der Nachricht verarbeitet werden, die die Einfügung dieses Datensatzes beschreibt. In einer lokalen Umgebung mit Warteschlangensteuerung wird dies oft durch die Anwendung durchgesetzt, die die Nachrichten aus der Warteschlange abruft, indem sie die Warteschlange mit der Option MQOO_INPUT_EXCLUSIVE öffnet. Dadurch wird verhindert, dass andere abrufende Anwendungen gleichzeitig auf die Warteschlange zugreifen.

Auf die gleiche Weise ermöglicht IBM MQ Anwendungen das exklusive Öffnen gemeinsam genutzter Warteschlangen. Dieser einfache Mechanismus des exklusiven Öffnens einer Warteschlange funktioniert allerdings nicht, wenn sich die Anwendung aus einer Partition einer Warteschlange bedient (z. B. befinden sich alle Datenbankaktualisierungen in der gleichen Warteschlange, diejenigen für Tabelle A haben jedoch die Korrelations-ID A und diejenigen für Tabelle B die Korrelations-ID B), und die Anwendung Nachrichten sowohl für Aktualisierungen an Tabelle A als auch für Aktualisierungen an Tabelle B gleichzeitig abrufen möchte.

Wenn eine solche Anwendung die Vorteile der Hochverfügbarkeit gemeinsam genutzter Warteschlangen nutzen soll, können Sie auch entscheiden, dass eine andere Instanz der Anwendung, die über einen sekundären Warteschlangenmanager auf die gleichen gemeinsam genutzten Warteschlangen zugreift, die Steuerung übernimmt, falls die primär abrufende Anwendung oder der primäre Warteschlangenmanager fehlschlägt.

Bei Fehlschlagen des primären Warteschlangenmanagers finden zwei Mechanismen statt:

- Die Peerwiederherstellung für gemeinsam genutzte Warteschlangen stellt sicher, dass unvollständige Aktualisierungen der primären Anwendung vollständig abgeschlossen oder zurückgesetzt werden.
- Die sekundäre Anwendung übernimmt die Warteschlangenverarbeitung.

Die sekundäre Anwendung beginnt eventuell bereits vor Abschluss der Verarbeitung der unvollständigen Arbeitseinheiten, was dazu führen kann, dass die sekundäre Anwendung die Nachrichten außer der Reihe abruft. Zur Umgehung dieses Problems kann die Anwendung als *serialisierte Anwendung* deklariert werden.

Eine serialisierte Anwendung verwendet für die Verbindung mit dem Warteschlangenmanager den Aufruf MQCONNX, wobei sie bei der Verbindung einen für die Anwendung eindeutigen Verbindungstag angibt. Alle von dieser Anwendung ausgeführten Arbeitseinheiten werden mit diesem Verbindungstag gekennzeichnet. IBM MQ stellt sicher, dass Arbeitseinheiten mit dem gleichen Verbindungstag innerhalb der Gruppe mit gemeinsamer Warteschlange serialisiert werden (gemäß den Serialisierungsoptionen des MQCONNX-Aufrufs).

Das bedeutet Folgendes: Wenn die primäre Anwendung den MQCONNX-Aufruf mit dem Verbindungstag `Database shadow retriever` verwendet und die übernehmende sekundäre Anwendung versucht, den MQCONNX-Aufruf mit dem gleichen Verbindungstag zu verwenden, kann die sekundäre Anwendung keine Verbindung zum zweiten IBM MQ-System herstellen, solange die ausstehenden primären Arbeitseinheiten nicht abgeschlossen sind - in diesem Fall per Peerwiederherstellung.

Diese Serialisierungsmethode empfiehlt sich für Anwendungen, für die die exakte Reihenfolge der Nachrichten in einer Warteschlange wichtig ist. Dies ist zum Beispiel bei folgenden Anwendungen der Fall:

- Anwendungen, die nach einem Anwendungs- oder Warteschlangenmanagerfehler erst dann wieder neu gestartet werden dürfen, wenn alle Commit- und Backoutoperationen für die vorherige Ausführung der Anwendung abgeschlossen sind.

  In diesem Fall kann das serialisierte Anwendungsverfahren nur verwendet werden, wenn die Anwendung mit Synchronisationspunkten arbeitet.
- Anwendungen, die nicht gestartet werden dürfen, solange eine andere Instanz der gleichen Anwendung ausgeführt wird.

  In diesem Fall ist das serialisierte Anwendungsverfahren nur erforderlich, wenn die Anwendung die Warteschlange nicht für die exklusive Eingabe öffnen kann.

Anmerkung: IBM MQ kann die Beibehaltung der Nachrichtenreihenfolge nur unter bestimmten Bedingungen garantieren. Auf diese wird in der Beschreibung des Aufrufs MQGET eingegangen.

Für gemeinsam genutzte Warteschlangen ungeeignete Anwendungen
Einige Funktionen von IBM MQ werden bei Verwendung gemeinsam genutzter Warteschlangen nicht unterstützt. Anwendungen, die diese Funktionen verwenden, sind also für Umgebungen mit gemeinsam genutzten Warteschlangen ungeeignet.

Beachten Sie bei der Entwicklung von Anwendungen, die gemeinsam genutzte Warteschlangen verwenden, die folgenden Punkte:

- Die Warteschlangenindexierung ist für gemeinsam genutzte Warteschlangen nur begrenzt möglich. Wenn Sie zur Auswahl der Nachricht, die Sie aus der Warteschlange abrufen wollen, die Nachrichten-ID oder die Korrelations-ID verwenden möchten, sollte die Warteschlange mit dem entsprechenden Wert indiziert sein. Wählen Sie Nachrichten nur anhand der Nachrichten-ID aus, benötigt die Warteschlange den Indextyp MQIT_MSG_ID (allerdings können Sie in diesem Fall auch MQIT_NONE verwenden). Wählen Sie Nachrichten nur anhand der Korrelations-ID aus, muss die Warteschlange den Indextyp MQIT_CORREL_ID haben.
- Temporäre dynamische Warteschlangen können nicht als gemeinsam genutzte Warteschlangen verwendet werden, permanente dynamische Warteschlangen hingegen schon. Die Modelle für gemeinsam genutzte dynamische Warteschlangen haben als DEFTYPE den Wert SHAREDYN (freigegeben und dynamisch), auch wenn sie auf die gleiche Weise erstellt und gelöscht werden, wie permanente dynamische Warteschlangen (PERMDYN).

Überlegungen zur gemeinsame Nutzung von Nicht-Anwendungswarteschlangen
In diesem Abschnitt erhalten Sie Informationen zur gemeinsamen Nutzung von Nicht-Anwendungswarteschlangen.

Eventuell möchten Sie neben Anwendungswarteschlangen auch andere Warteschlangen gemeinsam nutzen:

Initialisierungswarteschlangen

Wenn Sie eine Initialisierungswarteschlange als gemeinsam genutzt definieren, ist nicht auf jedem Warteschlangenmanager der Gruppe mit gemeinsamer Warteschlange ein Auslösemonitor erforderlich. Sie müssen lediglich sicherstellen, dass auf einem der Warteschlangenmanager ein Auslösemonitor läuft. (Selbstverständlich können Sie eine gemeinsam genutzte Initialisierungswarteschlange aber auch dann nutzen, wenn auf jedem Warteschlangenmanager der Gruppe mit gemeinsamer Warteschlange ein Auslösemonitor läuft.)

Wenn Sie eine gemeinsam genutzte Anwendungswarteschlange haben und den Auslösertyp EVERY verwenden (oder FIRST mit einem kleinen Auslöseintervall, das wie der Auslösertyp EVERY wirkt), muss Ihre Initialisierungswarteschlange auf jeden Fall eine gemeinsam genutzte Warteschlange sein. Weitere Informationen zur Verwendung einer gemeinsam genutzten Initialisierungswarteschlange finden Sie im Abschnitt Tabelle 116 auf Seite 937.

SYSTEM.* -Warteschlangen
Sie können die für Ereignisnachrichten verwendeten SYSTEM.ADMIN.*-Warteschlangen als gemeinsam genutzte Warteschlangen definieren. Dies kann bei Ausnahmebedingungen zur Überprüfung des Lastausgleichs sinnvoll sein. Jede von IBM MQ erstellte Ereignisnachricht enthält eine Korrelations-ID, die angibt, von welchem Warteschlangenmanager die Nachricht generiert wurde.

SYSTEM.QSG.*-Warteschlangen für gemeinsam genutzte Kanäle und die gruppeninterne Steuerung von Warteschlangen müssen Sie als gemeinsam genutzte Warteschlangen definieren.

Auch die Definitionen von SYSTEM.DEFAULT.LOCAL.QUEUE-Warteschlangen können Sie als gemeinsam genutzt deklarieren; ebenso können Sie aber auch eigene Definitionen für gemeinsam genutzte Standardwarteschlangen festlegen. Eine entsprechende Beschreibung finden Sie im Abschnitt **Defining system objects** im Planung für z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

Andere SYSTEM.*-Warteschlangen können nicht als gemeinsam genutzte Warteschlangen definiert werden.

Migration vorhandener Anwendungen für die Verwendung gemeinsam genutzter Warteschlangen
Ursachencodes, die Auslösefunktion und der API-Aufruf MQINQ können sich in einer Umgebung mit gemeinsam genutzten Warteschlangen anders verhalten.

Die Migration vorhandener Warteschlangen in gemeinsam genutzte Warteschlangen wird im IBM MQ for z/OS verwalten *IBM MQ for z/OS System Administration Guide* beschrieben.

Vor der Migration Ihrer vorhandenen Anwendungen sollten Sie bedenken, dass sich die folgenden Mechanismen in einer Umgebung mit gemeinsam genutzten Warteschlangen anders verhalten können:

Ursachencodes
Wenn Sie Ihre vorhandenen Anwendungen für die Verwendung gemeinsam genutzter Warteschlangen migrieren, überprüfen Sie, welche Ursachencodes in der neuen Umgebung ausgegeben werden können.

Auslösefunktion
Wenn Sie eine gemeinsam genutzte Anwendungswarteschlange verwenden, funktioniert der Auslösemechanismus nur bei bereits festgeschriebenen Nachrichten (bei nicht gemeinsam genutzten Anwendungswarteschlangen funktioniert das Auslösen bei allen Nachrichten).

Wenn Sie eine Anwendung über den Auslösemechanismus starten, empfiehlt sich unter Umständen die Verwendung einer gemeinsam genutzten Initialisierungswarteschlange. Tabelle 116 auf Seite 937 gibt Aufschluss über alle Aspekte, die Sie bei der Entscheidung über die Art der Initialisierungswarteschlange berücksichtigen müssen.

Tabelle 116. Wann sollten Sie eine gemeinsam genutzte Initialisierungswarteschlange verwenden

	Nicht gemeinsam genutzte Anwendungswarteschlange	**Gemeinsam genutzte Anwendungswarteschlange**
Nicht gemeinsam genutzte Initialisierungswarteschlange	Wie in früheren Releases.	Bei Verwendung des Auslösertyps FIRST oder DEPTH können Sie eine nicht gemeinsam genutzte Initialisierungswarteschlange mit einer gemeinsam genutzten Anwendungswarteschlange verwenden. Eventuell werden bei dieser Konfiguration mehr Auslösenachrichten generiert, jedoch ist die Konfiguration besonders gut für die Auslösung von Anwendungen mit langen Laufzeiten geeignet (wie die CICS bridge) und bietet eine hohe Verfügbarkeit. Beim Auslösertyp FIRST oder DEPTH löst eine Auslösenachricht eine Instanz der Anwendung auf jedem Warteschlangenmanager aus, auf dem ein Auslösemonitor ausgeführt wird und für den noch keine Anwendungswarteschlange für die Eingabe geöffnet ist. Pro Warteschlangenmanager wird eine Auslösenachricht generiert; wenn auf einem bestimmten Warteschlangenmanager mehrere Auslösemonitore für die nicht gemeinsam genutzte lokale Initialisierungswarteschlange ausgeführt werden, konkurrieren sie um die Verarbeitung der Nachricht.

Tabelle 116. Wann sollten Sie eine gemeinsam genutzte Initialisierungswarteschlange verwenden (Forts.)

	Nicht gemeinsam genutzte Anwendungswarteschlange	**Gemeinsam genutzte Anwendungswarteschlange**
Gemeinsam genutzte Initialisierungswarteschlange	Verwenden Sie keine gemeinsam genutzte Initialisierungswarteschlange mit einer nicht gemeinsam genutzten Anwendungswarteschlange.	Bei einer gemeinsam genutzten Anwendungswarteschlange mit dem Auslösertyp EVERY sollten Sie hingegen eine gemeinsam genutzte Initialisierungswarteschlange verwenden, um unter bestimmten Umständen, zum Beispiel, wenn ein Warteschlangenmanager fehlschlägt, keine Auslösenachrichten zu verlieren. Beim Auslösertyp FIRST oder DEPTH wird von jedem Warteschlangenmanager, der die angegebene Initialisierungswarteschlange für die Eingabe geöffnet hat, eine Auslösenachricht generiert. **Anmerkung:** Wenn bei Verwendung des Auslösertyps FIRST oder DEPTH eine Instanz des Auslösemonitors ausgelastet ist, erhalten weniger aktive Auslösemonitore die Gelegenheit, auch mehrere Auslösenachrichten aus der gemeinsam genutzten Initialisierungswarteschlange zu verarbeiten. Daher können für einen bestimmten Warteschlangenmanager auch mehrere Instanzen der Serveranwendung gestartet werden. Beachten Sie, dass diese Instanzen in Folge der Verarbeitung mehrerer Auslösenachrichten gestartet werden. Normalerweise ist es nämlich so, wenn beim Auslösertyp FIRST oder DEPTH eine Anwendungsinstanz bereits eine Anwendungswarteschlange bedient, dass von dem Warteschlangenmanager, mit dem die Anwendung verbunden ist, keine weitere Auslösenachricht generiert wird.

MQINQ

Wenn Sie den Aufruf MQINQ zum Anzeigen von Informationen zu einer gemeinsam genutzten Warteschlange verwenden, beziehen sich die Werte zur Anzahl der MQOPEN-Aufrufe, für die die Warteschlange für Ein- und Ausgaben geöffnet ist, nur auf den Warteschlangenmanager, der den Aufruf ausgegeben hat. Zu anderen Warteschlangenmanagern der Gruppe mit gemeinsamer Warteschlange, die die Warteschlange geöffnet haben, gibt der Aufruf keine Informationen zurück.

z/OS IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS

Die Informationen in diesem Abschnitt helfen Ihnen beim Schreiben von IMS-Anwendungen mit IBM MQ.

- Informationen zur Verwendung von Synchronisationspunkten und MQI-Aufrufen in IMS-Anwendungen finden Sie unter „IMS-Anwendungen mit IBM MQ schreiben” auf Seite 64.
- Informationen zum Schreiben von Anwendungen, die die IBM MQ - IMS-Brücke verwenden, finden Sie unter „IMS-Brückenanwendungen schreiben” auf Seite 69.

Unter den nachfolgenden Links erhalten Sie weitere Informationen zu IMS- und IMS-Brückenanwendungen unter IBM MQ for z/OS:

- „IMS-Anwendungen mit IBM MQ schreiben” auf Seite 64
- „IMS-Brückenanwendungen schreiben” auf Seite 69

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 744
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 760
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 769
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von IBM MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 781
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 797
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 885
Attribute sind Eigenschaften, die die Merkmale eines IBM MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 889
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 901
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 922
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

„Anwendungen unter IBM MQ for z/OS verwenden und schreiben” auf Seite 927
IBM MQ for z/OS-Anwendungen können aus Programmen zusammengestellt werden, die in verschiedenen Umgebungen ausgeführt werden können. Sie können daher die Funktionen mehrerer Umgebungen nutzen.

IMS-Anwendungen mit IBM MQ schreiben
Wenn Sie IBM MQ in IMS-Anwendungen verwenden, sind weitere Überlegungen zu berücksichtigen, z. B., welche MQ-API-Aufrufe und welche Mechanismen für Synchronisationspunkte verwendet werden können.

Unter den nachfolgenden Links erhalten Sie weitere Informationen zum Schreiben von IMS-Anwendungen unter IBM MQ for z/OS:

- „Synchronisationspunkte in IMS-Anwendungen” auf Seite 65
- „MQI-Aufrufe in IMS-Anwendungen” auf Seite 65

Einschränkungen
Es gibt Beschränkungen hinsichtlich der IBM MQ-API-Aufrufe, die eine Anwendung verwenden kann, die den IMS-Adapter verwendet.

Folgende IBM MQ-API-Aufrufe werden nicht in Anwendungen unterstützt, die den IMS-Adapter verwenden:

- MQCB
- MQCB_FUNCTION
- MQCTL

Zugehörige Konzepte

„IMS-Brückenanwendungen schreiben” auf Seite 69
Dieser Abschnitt enthält Informationen zum Schreiben von Anwendungen für die Verwendung der IBM MQ - IMS-Brücke.

Synchronisationspunkte in IMS-Anwendungen
Sie erstellen einen Synchronisationspunkt in einer IMS-Anwendung mithilfe von IMS-Aufrufen wie beispielsweise GU (Get Unique) eines primären E/A-Steuerblocks (IOPCB) und des CHKP (Prüfpunkt).

Wenn Sie alle Änderungen seit dem letzten Prüfpunkt zurücksetzen möchten, verwenden Sie den IMS-Aufruf ROLB (Rollback). Weitere Informationen finden Sie in folgender Dokumentation:

- IMS 13 Application Programming APG SC19-3646
- IMS 13 Application Programming APIs APR SC19-3647

Der Warteschlangenmanager ist Teilnehmer eines Protokolls für zweiphasige Festschreibung; Koordinator ist der IMS-Synchronisationspunktmanager.

Alle offenen Kennungen werden vom IMS-Adapter an einem Synchronisationspunkt geschlossen (außer in einer Stapel- oder nicht nachrichtengesteuerten BMP-Umgebung). Grund hierfür ist, dass ein anderer Benutzer die nächste Arbeitseinheit initiieren könnte und die Sicherheitsprüfung durch IBM MQ bei einem MQCONN-, MQCONNX- oder MQOPEN-Aufruf durchgeführt wird und nicht bei einem MQPUT- oder MQGET-Aufruf.

In einer Wait-for-Input-Umgebung (WFI) oder einer Pseudo-Wait-for-Input-Umgebung (PWFI) wird IBM MQ jedoch erst dann von IMS angewiesen, die Kennungen zu schließen, wenn die nächste Nachricht eintrifft oder ein QC-Statuscode an die Anwendung zurückgegeben wird. Wenn die Anwendung im IMS-Bereich wartet und solche Kennungen zu ausgelösten Warteschlangen gehören, findet keine Auslösung statt, da die Warteschlangen bereits offen sind. Aus diesem Grund sollen Anwendungen, die in einer WFI- oder PWFI-Umgebung ausgeführt werden, die Warteschlangenkennungen explizit mit MQCLOSE schließen, bevor sie den GU-Aufruf eines primären E/A-Steuerblocks (IOPCB) für die nächste Nachricht tätigen.

Wenn eine IMS-Anwendung (BMP oder MPP) einen MQDISC-Aufruf ausgibt, werden offene Warteschlangen geschlossen, jedoch kein impliziter Synchronisationspunkt eingerichtet. Wenn die Anwendung normaler beendet wird, werden alle geöffneten Warteschlangen geschlossen und eine implizite Festschreibung wird ausgeführt. Endet die Anwendung abnormal, werden alle geöffneten Warteschlangen geschlossen und eine implizite Zurücksetzung (Backout) wird ausgeführt.

MQI-Aufrufe in IMS-Anwendungen
Dieser Abschnitt enthält Informationen zur Verwendung von MQI-Aufrufen in Server- und Abfrageanwendungen.

In diesem Abschnitt wird die Verwendung von MQI-Aufrufen in folgenden IMS-Anwendungstypen behandelt:

- „Serveranwendungen” auf Seite 940
- „Abfrageanwendungen” auf Seite 943

Serveranwendungen

Nachfolgend sehen Sie die Gliederung des MQI-Serveranwendungsmodells:

```
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
```

Das Beispielprogramm CSQ4ICB3 zeigt, wie ein BMP mit diesem Modell in C/370 integriert wird. Das Programm stellt zunächst die Kommunikation mit IMS und dann mit IBM MQ her:

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

Return
```

Die IMS-Initialisierung ermittelt, ob das Programm als nachrichtengesteuerte oder als stapelorientiertes BMP aufgerufen wurde und steuert entsprechend die Verbindung zum IBM MQ-Warteschlangenmanager sowie die Warteschlangenkennung:

```
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
```

Die IBM MQ-Initialisierung stellt eine Verbindung zum Warteschlangenmanager her und öffnet die Warteschlange. Bei einem nachrichtengesteuerten BMP wird dies nach jedem IMS-Synchronisationspunkt aufgerufen, bei einem stapelorientierten BMP hingegen nur während des Programmstarts:

```
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
```

Die Implementierung des Servermodells in einem MPP wird von der Tatsache beeinflusst, dass das MPP eine einzelne Arbeitseinheit pro Aufruf verarbeitet. Wenn ein Synchronisationspunkt (GU) stattfindet, werden nämlich die Verbindung und die Warteschlangenkennungen geschlossen und die nächste IMS-Nachricht wird zugestellt. Diese Einschränkung kann zum Teil durch eine der folgenden Maßnahmen aufgehoben werden:

- **Verarbeitung vieler Nachrichten innerhalb einer einzelnen Arbeitseinheit**

  Hierzu gehören:

  – Nachricht lesen
  – Erforderliche Aktualisierungen verarbeiten
  – Antwort einreihen

  Dies findet in einer Schleife statt, bis alle Nachrichten verarbeitet wurden oder eine festgelegte maximale Anzahl von Nachrichten verarbeitet wurde, bis ein Synchronisationspunkt erreicht wird.

  Nur bestimmte Anwendungstypen (zum Beispiel eine einfache Datenbankaktualisierung oder Abfrage) können auf diese Weise bearbeitet werden. Obwohl die MQI-Antwortnachrichten mit der Berechtigung des Erstellers der zu bearbeitenden MQI-Nachricht eingereiht werden können, müssen die Sicherheitsüberlegungen zu allen IMS-Ressourcenaktualisierungen sorgfältig einbezogen werden.
- **Eine Nachricht pro Aufruf des MPP verarbeiten und die Mehrfachplanung des MPP sicherstellen, um alle verfügbaren Nachrichten zu verarbeiten.**

  Verwenden Sie das IBM MQ IMS-Auslösemonitorprogramm (CSQQTRMN) zur Planung der MPP-Transaktion, wenn Nachrichten in der IBM MQ-Warteschlange stehen und von keinen Anwendungen verarbeitet werden.

  Wenn der Auslösemonitor das MPP startet, werden der Name des Warteschlangenmanagers und der Name der Warteschlange an das Programm übergeben (siehe nachstehender COBOL-Codeextrakt):

```
* 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 '='.
```

Das Servermodell, bei dem es sich Erwartungen zufolge um eine Langzeittask handelt, wird besser in einem Stapelverarbeitungsbereich unterstützt, obwohl das BMP nicht mit CSQQTRMN ausgelöst werden kann.

Abfrageanwendungen

Eine typische IBM MQ-Anwendung, die eine Abfrage oder Aktualisierung initiiert, funktioniert wie folgt:

- Daten vom Benutzer zusammenstellen
- Mindestens eine IBM MQ-Nachricht einreihen
- Antwortnachricht abrufen (u. U. mit Wartezeit verbunden)
- Antwort an Benutzer übermitteln

Da Nachrichten, die in IBM MQ-Warteschlangen eingereiht wurden, für andere IBM MQ-Anwendungen erst zur Verfügung stehen, nachdem sie festgeschrieben wurden, müssen sie entweder außerhalb des Synchronisationspunkts eingereiht werden oder die IMS-Anwendung muss auf zwei Transaktionen aufgeteilt werden.

Wenn mit der Abfrage das Einreihen einer einzelnen Nachricht verbunden ist, können Sie die Option *no syncpoint* verwenden. Bei komplexeren Abfragen oder Ressourcenaktualisierungen entstehen jedoch unter Umständen Konsistenzprobleme, falls ein Fehler auftritt und Sie keinen Synchronisationspunkt verwenden.

Um dies zu vermeiden, können Sie IMS-MPP-Transaktionen mittels MQI-Aufrufen aufteilen, die einen Program-to-Program-Nachrichtenswitch verwenden. Informationen hierzu finden Sie unter *IMS/ESA Anwendungsprogrammierung: Datenkommunikation*. Auf diese Weise kann ein Abfrageprogramm in ein MPP implementiert werden:

```
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
```

IMS-Brückenanwendungen schreiben

Dieser Abschnitt enthält Informationen zum Schreiben von Anwendungen für die Verwendung der IBM MQ - IMS-Brücke.

Informationen zur IBM MQ-IMS-Bridge enthält der Abschnitt IMS-Bridge.

Unter den nachfolgenden Links erhalten Sie weitere Informationen zum Schreiben von IMS-Brückenanwendungen unter IBM MQ for z/OS:

- „Bearbeitung von Nachrichten durch die IMS-Bridge” auf Seite 69
- „IMS-Transaktionsprogramme in IBM MQ schreiben” auf Seite 951

Zugehörige Konzepte

„IMS-Anwendungen mit IBM MQ schreiben” auf Seite 64
Wenn Sie IBM MQ in IMS-Anwendungen verwenden, sind weitere Überlegungen zu berücksichtigen, z. B., welche MQ-API-Aufrufe und welche Mechanismen für Synchronisationspunkte verwendet werden können.

Bearbeitung von Nachrichten durch die IMS-Bridge
Wenn Sie mit der IBM MQ-IMS-Bridge Nachrichten an eine IMS-Anweisung senden, müssen Sie die Nachrichten in einem besonderen Format erstellen.

Sie müssen Ihre Nachrichten auch in IBM MQ-Warteschlangen einreihen, die mit einer Speicherklasse definiert wurden, welche die XCF-Gruppe und den Mitgliedsnamen des IMS-Zielsystems angibt. Diese werden als MQ-IMS-Bridge-Warteschlangen oder kurz als **Bridge-Warteschlangen** bezeichnet.

Die IBM MQ-IMS-Bridge erfordert exklusiven Eingabezugriff (MQOO_INPUT_EXCLUSIVE) für die Bridge-Warteschlange, wenn sie mit QSGDISP (QMGR) bzw. mit QSGDISP (SHARED) und der Option NOSHARE definiert ist.

Ein Benutzer muss sich nicht erst bei IMS anmelden, bevor er Nachrichten an eine IMS-Anwendung sendet. Die Benutzer-ID im Feld *UserIdentifier* der MQMD-Struktur wird für die Sicherheitsprüfung verwendet. Die Prüfungsstufe wird festgelegt, wenn IBM MQ die Verbindung zu IMS herstellt. Eine Beschreibung finden Sie im Abschnitt Anwendungszugriffssteuerung für die IMS-Bridge. So kann eine Pseudo-Anmeldung implementiert werden.

Die IBM MQ - IMS-Bridge akzeptiert folgende Nachrichtentypen:

- Nachrichten mit IMS-Transaktionsdaten und einer MQIIH-Struktur (siehe Beschreibung unter MQIIH):

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

  Anmerkung:

  1. Die eckigen Klammern [ ] stellen optionale Multisegmente dar.
  2. Setzen Sie das Feld *Format* der MQMD-Struktur auf MQFMT_IMS, wenn Sie die MQIIH-Struktur verwenden möchten.
- Nachrichten mit IMS-Transaktionsdaten, aber ohne MQIIH-Struktur:

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

IBM MQ validiert die Nachrichtendaten, um sicherzustellen, dass die Summe der LL-Bytes plus die Länge von MQIIH (falls vorhanden) der Nachrichtenlänge entspricht.

Wenn die IBM MQ - IMS-Bridge Nachrichten aus Bridge-Warteschlangen abruft, verarbeitet sie diese wie folgt:

- Enthält die Nachricht eine MQIIH-Struktur, verifiziert die Bridge den MQIIH (siehe MQIIH), erstellt die OTMA-Header und sendet die Nachricht an IMS. Der Transaktionscode wird in der Eingabenachricht angegeben. Handelt es sich dabei um einen LTERM, antwortet IMS mit einer DFS1288E-Nachricht. Stellt der Transaktionscode einen Befehl dar, führt IMS den Befehl aus; andernfalls wird die Nachricht für die Transaktion in IMS eingereiht.
- Enthält die Nachricht IMS-Transaktionsdaten, aber keine MQIIH-Struktur, nimmt die IMS-Bridge Folgendes an:
  - Der Transaktionscode befindet sich in den Bytes 5 bis 12 der Benutzerdaten.
  - Die Transaktion ist im nicht interaktiven Modus.
  - Die Transaktion ist im Festschreibungsmodus 0 (commit-then-send).
  - Der Wert im Feld *Format* des MQMD wird bei der Eingabe als *MFSMapName* verwendet.
  - Der Sicherheitsmodus ist MQISS_CHECK.

  Die Antwortnachricht wird ebenfalls ohne MQIIH-Struktur erstellt. Sie übernimmt das *Format* für den MQMD aus *MFSMapName* der IMS-Ausgabe.

Die IBM MQ - IMS-Bridge verwendet eine oder zwei Transaktionspipes für jede IBM MQ-Warteschlange:

- Eine synchronisierte Transaktionspipe wird für alle Nachrichten verwendet, die Festschreibungsmodus 0 (COMMIT_THEN_SEND) verwenden (für diese wird im Statusfeld des Befehls IMS /DIS TMEMBER client TPIPE xxxx der Status SYN angezeigt).
- Für alle Nachrichten, die Festschreibungsmodus 1 (SEND_THEN_COMMIT) verwenden, wird eine nicht synchronisierte Transaktionspipe verwendet.

Die Transaktionspipes werden bei ihrer ersten Verwendung von IBM MQ erstellt. Eine nicht synchronisierte Transaktionspipe ist bis zum Neustart von IMS vorhanden. Synchronisierte Transaktionspipes sind bis zum Kaltstart von IMS vorhanden. Diese Transaktionspipes können Sie nicht selber löschen.

Weitere Informationen zur Bearbeitung von Nachrichten durch die IBM MQ - IMS-Bridge finden Sie in den folgenden Abschnitten:

- „IBM MQ-Nachrichten zu IMS-Transaktionstypen zuordnen” auf Seite 70
- „Nachricht kann nicht in die IMS-Warteschlange eingereiht werden” auf Seite 71
- „Rückkopplungscodes der IMS-Brücke” auf Seite 72
- „MWMD-Felder in Nachrichten von der IMS-Brücke” auf Seite 72
- „MQIIH-Felder in Nachrichten von der IMS-Brücke” auf Seite 73
- „Antwortnachrichten aus IMS” auf Seite 74
- „Alternative Antwort-PCBs in IMS-Transaktionen verwenden” auf Seite 74
- „Nicht erwartete Nachrichten aus IMS senden” auf Seite 74
- „Nachrichtensegmentierung” auf Seite 75
- „Datenkonvertierung” auf Seite 75

Zugehörige Konzepte

„IMS-Transaktionsprogramme in IBM MQ schreiben” auf Seite 951

Die Codierung, die für die Verarbeitung von IMS-Transaktionen durch IBM MQ erforderlich ist, hängt von dem von der IMS-Transaktion benötigten Nachrichtenformat und von dem Nachrichtenspektrum ab, das die Transaktion zurückgeben kann. Wenn Ihre Anwendung die Bildschirmformatinformationen von IMS verarbeitet, müssen Sie jedoch einige Punkte berücksichtigen.

IBM MQ-Nachrichten zu IMS-Transaktionstypen zuordnen

Eine Tabelle mit Erläuterungen zur Zuordnung von IBM MQ-Nachrichten zu IMS-Transaktionstypen.

Tabelle 117. IBM MQ-Nachrichten zu IMS-Transaktionstypen zuordnen

IBM MQ-Nachrichtentyp	**Commit-then-send (mode 0) - verwendet synchronisierte IMS-Transaktionspipes**	**Send-then-commit (mode 1) - verwendet nicht synchronisierte IMS-Transaktionspipes**
Persistente IBM MQ-Nachrichten	• Wiederherstellbare Vollfunktion-Transaktionen • Nicht wiederherstellbare Transaktionen werden von IMS zurückgewiesen.	• Fastpath-Transaktionen • Dialogtransaktionen • Vollfunktion-Transaktionen
Nicht persistente IBM MQ-Nachrichten	• Nicht wiederherstellbare Vollfunktion-Transaktionen • Wiederherstellbare Transaktionen mit IMS V8 und APAR PQ61404 sowie allen späteren Versionen von IMS sind zulässig.	• Fastpath-Transaktionen • Dialogtransaktionen • Vollfunktion-Transaktionen

Anmerkung: IMS-Befehle können persistente IBM MQ-Nachrichten mit Commitmodus 0 nicht verwenden. Weitere Informationen hierzu enthält das *IMS/ESA-Benutzerhandbuch zu Open Transaction Manager Access*.

Nachricht kann nicht in die IMS-Warteschlange eingereiht werden
In diesem Abschnitt erfahren Sie, was zu tun ist, wenn eine Nachricht nicht in die IMS-Warteschlange eingereiht werden kann.

Wenn eine Nachricht nicht in die IMS-Warteschlange eingereiht werden kann, führt IBM MQ Folgendes aus:

- Wenn eine Nachricht nicht in IMS eingereiht werden kann, da sie ungültig ist, wird sie stattdessen in die Warteschlange für nicht zustellbare Nachrichten eingereiht und eine entsprechende Nachricht wird an die Systemkonsole gesendet.
- Lehnt IMS eine gültige Nachricht ab, sendet IBM MQ eine Fehlernachricht mit dem IMS-Prüfcode an die Systemkonsole. Die IBM MQ-Nachricht wird in die Warteschlange für nicht zustellbare Nachrichten eingereiht. Lautet der IMS-Prüfcode 001A, sendet IMS eine IBM MQ-Nachricht mit der Fehlerursache an die Empfangswarteschlange für Antworten.

  Anmerkung: Unter den oben genannten Umständen wird die Nachricht an die ursprüngliche IBM MQ-Warteschlange zurückgesendet, wenn sie von IBM MQ aus irgendeinem Grund nicht in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden kann. Eine Fehlernachricht wird an die Systemkonsole gesendet und es werden keine weiteren Nachrichten aus dieser Warteschlange gesendet.

  Führen Sie ***einen*** der folgenden Schritte aus, um die Nachrichten erneut zu senden:

  – Stopp und Neustart der Transaktionspipes in IMS entsprechend der Warteschlange
  – Ändern der Warteschlange in GET(DISABLED) und wieder in GET(ENABLED)
  – Stopp und Neustart von IMS oder von OTMA
  – Stopp und Neustart Ihres IBM MQ-Subsystems
- Lehnt IMS aus einem anderen Grund als ein Nachrichtenfehler ab, wird die IBM MQ-Nachricht an die ursprüngliche Warteschlange zurückgegeben. IBM MQ stoppt die Verarbeitung der Warteschlange und eine Fehlernachricht wird an die Systemkonsole gesendet.

  Wenn eine Ausnahmeberichtsnachricht erforderlich ist, reiht sie die Brücke mit der Berechtigung des Absenders die Empfangswarteschlange für Antworten ein. Kann die Nachricht nicht in die Warteschlange eingereiht werden, wird die Berichtsnachricht mit der Berechtigung der Brücke in die Warteschlange für nicht zustellbare Nachrichten eingereiht. Kann sie nicht in die DLQ eingereiht werden, wird sie gelöscht.

Rückkopplungscodes der IMS-Brücke
Normalerweise werden IMS-Prüfcodes in IBM MQ-Konsolennachrichten wie z. B. CSQ2001I im Hexadezimalformat ausgegeben (z. B. Prüfcode 0x001F). IBM MQ-Rückkopplungscodes im Header einer nicht zustellbaren Nachricht, die in die Warteschlange für nicht zustellbare Nachrichten eingereiht wurde, werden in Dezimalzahlen angegeben.

Die Rückkopplungscodes der IMS-Brücke liegen im Zahlenbereich von 301 bis 399, der NACK-Prüfcode 0x001A liegt im Bereich von 600 bis 855. Sie werden von den IMS-OTMA-Prüfcodes wie folgt zugeordnet:

1. Der IMS-OTMA-Prüfcode im Hexadezimalformat wird in eine Dezimalzahl konvertiert.
2. 300 wird zu der Zahl addiert, die sich aus der Berechnung in 1ergibt und ergibt den IBM MQ *Feedback* -Code.
3. Der IMS-OTMA-Prüfcode 0x001A, Dezimalzahl 26, ist ein Sonderfall. Ein *Rückkopplungscode* im Bereich von 600 bis 855 wird generiert.
   a. Der IMS-OTMA-Ursachencode im Hexadezimalformat wird in eine Dezimalzahl konvertiert.
   b. Zu dem Ergebnis der Berechnung in a wird 600 hinzuaddiert, was den IBM MQ-*Rückkopplungscode* ergibt.

Informationen zu den IMS-OTMA-Prüfcodes finden Sie unter OTMA-Prüfcodes für NAK-Nachrichten.

MWMD-Felder in Nachrichten von der IMS-Brücke
Dieser Abschnitt enthält Informationen zu den MQMD-Feldern in Nachrichten von der IMS-Brücke.

Der MQMD der ursprünglichen Nachricht wird von IMS in den Benutzerdatenbereich der OTMA-Header übertragen. Stammt die Nachricht ursprünglich aus IMS, wird dieser vom IMS-Exit für die Zielauflösung erstellt. Der MQMD einer aus IMS empfangenen Nachricht wird wie folgt erstellt:

StrucID
"MD "

Version
MQMD_VERSION_1

Bericht
MQRO_NONE

MsgType
MQMT_REPLY

Verfall
Wenn MQIIH_PASS_EXPIRATION im Flagfeld des MQIIH gesetzt ist, enthält dieses Feld die verbleibende Ablaufzeit. Andernfalls wird es auf MQEI_UNLIMITED gesetzt.

Feedback
MQFB_NONE

Encoding
MQENC.Native (Codierung des z/OS-Systems)

CodedCharSetId
MQCCSI_Q_MGR (ID des codierten Zeichensatzes des z/OS-Systems)

Format
MQFMT_IMS, wenn das MQMD.Format der Eingabenachricht MQFMT_IMS ist, ansonsten IOPCB.MODNAME

Priority
MQMD.Priority der Eingabenachricht

Permanenz
Hängt vom Commit-Knoten ab: MQMD.Persistence der Eingabenachricht, wenn CM-1; Persistenz entspricht der Wiederherstellbarkeit der IMS-Nachricht, wenn CM-0

MsgId
MQMD.MsgId wenn MQRO_PASS_MSG_ID, ansonsten New MsgId (Standardeinstellung)

CorrelId
MQMD.CorrelId aus der Eingabenachricht, wenn MQRO_PASS_CORREL_ID, ansonsten MQMD.MsgId aus der Eingabenachricht (Standardeinstellung)

BackoutCount
0

ReplyToQ
Leerzeichen

ReplyToQMgr
Leerzeichen (beim MQPUT-Aufruf vom Warteschlangenmanager auf den Namen des lokalen Warteschlangenmanagers gesetzt)

UserIdentifier
MQMD.UserIdentifier der Eingabenachricht

AccountingToken
MQMD.AccountingToken der Eingabenachricht

ApplIdentityData
MQMD.ApplIdentityData der Eingabenachricht

PutApplType
MQAT_XCF, wenn kein Fehler vorliegt, ansonsten MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName>, wenn kein Fehler vorliegt, ansonsten Name des Warteschlangenmanagers (QMGR name)

PutDate
Datum der Nachrichteneinreihung

PutTime
Uhrzeit, zu der die Nachricht eingereiht wurde

ApplOriginData
Leerzeichen

MQIIH-Felder in Nachrichten von der IMS-Brücke
Dieser Abschnitt enthält Informationen zu den MQIIH-Feldern in Nachrichten von der IMS-Brücke.

Der MQIIH einer aus IMS empfangenen Nachricht wird wie folgt erstellt:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat der Eingabenachricht, wenn MQIIH.ReplyToFormat nicht leer ist, ansonsten IOPCB.MODNAME

Flags
0

LTermOverride
LTERM-Name (Transaktionspipe) aus OTMA-Header

MFSMapName
Mapname aus OTMA-Header

ReplyToFormat
Leerzeichen

Authenticator
MQIIH.Authenticator der Eingabenachricht, wenn die Antwortnachricht in die Warteschlange einer MQ-IMS-Brücke eingereiht wird, ansonsten Leerzeichen.

TranInstanceId
Dialog-ID / Server-Token aus dem OTMA-Header, wenn im Dialog. In IMS-Versionen vor V14 enthält dieses Feld immer Null, wenn es sich nicht in einem Dialog befindet. Ab der IMS-Version 14 wird dieses Feld möglicherweise von IMS festgelegt, selbst wenn kein Dialog besteht.

TranState
"C", wenn im Dialog, ansonsten Leerzeichen

CommitMode
Commitmodus aus OTMA-Header ("0" oder "1")

SecurityScope
keinen

Reserved
keinen

Antwortnachrichten aus IMS
Wenn eine IMS-Transaktion den Befehl ISRT an ihren primären E/A-Steuerblock ausgibt, wird die Nachricht an das ursprüngliche logische Terminal (LTERM) oder an die Transaktionspipe (TPIPE) zurückgeleitet.

Sie werden in IBM MQ als Antwortnachrichten angezeigt. Antwortnachrichten aus IMS werden in die Empfangswarteschlange für Antworten eingereiht, die in der ursprünglichen Nachricht angegeben ist. Kann die Nachricht nicht in die Empfangswarteschlange für Antworten eingereiht werden, wird sie mit der Berechtigung der Brücke in die Warteschlange für nicht zustellbare Nachrichten eingereiht. Kann die Nachricht nicht in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden, wird sie an IMS gesendet, um mitzuteilen, dass die Nachricht nicht empfangen werden kann. Die Zuständigkeit für die Nachricht wird dann an IMS zurückgegeben. Wenn Sie Commitmodus 0 verwenden, werden Nachrichten aus dieser Transaktionspipe nicht an die Brücke gesendet und bleiben in der IMS-Warteschlange; d. h., bis zum Neustart werden keine weiteren Nachrichten gesendet. Wenn Sie Commitmodus 1 verwenden, können andere Vorgänge fortgesetzt werden.

Wenn die Antwort eine MQIIH-Struktur hat, lautet ihr Formattyp MQFMT_IMS; andernfalls wird ihr Formattyp vom IMS-MOD-Namen verwendet, der beim Einfügen der Nachricht verwendet wurde.

Alternative Antwort-PCBs in IMS-Transaktionen verwenden
Wenn eine IMS-Transaktion alternative Antwort-PCBs verwendet (Ausgabe von ISRT-Befehlen an den alternativen primären Steuerblock oder Ausgabe eines CHNG-Aufrufs an einen änderbaren primären Steuerblock), wird der Pre-Routing-Exit (DFSYPRX0) aufgerufen, um festzustellen, ob die Nachricht umgeleitet werden sollte.

Soll die Nachricht umgeleitet werden, wird der Exit für die Zielauflösung (DFSYDRU0) aufgerufen, um das Ziel zu bestätigen und die Headerinformationen vorzubereiten. Informationen zu diesen Exitprogrammen finden Sie unter OTMA-Exits in IMS verwenden und Pre-Routing-Exit DFSYPRX0.

Solange keine Aktion in den Exits vorgenommen wird, werden alle Ausgaben aus IMS-Transaktionen, die in einem IBM MQ-Warteschlangenmanager initialisiert wurden - ob nun Ausgaben an den primären E/A-Steuerblock oder an einen alternativen primären Steuerblock -, an denselben Warteschlangenmanager zurückgegeben.

Nicht erwartete Nachrichten aus IMS senden
Zum Senden von Nachrichten aus IMS an eine IBM MQ-Warteschlange müssen Sie eine IMS-Transaktion aufrufen, die den Befehl ISRT an einen alternativen primären Steuerblock (ALTPCB) ausgibt.

Sie müssen Pre-Routing-Exits und Exits für die Zielauflösung schreiben, um nicht erwartete Nachrichten aus IMS weiterzuleiten und die OTMA-Benutzerdaten zu erstellen, damit der MQMD der Nachricht ordnungsgemäß erstellt werden kann. Informationen zu diesen Exitprogrammen finden Sie unter Pre-Routing-Exit DFSYPRX0 und Benutzer-Exit für die Zielauflösung.

Anmerkung: Die IBM MQ - IMS-Brücke weiß nicht, ob es sich bei einer von ihr empfangenen Nachricht um eine Antwort um oder eine nicht erwartete Nachricht handelt. Sie bearbeitet die Nachricht in allen Fällen gleich und erstellt den MQMD und den MQIIH der Antwort auf Grundlage der OTMA-Benutzerdaten, die mit der Nachricht empfangen werden.

Nicht erwartete Nachrichten können neue Transaktionspipes erstellen. Wenn beispielsweise eine vorhandene IMS-Transaktion auf ein neues logisches Terminal wechselt (z. B. PRINT01), aber die Implementierung die Bereitstellung der Ausgabe über OTMA erfordert, wird eine neue Transaktionspipe erstellt (in diesem Beispiel PRINT01). Standardmäßig handelt es sich hierbei um eine nicht synchronisierte Transaktionspipe. Wenn die Implementierung eine wiederherstellbare Nachricht erfordert, legen Sie das Ausgabeattribut des Exits für die Zielauflösung fest. Weitere Informationen hierzu finden Sie im Handbuch *IMS Customization Guide*.

Nachrichtensegmentierung
Sie können IMS-Transaktionen so definieren, dass Eingaben in einzelnen oder mehreren Segmenten erwartet werden.

Die ursprüngliche IBM MQ-Anwendung muss die Benutzereingabe nach der MQIIH-Struktur als mindestens ein LLZZ-Datensegment konstruieren. Alle Segmente einer IMS-Nachricht müssen in einer einzelnen IBM MQ-Nachricht enthalten sein, die mit einem einzelnen MQPUT-Aufruf gesendet wird.

Die maximale Länge eines LLZZ-Datensegments wird durch IMS/OTMA (32767 Byte) definiert. Die Gesamtlänge der IBM MQ-Nachricht setzt sich aus der Summe der LL-Bytes plus der Länge der MQIIH-Struktur zusammen.

Alle Segmente der Antwort sind in einer einzelnen IBM MQ-Nachricht enthalten.

Es gibt eine weitere Beschränkung der Begrenzung auf 32 KB bei Nachrichten mit dem Format MQFMT_IMS_VAR_STRING. Wenn die Daten in einer ASCII-gemischten CCSID-Nachricht in eine EBCDIC-gemischte CCSID-Nachricht konvertiert werden, wird bei jedem Wechsel zwischen SBCS- und DBCS-Zeichen immer ein DBCS-Endebyte bzw. ein DBCS-Startbyte hinzugefügt. Die Einschränkung auf 32 KB gilt für die maximale Größe der Nachricht: Da das LL-Feld in der Nachricht 32 KB nicht überschreiten kann, darf die Nachrichtengröße maximal 32 KB einschließlich aller DBCS-Endezeichen und DBCS-Startzeichen betragen. Die Anwendung, mit der die Nachricht erstellt wird, muss dies unterstützen.

Datenkonvertierung
Die Datenkonvertierung wird entweder von der Funktion der verteilten Steuerung von Warteschlangen (die u.U. alle notwendigen Exits aufruft) oder vom gruppeninternen Warteschlangensteuerungsagenten vorgenommen (dieser unterstützt keine Verwendung von Exits), wenn eine Nachricht in eine Zielwarteschlange mit definierten XCF-Informationen für ihre Speicherklasse eingereiht wird. Die Datenkonvertierung findet nicht statt, wenn eine Nachricht mittels Publish/Subscribe an eine Warteschlange übermittelt wird.

Benötigte Exits müssen der Funktion für die verteilte Steuerung von Warteschlangen in der von der CSQXLIB-DD-Anweisung referenzierten Datei zur Verfügung stehen. Sie können also Nachrichten von jeder IBM MQ-Plattform unter Verwendung der IBM MQ - IMS-Bridge an eine IMS-Anwendung senden.

Kommt es zu Konvertierungsfehlern, wird die Nachricht unkonvertiert in die Warteschlange eingereiht; dies führt letztlich dazu, dass sie von der IBM MQ - IMS-Bridge als Fehler behandelt wird, da die Bridge das Headerformat nicht erkennen kann. Tritt ein Konvertierungsfehler auf, wird eine Fehlernachricht an die z/OS-Konsole gesendet.

Ausführliche Informationen zur Datenkonvertierung allgemein finden Sie unter „Datenkonvertierungsexits schreiben” auf Seite 1033.

Nachrichten an die IBM MQ - IMS-Bridge senden

Um sicherzustellen, dass die Konvertierung ordnungsgemäß ausgeführt wird, müssen Sie dem Warteschlangenmanager das Format der Nachricht mitteilen.

Wenn die Nachricht eine MQIIH-Struktur aufweist, muss das *Format* im MQMD auf das integrierte Format MQFMT_IMS gesetzt werden und das *Format* im MQIIH auf den Namen des Formats, das Ihre Nachrichtendaten beschreibt. Ist kein MQIIH vorhanden, setzen Sie das *Format* im MQMD auf Ihren Formatnamen.

Wenn es sich bei Ihren Daten (mit Ausnahme der LLZZs) nur um Zeichendaten (MQCHAR) handelt, verwenden Sie das integrierte Format MQFMT_IMS_VAR_STRING als Ihren Formatnamen (im MQIIH oder im MQMD). Andernfalls verwenden Sie Ihren eigenen Formatnamen. In diesem Fall müssen Sie ein Datenkonvertierungsexit für Ihr Format bereitstellen. Zusätzlich zu den eigentlichen Daten muss der Exit die Konvertierung der LLZZs in Ihrer Nachricht ausführen (muss aber keinen MQIIH am Anfang der Nachricht bearbeiten).

Wenn Ihre Anwendung *MFSMapName* verwendet, können Sie stattdessen Nachrichten mit MQFMT_IMS verwenden und den Mapnamen definieren, der im MQIIH-Feld MFSMapName an die IMS-Transaktion übermittelt wurde.

Nachrichten von der IBM MQ - IMS-Bridge empfangen

Ist eine MQIIH-Struktur in der ursprünglichen Nachricht vorhanden, die Sie an IMS senden, ist auch eine in der Antwortnachricht vorhanden.

So stellen Sie die ordnungsgemäße Konvertierung Ihrer Antwort sicher:

- Wenn Ihre ursprüngliche Nachricht eine MQIIH-Struktur enthält, geben Sie das gewünschte Format für Ihre Antwortnachricht im MQIIH-Feld *ReplytoFormat* der ursprünglichen Nachricht an. Dieser Wert wird in das MQIIH-Feld *Format* der Antwortnachricht eingetragen. Dies ist besonders nützlich, wenn alle Ihre Ausgabedaten das Format LLZZ<Zeichendaten> haben.
- Wenn Ihre ursprüngliche Nachricht keine MQIIH-Struktur enthält, geben Sie das gewünschte Format für Ihre Antwortnachricht als MFS MOD-Name im Befehl ISRT an, den Sie an den primären E/A-Steuerblock der IMS-Anwendung ausgeben.

IMS-Transaktionsprogramme in IBM MQ schreiben
Die Codierung, die für die Verarbeitung von IMS-Transaktionen durch IBM MQ erforderlich ist, hängt von dem von der IMS-Transaktion benötigten Nachrichtenformat und von dem Nachrichtenspektrum ab, das die Transaktion zurückgeben kann. Wenn Ihre Anwendung die Bildschirmformatinformationen von IMS verarbeitet, müssen Sie jedoch einige Punkte berücksichtigen.

Wenn eine IMS-Transaktion von einer 3270-Anzeige gestartet wird, wird die Nachricht über die Nachrichtenformatservices von IMS weitergeleitet. Aus Sicht der Transaktion können dadurch alle Anschlussabhängigkeiten vom Datenstrom entfernt werden. Wenn eine Transaktion über OTMA gestartet wird, ist MFS nicht eingebunden. Wenn in MFS Anwendungslogik implementiert ist, muss sie in der neuen Anwendung erneut erstellt werden.

Bei einigen IMS-Transaktionen kann die Endbenutzeranwendung gewisse Verhaltensweisen der 3270-Anzeige beeinflussen, beispielsweise ein Feld mit ungültigen Daten hervorheben. Diese Art der Informationen werden für jedes Anzeigefeld, das durch das Programm geändert werden muss, durch Hinzufügung eines zwei Byte großen Attributfelds zur IMS-Nachricht kommuniziert.

Wenn Sie daher eine Anwendung codieren, um einen 3270 nachzubilden, müssen Sie diese Felder beim Erstellen oder Empfangen von Nachrichten berücksichtigen.

Möglicherweise müssen Sie in Ihrem Programm Informationen für folgende Verarbeitungsvorgänge codieren:

- Welche Taste wird gedrückt (z. B. Eingabetaste und PF1)
- Wo ist der Cursor, wenn die Nachricht an Ihre Anwendung übermittelt wird
- Ob die Attributfelder durch die IMS-Anwendung festgelegt wurden
  - Intensität hoch, normal oder null
  - Farbe
  - Ob IMS das Feld beim nächsten Drücken der Eingabetaste zurück erwartet
- Ob die IMS-Anwendung in einem der Felder Nullzeichen (X'3F') verwendet hat.

Wenn Ihre IMS-Nachricht nur Zeichendaten enthält (abgesehen vom LLZZ-Datensegment) und Sie eine MQIIH-Struktur verwenden, müssen Sie das MQMD-Format auf MQFMT_IMS und das MQIIH-Format auf MQFMT_IMS_VAR_STRING setzen.

Wenn Ihre IMS-Nachricht nur Zeichendaten enthält (abgesehen vom LLZZ-Datensegment) und **keine** MQIIH-Struktur verwendet wird, setzen Sie das MQMD-Format auf MQFMT_IMS_VAR_STRING und stellen Sie sicher, dass die IMS-Anwendung in ihrer Antwort den MOD-Namen MQFMT_IMS_VAR_STRING angibt.Wenn IMS bei Auftreten eines Problems (z. B. Benutzer nicht zur Verwendung der Transaktion autorisiert) eine Fehlernachricht sendet, hat diese einen MODname im Format DFSMOx, wobei x eine Zahl zwischen 1 und 5 ist. Diese wird im MQMD-Format eingereiht.

Wenn Ihre IMS-Nachricht Binär-, gepackte oder Fließkommadaten enthält (abgesehen vom LLZZ-Datensegment), müssen Sie Ihre eigenen Datenkonvertierungsroutinen codieren. Informationen zur Bildschirmformatierung von IMS finden Sie im Abschnitt *IMS/ESA Application Programming: Transaction Manager*.

Wenn Sie Code für die Verarbeitung von IMS-Transaktionen durch IBM MQ schreiben, lesen Sie die folgenden Abschnitte.

- „IBM MQ-Anwendungen zum Aufrufen von IMS-Dialogtransaktionen schreiben” auf Seite 952
- „Programme schreiben, die IMS-Befehle enthalten” auf Seite 952
- „Auslösefunktion” auf Seite 952

IBM MQ-Anwendungen zum Aufrufen von IMS-Dialogtransaktionen schreiben

Die folgenden Informationen sind Anhaltspunkte beim Schreiben von IBM MQ-Anwendungen zum Aufrufen von IMS-Dialogtransaktionen.

Beachten Sie folgende Punkte beim Schreiben einer Anwendung, die einen IMS-Dialog aufruft:

- Schließen Sie in Ihrer Anwendungsnachricht eine MQIIH-Struktur ein.
- Setzen Sie den *CommitMode* in MQIIH auf MQICM_SEND_THEN_COMMIT.
- Setzen Sie *TranState* in MQIIH auf MQITS_NOT_IN_CONVERSATION, um einen neuen Dialog aufzurufen.
- Um einen zweiten und nachfolgenden Schritt eines Dialogs aufzurufen, setzen Sie *TranState* auf MQITS_IN_CONVERSATION und *TranInstanceId* auf den Wert des im vorherigen Schritt des Dialogs zurückgegebenen Felds.
- IMS bietet keine wirklich einfache Methode zur Ermittlung des Werts einer *TranInstanceId*, wenn Sie die ursprünglich von IMS übertragene Nachricht verloren haben.
- Die Anwendung muss den *TranState* der von IMS gesendeten Nachrichten überprüfen, um festzustellen, ob die IMS-Transaktion den Dialog beendet hat.
- Zum Beenden eines Dialogs können Sie /EXIT verwenden. Darüber hinaus müssen Sie in diesem Fall die *TranInstanceId* angeben, *TranState* auf MQITS_IN_CONVERSATION setzen und die IBM MQ-Warteschlange verwenden, über die der Dialog ausgeführt wird.
- Zum Anhalten oder Freigeben eines Dialogs können Sie nicht /HOLD oder /REL verwenden.
- Durch die IBM MQ/IMS-Bridge aufgerufene Dialoge werden beim Neustart von IMS beendet.

Programme schreiben, die IMS-Befehle enthalten

Ein Anwendungsprogramm kann anstelle einer Transaktion auch eine IBM MQ-Nachricht im Format LL*ZZBefehl* erstellen, wobei *Befehl* das Format /DIS TRAN PART oder /DIS POOL ALL hat.

Auf diese Weise können die meisten IMS-Befehle ausgegeben werden. Nähere Informationen hierzu finden Sie auch im Abschnitt *IMS V11 Communications and Connections*. Die Befehlsausgabe wird in der Antwortnachricht von IBM MQ im einem Textformat übermittelt, als ob sie zur Ausgabe an ein 3270-Terminal gesendet werden würde.

OTMA hat eine spezielle Form des IMS-Anzeigetransaktionsbefehls implementiert, der die Ausgabe in einem strukturierten Format zurückgibt. Eine ausführliche Beschreibung dieses Formats finden Sie in *IMS V11 Communications and Connections*. Zum Aufrufen dieses Formats aus einer IBM MQ-Nachricht erstellen Sie die Nachrichtendaten wie zuvor, zum Beispiel /DIS TRAN PART, setzen das Feld TranState in der MQIIH-Struktur dann aber auf MQITS_ARCHITECTED. IMS verarbeitet den Befehl und gibt die Antwort im strukturierten Format zurück. Eine strukturierte Antwort enthält alle Informationen, die im Textformat der Ausgabe zu finden sind, und eine zusätzliche Information: ob die Transaktion als wiederherstellbar oder nicht wiederherstellbar definiert wird.

Auslösefunktion

Die IBM MQ/IMS-Bridge unterstützt keine Auslösenachrichten.

Wenn Sie eine Initialisierungswarteschlange definieren, die eine Speicherklasse mit XCF-Parametern verwendet, werden Nachrichten, die in diese Warteschlange eingereiht werden, abgelehnt, wenn sie bei der Bridge eintreffen.

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 959.

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 962.

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 954.

In den folgenden Abschnitten finden Sie Informationen zum Vorbereiten und Ausführen anderer Anwendungstypen als Clientanwendungen:

- „CICS- und Tuxedo-Anwendungen vorbereiten und ausführen" auf Seite 975
- „Microsoft Transaction Server-Anwendungen vorbereiten und ausführen" auf Seite 47
- „IBM MQ JMS-Anwendungen vorbereiten und ausführen" auf Seite 978

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung" auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

„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 48
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 743
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 840
Beginnen Sie nun, Ihre eigenen IBM MQ-Publish/Subscribe-Anwendungen zu schreiben.

„Prozedurale Anwendung erstellen" auf Seite 1051
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 1103
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 1124

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

„Web-Services mit IBM MQ entwickeln” auf Seite 1368
Sie können IBM MQ-Anwendungen für Web-Services mithilfe des IBM MQ-Transports für SOAP entwickeln.

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 959
Mit dem Aufruf MQCONNX können Sie eine Struktur für die Kanaldefinition (MQCD) in der MQCNO-Struktur angeben.

Zugehörige Informationen

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. Zum Überschreiben der CCSID verwenden Sie die Umgebungsvariable MQCCSID. 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. Weitere Informationen finden Sie in „IBM MQ classes for Java-Clientverbindungen” auf Seite 363.

Der Clientcode geht davon aus, dass die auf dem Client über die MQI übertragenen Zeichendaten im CCSID dieser Workstation vorliegen. Wird die CCSID nicht unterstützt oder handelt es sich nicht um die erforderliche CCSID, kann sie mit einem der folgenden Befehle durch die Umgebungsvariable MQCCSID überschrieben werden:

- Windows

```
SET MQCCSID=850
```

- UNIX

```
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. Details enthält der Abschnitt 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:

- Sowohl der Client als auch der ferne Warteschlangenmanager müssen IBM WebSphere MQ 7 oder höher haben.
- 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 im Abschnitt „Leistung nicht persistenter Nachrichten verbessern” auf Seite 816.

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 1144.

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 im Abschnitt MQCNO-Struktur eines MQCONNX-Aufrufs verwenden. 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 1010.

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.

Weitere Informationen finden Sie unter „MQCONNX verwenden” auf Seite 959.

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 118. In Clientumgebungen unterstützte Programmiersprachen

Clientplattform	C	C + +	COBOL	pTAL	RPG	Visual Basic
AIX	Ja	Ja	Ja			
HP-UX	Ja	Ja	Ja			
IBM i	Ja		Ja		Ja	
Linux	Ja	Ja	Ja			
Solaris	Ja	Ja	Ja			

Tabelle 118. In Clientumgebungen unterstützte Programmiersprachen (Forts.)						
Clientplattform	**C**	**C + +**	**COBOL**	**pTAL**	**RPG**	**Visual Basic**
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

HP-UX **HP-UX**
Bibliothek libmqic.sl für Nicht-Thread-Anwendungen bzw. Bibliothek libmqic_r.sl 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.

Solaris **Solaris**
libmqic.so.

Wenn Sie die Programme auf einem System verwenden möchten, auf dem nur der IBM MQ MQI client for Solaris installiert ist, müssen Sie die Programme neu kompilieren, um sie mit der Clientbibliothek zu verknüpfen:

```
$ /opt/SUNWspro/bin/cc -o prog_name prog_name c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

Die Parameter müssen in der angegebenen Reihenfolge eingegeben werden.

Windows **Windows**
MQIC32.LIB.

ULW 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.

HP-UX **HP-UX**

Verknüpfen Sie eine Nicht-Thread-COBOL-Anwendung mit der Bibliothek libmqicb.sl und eine Thread-COBOL-Anwendung mit der Bibliothek libmqicb_r.sl.

HP-UX **Linux**

Verknüpfen Sie eine Nicht-Thread-COBOL-Anwendung mit der Bibliothek libmqicb.so und eine Thread-COBOL-Anwendung mit der Bibliothek libmqicb_r.so.

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.

Solaris **Solaris**

Verknüpfen Sie eine Nicht-Thread-COBOL-Anwendung mit der Bibliothek libmqicb.so und eine Thread-COBOL-Anwendung mit der Bibliothek libmqicb_r.so.

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.

V 9.0.0 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 1119
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 1084

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 im Abschnitt „Verbindungsdefinitionen unter Verwendung eines Vorverbindungsexits aus einem Repository referenzieren” auf Seite 1044.
2. Wenn die Umgebungsvariable MQSERVER gesetzt ist, wird der von ihr angegebene Kanal verwendet.
3. Wenn eine `mqclient.ini`-Datei definiert ist und ein ServerConnectionParms-Attribut enthält, wird der dort definierte Kanal verwendet. Weitere Informationen finden Sie in den Abschnitten Client mithilfe einer Konfigurationsdatei konfigurieren und CHANNELS-Zeilengruppe der Clientkonfigurationsdatei.
4. Wenn die Umgebungsvariablen MQCHLLIB und MQCHLTAB gesetzt sind, wird die Clientkanaldefinitionstabelle verwendet, auf die diese Variablen verweisen. V 9.0.0 Alternativ bietet ab IBM MQ 9.0 die Umgebungsvariable MQCCDTURL die funktional entsprechende Möglichkeit zum Festlegen einer Kombination der Umgebungsvariablen MQCHLLIB und MQCHLTAB. Ist MQCCDTURL gesetzt, wird die Clientkanaldefinitionstabelle verwendet, auf die die Variable verweist. Weitere Informationen finden Sie im Abschnitt Webadressierbarer Zugriff auf die Definitionstabelle für den Clientkanal.

5. Wenn eine `mqclient.ini`-Datei definiert ist und die Attribute ChannelDefinitionDirectory und ChannelDefinitionFile enthält, wird die Clientkanaldefinitionstabelle mithilfe dieser Attribute gesucht. Weitere Informationen finden Sie in den Abschnitten Client mithilfe einer Konfigurationsdatei konfigurieren und CHANNELS-Zeilengruppe der Clientkonfigurationsdatei.
6. Wurden die Umgebungsvariablen nicht gesetzt, sucht der Client nach einer Clientkanaldefinitionstabelle mit einem Pfad und Namen, der der Zeilengruppe `DefaultPrefix` in der Datei `mqs.ini` entnommen wird. Verläuft die Suche nach einer Clientkanaldefinitionstabelle ergebnislos, verwendet der Client folgende Pfade:
   - Linux UNIX Unter UNIX and Linux: `/var/mqm/AMQCLCHL.TAB`
   - Windows Unter Windows: `C:\Programme\IBM\MQ\amqclchl.tab`
   - IBM i 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 Informationen

Definitionstabelle für den Clientkanal

V 9.0.0 Webadressierbarer Zugriff auf die Definitionstabelle für den Clientkanal

MQSERVER

MQCHLLIB

MQCHLTAB

V 9.0.0 MQCCDTURL

Verbindungen zwischen dem Server und dem Client konfigurieren

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
Die Umgebungsvariable MQSERVER wird zur Definition eines minimalen Kanals verwendet. MQSERVER gibt die Position des IBM MQ-Servers sowie das zu verwendende Kommunikationsverfahren an.

MQCHLLIB
Die Umgebungsvariable MQCHLLIB gibt den Verzeichnispfad für die Datei an, die die Definitionstabelle für den Clientkanal (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 Definitionstabelle für den Clientkanal (CCDT) enthält.

V 9.0.0 Ab IBM MQ 9.0 bietet die Umgebungsvariable MQCCDTURL die funktional entsprechende Möglichkeit zum Festlegen einer Kombination der Umgebungsvariablen MQCHLLIB und MQCHLTAB. Mit MQCCDTURL können Sie eine Datei-, FTP- oder HTTP-URL als einzelnen Wert bereitstellen, von dem aus

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 im Abschnitt MQCNO-Struktur eines MQCONNX-Aufrufs verwenden.

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.

V 9.0.0 Ab IBM MQ 9.0 kann die CCDT an einer zentralen Position gehostet werden, auf die über eine URI zugegriffen werden kann, womit die Notwendigkeit entfällt, die CCDT für jeden implementierten Client einzeln zu aktualisieren.

Zugehörige Informationen
Definitionstabelle für den Clientkanal

V 9.0.0 Webadressierbarer Zugriff auf die Definitionstabelle für den Clientkanal

Zugreifen auf Clientverbindungskanaldefinitionen

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. Die automatische Wiederherstellung einer Clientverbindung 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 107 auf Seite 966 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`.

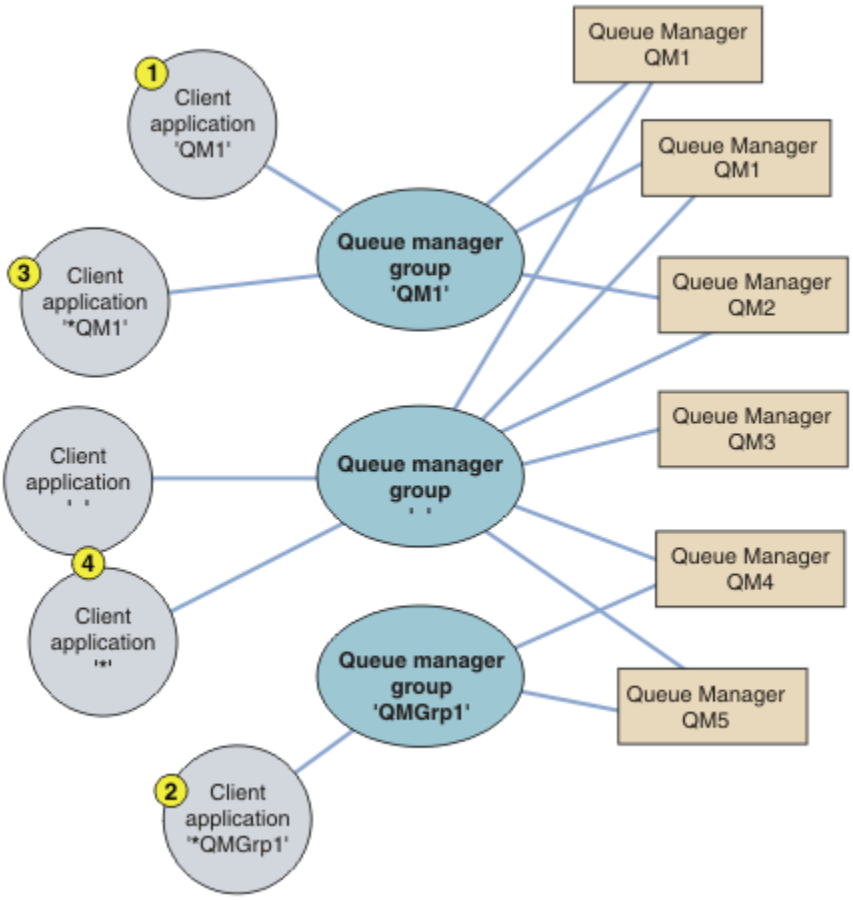


Abbildung 107. WS-Manager-Gruppen

Vier Beispiele für die Verwendung dieser Clientverbindungstabelle werden mithilfe der nummerierten Clientanwendungen in Abbildung 107 auf Seite 966 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 *Verbindung zu Gruppen mit gemeinsamer Warteschlange herstellen*

Sie können eine Anwendung mit einem Warteschlangenmanager verbinden, der Teil einer Gruppe mit gemeinsamer Warteschlange ist. Zu diesem Zweck geben Sie im MQCONN- oder MQCONNX-Aufruf statt des Warteschlangenmanagernamens den Namen der Gruppe mit gemeinsamer Warteschlange an.

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

Die Clientkanaldefinition sollte die generische Schnittstelle der Gruppe mit gemeinsamer Warteschlange zur Verbindung mit einem verfügbaren Warteschlangenmanager der Gruppe verwenden. Weitere Informationen finden Sie im Abschnitt Client mit einer Gruppe mit gemeinsamer Warteschlange verbinden. Durch eine Prüfung wird sichergestellt, dass der Warteschlangenmanager, zu dem das Empfangsprogramm eine Verbindung herstellt, ein Mitglied der Gruppe mit gemeinsamer Warteschlange ist.

Weitere Informationen zu gemeinsam genutzten Warteschlangen finden Sie im Abschnitt Gemeinsam genutzte Warteschlangen und Gruppen mit gemeinsamer Warteschlange.

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.

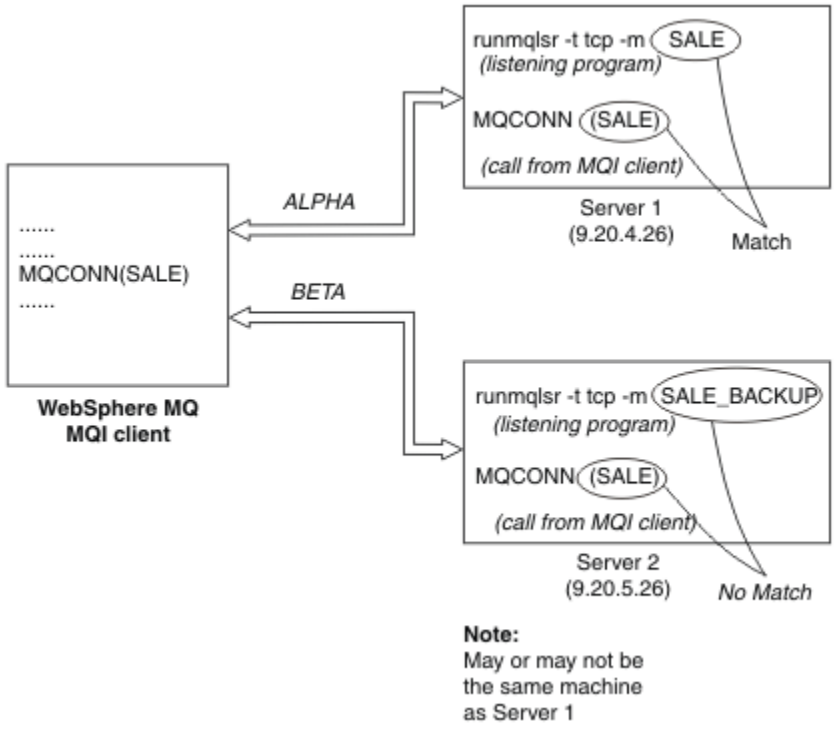


Abbildung 108. Beispiel für MQCONN-Aufruf

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 einen 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 Warteschlangenmanagers 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 einen Stern (*)" auf Seite 972.

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. Der `Warteschlangenmanagername` in der Definition lautet 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. Insbesondere werden die Nachrichten, die von einer Clientanwendung in eine ausgelöste Warteschlange eingereiht werden, die für einen z/OS-Warteschlangenmanager lokal ist, innerhalb einer Arbeitseinheit eingereiht, wenn in der Anwendung keine MQPMO-Optionen für die Synchronisationspunktsteuerung angegeben sind. 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 Windows-Client und möchte er eine Anforderung an einen Server unter einem anderen Betriebssystem senden, muss MQAT_WINDOWS_NT angegeben werden. Andernfalls verwendet das Betriebssystem des Servers seine Standarddefinitionen, und der Prozess schlägt fehl.

Auslösemonitor
Der von nicht unter z/OS ausgeführten IBM MQ-Produkten bereitgestellte Auslösemonitor wird in Clientumgebungen unter IBM i IBM i, UNIX, Linux, and Windows 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)
```

- ULW Unter Windows, UNIX and Linux:

```
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 UNIX 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 Clientverbindung mit der Umgebungsvariablen MQSERVER definiert wird, muss diese 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. Für die Anwendung, die auf einem IBM MQ MQI client ausgeführt wird, müssen **MQCMIT** und **MQBACK** verwendet werden.

Windows UNIX 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. Beispiel:

– UNIX Verwenden Sie für in C geschriebene Anwendungen unter UNIX die Bibliothek 'libmqic' (statt 'libmqm').

– Windows Oder unter Windows die Bibliothek mqic.lib (statt mqm.lib).

- Verwenden Sie statt der in Tabelle 119 auf Seite 976 und Tabelle 120 auf Seite 976 auf geführten Serversystembibliotheken die entsprechenden Clientsystembibliotheken. Ist eine Serversystembibliothek nicht in diesen Tabellen aufgeführt, verwenden Sie die entsprechende Bibliothek für Clientsysteme.

Tabelle 119. Clientsystembibliotheken unter UNIX

Bibliothek für ein IBM MQ-Serversystem	**Entsprechende Bibliothek für ein IBM MQ-Clientsystem**
libmqmxa	libmqcxa

Tabelle 120. 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.

Windows UNIX *Beispielprogramme für CICS und Tuxedo*

CICS- und Tuxedo-Beispielprogramme zur Verwendung auf Systemen mit UNIX und Windows.

In Tabelle 121 auf Seite 976 finden Sie eine Liste der CICS- und Tuxedo-Beispielprogramme für UNIX-Clientsysteme. In Tabelle 122 auf Seite 977 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 1147 und „TUXEDO-Beispiele unter UNIX und Windows verwenden” auf Seite 1193.

Tabelle 121. Beispielprogramme für UNIX-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	-

Tabelle 121. Beispielprogramme für UNIX-Clientsysteme (Forts.)		
Beschreibung	**Quelle**	**Ausführbares Modul**
Datei mit den Ansichtsdefinitionen für die Tuxedo-Programme	amqstxvx.v	-

Tabelle 122. 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	-

Windows UNIX *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

Beschreibung

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.

Benutzeraktion

Beheben Sie den Fehler und wiederholen Sie den Vorgang.

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 1081 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 553 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 1084 vorgehen, bei der Definition des Visual Basic-Projekts im Feld **Argumente für bedingte Kompilierung** jedoch `MqType=3` eingeben.
- Verwendet die Komponente IBM MQ Automation Classes for ActiveX (MQAX), definieren Sie die Umgebungsvariable GMQ_MQ_LIB mit dem Wert `mqic32xa.dll`.

  Sie können die Umgebungsvariable in der Anwendung definieren oder so, dass sie systemweit gilt. Eine systemweit geltende Definition kann allerdings zu nicht ordnungsgemäßem Verhalten von MQAX-Anwendungen führen, bei denen die Umgebungsvariable nicht innerhalb der Anwendung definiert wurde.

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 verwenden” auf Seite 77.

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 Informationen
Kanalexitprogramme für MQI-Kanäle
API-Exitreferenz
Referenzinformationen zu installierbaren Services
IBM i Referenzinformationen zu installierbaren Services unter IBM i

ULW Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben

Sie können Exits schreiben und kompilieren, ohne eine Verbindung zu IBM MQ -Bibliotheken unter UNIX, Linux und Windowsherzustellen.

Informationen zu diesem Vorgang

Dieser Abschnitt gilt nur für Systeme unter UNIX, 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 UNIX, 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 UNIX 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 1001.

- 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 1012.
- 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 auf UNIX and Linux-Systemen schreiben" auf Seite 1038.
  - 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 1033.
- 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 1044.
- 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 1124.

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 1003.
     - Kanalexits, Veröffentlichungsexits, Exits für Clusterauslastung: „Kanalexitprogramme auf Windows, UNIX and Linux-Systemen kompilieren” auf Seite 1032.
     - Datenkonvertierungsexits: „Datenkonvertierungsexits schreiben” auf Seite 1033.
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 990
   - API-Exits: „API-Exits konfigurieren” auf Seite 1007.
   - Kanalexits: „Kanalexits konfigurieren” auf Seite 1033
   - Veröffentlichungsexits: „Veröffentlichungsexits konfigurieren” auf Seite 1047
   - Pre-Connect-Exits: Pre-Connect-Zeilengruppe der Clientkonfigurationsdatei.

ULW *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 Än-

derung des Programmcodes des Exits planen, damit stattdessen die MQIEP-Eingangspunkte verwendet werden.

Ab IBM MQ 8.0 ist **runmqsc** ein Beispiel für ein Programm, das nicht direkt mit einer MQI-Bibliothek verlinkt 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

„Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben” auf Seite 979
Sie können Exits schreiben und kompilieren, ohne eine Verbindung zu IBM MQ -Bibliotheken unter UNIX, Linux und Windowsherzustellen.

ULW Installierbare Services und Komponenten für UNIX, Linux und 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 123 auf Seite 983 sehen Sie eine Zusammenfassung der installierbaren Services und ihrer Komponenten.

Tabelle 123. Zusammenfassung der installierbaren Services und ihrer Komponenten			
Installierbarer Service	**Bereitgestellte Komponente**	**Funktion**	**Anforderungen**
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)
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 Informationen

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 987. 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 109 auf Seite 984 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.

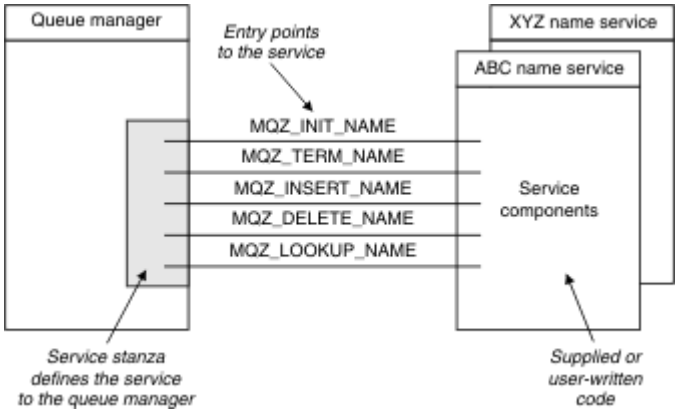


Abbildung 109. Zusammenhang zwischen Services, Komponenten und Eingangspunkten

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 UNIX 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 UNIX 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.

Rückgabecodes

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)
Mit den IBM MQ-Produkten wird als Berechtigungsservicekomponente der Objektberechtigungsmanager (OAM, Object Authority Manager) bereitgestellt.

Der Objektberechtigungsmanager ist standardmäßig aktiv und arbeitet mit den Steuerbefehlen **dspmqaut** (Bildschirmberechtigung), **dmpmqaut** (Speicherauszugsberechtigung) und **setmqaut** (Einstellungs- und Zurücksetzungsberechtigung).

Die Syntax dieser Befehle und deren Verwendung werden im Abschnitt Referenzinformationen zu IBM MQ-Steuerbefehlen beschrieben.

Der Objektberechtigungsmanager arbeitet mit der *Entität* eines Principals oder einer Gruppe:

- Linux UNIX Unter UNIX 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 124 auf Seite 990 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 124 auf Seite 990). 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 124. 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 seine eigene Zeilengruppe in der Registrierung hat.

1. Fügen Sie der Konfigurationsdatei des Warteschlangenmanagers Zeilengruppen hinzu, mit denen der Service gegenüber dem Warteschlangenmanager definiert und die Speicherposition des Moduls angegeben wird.

   Diese Datei muss für jeden verwendeten Service die Zeilengruppe *Service* enthalten, die den Service für den Warteschlangenmanager definiert.

   Für jede Komponente innerhalb eines Service muss die Zeilengruppe *ServiceComponent* vorhanden sein. Sie gibt den Namen und den Pfad des Moduls an, in dem der Code für die betreffende Komponente enthalten ist.

   Weitere Informationen finden Sie in den Abschnitten „Format der Zeilengruppe 'Service'" auf Seite 991 und „Format der Zeilengruppe 'ServiceComponent'" auf Seite 991.

   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.

Format der Zeilengruppe 'Service'
Die Zeilengruppe 'Service' enthält den Namen des Service und die Anzahl der für den Service definierten Eingangspunkte.

Die Zeilengruppe hat folgendes Format:

```
Service:
  Name=service_name
  EntryPoints=entries
  SecurityPolicy=policy
```

Dabei gilt:

service_name
Der Name des Service. Dieser wird durch den Service definiert.

entries
Die Anzahl der Eingangspunkte, die für den Service definiert wurden. Dazu gehören die Initialisierungs-und Beendigungseingangspunkte

policy

Linux UNIX Unter UNIX and Linux: `user`, `group` oder `default`. Der Wert gibt an, ob der Warteschlangenmanager benutzerbasierte oder gruppenbasierte Berechtigungen verwendet. Bei Werten wird die Groß-/Kleinschreibung nicht beachtet. Fehlt dieses Attribut, werden standardmäßig gruppenbasierte Berechtigungen verwendet. Starten Sie den Warteschlangenmanager neu, damit die Änderungen wirksam werden. Weitere Informationen hierzu finden Sie im Abschnitt „Zeilengruppen für Berechtigungsservice unter UNIX and Linux konfigurieren” auf Seite 992.

Windows Unter Windows: `NTSIDsRequired` (die Sicherheitskennung von Windows) oder `Default`. Ohne Angabe von `NTSIDsRequired` wird der Wert `Default` verwendet. Dieses Attribut ist nur gültig, wenn **Name** den Wert `AuthorizationService` hat. Weitere Informationen hierzu finden Sie im Abschnitt „Zeilengruppen für Berechtigungsservice unter Windows konfigurieren” auf Seite 992.

Format der Zeilengruppe 'ServiceComponent'
Die Zeilengruppen **Service** und **ServiceComponent** können in beliebiger Reihenfolge aufgeführt werden.

Die Zeilengruppe 'ServiceComponent' hat folgendes Format:

```
ServiceComponent:
  Service=service_name
  Name=component_name
  Module=module_name
  ComponentDataSize=size
```

Dabei gilt:

Servicename
Der Name des Service. Dieser muss dem in einer 'Service'-Zeilengruppe angegebenen Wert für `Name` entsprechen.

Komponentenname
Der beschreibende Name der Servicekomponente. Dieser Name muss eindeutig sein und darf nur Zeichen enthalten, die für die Namen von IBM MQ-Objekten (z. B. Warteschlangennamen) gültig sind. Dieser Name tritt in Bedienernachrichten auf, die durch den Service generiert werden. Es wird empfohlen, einen Namen zu verwenden, der mit einer Unternehmensmarke oder einer ähnlichen individuellen Zeichenfolge beginnt.

Modulname
Der Name des Moduls, das den Code für diese Komponente enthält.

Größe
Die Größe (in Byte) des Komponentendatenbereichs, der bei jedem Aufruf an die Komponente übergeben wird. Geben Sie Null an, wenn keine Komponentendaten erforderlich sind.

Die beiden Zeilengruppen **Service** und **ServiceComponent** können ebenso wie die darin enthaltenen Zeilengruppenschlüssel in beliebiger Reihenfolge aufgeführt werden. Für jede dieser Zeilengruppen müssen alle Zeilengruppenschlüssel vorhanden sein. Falls ein Zeilengruppenschlüssel mehrfach vorkommt, wird der letzte verwendet.

Beim Start verarbeitet der Warteschlangenmanager der Reihe nach alle Servicekomponenteneinträge der Konfigurationsdatei. Anschließend lädt er das angegebene Komponentenmodul, wobei er den Eingangspunkt für die Initialisierung der Komponente aufruft und ihm eine Konfigurationskennung übergibt.

Linux UNIX *Zeilengruppen für Berechtigungsservice unter UNIX and Linux konfigurieren*

Unter UNIX and Linux hat jeder Warteschlangenmanager seine eigene Konfigurationsdatei.

Standardpfad und Dateiname der warteschlangenmanagerspezifischen Konfigurationsdatei für den Warteschlangenmanager QMNAME lauten beispielsweise `/var/mqm/qmgrs/QMNAME/qm.ini`.

Die Zeilengruppen *Service* und *ServiceComponent* für die Standardberechtigungskomponente werden der Datei `qm.ini` automatisch hinzugefügt, können aber durch `mqsnoaut` überschrieben werden. Alle anderen *ServiceComponent*-Zeilengruppen müssen manuell hinzugefügt werden.

Die folgenden Zeilengruppen in der Konfigurationsdatei für Warteschlangenmanager definieren beispielsweise unter IBM MQ for AIX zwei Berechtigungsservicekomponenten: *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

```
  Service:
     Name=AuthorizationService
     EntryPoints=13

  ServiceComponent:
     Service=AuthorizationService
     Name=MQSeries.UNIX.auth.service
Module= MQ_INSTALLATION_PATH/lib/amqzfu
     ComponentDataSize=0

  ServiceComponent:
     Service=AuthorizationService
     Name=user.defined.authorization.service
     Module=/usr/bin/udas01
     ComponentDataSize=96
```

Abbildung 110. Zeilengruppen des UNIX and Linux-Berechtigungsservice in der Datei qm.ini

Die Zeilengruppe 'ServiceComponent' (`MQSeries.UNIX.auth.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.

Windows *Zeilengruppen für Berechtigungsservice unter Windows konfigurieren*

Unter IBM MQ for Windows hat jeder Warteschlangenmanager in der Registrierung seine eigene Zeilengruppe.

Die Zeilengruppen *Service* und *ServiceComponent* für die Standardberechtigungskomponente werden der Registrierungsdatenbank automatisch hinzugefügt, können aber durch `mqsnoaut` überschrieben werden. Alle anderen *ServiceComponent*-Zeilengruppen müssen manuell hinzugefügt werden.

Sie können das Attribut `SecurityPolicy` auch mithilfe der IBM MQ -Services hinzufügen. Das Attribut `SecurityPolicy` gilt nur, wenn der in der Zeilengruppe *Service* angegebene Service der Berechtigungsservice ist, d. h. der Standard-OAM. Mit dem Attribut `SecurityPolicy` können Sie für jeden Warteschlangenmanager die Sicherheitsrichtlinie angeben. Folgende Werte sind möglich:

Default
Geben Sie `Default` an, wenn die Standardsicherheitsrichtlinie gelten soll. Wenn dem OAM für eine bestimmte Benutzer-ID keine Windows-Sicherheitskennung (NT SID) übergeben wird, wird die passende SID in den relevanten Sicherheitsdatenbanken gesucht.

NTSIDsRequired
Verlangt bei Sicherheitsprüfungen, dass dem OAM eine NT SID übergeben wird.

Informationen zum Format der Zeilengruppe 'Service' finden Sie im Abschnitt „Format der Zeilengruppe 'Service'" auf Seite 991. Allgemeine Informationen zur Sicherheit erhalten Sie im Abschnitt Sicherheit auf Windows-, UNIX and Linux-Systemen einrichten.

Die Zeilengruppe 'ServiceComponent' (`MQSeries.WindowsNT.auth.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.

Linux UNIX *Zeilengruppen für Namensservice konfigurieren: Systeme unter UNIX and Linux*

Unter UNIX and Linux hat jeder Warteschlangenmanager seine eigene Konfigurationsdatei.

Die folgenden Zeilengruppenbeispiele für den Namensservice aus einer UNIX and Linux-Konfigurationsdatei geben eine Namensservicekomponente des (fiktiven) Unternehmens ABC an.

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=/usr/lib/abcname
    ComponentDataSize=1024
```

Abbildung 111. Zeilengruppen für den Namensservice aus der Datei 'qm.ini' (für UNIX and Linux-Systeme)

Anmerkung: Windows Unter Windows werden die Zeilengruppeninformationen für den Namensservice in der Registrierung gespeichert.

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 Informationen
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 125 auf Seite 994 enthält eine Zusammenfassung der Unterstützung für den Berechtigungsservice.

Tabelle 125. Zusammenfassung der Komponenten des Berechtigungsservice

Bereitgestellte Komponente	**Funktion**	**Anforderungen**
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 in Versionen von IBM MQ vor 6.0unterstützt.	• 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 über die Konfigurationsdateien des Warteschlangenmanagers konfiguriert. Diese Datei muss für jeden verwendeten Service die Zeilengruppe *Service* enthalten, die den Service für den Warteschlangenmanager definiert.

Für jede Komponente innerhalb eines Service muss die Zeilengruppe *ServiceComponent* vorhanden sein. Sie gibt den Namen und den Pfad des Moduls an, in dem der Code für die betreffende Komponente enthalten ist.

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.

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.

Format der Zeilengruppe 'Service'

Die Zeilengruppe **Service** hat folgendes Format:

```
Service:
  Name=service_name
  EntryPoints=entries
```

Dabei gilt:

Servicename
Der Name des Service. Dieser wird durch den Service definiert.

Eingangspunkte
Die Anzahl der Eingangspunkte, die für den Service definiert wurden. Dazu gehören die Initialisierungs-und Beendigungseingangspunkte

Format der Zeilengruppe 'ServiceComponent'

Die Zeilengruppe **Service component** hat folgendes Format:

```
ServiceComponent:
  Service=service_name
  Name=component_name
  Module=module_name
  ComponentDataSize=size
```

Dabei gilt:

Servicename
Der Name des Service. Dieser muss dem in einer 'Service'-Zeilengruppe angegebenen Wert für *Name* entsprechen.

Komponentenname
Der beschreibende Name der Servicekomponente. Dieser Name muss eindeutig sein und darf nur Zeichen enthalten, die für die Namen von IBM MQ-Objekten (z. B. Warteschlangennamen) gültig sind. Dieser Name tritt in Bedienernachrichten auf, die durch den Service generiert werden. Es wird empfohlen, einen Namen zu verwenden, der mit einer Unternehmensmarke oder einer ähnlichen individuellen Zeichenfolge beginnt.

Modulname
Der Name des Moduls, das den Code für diese Komponente enthält. Geben Sie einen vollständigen Pfadnamen an.

Größe
Die Größe (in Byte) des Komponentendatenbereichs, der bei jedem Aufruf an die Komponente übergeben wird. Geben Sie Null an, wenn keine Komponentendaten erforderlich sind.

Diese beiden Zeilengruppen können wie auch die darin enthaltenen Zeilengruppenschlüssel in jeder Reihenfolge auftreten. Für jede dieser Zeilengruppen müssen alle Zeilengruppenschlüssel vorhanden sein. Falls ein Zeilengruppenschlüssel mehrfach vorkommt, wird der letzte verwendet.

Beim Start verarbeitet der Warteschlangenmanager der Reihe nach alle Servicekomponenteneinträge der Konfigurationsdatei. Anschließend lädt er das angegebene Komponentenmodul, wobei er den Eingangspunkt für die Initialisierung der Komponente aufruft und ihm eine Konfigurationskennung übergibt.

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 998
- „Service im Betriebssystem definieren" auf Seite 998
- „Zeilengruppen für Berechtigungsservice konfigurieren" auf Seite 998
- „Berechtigungsserviceschnittstelle unter IBM i" auf Seite 999

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 996.

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 112. 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 997.

API-Exits 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: Nicht unterstützt unter IBM MQ for z/OS.

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 anzulegen. 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:

- Für eine erhöhte *Sicherheit* können Sie eine Authentifizierung bereitstellen, um zu überprüfen, ob Anwendungen berechtigt sind, auf eine Warteschlange oder einen Warteschlangenmanager zuzugreifen. Sie können auch die Verwendung der API durch die Anwendungen überwachen und die einzelnen API-Aufrufe oder sogar die dabei verwendeten Parameter authentifizieren.
- Für mehr *Flexibilität* können Sie auf kurzfristige Veränderungen in Ihrem Geschäftsumfeld reagieren, ohne die Anwendungen, die auf die Daten aus dem Geschäftsumfeld angewiesen sind, ändern zu müssen. Sie können beispielsweise API-Exits einsetzen, die auf Änderungen von Zinssätzen oder Wechselkursen oder auf Preisänderungen von Komponenten in einer Produktionsumgebung reagieren.
- Zur *Überwachung* einer Warteschlange oder eines Warteschlangenmanagers können Sie den Datenfluss von Anwendungen und Nachrichten verfolgen, Fehler in den API-Aufrufen protokollieren, Prüflisten zu Buchhaltungszwecken einrichten oder Nutzungsstatistiken zu Planungszwecken 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-Exitroutinen werden auf IBM i, Windows, UNIX and Linux-Systemen in Zeilengruppen angegeben. Dieser Abschnitt bezieht sich auf die Zeilengruppen in den Konfigurationsdateien 'mqs.ini' und 'qm.ini'.

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 1001 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 1001 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 1007 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 „Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben” auf Seite 979.

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.

API-Exits schreiben

Mithilfe der Programmiersprache C können Exits für jeden API-Aufruf geschrieben werden.

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.

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 1142.

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 „Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben” auf Seite 979.

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.

Anmerkung: Für die Verarbeitung von Exitnachrichteneigenschaften in:

- Eine Funktion nach MQ_GET_EXIT müssen Sie eine Funktion vor MQ_GET_EXIT für den Exit definieren.
- Eine Funktion vor MQ_CALLBACK_EXIT müssen Sie eine Funktion vor MQ_CB_EXIT für den Exit definieren.

Ein Musterprogramm (amqsaem0.c) wird bereitgestellt, um die Verwendung von Nachrichtenkennungen in API-Exits zu veranschaulichen.

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 1142 beschriebene Musterprogramm verwendet werden. Für Plattformen mit Ausnahme von Windows-Systemen finden Sie den API-Exit-Mustercode in *MQ_INSTALLATION_PATH*/samp und die kompilierte und verbundene gemeinsam genutzte Bibliothek in *MQ_INSTALLATION_PATH*/samp/bin. 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.

Hinweis für Benutzer:

1. Eine Anleitung zur Programmierung von 64-Bit-Anwendungen finden Sie im Abschnitt Codierungsstandards auf 64-Bit-Plattformen.

Durch die Einführung von Multicasting-Clients müssen API-Exits und Datenkonvertierungsexits clientseitig ausgeführt werden können, da manche Nachrichten den Warteschlangenmanager unter Umständen nicht durchlaufen. Die folgenden Bibliotheken sind jetzt sowohl Teil der Client- als auch der Serverpakete:

Tabelle 126. Bibliotheken, die jetzt in den Client- und Serverpaketen enthalten sind.

Betriebssystem	**Bibliotheken**
Windows	32 Bit & 64 Bit: mqm.dll & mqm.pdb
Linux & HP-UX	32 Bit & 64 Bit: libmqm.so & libmqm_r.so
AIX	32 Bit & 64 Bit: libmqm.a & libmqm_r.a
Solaris	32 Bit & 64 Bit: libmqm.so
IBM i IBM i	LIBMQM & LIBMQM_R

API-Exits auf Unix- und Linux-Systemen kompilieren

Beispiele zur Kompilierung von API-Exits auf UNIX 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

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
```

Auf HP-UX Itanium-Plattformen

32-Bit-Anwendungen

Nicht-thread-basiert

Kompilieren Sie den API-Exit-Quellcode:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie den API-Exit-Quellcode

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe
rm amqsaxe.o
```

Thread-basiert

Kompilieren Sie den API-Exit-Quellcode:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie den API-Exit-Quellcode

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe_r
rm amqsaxe.o
```

64-Bit-Anwendungen

Nicht-thread-basiert

Kompilieren Sie den API-Exit-Quellcode:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie den API-Exit-Quellcode

```
ld -b amqsaxe.o +ee MQStart -o /var/mqm/exits64/amqsaxe
rm amqsaxe.o
```

Thread-basiert

Kompilieren Sie den API-Exit-Quellcode:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie den API-Exit-Quellcode

```
ld -b amqsaxe.o +ee MQStart -o /var/mqm/exits64/amqsaxe_r
rm amqsaxe.o
```

unter 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
```

unter Solaris

Kompilieren Sie den API-Exit-Quellcode, indem Sie einen der folgenden Befehle ausgeben:

32-Bit-Anwendungen

SPARC-Plattform

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

x86-64-Plattform

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

64-Bit-Anwendungen

SPARC-Plattform

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
```

```
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

x86-64-Plattform

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

auf Windows-Systemen
So kompilieren und verbinden Sie das API-Exit-Musterprogramm `amqsaxe0.c` unter Windows

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 113 auf Seite 1006 oder Abbildung 114 auf Seite 1006 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 113. `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 114. `amqsaxe0.c` unter einer 64-Bit-Version von Windows kompilieren und verbinden

Zugehörige Konzepte

„API Exit-Beispielprogramm” auf Seite 1142
Das API Exit-Beispielprogramm generiert einen MQI-Trace zu einer benutzerdefinierten Datei mit einem in der Umgebungsvariable MQAPI_TRACE_LOGFILE definierten Präfix.

unter IBM i
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:

- Über den IBM MQ Explorer (Unter Windows und Linux (x86- und x86-64-Plattformen))
- Mithilfe des Befehls **amqmdain** (Unter Windows)
- Direkt über die Dateien 'mqs.ini' und 'qm.ini' (auf Systemen unter Windows, IBM i IBM i sowie unter UNIX and Linux).

  Die Datei 'mqs.ini' enthält Informationen, die für alle Warteschlangenmanager auf einem bestimmten Knoten relevant sind. Sie befindet sich im Verzeichnis `/var/mqm` unter UNIX and Linux IBM i, im Verzeichnis `/QIBM/UserData/mqm` unter IBM i und im Verzeichnis WorkPath , das im Schlüssel `HKLM\SOFTWARE\IBM\WebSphere MQ` auf Windows -Systemen angegeben ist.

  Die Datei 'qm.ini' enthält für einen bestimmten Warteschlangenmanager relevante Informationen. 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:

  Auf UNIX and Linux-Systemen:

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  IBM i Auf IBM i-Systemen:

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  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.

  Sie können die Konfigurationsdateien entweder wie folgt bearbeiten:

  – Automatisch unter Verwendung von Befehlen, die die Konfiguration von Warteschlangenmanagern auf dem Knoten ändern
  – Manuell unter Verwendung eines Standardtexteditors

  Wenn Sie einen falschen Wert in einem Konfigurationsdateiattribut festlegen, wird der Wert ignoriert und eine Bedienernachricht ausgegeben, um das Problem anzugeben. (Der Effekt ist der gleiche wie das Fehlen des Attributs vollständig.)

Zu konfigurierende Zeilengruppen

Folgende Zeilengruppen müssen geändert werden:

ApiExitCommon

In der Datei 'mqs.ini' sowie im IBM MQ Explorer auf der IBM MQ-Eigenschaftenseite unter 'Exits' definiert.

Wenn ein Warteschlangenmanager gestartet wird, werden die Attribute in dieser Zeilengruppe gelesen und dann mit den in der Datei 'qm.ini' definierten API-Exits überschrieben.

ApiExitTemplate

In der Datei 'mqs.ini' sowie 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

In der Datei 'qm.ini' sowie im IBM MQ Explorer auf der Warteschlangenmanager-Eigenschaftenseite 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 1033.

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 127 auf Seite 1010 zeigt die Kanalexitarten, die für die einzelnen Kanaltypen verfügbar sind.

Tabelle 127. Für die einzelnen Kanaltypen verfügbare Kanalexits

Kanaltyp	Nachrichtenexit	Nachrichtenwiederholungsexit	Empfangsexit	Sicherheitsexit	Sendeexit	Autodefinitionsexit
Senderkanal	Ja		Ja	Ja	Ja	
Serverkanal	Ja		Ja	Ja	Ja	
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

Hinweise: 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 115 auf Seite 1011 dargestellt.

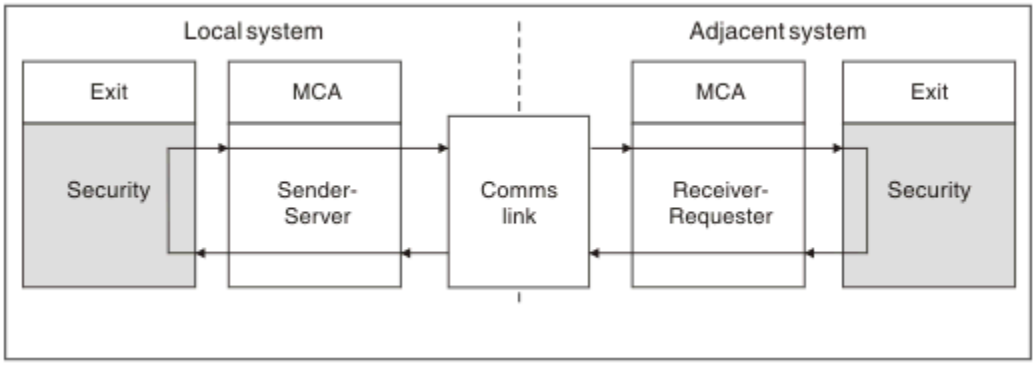


Abbildung 115. Sicherheitsexitschleife

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 116 auf Seite 1011 dargestellt an den empfangenden Nachrichtenkanalagenten.

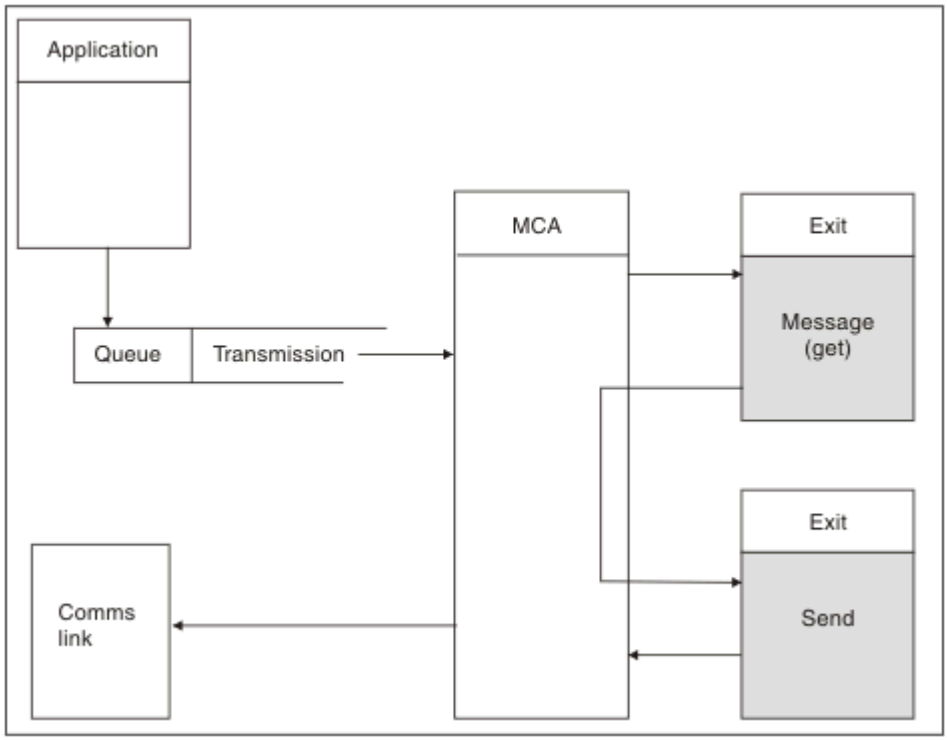


Abbildung 116. Beispiel eines Sendeexits auf der Sendeseite des Nachrichtenkanals

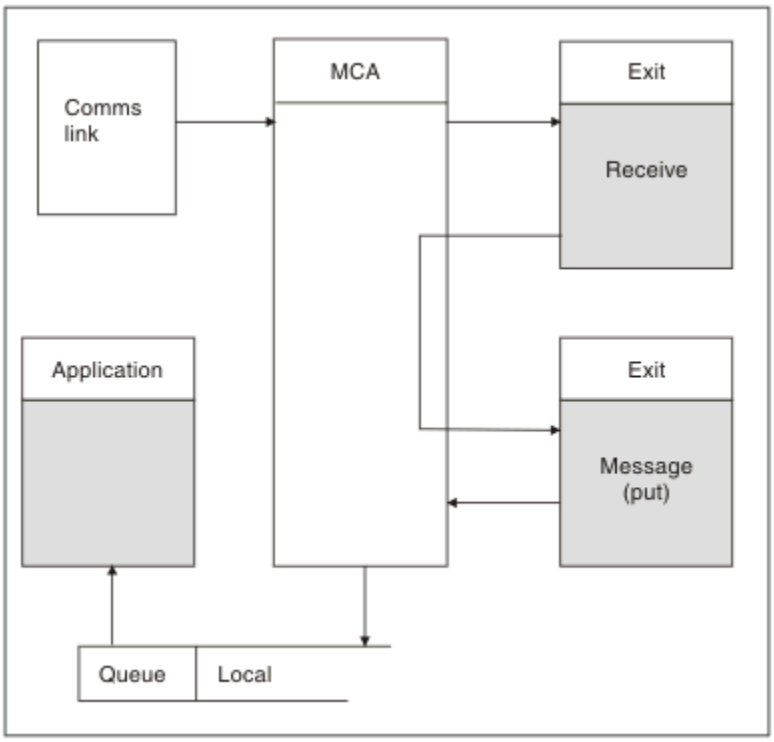


Abbildung 117. Beispiel eines Empfangsexits auf der Empfangsseite des Nachrichtenkanals

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 117 auf Seite 1012 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 UNIX 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. Details finden Sie durch Überprüfung des in 'MaxSegmentLength' zurückgegebenen Werts. Weitere Informationen zu 'MaxSegmentLength' finden Sie unter MaxSegmentLength.

  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 1010 finden Sie Details zum Aufruf.

z/OS *Kanalexitprogramme unter z/OS schreiben*

Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für z/OS schreiben und kompilieren.

Die Exits werden wie von einem z/OS-LINK in folgenden Modi gestartet:

- Bei Auftreten eines Fehlers im Programm aufgrund einer fehlenden Zugriffsberechtigung
- Im Steuermodus für den Primäradressraum
- Im nicht speicherübergreifenden Modus
- Im Modus ohne Registerzugriff
- Im 31-Bit-Adressierungsmodus

Die programmverbundenen Module müssen in die von der CSQXLIB-DD-Anweisung der Adressraumprozedur des Kanalinitiators angegebene Datei gestellt werden. Die Namen der Lademodule werden als die Exitnamen in der Kanaldefinition angegeben.

Beim Schreiben von Kanalexits für z/OS gelten folgende Regeln:

- Exits müssen in Assembler oder C geschrieben werden. Bei Verwendung der Programmiersprache C muss diese der Programmierumgebung der C-Systeme für Systemexits entsprechen, die im z/OSC/C++ Programming Guide beschrieben sind.
- Die Exits werden aus den in der Anweisung CSQXLIB DD angegebenen nicht autorisierten Bibliotheken geladen. Wenn CSQXLIB den Wert DISP=SHR hat, können Exits aktualisiert werden, während der Kanalinitiator ausgeführt wird. Beim Neustart des Kanals wird dann die neue Version verwendet.
- Exits müssen wiedereintrittsfähig und überall im virtuellen Speicher ausführbar sein.
- Exits müssen die Umgebung bei Rückgabe auf den Zustand zum Zeitpunkt der Eingabe zurücksetzen.
- Exits müssen erhaltenen Speicherplatz frei machen oder sicherstellen, dass er von einem nachfolgenden Exitaufruf frei gemacht wird.

  Verwenden Sie bei Speicherplatz, der zwischen Aufrufen bestehen bleibt, den z/OS STORAGE-Service; bei Systemprogrammierung C die '4kmalc'-Bibliotheksfunktion.

  Weitere Informationen zu dieser Funktion finden Sie im Abschnitt 4kmalc(-- Allocate Page-Aligned Storage.
- Mit Ausnahme von MQCMIT bzw. CSQBCMT und MQBACK bzw. CSQBBAK können alle IBM MQ MQI-Aufrufe verwendet werden. Sie müssen hinter MQCONN (mit einem leeren Warteschlangenmanagernamen) enthalten sein. Wenn diese Aufrufe verwendet werden, muss der Exit mit dem Stub CSQXSTUB über eine Programmverknüpfung verbunden sein.

  Als Ausnahme dieser Regel können Sicherheitskanalexits 'Commit'- und 'Backout'-MQI-Aufrufe ausgeben. Codieren Sie, um solche Aufrufe auszugeben, die Verben CSQXCMT und CSQXBAK anstelle von MQCMIT bzw. CSQBCMT und MQBACK bzw. CSQBBAK.
- Alle Exits, die den Stub CSQXSTUB aus IBM WebSphere MQ 7.0 oder höher verwenden, müssen über eine Programmverknüpfung mit einer CSQXLIB-Ladebibliothek des Formats PDS-E verbunden sein.
- Exits dürfen keine Systemservices verwenden, die einen Wartezyklus verursachen, da die Verwendung von Systemservices schwerwiegende Auswirkungen auf die Handhabung einiger oder aller anderen

Kanäle haben würde. Viele Kanäle werden üblicherweise unter einem einzigen Tasksteuerblock ausgeführt. Wenn Sie eine Aktion in einem Exit ausführen, die einen Wartezyklus verursacht, und Sie nicht MQXWAIT verwenden, wird ein Wartezyklus für alle diese Kanäle verursacht. Wartezyklen bei Kanälen führen zwar nicht zu funktionalen Problemen, können sich jedoch nachteilig auf die Leistung auswirken. Die meisten SVCs sind mit Wartezyklen verbunden und sollten, mit Ausnahme der folgenden SVCs, vermieden werden:

  - GETMAIN/FREEMAIN/STORAGE
  - LOAD/DELETE

  SVCs, PCs und Ein-/Ausgabe (I/O) sind daher im Allgemeinen zu vermeiden. Verwenden Sie stattdessen den Aufruf MQXWAIT.

- Exits geben, ausgenommen in Subtasks, keine ESTAEs oder SPIEs aus, da ihre Fehlerbehandlung die Fehlerbehandlung von IBM MQ beeinträchtigen könnte. Das bedeutet, dass IBM MQ nach einem Fehler möglicherweise nicht wiederhergestellt werden kann oder dass Ihr Exitprogramm eventuell nicht alle Fehlerinformationen erhält.
- Der Aufruf MQXWAIT (siehe MQXWAIT) stellt einen Warteservice bereit, der auf Ein-/Ausgabe und andere Ereignisse wartet. Bei Verwendung dieses Service ist die Verwendung des Verbindungsstacks durch die Exits nicht möglich.

  Bei Ein-/Ausgaben und anderen Funktionen, die keine nicht blockierenden Funktionen oder einen Ereignissteuerblock zum Warten bereitstellen, muss eine separate Subtask angehängt werden, auf deren Beendigung der Aufruf MQXWAIT wartet. Wegen der Verarbeitung dieser Technik darf die Funktion nur vom Sicherheitsexit verwendet werden.

- Der MQI-Aufruf MQDISC führt nicht dazu, dass eine implizite Commitoperation innerhalb des Exitprogramms auftritt. Eine Commitoperation des Kanalprozesses wird nur ausgeführt, wenn es das Kanalprotokoll so vorsieht.

IBM MQ for z/OS enthält folgende Exitbeispiele:

CSQ4BAX0
Dieses Beispiel ist in der Assemblersprache geschrieben und veranschaulicht die Verwendung von MQXWAIT.

CSQ4BCX1 und CSQ4BCX2
Diese Beispiele sind in der Programmiersprache C geschrieben und veranschaulichen den Zugriff auf die Parameter.

CSQ4BCX3 und CSQ4BAX3
Diese Beispiele sind in der Programmiersprache C bzw. in der Assemblersprache geschrieben.

Das Beispiel CSQ4BCX3 (das in die SCSQAUTH LOADLIB vorkompiliert wird, sollte funktionieren, ohne dass im Exit selbst Änderungen nötig sind. Sie können eine LOADLIB (z. B. mit dem Namen MY.TEST.LOADLIB) erstellen und das Member SCSQAUTH (CSQ4BCX3) hineinkopieren.

Gehen Sie wie folgt vor, um einen Sicherheitsexit für eine Clientverbindung zu konfigurieren:

1. Erstellen Sie ein gültiges OMVS-Segment für die Benutzer-ID, die der Kanalinitiator verwendet.

   Dies ermöglicht es dem IBM MQ for z/OS-Kanalinitiator, TCP/IP mit der UNIX System Services-(USS-)Socketschnittstelle zu verwenden, um die Exitverarbeitung zu vereinfachen. Beachten Sie, dass kein OMVS-Segment für die Benutzer-ID irgendeines verbindenden Clients definiert werden muss.
2. Stellen Sie sicher, dass der Exit-Code selbst nur in einer programmgesteuerten Umgebung ausgeführt wird.

   Dies bedeutet, dass alles, was in den CHINIT-Adressraum geladen wird, aus einer programmgesteuerten Bibliothek geladen werden muss (also alle Bibliotheken in der STEPLIB) und alle Bibliotheken benannt in CSQXLIB und

```
++hlq++.SCSQANLx
```

```
++hlq++.SCSQMVR1
++hlq++.SCSQAUTH
```

Verwenden Sie einen Befehl wie den folgenden, um eine Ladebibliothek als programmgesteuert zu definieren:

```
RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
```

Danach können Sie die programmgesteuerte Umgebung aktivieren oder aktualisieren, indem Sie folgenden Befehl ausgeben:

```
SETROPTS WHEN(PROGRAM) REFRESH
```

3. Fügen Sie den Exit LOADLIB zur CSQXLIB DD hinzu (in der gestarteten CHINIT-Prozedur) hinzu, indem Sie folgenden Befehl ausgeben:

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

Dadurch wird der Exit für den benannten Kanal aktiviert.

4. Ihr externer Sicherheitsmanager (ESM) listet alle anderen Bibliotheken auf, die programmgesteuert werden sollen. Beachten Sie jedoch, dass keine der ESM- oder C-Bibliotheken programmgesteuert sein muss.

Weitere Informationen zur Konfiguration eines Sicherheitsexits mit dem Beispiel CSQ4BCX3 finden Sie im Abschnitt IBM MQ for z/OS-Serververbindungskanal.

CSQ4BCX4
Dieses Beispiel ist in C geschrieben und veranschaulicht die Verwendung der Felder **RemoteProduct** und **RemoteVersion** in MQCXP.

Zugehörige Konzepte

„Kanalexitprogramme unter IBM i schreiben” auf Seite 1016
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für IBM i schreiben und kompilieren.

„Kanalexitprogramme unter UNIX, Linux, and Windows schreiben” auf Seite 1017
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für Systeme unter UNIX, Linux, and Windows schreiben.

Zugehörige Informationen

IBM MQ for z/OS-Serververbindungskanal

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 UNIX, Linux, and Windows schreiben” auf Seite 1017
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für Systeme unter UNIX, Linux, and Windows schreiben.

„Kanalexitprogramme unter z/OS schreiben” auf Seite 1014
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für z/OS schreiben und kompilieren.

ULW *Kanalexitprogramme unter UNIX, Linux, and Windows schreiben*

Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für Systeme unter UNIX, Linux, and Windows schreiben.

Folgen Sie den unter „Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben” auf Seite 979 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 118 auf Seite 1017 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 118. 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 119 auf Seite 1018. Weitere Informationen zum Schreiben von Kanalexitprogrammen finden Sie im Abschnitt „Kanalexitprogramme schreiben” auf Seite 1012.

```
EXPORTS
ChannelExit
```

Abbildung 119. DEF-Musterdatei für Windows

Zugehörige Konzepte

„Kanalexitprogramme unter IBM i schreiben” auf Seite 1016
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für IBM i schreiben und kompilieren.

„Kanalexitprogramme unter z/OS schreiben” auf Seite 1014
Mithilfe der folgenden Informationen können Sie Kanalexitprogramme für z/OS schreiben und kompilieren.

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. Wenn Sie festlegen möchten, dass ein Kanal einen Sicherheitsexit verwenden muss, geben Sie im Feld SCYEXIT der Kanaldefinition den Exitnamen 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.

Bei Sicherheitsexits, die vor IBM WebSphere MQ 7.1 geschrieben wurden, ist zu beachten, dass in früheren Versionen von IBM MQ der zugrunde liegende sichere Sockets-Provider (z. B. GSKit) abgefragt wurde, um den registrierten Namen des Zertifikatsinhabers (SSLPEER) und den registrierten Namen des Zertifikatsausstellers (SSLCERTI) des fernen Partners zu ermitteln. IBM WebSphere MQ 7.1 unterstützt nun eine Reihe neuer Sicherheitsattribute. Um auf diese Attribute zuzugreifen, ruft IBM WebSphere MQ 7.1 die DER-Codierung (Distinguished Encoding Rules) des Zertifikats ab und ermittelt damit den registrierten Namen des Zertifikatsinhabers und den registrierten Namen des Zertifikatsausstellers. 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

In IBM WebSphere MQ 7.1 7.1 ist im registrierten Namen des Zertifikatsinhabers auch das Attribut SERIALNUMBER mit der Seriennummer des Zertifikats des fernen Partners enthalten. Außerdem werden einige Attribute für registrierte Namen aus früheren Releases in einer anderen Reihenfolge zurückgegeben. Infolgedessen wurde die Zusammensetzung der Felder SSLPEER und SSLCERTI in IBM WebSphere MQ 7.1 im Vergleich zu früheren Releases geändert und es wird empfohlen, alle Sicherheitsexits oder Anwendungen, die von diesen Feldern abhängig sind, zu prüfen und zu aktualisieren.

Vorhandene IBM MQ-Peernamensfilter, die über das Feld SSLPEER einer Kanaldefinition angegeben werden, sind davon nicht betroffen und funktionieren genauso wie in früheren Releases. Das liegt daran, dass der Algorithmus zum Abgleich des IBM MQ-Peernamens aktualisiert wurde und bestehende SSLPEER-Filter bearbeiten kann, ohne dass eine Änderung der Kanaldefinitionen erforderlich ist. Diese Änderung wirkt sich höchstwahrscheinlich auf Sicherheitsexits und -anwendungen aus, die von den Werten für den

registrierten Namen des Zertifikatsinhabers und den registrierten Namen des Zertifikatsausstellers, die von der PCF-Programmierschnittstelle zurückgegeben werden, abhängig sind.

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 120 auf Seite 1020bis Abbildung 123 auf Seite 1022 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 120 auf Seite 1020:

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 120. 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 121. 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 122. 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 123. 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 Informationen

Kanalauthentifizierungsdatensätze

Konzepte der Transport Layer Security (TLS)

Sicherheitsexit schreiben

Sie können den Sicherheitsexit-Entwurfscode zum Schreiben von Sicherheitsexits verwenden.

Abbildung 124 auf Seite 1023 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 124. 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).

Kanaltyp
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).

MCA User Identifier
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 1209

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 32 KB 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.

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. Er ermittelt das Objekt, ruft es auf geeignete Weise ab, fügt es an den Header an und übergibt es an den Nachrichtenkanalagenten, der es wiederum an den empfangenden Nachrichtenkanalagenten überträgt. 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” auf Seite 825 und „Referenznachrichtenbeispiel ausführen” auf Seite 1174.

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 UNIX Wenn Sie unter UNIX 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, wird unter z/OS standardmäßig ein verteilter Exitname des Formats *[Pfad]/Bibliotheksname(Funktion)* geprüft und bis zu 8 Zeichen von 'Funktion' (falls vorhanden) oder 'Bibliotheksname' verwendet. 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.

Kanalexitprogramme auf Windows, UNIX and Linux-Systemen kompilieren

Mithilfe der folgenden Beispiele können Sie Kanalexitprogramme für Windows, UNIX and Linux-Systeme kompilieren.

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 UNIX and Linux

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.

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
```

Compiler- und Linker-Musterbefehle für Kanalexits unter HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o exit.o exit.c -I/opt/mqm/inc
$ ld -b exit.o +ee MQStart +ee ChannelExit -o
/var/mqm/exits64/exit -L/usr/lib/pa20_64 -lpthread
$ rm exit.o
```

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
```

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
```

Compiler- und Linker-Beispielbefehle für Kanalexits unter Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-R/usr/lib/64 -lsocket -lnsl -ldl
```

Auf dem Client kann sowohl ein 32-Bit- als auch ein 64-Bit-Exit verwendet werden. Dieser Exit muss mit 'mqic_r' verknüpft sein.

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 den Abschnitten MQCD oder Script (MQSC) Commands.

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.

Durch die Einführung von Multicasting-Clients müssen API-Exits und Datenkonvertierungsexits clientseitig ausgeführt werden können, da manche Nachrichten den Warteschlangenmanager unter Umständen nicht durchlaufen. Die folgenden Bibliotheken sind jetzt sowohl Teil der Client- als auch der Serverpakete:

Tabelle 128. Bibliotheken, die jetzt in den Client- und Serverpaketen enthalten sind.

Betriebssystem	**Bibliotheken**
Windows	32 Bit & 64 Bit: mqm.dll & mqm.pdb
Linux & HP-UX	32 Bit & 64 Bit: libmqm.so & libmqm_r.so
AIX	32 Bit & 64 Bit: libmqm.a & libmqm_r.a
Solaris	32 Bit & 64 Bit: libmqm.so
IBM i IBM i IBM i	LIBMQM & LIBMQM_R

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 1036 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 1036 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 1036 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.

Datenkonvertierungsexitprogramm für IBM MQ for z/OS schreiben

Schritte, die Sie beim Schreiben von Datenkonvertierungsprogrammen für IBM MQ for z/OS beachten sollten.

Gehen Sie wie folgt vor:

1. Verwenden Sie die bereitgestellte Quellenentwurfsdatei CSQ4BAX9 (für nicht-CICS-Umgebungen) bzw. CSQ4CAX9 (für CICS) als Ausgangspunkt.
2. Führen Sie das Dienstprogramm CSQUCVX aus.
3. Befolgen Sie die Anweisungen im Prolog von CSQ4BAX9 oder CSQ4CAX9, um die vom Dienstprogramm CSQUCVX erstellten Routinen in der Reihenfolge aufzunehmen, in der die Strukturen in der Nachricht, die Sie konvertieren möchten, auftreten.
4. Das Dienstprogramm setzt voraus, dass die Datenstrukturen nicht gepackt sind, die implizierte Ausrichtung der Daten berücksichtigt wird und die Strukturen an einer Maschinenwortgrenze anfangen, wobei Bytes bei Bedarf übersprungen werden (wie zwischen ID und VERSION im Beispiel im Abschnitt Valid syntax). Wenn die Strukturen gepackt sind, schließen Sie die erstellten CMQXCALA-Makros aus. Erwägen Sie daher, Ihre Strukturen so zu deklarieren, dass alle Felder benannt und keine Bytes übersprungen werden; fügen Sie zwischen ID und VERSION im Beispiel im Abschnitt Valid syntax ein Feld "MQBYTE DUMMY;" hinzu.
5. Der bereitgestellte Exit gibt einen Fehler zurück, wenn der Eingabepuffer kürzer als das Nachrichtenformat ist, in das konvertiert werden soll. Obwohl der Exit so viele vollständige Felder wie möglich konvertiert, wird durch den Fehler eine nicht konvertierte Nachricht an die Anwendung zurückgegeben. Wenn Sie zulassen möchten, dass kurze Eingabepuffer einschließlich partieller Felder soweit wie möglich konvertiert werden, ändern Sie den Wert TRUNC= im CSQXCDFA-Makro in YES: Es wird kein Fehler zurückgegeben und die Anwendung empfängt somit eine konvertierte Nachricht. Die Anwendung muss die Verkürzung handhaben.
6. Fügen Sie nach Bedarf weiteren Sonderverarbeitungscode hinzu.
7. Benennen Sie das Programm in den Namen Ihres Datenformats um.
8. Kompilieren Sie Ihr Programm wie ein Stapelanwendungsprogramm und stellen Sie eine Programmverbindung wie bei einem solchen Programm her (es sei denn, es soll mit CICS-Anwendungen verwendet werden). Die Makros, die vom Dienstprogramm im Code erstellt werden, befinden sich in der Bibliothek **thlqual**.SCSQMACS.

   Wenn die Nachricht Zeichendaten enthält, ruft der generierte Code MQXCNVC auf. Wenn Ihr Exit diesen Aufruf verwendet, stellen Sie mit dem Exit-Stubprogramm CSQASTUB eine Programmverbindung her. Der Stub ist sprachunabhängig und umgebungsunabhängig. Wahlweise können Sie den Stub dyna-

misch laden, indem Sie den dynamischen Aufrufnamen CSQXCNVC verwenden. Weitere Informationen finden Sie im Abschnitt „IBM MQ-Stub dynamisch aufrufen" auf Seite 1094.

Positionieren Sie das Modul mit Programmverbindung in Ihrer Anwendungsladebibliothek und in einem Datensatz, der durch die CSQXLIB-Datendefinitionsanweisung Ihrer Taskprozedur referenziert ist, die durch Ihren Kanalinitiator gestartet wurde.

9. Wenn der Exit mit CICS-Anwendungen verwendet werden soll, kompilieren Sie ihn wie ein CICS-Anwendungsprogramm und stellen Sie wie bei einem solchen Programm eine Programmverbindung her (falls erforderlich mit CSQASTUB). Stellen Sie ihn in die Bibliothek Ihres CICS -Anwendungsprogramms. Definieren Sie das Programm wie üblich für CICS, indem Sie EXECKEY(CICS) in der Definition angeben.

Anmerkung: Die LE/370-Laufzeitbibliotheken sind zwar zur Ausführung des Dienstprogramms CSQUCVX (siehe Schritt „2" auf Seite 1037), jedoch nicht zur Herstellung einer Programmverbindung oder Ausführung des eigentlichen Datenkonvertierungsexits erforderlich (siehe Schritte „8" auf Seite 1037 und „9" auf Seite 1038).

Informationen zur Datenkonvertierung in der IBM MQ-IMS-Bridge finden Sie im Abschnitt „IMS-Brückenanwendungen schreiben" auf Seite 69.

Datenkonvertierungsexit für IBM MQ auf UNIX and Linux-Systemen schreiben

Schritte, die Sie beim Schreiben von Datenkonvertierungsexitprogrammen für IBM MQ auf UNIX and Linux-Systemen beachten sollten.

Führen Sie folgende Schritte aus:

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 1038 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 1038 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 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 „3” auf Seite 1038 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 UNIX and Linux-Systemen kompilieren” auf Seite 1039.
9. Stellen Sie die Ausgabe in das Exitverzeichnis. Das Standardexitverzeichnis lautet `/var/mqm/exits` für 32-Bit-Systeme und `/var/mqm/exits64` für 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. Wenn `crtmqcvx` gepackte Strukturen verwendet, müssen alle IBM MQ -Anwendungen auf diese Weise kompiliert werden.
2. Datenkonvertierungsexitprogramme müssen simultan verwendbar sein.
3. Von einem Datenkonvertierungsexit aus kann nur der MQI-Aufruf MQXCNVC ausgegeben werden.

Datenkonvertierungsexits auf UNIX and Linux-Systemen kompilieren
Beispiele zur Kompilierung von Datenkonvertierungsexits auf UNIX 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

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
```

HP-UX Itanium-Plattform

Kompilieren und verbinden Sie den Exit-Quellcode, indem Sie eine der folgenden Befehlsgruppen ausgeben:

32-Bit-Anwendungen

Nicht-thread-basiert

Kompilieren Sie den Exit-Quellcode:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie das Exitobjekt:

```
ld +b: -b MYFORMAT.o +ee MQStart -o \
    /var/mqm/exits/MYFORMAT -L/usr/lib/hpux32
rm MYFORMAT.o
```

Thread-basiert

Kompilieren Sie den Exit-Quellcode:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie das Exitobjekt:

```
ld +b: -b MYFORMAT.o +ee MQStart -o \
    /var/mqm/exits/MYFORMAT_r -L/usr/lib/hpux32 \
    -lpthread
rm MYFORMAT.o
```

64-Bit-Anwendungen

Nicht-thread-basiert

Kompilieren Sie den Exit-Quellcode:

```
 c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie das Exitobjekt:

```
ld -b MYFORMAT.o +ee MQStart \
    -o  /var/mqm/exits64/MYFORMAT \
    -L/usr/lib/hpux64
rm MYFORMAT.o
```

Thread-basiert

Kompilieren Sie den Exit-Quellcode:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Verknüpfen Sie das Exitobjekt:

```
ld -b MYFORMAT.o +ee MQStart  \
    -o  /var/mqm/exits64/MYFORMAT_r  \
    -L/usr/lib/hpux64 -lpthread
rm MYFORMAT.o
```

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
```

Solaris

Kompilieren Sie den Exit-Quellcode, indem Sie einen der folgenden Befehle ausgeben:

32-Bit-Anwendungen

SPARC-Plattform

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

x86-64-Plattform

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

64-Bit-Anwendungen

SPARC-Plattform

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

x86-64-Plattform

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

Datenkonvertierungsexit für IBM MQ for Windows schreiben

Schritte, die Sie beim Schreiben von Datenkonvertierungsprogrammen für IBM MQ for Windows beachten sollten.

Führen Sie folgende Schritte aus:

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.

   Im Beispielverzeichnis *MQ_INSTALLATION_PATH*\Tools\C\Samples steht zudem eine .DEF-Datei namens 'amqsvfcn.def' bereit. *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 unter *MQ_INSTALLATION_PATH*\Tools\C\Samples, wobei *MQ_INSTALLATION_PATH* für das Verzeichnis steht, in dem IBM MQ installiert ist. (Das Standardinstallationsverzeichnis ist C:\Programme\IBM\MQ.)

Die Entwurfsdatei enthält eine Musterheaderdatei '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 1042 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 1042 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.

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. Diese 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
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 einer Eingabe handelt es sich bei diesem Parameter um die Filterzeichenfolge, die für den MQCONN API-Aufruf über den Parameter **QMgrName** bereitgestellt 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 zur Steuerung der Aktion von MQCONNX.

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 „Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben” auf Seite 979 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 „Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben” auf Seite 979.

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 UNIX 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 Übergehen Sie `PublishExitFunction` in IBM i, falls ein Programm verwendet wird.

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 „Programmierung des Exits für Clusterauslastung für IBM MQ for z/OS" auf Seite 1050.

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 „Installierbare Services und Komponenten für UNIX, Linux und Windows schreiben" auf Seite 979 und verwenden Sie das Beispielprogramm „Beispielprogramm für Exit für Clusterauslastung" auf Seite 1049, das Sie beim Schreiben und Kompilieren Ihres Exits unterstützt.

Weitere Informationen zu Kanalexits finden Sie im Abschnitt „Kanalexitprogramme schreiben" auf Seite 1012.

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. Beispiel:

```
ALTER QMGR CLWLEXIT(myexit)
```

Zugehörige Informationen

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 129. Speicherposition des Beispielprogramms für den Exit für Clusterauslastung auf Multiplatforms-Systemen

Plattform	Dateipfad
AIX, HP-UX, Sun Solaris	*MQ_INSTALLATION_PATH*/samp
Windows	*MQ_INSTALLATION_PATH* \Tools\c\Samples
IBM i IBM i	Die qmqm-Bibliothek

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. Beispiel:

```
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 UNIX Unter UNIX 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.

Programmierung des Exits für Clusterauslastung für IBM MQ for z/OS

Exits für Clusterauslastung werden wie von einem z/OS **LINK**-Befehl aufgerufen. Exits unterliegen einer Reihe von strikten Programmierungsregeln. Vermeiden Sie einen Großteil der SVC-Befehle, die Wartezeiten beinhalten, und vermeiden Sie die Verwendung von STAE oder ESTAE in einem Exit für Clusterauslastung.

Exits für Clusterauslastung werden in folgenden Fällen wie von einem z/OS **LINK**-Befehl aufgerufen:

- Bei Auftreten eines Fehlers im Programm aufgrund einer fehlenden Zugriffsberechtigung
- Im Steuermodus für den Primäradressraum
- Im nicht speicherübergreifenden Modus

- Im Modus ohne Registerzugriff
- Im 31-Bit-Adressierungsmodus
- Über Speicherschlüssel 8
- Über Programmschlüsselmaske 8
- Über TCB-Schlüssel 8

Fügen Sie die verbundenen Module in den Datensatz ein, der mit der Anweisung CSQXLIB DD der Prozedur für den Adressraum des Warteschlangenmanagers angegeben wird. Die Namen der Lademodule werden in der Warteschlangenmanagerdefinition als die Namen von Auslastungsexits angegeben.

Beim Schreiben von Exits für Auslastung für IBM MQ for z/OS gelten die folgenden Regeln:

- Sie müssen Exits in der Programmiersprache Assembler oder C schreiben. Bei Verwendung von C muss der Exit der Programmierumgebung von C-Systemen für Systemexits entsprechen, wie im Handbuch *z/OS C/C++ Programming Guide* (SC09-4765) beschrieben.
- Stellen Sie bei Verwendung des Aufrufs MQXCLWLN eine Programmverknüpfung mit der Anweisung CSQMFCLW her, die in *thlqual*.SCSQLOAD bereitgestellt wird.
- Exits werden aus den nicht autorisierten Bibliotheken geladen, die durch eine CSQXLIB DD-Anweisung definiert werden. Wenn für CSQXLIB der Wert DISP=SHR festgelegt ist, können Exits mit der neuen Version aktualisiert werden, die im nächsten vom Warteschlangenmanager gestarteten MQCONN-Thread verwendet wird, während der Warteschlangenmanager ausgeführt wird.
- Exits müssen wiedereintrittsfähig und überall im virtuellen Speicher ausführbar sein.
- Exits müssen die Umgebung bei Rückgabe auf den Zustand zum Zeitpunkt der Eingabe zurücksetzen.
- Exits müssen gesammelten Speicherplatz freigeben oder sicherstellen, dass er von einem nachfolgenden Exitaufruf freigegeben wird.
- Es sind keine MQI-Anrufe zulässig.
- Exits dürfen keine Systemservices verwenden, die eine Wartezeit verursachen können, da eine Wartezeit die Leistung des Warteschlangenmanagers deutlich vermindert. Vermeiden Sie daher im Allgemeinen SVCs, PCs und Ein-/Ausgabe.
- Exits dürfen ein ESTAE oder SPIE nicht ausgeben, außer innerhalb von Subtasks, die sie anhängen.

Anmerkung: Es gibt keine absoluten Einschränkungen dessen, was Sie in einem Exit ausführen können. Allerdings betreffen die meisten SVCs Wartezeiten, daher sollten Sie sie vermeiden, mit Ausnahme der folgenden Befehle:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

Verwenden Sie keine ESTAEs und ESPIEs, da ihre Fehlerbehandlung die von IBM MQ ausgeführte Fehlerbehandlung beeinträchtigen könnte. IBM MQ kann nach einem Fehler möglicherweise nicht wiederhergestellt werden oder Ihr Exitprogramm erhält eventuell nicht alle Fehlerinformationen.

Der Systemparameter EXITLIM beschränkt den Zeitraum, in dem ein Exit ausgeführt werden kann. Der Standardwert für EXITLIM ist 30 Sekunden. Wenn der Rückgabecode MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA' angezeigt wird, befindet sich Ihr Exit möglicherweise in einer Schleife. Wenn Sie glauben, dass der Exit mehr als 30 Sekunden benötigt, um erfolgreich beendet zu werden, erhöhen Sie den Wert von 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 740.

Legen Sie bei Ausführung eines Thread-Servers oder einer Thread-Clientanwendung die Umgebungsvariable THREAD_SCOPE=S fest.

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.

Wenn Sie den VisualAge C/C++-Compiler für C++-Programme verwenden, müssen Sie die Option `-q namemangling=v5` einschließen, damit beim Verknüpfen der Bibliotheken alle IBM MQ-Symbole aufgelöst werden.

Wenn Sie die Programme auf einer Maschine verwenden möchten, auf der nur IBM MQ MQI client for AIX installiert ist, kompilieren Sie die Programme erneut, um sie 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.

Wenn Sie die Programme auf einer Maschine verwenden möchten, auf der nur IBM MQ MQI client for AIX installiert ist, kompilieren Sie die Programme erneut, um sie 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 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. In IBM MQ auf UNIX and 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.

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 ist `amqminqx` ein Beispielprogramm.

In der Dokumentation zu Micro Focus COBOL finden Sie eine Beschreibung der Umgebungsvariablen, die Sie einrichten müssen.

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 130 auf Seite 1056 aufgeführt.

Tabelle 130. Essenzieller Code für CICS-Anwendungsprogramme unter AIX: XA-Initialisierungsroutine

Beschreibung	**C (Quelle)**	**C (Exec) - zu Ihrer XAD.Stanza**
XA-Initialisierungsroutine	amqzscix.c	*amqzsc* - CICS for 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 Verwalten IBM MQ-Handbuch zur Systemverwaltung.

TXSeries CICS-Unterstützung

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 on UNIX and 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* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -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 reiht sie in die lokale Warteschlange mit dem Warteschlangennamen ein, der im Übertragungsheader der Nachricht enthalten ist. Mögliche Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICS.DLQ gesendet. Erstellen Sie mithilfe des MQSC-Beispielscripts AMQSCIC0.TST diese Warteschlangen und die Beispieleingabewarteschlangen.

Prozedurale Anwendung auf HP-UX erstellen

In diesen Informationen werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie beim Erstellen von IBM MQ for HP-UX-Anwendungen für die Ausführung unter HP-UX 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 HP-UX 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 HP-UX-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 740.

In diesem Abschnitt wird durchgängig der umgekehrte Schrägstrich (\) verwendet, um lange Befehle über mehr als eine Zeile zu trennen. Geben Sie dieses Zeichen nicht ein; geben Sie jeden Befehl in einer einzigen Zeile ein.

C-Programme in HP-UX vorbereiten

In diesem Abschnitt finden Sie Informationen, die Sie beim Vorbereiten von C-Programmen in HP-UX berücksichtigen müssen. Es sind auch Beispiele für die IA64-Plattform (IPF) vorhanden.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Arbeiten Sie in Ihrer normalen Umgebung. Vorkompilierte C-Programme werden im Verzeichnis *MQ_INSTALLATION_PATH* /samp/bin bereitgestellt.

Weitere Informationen zur Programmierung von 64-Bit-Anwendungen finden Sie in Codierungsstandards auf 64-Bit-Plattformen.

Für die Verwendung von TLS müssen IBM MQ MQI clients unter HP-UX mithilfe von POSIX-Threads erstellt werden.

Folgende Beispiele sollten beachtet werden:

- „IA64-Plattform (IPF)” auf Seite 1058
- „Bibliotheken verknüpfen” auf Seite 1060

IA64-Plattform (IPF)

Beispiele zur Erstellung von amqsput0, cliexit und srvexit auf der IA64-Plattform (IPF).

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Clientanwendung in einer 32-Bit-Umgebung ohne Thread erstellt:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic
```

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Clientanwendung in einer 32-Bit-Umgebung mit Thread erstellt:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic_r -lpthread
```

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Clientanwendung in einer 64-Bit-Umgebung ohne Thread erstellt:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Clientanwendung in einer 64-Bit-Umgebung mit Thread erstellt:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Serveranwendung in einer 32-Bit-Umgebung ohne Thread erstellt:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm
```

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Serveranwendung in einer 32-Bit-Umgebung mit Thread erstellt:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm_r -lpthread
```

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Serveranwendung in einer 64-Bit-Umgebung ohne Thread erstellt:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

Im folgenden Beispiel wird das Beispielprogramm 'amqsput0' als Serveranwendung in einer 64-Bit-Umgebung mit Thread erstellt:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

Im folgenden Beispiel wird der Client-Exit 'cliexit' in einer 32-Bit-Umgebung ohne Thread erstellt:

```
c89 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic
```

Im folgenden Beispiel wird der Client-Exit 'cliexit' in einer 32-Bit-Umgebung mit Thread erstellt:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic_r -lpthread
```

Im folgenden Beispiel wird der Client-Exit 'cliexit' in einer 64-Bit-Umgebung ohne Thread erstellt:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
```

```
ld -b cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

Im folgenden Beispiel wird der Client-Exit 'cliexit' in einer 64-Bit-Umgebung mit Thread erstellt:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
ld -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_64_r \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

Im folgenden Beispiel wird der Server-Exit 'srvexit' in einer 32-Bit-Umgebung ohne Thread erstellt:

```
c89 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm
```

Im folgenden Beispiel wird der Server-Exit 'srvexit' in einer 32-Bit-Umgebung mit Thread erstellt:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm_r -lpthread
```

Im folgenden Beispiel wird der Server-Exit 'srvexit' in einer 64-Bit-Umgebung ohne Thread erstellt:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH  MQ_INSTALLATI
ON_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

Im folgenden Beispiel wird der Server-Exit 'srvexit' in einer 64-Bit-Umgebung mit Thread erstellt:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_64_r \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

Bibliotheken verknüpfen

Sie müssen Ihre Programme mit einer der Bibliotheken verknüpfen, die von IBM MQ bereitgestellt werden.

In der folgenden Tabelle wird gezeigt, welche Bibliothek in den unterschiedlichen Umgebungen verwendet werden soll:

Hardwareplattform	Umgebung mit oder ohne Thread	Programm/Exit-Typ	Bibliotheksdatei
IA64 (IPF)	Mit Thread	Server & Client für C	libmqm_r.so
IA64 (IPF)	Mit Thread	Client für C	libmqic_r.so
IA64 (IPF)	Ohne Thread	Server & Client für C	libmqm.so
IA64 (IPF)	Ohne Thread	Client für C	libmqic.so

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 Verwalten), müssen Sie eine Verknüpfung zu der Bibliothek `libmqmzf.sl` 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.sl` (oder `libmqmxa64.sl`, falls Ihr Transaktionsmanager den Typ 'long' als 64-Bit behandelt) und `libmqz.sl` in einer anderen Anwendung als einer Thread-Anwendung bzw. zu den Bibliotheken `libmqmxa_r.sl` (oder `libmqmxa64_r.sl`) und `libmqz_r.sl` in einer Thread-Anwendung herstellen.
4. Sie müssen IBM MQ-Bibliotheken vor allen anderen Produktbibliotheken verknüpfen.

COBOL-Programme in HP-UX vorbereiten

Hier finden Sie Informationen zur Vorbereitung von COBOL-Programmen in HP-UX mithilfe von Micro Focus Server Express mit IBM MQ auf der IA64-Plattform (IPF) und zur Ausführung von Programmen in der IBM MQ MQI client-Umgebung.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Hinweise zur Verwendung:

1. 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
   ```

2. COBOL-Copybooks im 64-Bit-Format sind 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 auf

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   für 64-Bit-Anwendungen.

Kompilieren Sie die Programme mithilfe des Micro Focus-Compilers. Die Kopierdateien, in denen die Strukturen deklariert werden, befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB= MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="COBCPY_VALUE"
```

Kompilieren von 32-Bit-Programmen:

```
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Kompilieren von 64-Bit-Programmen:

```
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

Dabei ist `amqsput` ein Beispielprogramm

Stellen Sie sicher, dass Sie angemessene Stackgrößen für die Laufzeit angegeben haben; die Mindestempfehlung liegt bei 16 KB.

Sie müssen Ihre Programme mit der entsprechenden Bibliothek verknüpfen, die von IBM MQ bereitgestellt wird. In der folgenden Tabelle wird gezeigt, welche Bibliothek in den unterschiedlichen Umgebungen verwendet werden soll

Hardwareplattform	Programm/Exit-Typ	Bibliotheksdatei
IA64 (IPF)	Server für COBOL	libmqmcb.so
IA64 (IPF)	Client für COBOL	libmqicb.so
IA64 (IPF)	Thread-Anwendungen	libmqmcb_r.so

Micro Focus Server Express mit IBM MQ auf der IA64-Plattform (IPF) verwenden

Unter „Von IBM MQ for HP-UX on IA64 (IPF) unterstütztes Modell für Adressräume” auf Seite 1063 finden Sie Einzelheiten zur Verwendung von Micro Focus Server Express in Verbindung mit IBM MQ auf der HP/IPF-Plattform.

Programme für die Ausführung in der IBM MQ MQI client-Umgebung

Wenn Sie LU 6.2 verwenden, um Ihren MQI-Client mit einem Server zu verbinden, verknüpfen Sie Ihre Anwendung mit der Bibliothek 'libsna.a', die Teil des SNAplusAPI-Produkts ist. Verwenden Sie die Optionen `-lV3` und `-lstr` in Ihrem Befehl zum Kompilieren und Verknüpfen.

- Die Option `-lV3` ermöglicht Ihrem Programm den Zugriff auf die AT & T-Signalisierungsbibliothek (SNAplusAPI verwendet AT & T-Signale)
- Die Option `-lstr` verknüpft Ihr Programm mit den Komponenten der Datenströme

CICS-Programme in HP-UX vorbereiten

Hier finden Sie Informationen zum Erstellen von CICS-Transaktionsprogrammen in HP-UX.

Zum Erstellen der CICS-Beispieltransaktion 'amqscic0.ccs' führen Sie den folgenden Befehl aus:

```
$ export USERLIB="-lmqm_r"
$ cicstcl -l C amqscic0.ccs
```

Es wird ein XA-Switch-Modul bereitgestellt, damit Sie CICS mit IBM MQ verknüpfen können:

Tabelle 131. Essenzieller Code für CICS-Anwendungen (HP-UX)

Beschreibung	C (Quelle)	C (Exec)
XA-Initialisierungsroutine	amqzscix.c	amqzsc

Weitere Informationen zu unterstützenden CICS-Transaktionen finden Sie unter Verwalten.

TXSeries CICS-Unterstützung
IBM MQ on HP-UX unterstützt TXSeries CICS mithilfe der XA-Schnittstelle. Stellen Sie sicher, dass CICS-Anwendungen mit der Thread-Version der MQ-Bibliotheken verknüpft sind.

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 C-Beispieltransaktion

Ein Beispiel für einen C-Quellcode für eine CICS IBM MQ-Transaktion wird in AMQSCIC0.CCS bereitgestellt. Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE im Standardwarteschlangenmanager und reiht sie in die lokale Warteschlange mit dem Warteschlangennamen ein, der im Übertragungsheader der Nachricht enthalten ist. Mögliche Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICS.DLQ gesendet. Erstellen Sie mithilfe des MQSC-Beispielscripts AMQSCIC0.TST diese Warteschlangen und die Beispieleingabewarteschlangen.

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 dieser 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 on UNIX and 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 HP-UX
   - 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
   ```

Von IBM MQ for HP-UX on IA64 (IPF) unterstütztes Modell für Adressräume

In HP-UX werden mehrere Modelle für Adressräume bereitgestellt, die von IBM MQ-Anwendungen genutzt werden können.

HP-UX unterstützt zwei Adressraummodelle:

- MGAS - Mostly Global Address Space (dies ist die Standardeinstellung, die von IBM MQ verwendet wird)
- MPAS - Mostly Private Address Space

Anwendungen, die eine Verbindung zu IBM MQ herstellen, können die Adressraummodelle MGAS oder MPAS verwenden. Bei Anwendungen, die zur Verwendung des MPAS-Modells erstellt wurden und eine Verbindung zu IBM MQ mithilfe eines gemeinsam genutzten Speichers herstellen, kann es aufgrund der Ineffizienz bei der Zuordnung der von IBM MQ verwendeten Seiten für den gemeinsam genutzten Speicher zum virtuellen Adressraum des MPAS-Programms zu geringfügigen Leistungsbeeinträchtigungen kommen.

COBOL-Anwendungen, die mithilfe von Micro Focus Server Express erstellt wurden, verwenden standardmäßig das MPAS-Modell.

Mit dem Programm **chatr** können Sie das von einem Programm verwendete Adressmodell überprüfen und ändern.

Wenn beim Herstellen einer Verbindung zu IBM MQ aus 32-Bit-MPAS-Programmen Probleme auftreten, können Sie das MGAS-Adressmodell verwenden oder Ihre Anwendung statt als 32-Bit-MPAS-Anwendung als 64-Bit-MPAS-Anwendung erstellen.

Weitere Informationen zu den Modellen für MGAS- und MPAS-Adressräume finden Sie in der Dokumentation zu HP-UX.

Linux Prozedurale Anwendung auf Linux erstellen

In diesen Informationen werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie ausführen müssen, wenn Sie IBM MQ for Linux -Anwendungen für die Ausführung erstellen.

C und C++ werden unterstützt. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter C++ verwenden.

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 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.

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
```

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
```

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 ist `amqsput` ein Beispielprogramm

In der Dokumentation zu Micro Focus COBOL finden Sie eine Beschreibung der erforderlichen Umgebungsvariablen.

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 Codie-

rung 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 740.

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 verwenden:CRTCMOD. 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.

Hier finden Sie ein Beispiel für den Befehl in einer Umgebung ohne Threads:

Tabelle 132. Beispiel für CRTPGM in der Umgebung ohne Threads

Befehl	**Programm/Exit-Typ**
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM)`	Server oder Client für C

Dabei ist *pgmname* der Name Ihres Programms.

Hier finden Sie ein Beispiel für den Befehl in einer Threadumgebung:

Tabelle 133. Beispiel für CRTPGM in der Threadumgebung

Befehl	**Programm/Exit-Typ**
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM_R)`	Server oder Client für C

Dabei ist *pgmname* der Name Ihres Programms.

In den folgenden Tabellen werden die Bibliotheken aufgeführt, die zur Vorbereitung von Programmen in der Programmiersprache C in IBM i in einer Umgebung mit bzw. ohne Threads erforderlich sind.

Tabelle 134. Umgebung ohne Threads

Bibliotheksdatei	**Programm/Exit-Typ**
LIBMQM	Server für C
LIBMQIC und LIBMQM	Client für C

Tabelle 135. Umgebung mit Threads

Bibliotheksdatei	**Programm/Exit-Typ**
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 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.

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 oder CRTCICSCBL ein. Dadurch wird die Verarbeitung gestoppt und Sie können die entsprechenden CICS- und IBM MQ-Serviceprogramme einschließen. Mit diesem Befehl wird der Code standardmäßig in QTEMP/QACYCICS integriert.
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 1123 und Referenzinformationen zur Anwendungsprogrammierung für IBM i (ILE/RPG).

Überlegungen zur Programmierung von SQL

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)
```

Solaris Prozedurale Anwendung auf Solaris erstellen

In diesen Informationen werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie beim Erstellen von IBM MQ for Solaris-Anwendungen für die Ausführung unter Solaris durchführen müssen.

Die Programmiersprachen COBOL, C und C++ werden unterstützt. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter C++ verwenden.

Zusätzlich zur Codierung der MQI-Aufrufe in Ihrem Quellcode müssen Sie die entsprechenden Include-Dateien hinzufügen. Machen Sie sich mit den Inhalten dieser Dateien vertraut. Eine vollständige Beschreibung finden Sie unter „IBM MQ-Datendefinitionsdateien” auf Seite 740.

In diesem Abschnitt wird durchgängig der umgekehrte Schrägstrich (\) verwendet, um lange Befehle über mehr als eine Zeile zu trennen. Geben Sie dieses Zeichen nicht ein, geben Sie jeden Befehl in einer einzigen Zeile ein.

C-Programme in Solaris vorbereiten

Vorkompilierte C-Programme werden im Verzeichnis *MQ_INSTALLATION_PATH* `/samp/bin` bereitgestellt.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Weitere Informationen zur Programmierung von 64-Bit-Anwendungen finden Sie unter Codierungsstandards auf 64-Bit-Plattformen.

Wenn Sie die Programme auf einem System verwenden möchten, auf dem nur der IBM MQ MQI client for Solaris installiert ist, kompilieren Sie die Programme so, dass diese mit der Clientbibliothek (`-lmqic`) verknüpft sind.

Wenn Sie den nicht unterstützten Compiler `/usr/ucb/cc` verwenden, führt Ihre Anwendung das Kompilieren und Verknüpfen möglicherweise erfolgreich aus. Wenn Sie die Anwendung ausführen, schlägt sie allerdings beim Versuch, eine Verbindung zum Warteschlangenmanager herzustellen, fehl.

Anmerkung: Die für einen mit FIPS 140-2-konformen Betrieb konfigurierten Solaris x86-SSL- und TLS-Clients (32-Bit) schlagen bei der Ausführung auf Intel-Systemen fehl. Dieser Fehler tritt auf, da die FIPS 140-2-konforme GSKit-Crypto Solaris x86 32-Bit-Bibliotheksdatei auf dem Intel-Chipsatz nicht geladen werden kann. Auf betroffenen Systemen wird im Clientfehlerprotokoll der Fehler AMQ9655 gemeldet. Um dieses Problem zu beheben, müssen Sie die FIPS 140-2-Konformität inaktivieren oder die Clientanwendung erneut für 64-Bit kompilieren, da der 64-Bit-Code hiervon nicht betroffen ist.

Bibliotheken verknüpfen

Sie müssen eine Verknüpfung zu den IBM MQ-Bibliotheken herstellen, die für den Typ Ihrer Anwendung geeignet sind:

Bibliotheksdateien	Programm/Exit-Typ
libmqm.so	Server für C
libmqic.so & libmqm.so	Client für C

Anmerkung:

1. Wenn Sie einen installierbaren Service schreiben (weitere Informationen finden Sie unter Verwalten), stellen Sie eine Verknüpfung zu der Bibliothek `libmqmzf.so` her.

2. 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` herstellen.
3. Sie müssen IBM MQ-Bibliotheken vor allen anderen Produktbibliotheken verknüpfen.

Anwendungen auf der x86-64-Plattform erstellen
In diesem Abschnitt finden Sie Beispiele für die Befehle, die zum Erstellen von Programmen in verschiedenen Umgebungen auf der x86-64-Plattform verwendet werden.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

C-Clientanwendung, 32-Bit

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C-Clientanwendung, 64-Bit

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

C-Serveranwendung, 32-Bit

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C-Serveranwendung, 64-Bit

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

C++-Clientanwendung, 32-Bit

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as -lmqic -lsocket -lnsl -ldl
```

C++-Clientanwendung, 64-Bit

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

C++-Serveranwendung, 32-Bit

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

C++-Serveranwendung, 64-Bit

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

C-Client-Exit, 32-Bit

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

C-Client-Exit, 64-Bit

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

C-Server-Exit, 32-Bit

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

C-Server-Exit, 64-Bit

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

Anwendungen auf der SPARC-Plattform erstellen
In diesem Abschnitt finden Sie Beispiele für die Befehle, die zum Erstellen von Programmen in verschiedenen Umgebungen auf der SPARC-Plattform verwendet werden.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

C-Clientanwendung, 32-Bit

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C-Clientanwendung, 64-Bit

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

C-Serveranwendung, 32-Bit

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C-Serveranwendung, 64-Bit

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

C++-Clientanwendung, 32-Bit

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
```

```
-lmqic
-lsocket -lnsl -ldl
```

C++-Clientanwendung, 64-Bit

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

C++-Serveranwendung, 32-Bit

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATI
ON_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

C++-Serveranwendung, 64-Bit

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

C-Client-Exit, 32-Bit

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

C-Client-Exit, 64-Bit

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

C-Server-Exit, 32-Bit

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

C-Server-Exit, 64-Bit

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

COBOL-Programme in Solaris vorbereiten

Hier finden Sie Informationen zum Vorbereiten von COBOL-Programmen in Solaris.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

1. 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
```

2. COBOL-Copybooks im 64-Bit-Format sind 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 auf

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

für 64-Bit-Anwendungen.

Kompilieren Sie die Programme mithilfe des Micro Focus-Compilers. Die Kopierdateien, in denen die Strukturen deklariert werden, befinden Sie im Verzeichnis *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB= MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="COBCPY_VALUE"
```

Kompilieren von 32-Bit-Programmen:

- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb`

  Server für COBOL
- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb`

  Client für COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r`

  Thread-Server für COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r`

  Thread-Client für COBOL

Kompilieren von 64-Bit-Programmen:

- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb`

  Server für COBOL
- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb`

  Client für COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r`

  Thread-Server für COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r`

  Thread-Client für COBOL

Dabei ist *amqs0put0.cbl* ein Beispielprogramm.

Sie müssen Ihr Programm mit einer der folgenden Bibliotheksdateien verknüpfen:

- libmqmcb.so

  Server für COBOL
- libmqicb.so

  Client für COBOL

CICS-Programme in Solaris vorbereiten

Hier finden Sie Informationen zum Vorbereiten von CICS-Programmen in Solaris.

Es wird ein XA-Switch-Modul bereitgestellt, damit Sie CICS mit IBM MQ verknüpfen können:

Tabelle 136. Essenzieller Code für CICS-Anwendungen (Solaris)

Beschreibung	**C (Quelle)**	**C (Exec)**
XA-Initialisierungsroutine	amqzscix.c	amqzsc - TXSeries für Solaris

Verbinden Sie Ihre Transaktionen immer mit der threadsicheren IBM MQ-Bibliothek 'libmqm.so'.

Weitere Informationen zu unterstützenden CICS-Transaktionen finden Sie unter Verwalten.

TXSeries CICS-Unterstützung
IBM MQ for Solaris unterstützt TXSeries CICS über die XA-Schnittstelle.

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 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
   ```

   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, verschieben Sie diese Funktionen aus den COBOL-Anwendungen in Ihre eigene Bibliothek (z. B. `myMQ.so`). Nach dem Verschieben dieser 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 on UNIX and 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 TXSeries for Solaris
   - 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

Erstellen Sie CICS C-Programme mithilfe der CICS-Standardfunktionen:

1. Exportieren Sie ***eine*** der folgenden Umgebungsvariablen:
   - LDFLAGS = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -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 CICS IBM MQ-Transaktion wird in AMQSCIC0.CCS bereitgestellt. Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE im Standardwarteschlangenmanager und reiht sie in die lokale Warteschlange mit dem Warteschlangennamen ein, der im Übertragungsheader der Nachricht enthalten ist. Mögliche Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICS.DLQ gesendet. Erstellen Sie mithilfe des MQSC-Beispielscripts AMQSCIC0.TST diese Warteschlangen und die Beispieleingabewarteschlangen.

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 ActiveX, C, C++, COBOL und Visual Basic werden unterstützt. Informationen zur Vorbereitung Ihrer ActiveX-Programme finden Sie unter Component Object Model-Schnittstelle verwenden (WebSphere MQ Automation Classes for ActiveX). 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 740.

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 137. Position der IBM MQ-Bibliotheken

Bibliotheksversion	**Verzeichnis mit den Bibliotheksdateien**
32 Bit	*MQ_INSTALLATION_PATH* \Tools\Lib

Tabelle 137. Position der IBM MQ-Bibliotheken (Forts.)	
Bibliotheksversion	**Verzeichnis mit den Bibliotheksdateien**
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.

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\mqic-xa.lib	Client für 32-Bit-C mit Transaktionskoordination
MQ_INSTALLATION_PATH \Tools\Lib64\mqm.lib	Server für 64-Bit-C
MQ_INSTALLATION_PATH \Tools\Lib64\mqic.lib	Client für 64-Bit-C
MQ_INSTALLATION_PATH \Tools\Lib64\mqic-xa.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 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 nach Möglichkeit Kleinbuchstaben verwenden, können Sie auf UNIX and Linux -Systemen von IBM MQ for Windows nach IBM MQ 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 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.

V 9.0.0 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 1119 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 wie folgt auf 1 für einen Server oder 2 für einen Client:

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 1119
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 961
Unter Windows können Sie Microsoft Visual Basic-Anwendungen mit dem IBM MQ MQI client-Code verknüpfen.

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 1209.

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 138. 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 1087). 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 Prozedurale Anwendung auf z/OS erstellen

In den Veröffentlichungen zu CICS, IMS und z/OS ist beschrieben, wie Sie Anwendungen erstellen können, die in diesen Umgebungen ausgeführt werden.

In dieser Sammlung von Themen werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie beim Erstellen von IBM MQ for z/OS-Anwendungen für diese Umgebungen ausführen müssen. Die Programmiersprachen COBOL, C, C++, Assembler und PL/I werden unterstützt. (Informationen zur Erstellung von C++-Anwendungen finden Sie unter C++ verwenden.)

Die Tasks, die Sie zum Erstellen einer ausführbaren IBM MQ for z/OS-Anwendung ausführen müssen, sind von der Programmiersprache, in der das Programm geschrieben ist, und der Umgebung abhängig, in der die Anwendung ausgeführt werden soll.

Zusätzlich zur Codierung der MQI-Aufrufe in Ihrem Programm müssen Sie die entsprechenden Sprachanweisungen hinzufügen, um die IBM MQ for z/OS-Datendefinitionsdatei 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 740.

Hinweis

Der Name **thlqual** gibt das übergeordnete Qualifikationsmerkmal der Installationsbibliothek unter z/OS an.

Programm für die Ausführung vorbereiten

Nachdem Sie das Programm zum Erstellen einer ausführbaren Anwendung für Ihre IBM MQ-Anwendung geschrieben haben, müssen Sie es kompilieren oder assemblieren und für den entstandenen Objektcode eine Programmverknüpfung zu dem Stubprogramm herstellen, das von IBM MQ for z/OS für jede unterstützte Umgebung bereitgestellt wird.

Die Vorbereitung Ihres Programms ist von der Umgebung (Batch, CICS, IMS (BMP oder MPP), Linux oder UNIX System Services), in der die Anwendung ausgeführt wird, und der Struktur der Datensätze in Ihrer z/OS-Installation abhängig.

Im Abschnitt „IBM MQ-Stub dynamisch aufrufen” auf Seite 1094 wird eine alternative Methode zum Ausgeben von MQI-Aufrufen in Ihren Programmen beschrieben, damit Sie keine Programmverknüpfung zu einem IBM MQ-Stub herstellen müssen. Diese Methode ist nicht für alle Sprachen und Umgebungen verfügbar.

Stellen Sie keine Programmverknüpfung eines Stubprogramms mit einer höheren Version als der Version von IBM MQ for z/OS her, mit der Ihr Programm ausgeführt wird. Beispielsweise darf für ein Programm, das unter MQSeries for OS/390 5.2 ausgeführt wird, keine Programmverbindung mit einem Stubprogramm hergestellt werden, das mit IBM MQ for z/OS 7 bereitgestellt wird.

64-Bit-C-Anwendungen erstellen

In z/OS werden 64-Bit-C-Anwendungen unter Verwendung der LP64-Compiler- und Bindeoptionen erstellt. Die IBM MQ for z/OS-Headerdatei *cmqc.h* erkennt es, wenn diese Option für den Compiler angegeben wird, und generiert entsprechende IBM MQ-Datentypen und -Strukturen für den Betrieb mit 64 Bit.

Mit dieser Option erstellter C-Code muss so erstellt werden, dass die entsprechenden DLL-Dateien für die erforderliche Koordinationssemantik verwendet werden. Wenn der kompilierte Code mit dem entsprechenden im Abschnitt Für jede Koordinationssemantik erforderlicher Sidedeckname definierten Sidedeck gebunden wird, wird die spezielle erforderliche DLL angezeigt.

Tabelle 139. Für jede Koordinationssemantik erforderlicher Sidedeckname

Koordination	**Sidedeckname**
Einphasencommit-MQI	CSQBMQ2X
Zweiphasiges Commit mit RRS-Koordination unter Verwendung von RRS-Verben	CSQBRR2X
Zweiphasiges Commit mit RRS-Koordination unter Verwendung von MQI-Verben	CSQBRI2X

Verwenden Sie die mit *z/OS XL C/C++* bereitgestellte EDCQCB-JCL-Prozedur, um wie folgt ein Einphasencommit-IBM MQ-Programm als Batch-Job zu erstellen:

```
//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
```

Führen Sie zum Erstellen eines koordinierten RRS-Programms in z/OS Unix System Services wie folgt eine Kompilierung und Verknüpfung durch:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *z/OS-Batchanwendungen erstellen*

In diesem Abschnitt finden Sie Informationen zum Erstellen von z/OS-Batchanwendungen und die Schritte, die Sie dabei berücksichtigen sollten.

Zum Erstellen einer Anwendung für IBM MQ for z/OS, die auf einem z/OS-Batch ausgeführt wird, erstellen Sie die JCL (Job Control Language), mit der diese Tasks ausgeführt werden:

1. Kompilieren (oder Assemblieren) des Programms zum Erzeugen des Objektcodes. Die JCL für Ihre Kompilierung muss SYSLIB-Anweisungen enthalten, mit denen die Definitionsdateien der Produktdaten für den Compiler verfügbar werden. Die Datendefinitionen werden in den folgenden IBM MQ for z/OS-Bibliotheken bereitgestellt:
   - Für COBOL: **thlqual**.SCSQCOBC
   - Für Assemblersprache: **thlqual**.SCSQMACS
   - Für C: **thlqual**.SCSQC370
   - Für PL/I: **thlqual**.SCSQPLIC
2. Herstellen einer Programmverknüpfung mit dem in Schritt „1" auf Seite 1089 erstellten Objektcode in einer C-Anwendung.
3. Verwenden der Compileroption EXTRN(SHORT) für PL/I-Anwendungen.
4. Herstellen einer Programmverknüpfung mit dem in Schritt „1" auf Seite 1089 (oder Schritt „2" auf Seite 1089 für eine C-Anwendung) erstellten Objektcodes zum Erzeugen eines Lademoduls. Wenn Sie eine Programmverknüpfung mit dem Code herstellen, müssen Sie eines der Stubprogramme für den IBM MQ for z/OS-Batch (CSQBSTUB oder eines der RRS-Stubprogramme: CSQBRRSI oder CSQBRSTB) einschließen.

   CSQBSTUB
   Einphasige Festschreibung, die von IBM MQ for z/OS bereitgestellt wird.

   CSQBRRSI
   Zweiphasige Festschreibung, die von RRS über die MQI bereitgestellt wird.

   CSQBRSTB
   Zweiphasige Festschreibung, die direkt von RRS bereitgestellt wird.

   Anmerkungen:

   a. Wenn Sie CSQBRSTB verwenden, müssen Sie für Ihre Anwendung außerdem eine Programmverknüpfung mit ATRSCSS aus SYS1.CSSLIB herstellen. In Abbildung 125 auf Seite 1089 und Abbildung 126 auf Seite 1090 werden Fragmente des JCL-Codes für diesen Vorgang gezeigt. Die Stubs sind sprachunabhängig und werden in der Bibliothek **thlqual**.SCSQLOAD bereitgestellt.

   b. Wenn Ihre Anwendung unter Language Environment ausgeführt wird, sollten Sie sicherstellen, dass stattdessen eine Programmverbindung mit der Language Environment-DLL hergestellt wird, wie im Abschnitt „z/OS-Batchanwendungen mithilfe von Language Environment erstellen" auf Seite 1090 beschrieben.
5. Speichern des Lademoduls in einer Ladebibliothek für die Anwendung.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

Abbildung 125. Fragmente des JCL-Codes zum Herstellen einer Programmverknüpfung mit dem Objektmodul in der Batchumgebung mithilfe einer einphasigen Festschreibung

```
⋮
//*
//* 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)
⋮
/*
```

Abbildung 126. Fragmente des JCL-Codes zum Herstellen einer Programmverknüpfung mit dem Objektmodul in der Batchumgebung mithilfe einer zweiphasigen Festschreibung

Zur Ausführung eines Batch- oder RRS-Programms müssen Sie die Bibliotheken **thlqual**.SCSQAUTH und **thlqual**.SCSQLOAD in die STEPLIB- oder JOBLIB-Verkettung des Datensatzes einschließen.

Für die Ausführung eines TSO-Programms müssen Sie die Bibliotheken **thlqual**.SCSQAUTH und **thlqual**.SCSQLOAD in die von der TSO-Sitzung verwendete STEPLIB integrieren.

Für die Ausführung eines Batchprogramms unter UNIX System Services aus der UNIX System Services-Shell fügen Sie die Bibliotheken **thlqual**.SCSQAUTH und **thlqual**.SCSQLOAD der STEPLIB-Spezifikation in $HOME?.profile folgendermaßen hinzu:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *z/OS-Batchanwendungen mithilfe von Language Environment erstellen*

In IBM MQ for z/OS wird eine Gruppe von DLL-Dateien (Dynamic Link Libraries, DLLs) bereitgestellt, die beim Herstellen einer Programmverknüpfung zu Ihren Anwendungen verwendet werden können.

Es sind zwei Varianten der Bibliotheken vorhanden, die es der Anwendung ermöglichen, eine der folgenden Aufrufschnittstellen zu verwenden:

- Die 32-Bit-Aufrufschnittstelle Language Environment.
- Die 31-Bit Aufrufschnittstelle XPLINK. Bei z/OS XPLINK handelt es sich um eine Anrufvereinbarung mit hoher Leistung, die für C-Anwendungen verfügbar ist.

Für die Verwendung der DLL-Dateien wird die Anwendung mit sogenannten *Sidedecks* verbunden oder verknüpft, statt mit den Stubs, die in früheren Versionen bereitgestellt wurden. Die Sidedecks befinden sich in der Bibliothek SCSQDEFS (statt in der Bibliothek SCSQLOAD).

Tabelle 140. Varianten von DDL-Dateien (Dynamic Link Library)

Commit	31-Bit-DLL für Language Environment	31-Bit-DLL für XPLINK	Äquivalenter Stubname
Einphasige Festschreibung von MQI-Bibliotheken	CSQBMQ1	CSQBMQ1X	CSQBSTUB
Zweiphasige Festschreibung mit RRS-Koordination mithilfe von RRS-Transaktionssteuerungsverben	CSQBRR1	CSQBRR1X	CSQBRSTB

Tabelle 140. Varianten von DDL-Dateien (Dynamic Link Library) (Forts.)			
Commit	**31-Bit-DLL für Language Environment**	**31-Bit-DLL für XPLINK**	**Äquivalenter Stubname**
Zweiphasige Festschreibung mit RRS-Koordination mithilfe von MQI-Transaktionssteuerungsverben	CSQBRI1	CSQBRI1X	CSQBRRSI

Anmerkung: Alle Sidedecks enthalten eine Definition des Datenkonvertierungseingangspunkts (MQXCNVC), der zuvor durch Integration von CSQASTUB aufgelöst wurde.

Allgemeine Probleme:

- Die folgende Nachricht wird im Jobprotokoll angezeigt, wenn Ihre Anwendung asynchrone Nachrichtenverarbeitung verwendet (MQCB-, MQCTL- oder MQSUB-Aufrufe) und die vorherige DLL-Schnittstelle nicht verwendet wird:

  ```
  CSQB001E Language environment programs running in z/OS batch or USS must use the DLL interface
  to IBM MQ
  ```

  Lösung: Erstellen Sie Ihre Anwendung erneut und verwenden Sie Sidedecks statt Stubs, wie oben ausführlich beschrieben.

- Bei der Programmerstellung wird folgende Nachricht angezeigt:

  ```
  IEW2469E The Attributes of a reference to MQAPI-NAME from section your-code do not match the
  attributes of
  the target symbol
  ```

  Ursache: Dies bedeutet, dass Sie Ihr XPLINK-Programm mit Version V701 (oder höher) von cmqc.h kompiliert haben, aber keine Bindung mit Sidedecks vorhanden ist.

  Lösung: Ändern Sie die Builddatei Ihres Programms, indem Sie eine Bindung mit dem geeigneten Sidedeck aus SCSQDEFS statt mit einem Stub aus SCSQLOAD durchführen.

In der folgenden Beispiel-JCL wird veranschaulicht, wie Sie ein C-Programm kompilieren und eine Programmverknüpfung dafür herstellen können, um die Aufrufschnittstelle für die 31-Bit-DLL 'Language Environment' zu verwenden:

```
//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)
//
```

Anmerkung: Bei der Kompilierung wird die Option **DLL** verwendet. Die Programmverknüpfung verwendet die Option **DYNAM=DLL** und verweist auf die Bibliothek **CSQBMQ1**.

In der folgenden Beispiel-JCL wird veranschaulicht, wie Sie ein C-Programm kompilieren und eine Programmverknüpfung dafür herstellen können, um die Aufrufschnittstelle für die 31-Bit-DLL XPLINK zu verwenden:

```
//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)
//
```

Anmerkung: Bei der Kompilierung werden die Optionen **XPLINK** und **DLL** verwendet. Die Programmverknüpfung verwendet die Option **DYNAM=DLL** und verweist auf die Bibliothek **CSQBMQ1X**.

Stellen Sie sicher, dass Sie die Kompilierungsoption 'DLL' jedem Programm im Modul hinzufügen. Durch Nachrichten wie beispielsweise IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED wird angezeigt, dass Sie prüfen müssen, ob alle Programme mit der DLL-Option kompiliert wurden.

CICS-Anwendungen in z/OS erstellen
Verwenden Sie diese Informationen zum Erstellen von CICS-Anwendungen in z/OS.

Zum Erstellen einer Anwendung für IBM MQ for z/OS die unter CICS ausgeführt wird, müssen Sie folgendermaßen vorgehen:

- Übersetzen Sie die CICS-Befehle in Ihrem Programm in die Sprache, in der der Rest Ihres Programms geschrieben ist.
- Kompilieren oder Assemblieren Sie die Ausgabe des Umsetzungsprogramms, um einen Objektcode zu erstellen.
  - Verwenden der Compileroption EXTRN(SHORT) für PL/I-Programme.
  - Wenn die Anwendung in C-Anwendungen XPLINK nicht verwendet, verwenden Sie die Compileroption DEFINE(MQ_OS_LINKAGE=1).
- Stellen Sie eine Programmverknüpfung zum Objektcode her, um ein Lademodul zu erzeugen.

In CICS wird eine Prozedur bereitgestellt, mit der diese Schritte nacheinander für jede unterstützte Programmiersprache ausgeführt werden.

- Informationen zur Verwendung dieser Prozeduren in CICS Transaction Server for z/OS finden Sie im Handbuch *CICS Transaction Server for z/OS System Definition Guide*; zusätzliche Angaben über den Umsetzungsprozess bietet das Handbuch *CICS/ESA Application Programming Guide*.

Sie müssen die folgenden Komponenten integrieren:

- In die Anweisung SYSLIB für die Kompilierungsphase (oder Assemblierungphase) müssen Anweisungen integriert werden, durch die die Definitionsdateien der Produktdaten für den Compiler verfügbar werden. Die Datendefinitionen werden in den folgenden IBM MQ for z/OS-Bibliotheken bereitgestellt:
  - Für COBOL: **thlqual**.SCSQCOBC
  - Für Assemblersprache: **thlqual**.SCSQMACS
  - Für C: **thlqual**.SCSQC370
  - Für PL/I: **thlqual**.SCSQPLIC
- In Ihre JCL zum Herstellen einer Programmverknüpfung: Das IBM MQ for z/OS CICS-Stubprogramm (CSQCSTUB). In Abbildung 127 auf Seite 1093 werden Fragmente des JCL-Codes für diesen Vorgang gezeigt. Der Stub ist sprachunabhängig und wird in der Bibliothek **thlqual**.SCSQLOAD bereitgestellt.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

Abbildung 127. Fragmente des JCL-Codes zum Herstellen einer Programmverknüpfung mit dem Objektmodul in der CICS-Umgebung

- Wenn Sie eine höhere CICS-Version als CICS TS 3.2 verwenden oder APIs für die IBM MQ-Nachrichteneigenschaft oder die IBM MQ-APIs MQCB, MQCTL, MQSTAT, MQSUB oder MQSUBR verwenden, müssen Sie eine Programmverknüpfung von Ihrem Objektcode zu dem mit CICS bereitgestellten Stub DFHMQSTB und nicht zu dem mit IBM MQ bereitgestellten Stub CSQCSTUB herstellen. Weitere Informationen zum Erstellen von IBM MQ-Programmen für CICS finden Sie in der CICS-Produktdokumentation unter API stub program to access IBM MQ MQI calls.

Nach dem Ausführen dieser Schritte speichern Sie das Lademodul in einer Ladebibliothek für die Anwendung und definieren das Programm für CICS in der gewohnten Weise.

Vor der Ausführung eines CICS-Programms muss Ihr Systemadministrator es für CICS als Programm und Transaktion für IBM MQ definieren. Anschließend können Sie es wie gewohnt ausführen.

BMP- oder MPP-Anwendungen unter IMS erstellen
Verwenden Sie diese Informationen zum Erstellen von BMP- oder MPP-Anwendungen unter IMS.

Wenn Sie DL/I-Batchprogramme erstellen, lesen Sie den Abschnitt „z/OS-Batchanwendungen erstellen” auf Seite 1089. Zum Erstellen anderer Anwendungen, die unter IMS ausgeführt werden (als BMP- oder MPP-Anwendung), erstellen Sie die JCL, mit der die folgenden Tasks ausgeführt werden:

1. Kompilieren (oder Assemblieren) des Programms zum Erzeugen des Objektcodes. Die JCL für Ihre Kompilierung muss SYSLIB-Anweisungen enthalten, mit denen die Definitionsdateien der Produktdaten für den Compiler verfügbar werden. Die Datendefinitionen werden in den folgenden IBM MQ for z/OS-Bibliotheken bereitgestellt:
   - Für COBOL: **thlqual**.SCSQCOBC
   - Für Assemblersprache: **thlqual**.SCSQMACS
   - Für C: **thlqual**.SCSQC370
   - Für PL/I: **thlqual**.SCSQPLIC
2. Herstellen einer Programmverknüpfung mit dem in Schritt „1” auf Seite 1093 erstellten Objektmodul in einer C-Anwendung.
3. Verwenden der Compileroption EXTRN(SHORT) für PL/I-Programme.
4. In einer C-Anwendung wird die Compileroption DEFINE(MQ_OS_LINKAGE=1) verwendet, wenn die Anwendung nicht XPLINK verwendet.
5. Herstellen einer Programmverknüpfung mit dem in Schritt „1” auf Seite 1093 (oder Schritt „2” auf Seite 1093 bei einer C/370-Anwendung) erstellten Objektcodes zum Erzeugen eines Lademoduls:
   a. Integrieren des IMS-Sprachschnittstellenmoduls (DFSLI000).
   b. Integrieren des Stubprogramms für IBM MQ for z/OS IMS (CSQQSTUB). In Abbildung 128 auf Seite 1094 werden Fragmente des JCL-Codes für diesen Vorgang gezeigt. Der Stub ist sprachunabhängig und wird in der Bibliothek **thlqual**.SCSQLOAD bereitgestellt.

      Anmerkung: Wenn Sie COBOL verwenden, wählen Sie zum Aktivieren des Verbindungseditors die Compileroption NODYNAM aus, um Verweise auf CSQQSTUB aufzulösen, es sei denn, Sie möchten die im Abschnitt „IBM MQ-Stub dynamisch aufrufen” auf Seite 1094 beschriebene dynamische Verknüpfung verwenden.

6. Speichern des Lademoduls in einer Ladebibliothek für die Anwendung.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Abbildung 128. Fragmente des JCL-Codes zum Herstellen einer Programmverknüpfung mit dem Objektmodul in der IMS-Umgebung

Vor der Ausführung eines IMS-Programms muss Ihr Systemadministrator es für IMS als Programm und Transaktion für IBM MQ definieren. Anschließend können Sie es wie gewohnt ausführen.

Anwendungen für z/OS UNIX System Services erstellen
Verwenden Sie diese Informationen zum Erstellen von Anwendungen für z/OS UNIX System Services.

Zum Erstellen einer C-Anwendung für IBM MQ for z/OS, die auf UNIX System Services ausgeführt wird, kompilieren und verknüpfen Sie Ihre Anwendung folgendermaßen:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

Dabei ist **thlqual** das von Ihrer Installation verwendete übergeordnete Qualifikationsmerkmal.

Zur Ausführung des C-Programms müssen Sie den folgenden Code Ihrer `.profile`-Datei hinzufügen, die sich im Stammverzeichnis befinden sollte:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Beachten Sie, dass Sie UNIX System Services beenden und UNIX System Services erneut starten müssen, damit die Änderung erkannt wird.

Wenn Sie mehrere Shells ausführen möchten, fügen Sie den Ausdruck 'export' am Anfang jeder Zeile hinzu:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Nach erfolgreichem Abschluss können Sie die CSQBSTUB verlinken und IBM MQ -Aufrufe absetzen.

Im Abschnitt „IBM MQ-Stub dynamisch aufrufen” auf Seite 1094 wird eine alternative Methode zum Ausgeben von MQI-Aufrufen in Ihren Programmen beschrieben, damit Sie keine Programmverknüpfung zu einem IBM MQ-Stub herstellen müssen. Diese Methode ist nicht für alle Sprachen und Umgebungen verfügbar.

Stellen Sie keine Programmverknüpfung eines Stubprogramms mit einer höheren Version als der Version von IBM MQ for z/OS her, mit der Ihr Programm ausgeführt wird. Beispielsweise darf ein Programm, das in IBM WebSphere MQ for z/OS 7.1 ausgeführt wird, nicht mit einem mit IBM MQ for z/OS 8.0 bereitgestellten Stubprogramm verknüpft sein.

IBM MQ-Stub dynamisch aufrufen

Statt eine Programmverbindung des IBM MQ-Stubprogramms mit Ihrem Objektcode herzustellen, können Sie den Stub aus dem Programm dynamisch aufrufen.

Sie können dies im Batch, in IMS und in CICS-Umgebungen vornehmen. Diese Funktion wird in der RRS-Umgebung nicht unterstützt. Wenn Ihr Anwendungsprogramm Aktualisierungen mithilfe von RRS koordiniert, lesen Sie den Abschnitt „Überlegungen zu RRS” auf Seite 1099.

Diese Methode hat allerdings die folgenden Auswirkungen:

- Zunahme der Komplexität Ihrer Programme
- Zunahme des für Ihre Programme während der Ausführung erforderlichen Speichers
- Verringerung der Leistung Ihrer Programme
- Die gleichen Programme können nicht in anderen Umgebungen verwendet werden

Wenn Sie den Stub dynamisch aufrufen, müssen das entsprechende Stubprogramm und die zugehörigen Aliasnamen während der Ausführung verfügbar sein. Um dies sicherzustellen, schließen Sie den IBM MQ for z/OS-Datensatz SCSQLOAD ein:

- Für Stapelverarbeitung und IMS in die Verkettung STEPLIB des JCL-Codes.
- Für CICS in die CICS-Verkettung DFHRPL.

Stellen Sie für IMS sicher, dass sich die Bibliothek mit dem dynamischen Stub (der wie in den Informationen zur Installation des IMS-Adapters im Abschnitt IMS-Adapter einrichten beschrieben erstellt wurde) vor dem Datensatz SCSQLOAD in der STEPLIB-Verkettung der JCL für die Region befindet.

Verwenden Sie beim dynamischen Aufrufen des Stubs die in Tabelle 141 auf Seite 1095 gezeigten Namen. Deklarieren Sie in PL/I nur die Aufrufnamen, die in Ihrem Programm verwendet werden.

Tabelle 141. Aufrufnamen für dynamisches Verbinden

MQI-Aufruf	**Dynamische Aufrufnamen für die Stapelverarbeitung (nicht RRS)**	**Dynamische Aufrufnamen für CICS**	**Dynamische Aufrufnamen für IMS**
MQBACK	CSQBBACK	nicht unterstützt	Nicht unterstützt
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	Nicht unterstützt
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	nicht unterstützt	Nicht unterstützt
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	Nicht unterstützt
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

Tabelle 141. Aufrufnamen für dynamisches Verbinden (Forts.)			
MQI-Aufruf	**Dynamische Aufrufnamen für die Stapelverarbeitung (nicht RRS)**	**Dynamische Aufrufnamen für CICS**	**Dynamische Aufrufnamen für IMS**
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Anmerkung: 1. Diese API-Aufrufe sind nur verfügbar, wenn CICS TS 3.2 oder höher verwendet wird und das mit CICS bereitgestellte Stubprogramm CSQCSTUB verwendet werden muss. Für CICS TS 3.2 muss APAR PK66866 angewendet werden. Für CICS TS 4.1 muss APAR PK89844 angewendet werden.

Beispiele zur Verwendung dieses Verfahrens finden Sie in den folgenden Abbildungen:

- Stapelverarbeitung und COBOL: siehe Abbildung 129 auf Seite 1096
- CICS und COBOL: siehe Abbildung 130 auf Seite 1096
- IMS und COBOL: siehe Abbildung 131 auf Seite 1097
- Stapelverarbeitung und Assembler: siehe Abbildung 132 auf Seite 1097
- CICS und Assembler: siehe Abbildung 133 auf Seite 1097
- IMS und Assembler: siehe Abbildung 134 auf Seite 1097
- Stapelverarbeitung und C: siehe Abbildung 135 auf Seite 1098
- CICS und C: siehe Abbildung 136 auf Seite 1098
- IMS und C: siehe Abbildung 137 auf Seite 1098
- Stapelverarbeitung und PL/I: siehe Abbildung 138 auf Seite 1098
- IMS und PL/I: siehe Abbildung 139 auf Seite 1099

```
...
       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.
...
```

Abbildung 129. Dynamisches Verbinden mithilfe von COBOL in der Batchumgebung

```
...
       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.
...
```

Abbildung 130. Dynamisches Verbinden mithilfe von COBOL in der CICS-Umgebung

```
...
      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.
...
```

Abbildung 131. Dynamisches Verbinden mithilfe von COBOL in der IMS-Umgebung

```
...
        LOAD   EP=CSQBOPEN
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        DELETE EP=CSQBOPEN
...
```

Abbildung 132. Dynamisches Verbinden mithilfe der Assemblersprache in der Batchumgebung

```
...
        EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

Abbildung 133. Dynamisches Verbinden mithilfe der Assemblersprache in der CICS-Umgebung

```
...
        LOAD   EP=MQOPEN
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        DELETE EP=MQOPEN
...
```

Abbildung 134. Dynamisches Verbinden mithilfe der Assemblersprache in der IMS-Umgebung

```
...
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);
...
```

Abbildung 135. Dynamisches Verbinden mithilfe der Programmiersprache C in der Batchumgebung

```
...
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);
...
```

Abbildung 136. Dynamisches Verbinden mithilfe der Programmiersprache C in der CICS-Umgebung

```
...
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);
...
```

Abbildung 137. Dynamisches Verbinden mithilfe der Programmiersprache C in der IMS-Umgebung

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

Abbildung 138. Dynamisches Verbinden mithilfe von PL/I in der Batchumgebung

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

Abbildung 139. Dynamisches Verbinden mithilfe von PL/I in der IMS-Umgebung

Überlegungen zu RRS
Sie können diese Informationen verwenden, wenn Ihr Anwendungsprogramm RRS zum Koordinieren von Aktualisierungen verwendet.

In IBM MQ werden zwei unterschiedliche Stubs für Batchprogramme bereitgestellt, für die RRS-Koordination erforderlich ist - siehe „RRS-Batch-Adapter” auf Seite 930. Diese Unterschiede im Verhalten späterer API-Aufrufe werden beim Aufruf von MQCONN durch den Batchadapter aus Informationen ermittelt, die von der Stubroutine in der MQCONN- oder MQCONNX-API übergeben wurden. Das bedeutet, dass dynamische API-Aufrufe für Batchprogramme verfügbar sind, für die die RRS-Koordination erforderlich ist, vorausgesetzt, die einleitende Verbindung mit IBM MQ wurde mithilfe des entsprechenden Stubs vorgenommen. Dies wird im folgenden Beispiel veranschaulicht:

```
       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.
```

Fehlerbehebung für Ihre Programme ausführen

In diesem Abschnitt finden Sie Informationen zur Fehlerbehebung in TSO- und CICS-Programmen und erhalten Einblicke in den CICS-Trace.

Die wichtigste Unterstützung bei der Fehlerbehebung für IBM MQ for z/OS-Anwendungsprogramme sind die von jedem API-Aufruf zurückgegebenen Ursachencodes. Eine Liste der Ursachencodes und Konzepte zur Fehlerberichtigung finden Sie in den folgenden Abschnitten:

- IBM MQ for z/OS -Nachrichten, -Beendigungscodes und -Ursachencodes for IBM MQ for z/OS
- Nachrichten und Ursachencodes für alle anderen IBM MQ-Plattformen

In diesem Abschnitt werden auch andere Tools zur Fehlerbehebung für die Verwendung in bestimmten Umgebungen empfohlen.

Fehlerbehebung in TSO-Programmen

Die folgenden interaktiven Tools zur Fehlerbehebung sind für TSO-Programme verfügbar:

- TEST-Tool
- Interaktives Tool zur Fehlerbehebung für VS COBOL II
- Interaktives INSPECT-Tool zur Fehlerbehebung für C- und PL/I-Programme

Fehlerbehebung für CICS-Programme

Mit dem Tool CICS Execution Diagnostic Facility (CEDF) können Sie Ihre CICS-Programme interaktiv testen, ohne das Programm oder die Prozedur zur Programmerstellung ändern zu müssen.

Weitere Informationen zu EDF finden Sie im Handbuch *CICS Transaction Server for z/OS CICS Application Programming Guide*.

CICS-Trace

Die Transaktion zur Tracesteuerung von CICS (CETR), mit der die CICS-Traceaktivität gesteuert wird, ist möglicherweise ebenfalls hilfreich.

Weitere Informationen zu CETR finden Sie im Handbuch *CICS Transaction Server for z/OS CICS-Supplied Transactions*.

Um festzustellen, ob der CICS-Trace aktiv ist, zeigen Sie den Verbindungsstatus über die CKQC-Anzeige an. In dieser Anzeige wird auch die Tracenummer dargestellt.

Informationen zum Interpretieren von CICS-Traceeinträgen finden Sie in Tabelle 142 auf Seite 1100.

Der CICS-Traceeintrag für diese Werte lautet AP0 *xxx* (wobei *xxx* die Tracenummer ist, die Sie bei der Aktivierung des CICS-Adapters angegeben haben). Mit Ausnahme von CSQCTEST werden alle Traceeinträge von CSQCTRUE ausgegeben. CSQCTEST wird von CSQCRST und CSQCDSP ausgegeben.

Tabelle 142. Traceeinträge des CICS-Adapters

Name	Beschreibung	Tracesequenz	Tracedaten
CSQCABNT	Abnormale Beendigung	Vor der Ausgabe von END_THREAD ABNORMAL an IBM MQ. Der Grund dafür ist, dass das Ende der Task und ein implizites Backout von der Anwendung ausgeführt werden konnten. In diesem Fall wird in den Aufruf 'END_THREAD' die Anfrage 'ROLLBACK' einbezogen.	Informationen zur Arbeitseinheit. Anhand dieser Informationen können Sie den Arbeitsstatus überprüfen. (Er kann beispielsweise anhand der Ausgabe geprüft werden, die vom Befehl DISPLAY THREAD erzeugt wurde, oder anhand des Druckdienstprogramms für das IBM MQ for z/OS-Protokoll.)
CSQCBACK	Synchronisationspunkt-Backout	Vor der Ausgabe von BACKOUT an IBM MQ for z/OS. Dies ist auf eine explizite Backout-Anforderung der Anwendung zurückzuführen.	Informationen zur Arbeitseinheit.
CSQCCCRC	Beendigungscode und Ursachencode	Nach der nicht erfolgreichen Rückgabe von einem API-Aufruf	Beendigungscode und Ursachencode

Tabelle 142. Traceeinträge des CICS-Adapters (Forts.)

Name	Beschreibung	Tracesequenz	Tracedaten
CSQCCOMM	Synchronisationspunkt-Festschreibung	Vor der Ausgabe von COMMIT an IBM MQ for z/OS. Dies kann auf eine einphasige Festschreibungsanfrage oder die zweite Phase einer zweiphasigen Festschreibungsanfrage zurückzuführen sein. Die Anforderung wird aufgrund einer expliziten Synchronisationspunktanforderung von der Anwendung ausgegeben.	Informationen zur Arbeitseinheit.
CSQCEXER	Auflösung ausführen	Vor der Ausgabe von EXECUTE_RESOLVE an IBM MQ for z/OS.	Die Arbeitseinheitsinformationen der Arbeitseinheit, die EXECUTE_RESOLVE ausgibt. Dies ist die letzte unbestätigte Arbeitseinheit im Resynchronisationsprozess.
CSQCGETW	Wartestatus abrufen	Vor der Ausgabe des CICS-Wartestatus.	Adresse des Ereignissteuerblocks, auf den gewartet werden soll.
CSQCGMGD	Nachrichtendaten abrufen	Nach erfolgreicher Rückgabe aus MQGET.	Bis zu 40 Byte der Nachrichtendaten.
CSQCGMGH	Nachrichtenhandle abrufen	Vor der Ausgabe von MQGET an IBM MQ for z/OS.	Objektkennung.
CSQCGMGI	Nachrichten-ID abrufen	Nach erfolgreicher Rückgabe aus MQGET.	Nachrichten-ID und Korrelations-ID der Nachricht
CSQCINDL	Unbestätigte Liste	Nach der zweiten erfolgreichen Rückgabe von INQUIRE_INDOUBT.	Liste der unbestätigten Arbeitseinheiten.
CSQCINDO	Nur Verwendung durch IBM		
CSQCINDS	Größe der unbestätigten Liste	Nach erfolgreicher Rückgabe aus der ersten INQUIRE_INDOUBT und die unbestätigte Liste nicht leer ist.	Länge der Liste. Durch das Dividieren mit 64 wird die Anzahl der unbestätigten Arbeitseinheiten ermittelt.
CSQCINQH	INQ-Kennung	Vor der Ausgabe von MQINQ an IBM MQ for z/OS.	Objektkennung.
CSQCLOSH	CLOSE-Kennung	Vor der Ausgabe von MQCLOSE an IBM MQ for z/OS.	Objektkennung.
CSQCLOST	Disposition verloren	Während des Resynchronisationsprozesses informiert CICS den Adapter, dass er erneut gestartet wurde, damit keine Dispositionsinformationen zu den Arbeitseinheiten verfügbar sind, die resynchronisiert werden.	ID der Arbeitseinheit, die CICS für die resynchronisierte Arbeitseinheit bekannt ist.

Tabelle 142. Traceeinträge des CICS-Adapters (Forts.)

Name	Beschreibung	Tracesequenz	Tracedaten
CSQCNIND	Disposition nicht unbestätigt	Während des Resynchronisationsprozesses informiert CICS den Adapter, dass die Arbeitseinheit, die resynchronisiert wird, nicht unbestätigt sein sollte (d. h. sie ist vielleicht noch aktiv).	ID der Arbeitseinheit, die CICS für die resynchronisierte Arbeitseinheit bekannt ist.
CSQCNORT	Normale Beendigung	Vor der Ausgabe von END_THREAD NORMAL an IBM MQ for z/OS. Der Grund ist das Ende der Task, wodurch die Anwendung möglicherweise eine Festschreibung des impliziten Synchronisationspunkts ausführt. In diesem Fall wird in den Aufruf 'END_THREAD' die Anfrage 'COMMIT' einbezogen.	Informationen zur Arbeitseinheit.
CSQCOPNH	Kennung öffnen	Nach erfolgreicher Rückgabe aus MQOPEN.	Objektkennung.
CSQCOPNO	Objekt öffnen	Vor der Ausgabe von MQOPEN an IBM MQ for z/OS.	Objektname
CSQCPMGD	Nachrichtendaten einreihen	Vor der Ausgabe von MQPUT an IBM MQ for z/OS.	Bis zu 40 Byte der Nachrichtendaten.
CSQCPMGH	Nachrichtenhandle einreihen	Vor der Ausgabe von MQPUT an IBM MQ for z/OS.	Objektkennung.
CSQCPMGI	Nachrichten-ID einreihen	Nach erfolgreichem MQPUT-Aufruf aus IBM MQ for z/OS.	Nachrichten-ID und Korrelations-ID der Nachricht
CSQCPREP	Synchronisationspunktvorbereitung	Vor der Ausgabe von PREPARE an IBM MQ for z/OS in der ersten Phase der zweiphasigen Festschreibung. Dieser Aufruf kann auch von der Komponente zur verteilten Steuerung von Warteschlangen als ein API-Aufruf ausgegeben werden.	Informationen zur Arbeitseinheit.
CSQCP1MD	Einzelne Nachrichtendaten einreihen	Vor der Ausgabe von MQPUT1 an IBM MQ for z/OS.	Bis zu 40 Byte der Nachrichtendaten.
CSQCP1MI	Einzelne Nachrichten-ID einreihen	Nach erfolgreicher Rückgabe aus MQPUT1.	Nachrichten-ID und Korrelations-ID der Nachricht
CSQCP1ON	Einzelnen Objektnamen einreihen	Vor der Ausgabe von MQPUT1 an IBM MQ for z/OS.	Objektname
CSQCRBAK	Backout aufgelöst	Vor der Ausgabe von RESOLVE_ROLLBACK an IBM MQ for z/OS.	Informationen zur Arbeitseinheit.
CSQCRCMT	Festschreibung aufgelöst	Vor der Ausgabe von RESOLVE_COMMIT an IBM MQ for z/OS.	Informationen zur Arbeitseinheit.

Tabelle 142. Traceeinträge des CICS-Adapters (Forts.)

Name	Beschreibung	Tracesequenz	Tracedaten
CSQCRMIR	RMI-Antwort	Vor der Rückgabe an die CICS-Ressourcenmanagerschnittstelle (RMI) von einem bestimmten Aufruf.	Wert einer strukturierten RMI-Antwort. Seine Bedeutung hängt vom Typ des Aufrufs ab. Diese Werte sind in der Veröffentlichung *CICS Transaction Server for z/OS Customization Guide* dokumentiert. Um den Typ des Aufrufs zu bestimmen, prüfen Sie die Traceeinträge, die zuvor von der CICS-RMI-Komponente erstellt wurden.
CSQCRSYN	Resynchronisation	Bevor der Resynchronisationsprozess für die Task startet.	ID der Arbeitseinheit, die CICS für die resynchronisierte Arbeitseinheit bekannt ist.
CSQCSETH	Kennung festlegen	Vor der Ausgabe von MQSET an IBM MQ for z/OS.	Objektkennung.
CSQCTASE	Nur Verwendung durch IBM		
CSQCTEST	Tracetest	Verwendung im Aufruf EXEC CICS ENTER TRACE, um die vom Benutzer angegebene Tracenummer oder den Tracestatus der Verbindung zu prüfen.	Keine Daten.
CSQCDCFF	Nur Verwendung durch IBM		

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 for IBM 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 889) 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 „Transaktionsmanagement und Recoverable Resource Manager Services (RRS)” auf Seite 894.

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.

Systeme mit UNIX, Linux, and Windows

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 889). 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. ULW Bei IBM MQ on UNIX, 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 UNIX, Linux, and Windows finden Sie im Abschnitt 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, UNIX, 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 Systemen mit IBM MQ for IBM i IBM i, Windows, UNIX and Linux wird der Rücksetzungszähler auch nach dem Neustart des Warteschlangenmanagers fortgesetzt.

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 829.

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 16.

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 1010.

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 16.

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 1108.

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 1109) 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 28.

In der folgenden Tabelle werden alle wichtigen MQI-Konzepte und deren Beziehung zu Multicast aufgeführt:

Tabelle 143. MQI-Konzepte und deren Beziehung zu Multicast

MQI-Konzept	**Aktion beim Versuch, Multicast zu verwenden**	**Ursachencode**
Nachricht mit Nulllänge einreihen	Zurückgewiesen	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
Gruppe	Zurückgewiesen	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentation	Zurückgewiesen	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
Verteilerlisten	Zurückgewiesen	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)	Zurückgewiesen	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
Nachricht durchsuchen	Zurückgewiesen	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE

Tabelle 143. MQI-Konzepte und deren Beziehung zu Multicast (Forts.)		
MQI-Konzept	**Aktion beim Versuch, Multicast zu verwenden**	**Ursachencode**
Nachrichten sperren	Zurückgewiesen	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Mit Markierung durchsuchen	Zurückgewiesen	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Kontext übergeben	Zurückgewiesen	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	Zurückgewiesen	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-Konzepte 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 1033 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 in der Datei ClientExitPath-Zeilengruppe der 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 144. 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 144. 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 144. 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 1116
- „Parameter mit einem nicht definierten Datentyp” auf Seite 1117
- „Datentypen” auf Seite 1117
- „Binärzeichenfolgen bearbeiten” auf Seite 1117
- „Zeichenfolgen bearbeiten” auf Seite 1117
- „Anfangswerte für Strukturen” auf Seite 1118
- „Anfangswerte für dynamische Strukturen” auf Seite 1118
- „Verwendung in der Programmiersprache C++” auf Seite 1119

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. Das Symbol ¬ steht beispielsweise für ein Leerzeichen:

  ```
  #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. Beispiel:

  ```
  #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: 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 im Abschnitt .NET-Anwendungen entwickeln.

V 9.0.0 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 1084
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 961
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.

Codierung in Assemblersprache System/390 (Schnittstelle für Nachrichtenwarteschlangen)

Beachten Sie bei der Codierung von IBM MQ for z/OS-Programmen in Assemblersprache die Informationen in den folgenden Abschnitten.

- „Namen” auf Seite 1121
- „MQI-Aufrufe verwenden” auf Seite 1121
- „Konstanten deklarieren” auf Seite 1121
- „Name einer Struktur angeben” auf Seite 1122
- „Form einer Struktur angeben” auf Seite 1122
- „Liste steuern” auf Seite 1122
- „Anfangswerte für Felder angeben” auf Seite 1122
- „Simultan verwendbare Programme schreiben” auf Seite 1123

- „CEDF verwenden” auf Seite 1123

Namen

Die Namen von Parametern in den Beschreibungen von Aufrufen und die Namen von Feldern in den Beschreibungen von Strukturen werden in gemischter Groß- und Kleinschreibung angezeigt. In den Assemblersprache-Makros von IBM MQ sind alle Namen großgeschrieben.

MQI-Aufrufe verwenden

Da die MQI eine Aufrufschnittstelle ist, müssen Programme in Assemblersprache die Verbindungskonvention des Betriebssystems beachten.

Insbesondere vor Ausgabe eines MQI-Aufrufs müssen Programme in Assemblersprache das Register R13 auf einen Sicherungsbereich von mindestens 18 vollständigen Wörtern verweisen. Dieser Sicherungsbereich dient als Speicher für das aufgerufene Programm. Er speichert die Register des Aufrufenden, bevor deren Inhalte zerstört werden, und stellt den Inhalt der Register des Aufrufenden bei Rückgabe wieder her.

Anmerkung: Dies ist wichtig für CICS-Programme in Assemblersprache, die mit dem DFHEIENT-Makro ihren dynamischen Speicher einrichten, aber das standardmäßige DATAREG von R13 mit anderen Registern überschreiben. Wenn die CICS Resource Manager-Schnittstelle die Kontrolle vom Stub erhält, speichert sie den aktuellen Inhalt der Register unter der Adresse, auf die R13 verweist. Wird kein Sicherungsbereich zu diesem Zweck reserviert, führt dies zu unvorhersehbaren Ergebnissen und möglicherweise zu einem Abbruch in CICS.

Konstanten deklarieren

Die meisten Konstanten sind im Makro CMQA als Gleichsetzung deklariert.

Folgende Konstanten können jedoch nicht als Gleichsetzungen deklariert werden und werden nicht einbezogen, wenn Sie das Makro unter Verwendung der Standardoptionen aufrufen:

- 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

Um sie einzuschließen, fügen Sie das Schlüsselwort EQUONLY=NO hinzu, wenn Sie das Makro aufrufen.

CMQA ist vor Mehrfachdeklarationen geschützt und kann daher viele Male einbezogen werden. Das Schlüsselwort EQUONLY ist allerdings nur beim ersten Einbeziehen des Makros gültig.

Name einer Struktur angeben

Damit mehrere Instanzen einer Struktur deklariert werden können, setzt das Makro, das die Struktur generiert, dem Namen jedes Feld mit einer benutzerdefinierbaren Zeichenfolge und einem Unterstrich (_) ein Präfix voran.

Geben Sie beim Aufrufen des Makros die Zeichenfolge an. Andernfalls verwendet das Makro den Namen der Struktur, um das Präfix zu erstellen:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

Die Strukturdeklarationen in Aufrufbeschreibungen zeigen das Standardpräfix an.

Form einer Struktur angeben

Die Makros können Strukturdeklarationen in einer von zwei Formen generieren, gesteuert durch den Parameter DSECT:

DSECT=YES
Mit einer DSECT-Anweisung in Assemblersprache wird ein neuer Datenabschnitt gestartet. Die Strukturdefinition folgt unmittelbar auf die DSECT-Anweisung. Da kein Speicher zugeordnet wird, ist auch keine Initialisierung möglich. Der Kennsatz im Makroaufruf wird als Name des Datenabschnitts verwendet; wird kein Kennsatz angegeben, wird der Name der Struktur verwendet.

DSECT=NO
Mit DC-Anweisungen in Assemblersprache wird de Struktur an der aktuellen Position in der Routine definiert. Die Felder werden mit Werten initialisiert, die Sie durch das Codieren der dazugehörigen Parameter bei Makroaufruf angeben können. Felder, für die im Makroaufruf keine Werte angegeben werden, werden mit den jeweiligen Standardwerten initialisiert.

DSECT=NO wird angenommen, wenn der Parameter DSECT nicht angegeben ist.

Liste steuern

Mit dem Parameter LISTEN können Sie die Darstellung der Strukturdeklaration in der Assemblersprachenliste steuern:

LIST=YES
Die Strukturdeklaration wird in der Assemblersprachenliste angezeigt.

LIST=NO
Die Strukturdeklaration wird nicht in der Assemblersprachenliste angezeigt. Dies wird angenommen, wenn der Parameter LISTEN nicht angegeben wird.

Anfangswerte für Felder angeben

Sie können den Wert für die Initialisierung eines Felds in einer Struktur angeben, indem Sie den Namen des betreffenden Felds (ohne das Präfix) als Parameter bei Makroaufruf codieren, gemeinsam mit dem erforderlichen Wert.

Beispiel: Deklarieren Sie die Struktur eines Nachrichtendeskriptors, wobei das Feld *MsgType* mit MQMT_REQUEST und das Feld *ReplyToQ* mit der Zeichenfolge MY_REPLY_TO_QUEUE initialisiert wird, indem Sie folgenden Code verwenden:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

Wenn Sie eine benannte Konstante (oder Gleichsetzung) als einen Wert beim Makroaufruf angeben, verwenden Sie das CMQA-Makro, um die benannte Konstante zu definieren. Werte, bei denen es sich um Zeichenfolgen handelt, dürfen nicht in einfachen Anführungszeichen (' ') stehen.

Simultan verwendbare Programme schreiben

IBM MQ verwendet seine Strukturen für Ein- und Ausgaben. Beachten Sie Folgendes, wenn Ihr Programm simultan verwendbar bleiben soll:

1. Definieren Sie Arbeitsspeicherversionen der Strukturen als DSECTs oder definieren Sie die Strukturen innerhalb eines bereits definierten DSECT. Kopieren Sie dann den DSECT unter Verwendung folgender Makros oder Befehle in den erhaltenen Speicher:
   - Für Stapel- und TSO-Programme die z/OS-Assembler-Makros STORAGE oder GETMAIN
   - Für CICS der Arbeitsspeicherbefehl DSECT (DFHEISTG) oder der Befehl EXEC CICS GETMAIN

   Kopieren Sie zur ordnungsgemäßen Initialisierung dieser Arbeitsspeicherstrukturen eine konstante Version der entsprechenden Struktur in die Arbeitsspeicherversion.

   Anmerkung: Die MQMD- und MQXQH-Strukturen sind jeweils länger als 256 Bytes. Verwenden Sie zum Kopieren dieser Strukturen in den Speicher die Assembleranweisung MVCL.
2. Reservieren Sie Speicherplatz, indem Sie die LIST-Form (`MF=L`) des CALL-Makros verwenden. Wenn Sie mit dem CALL-Makro einen MQI-Aufruf vornehmen, verwenden Sie die EXECUTE-Form (`MF=E`) des Makros und den zuvor reservierten Speicherplatz (siehe Beispiel unter „CEDF verwenden” auf Seite 1123). Weitere Beispiele hierzu finden Sie in den Beispielprogrammen in Assemblersprache von IBM MQ.

Verwenden Sie mit der Assemblersprachenoption RENT, um zu ermitteln, ob Ihr Programm simultan verwendbar ist.

Informationen zum Schreiben von erneut eingabefähigen Programmen finden Sie in *z/OS MVS Application Development Guide: Assembler Language Programs*.

CEDF verwenden

Wenn Sie Ihr Programm mit der CICS-Transaktion CEDF (CICS Execution Diagnostic Facility) debuggen möchten, fügen Sie jeder `CALL`-Anweisung das Schlüsselwort `,VL` hinzu, beispielsweise wie folgt:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

Bei dem vorherigen Beispiel handelt es sich um simultan verwendbaren Assemblersprachencode, wobei PARMAREA für einen von Ihnen angegebenen Bereich im Arbeitsspeicher steht.

MQI-Aufrufe verwenden

Da die MQI eine Aufrufschnittstelle ist, müssen Programme in Assemblersprache die Verbindungskonvention des Betriebssystems beachten. Insbesondere vor Ausgabe eines MQI-Aufrufs müssen Programme in Assemblersprache das Register R13 auf einen Sicherungsbereich von mindestens 18 vollständigen Wörtern verweisen. Dieser Sicherungsbereich dient als Speicher für das aufgerufene Programm. Er speichert die Register des Aufrufenden, bevor deren Inhalte zerstört werden, und stellt den Inhalt der Register des Aufrufenden bei Rückgabe wieder her.

Anmerkung: Dies ist wichtig für CICS-Programme in Assemblersprache, die mit dem DFHEIENT-Makro ihren dynamischen Speicher einrichten, aber das standardmäßige DATAREG von R13 mit anderen Registern überschreiben. Wenn die CICS Resource Manager-Schnittstelle die Kontrolle vom Stub erhält, speichert sie den aktuellen Inhalt der Register unter der Adresse, auf die R13 verweist. Wird kein ordnungsgemäßer Sicherungsbereich zu diesem Zweck reserviert, führt dies zu unvorhersehbaren Ergebnissen und möglicherweise zu einem Abbruch 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).

Codierung in PL/I (nur unter z/OS)

Informationen zur Codierung in PL/I für IBM MQ

Strukturen

Strukturen werden mit dem Attribut BASED deklariert und belegen somit erst dann Speicherkapazitäten, wenn das Programm mindestens eine Instanz einer Struktur deklariert.

Eine Strukturinstanz kann mit dem Attribut `like` deklariert werden, zum Beispiel:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

Die Strukturfelder werden mit dem Attribut INITIAL deklariert; wenn die Instanz einer Struktur mit dem Attribut `like` deklariert wird, übernimmt diese Instanz die für diese Struktur definierten Anfangswerte. Sie müssen nur die Felder einstellen, in denen der Anfangswert nicht dem erforderlichen Wert entspricht.

In PL/I muss keine Groß-/Kleinschreibung beachtet werden. Sie können also die Namen von Aufrufen, Strukturfeldern und Konstanten wahlweise nur in Kleinbuchstaben, nur in Großbuchstaben oder gemischt codieren.

Benannte Konstanten

Die benannten Konstanten werden als Makrovariablen deklariert; benannte Konstanten, auf die das Programm nicht verweist, belegen also keinen Speicher in der kompilierten Prozedur.

Allerdings muss beim Kompilieren des Programms die Compileroption angegeben werden, die anweist, dass die Quelle vom Makrovorprozessor verarbeitet wird.

Alle Makrovariablen sind Zeichenvariablen, sogar diejenigen, die numerische Werte darstellen. Obwohl dies nicht intuitiv erscheinen könnte, führt es zu keinem Datentypkonflikt, nachdem die Makrovariablen vom Makroumwandler ersetzt wurden, zum Beispiel:

```
%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:

- Beispielprogramme für verteilte Systeme und IBM i.
- Beispielprogramme für z/OS.

Prozedur

- Weitere Informationen zu den Beispielprogrammen finden Sie unter den folgenden Links:
  - „Beispielprogramme plattformübergreifend verwenden” auf Seite 1125

– z/OS „Beispielprogramme für z/OS verwenden” auf Seite 1236

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

„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 48
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 743
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 953
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„Publish/Subscribe-Anwendungen schreiben” auf Seite 840
Beginnen Sie nun, Ihre eigenen IBM MQ-Publish/Subscribe-Anwendungen zu schreiben.

„Prozedurale Anwendung erstellen” auf Seite 1051
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 1103
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

„Web-Services mit IBM MQ entwickeln” auf Seite 1368
Sie können IBM MQ-Anwendungen für Web-Services mithilfe des IBM MQ-Transports für SOAP entwickeln.

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 1135.

Prozedur

- Weitere Informationen zu den Beispielprogrammen finden Sie unter den folgenden Links:
  - „In den Beispielprogrammen für Multiplatforms veranschaulichte Funktionen” auf Seite 1127
  - „Publish/Subscribe-Beispielprogramme” auf Seite 1166
  - „Die Put-Beispielprogramme” auf Seite 1171
  - „Das Distribution List-Beispielprogramm” auf Seite 1157
  - „Die Browse-Beispielprogramme” auf Seite 1145
  - „Das Beispielprogramm 'Browser'” auf Seite 1146
  - „Die Get-Beispielprogramme” auf Seite 1159
  - „Die Reference Message-Beispielprogramme” auf Seite 1173
  - „Die Request-Beispielprogramme” auf Seite 1181
  - „Die Inquire-Beispielprogramme” auf Seite 1164
  - „Beispielprogramm zum Abfragen der Eigenschaften eines Nachrichtenhandles” auf Seite 1165
  - „Die Set--Beispielprogramme” auf Seite 1187
  - „Die Echo-Beispielprogramme” auf Seite 1158
  - „Das Data-Conversion-Beispielprogramm” auf Seite 1149
  - „Auslösebeispielprogramme” auf Seite 1191
  - „Das Asynchronous Put-Beispielprogramm” auf Seite 1144
  - „Beispiele zur Datenbankkoordination” auf Seite 1149
  - „Das CICS Transaction-Beispielprogramm” auf Seite 1147
  - „TUXEDO-Beispiele unter UNIX und Windows verwenden” auf Seite 1193
  - „Beispielprogramm für eine Steuerroutine der Warteschlange für nicht zustellbare Nachrichten” auf Seite 1156
  - „Das Connect-Beispielprogramm” auf Seite 1147
  - „API Exit-Beispielprogramm” auf Seite 1142
  - „SSPI-Sicherheitsexit unter Windows verwenden” auf Seite 1209
  - „Beispiele unter Verwendung ferner Warteschlangen ausführen” auf Seite 1210
  - „Das Beispielprogramm 'Clusterwarteschlangenüberwachung' (AMQSCLM)” auf Seite 1210
  - „Das Beispielprogramm für die Suche nach Verbindungsendpunkten (CEPL)” auf Seite 1220

Zugehörige Konzepte

„Beispielprogramme in C++” auf Seite 530
In vier bereitgestellten Beispielprogrammen wird das Abrufen und Einreihen von Nachrichten veranschaulicht.

Zugehörige Tasks

„Beispielprogramme für z/OS verwenden” auf Seite 1236
Die prozeduralen Beispielanwendungen, die mit IBM MQ for z/OS bereitgestellt werden, veranschaulichen typische Verwendungsmöglichkeiten der MQI (Message Queue Interface).

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 „Beispielprogramme für z/OS verwenden” auf Seite 1236.

Linux UNIX *Beispiele für Systeme mit UNIX and Linux*

In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit UNIX and Linux veranschaulicht werden.

Informationen dazu, wo die Beispielprogramme für IBM MQ auf UNIX and Linux -Systemen gespeichert sind, finden Sie unter „Beispielprogramme unter UNIX and Linux vorbereiten und ausführen” auf Seite 1139 .

Tabelle 145 auf Seite 1127 Die Tabelle führt auf, welche C- und COBOL-Quellendateien bereitgestellt werden und ob eine ausführbare Server- oder Clientdatei enthalten ist.

Tabelle 145. Beispielprogramme für IBM MQ auf UNIX and Linux zur Veranschaulichung der Verwendung der MQI (C und COBOL)

Verfahren	**C (Quelle) („1” auf Seite 1129)**	**COBOL (Quelle) („2” auf Seite 1129)**	**Server (ausführbar in C)**	**Client (ausführbar in C) („3” auf Seite 1129)**
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 („4” auf Seite 1129)	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 145. Beispielprogramme für IBM MQ auf UNIX and Linux zur Veranschaulichung der Verwendung der MQI (C und COBOL) (Forts.)

Verfahren	**C (Quelle) („1” auf Seite 1129)**	**COBOL (Quelle) („2” auf Seite 1129)**	**Server (ausführbar in C)**	**Client (ausführbar in C) („3” auf Seite 1129)**
Abrufen von Nachrichten (mit Datenkonvertierung)	amqsecha	kein Beispiel	amqsech	kein Beispiel
Referenznachrichten in eine Warteschlange einreihen („4” auf Seite 1129)	amqsprma	kein Beispiel	amqsprm	amqsprmc
Referenznachrichten aus einer Warteschlange abrufen („4” auf Seite 1129)	amqsgrma	kein Beispiel	amqsgrm	amqsgrmc
Referenznachricht-Kanalexit („4” auf Seite 1129)	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	(„5” auf Seite 1129)	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
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 („6” auf Seite 1129)	amqscic0.ccs	kein Beispiel	amqscic0	kein Beispiel
Encina-Transaktion („4” auf Seite 1129)	amqsxae0	kein Beispiel	amqsxae0	kein Beispiel

Tabelle 145. Beispielprogramme für IBM MQ auf UNIX and Linux zur Veranschaulichung der Verwendung der MQI (C und COBOL) (Forts.)

Verfahren	**C (Quelle) („1” auf Seite 1129)**	**COBOL (Quelle) („2” auf Seite 1129)**	**Server (ausführbar in C)**	**Client (ausführbar in C) („3” auf Seite 1129)**
TUXEDO-Transaktion zum Einreihen von Nachrichten („7” auf Seite 1130)	amqstxpx	kein Beispiel	kein Beispiel	kein Beispiel
TUXEDO-Transaktion zum Abrufen von Nachrichten („7” auf Seite 1130)	amqstxgx	kein Beispiel	kein Beispiel	kein Beispiel
Server für TUXEDO („7” auf Seite 1130)	amqstxsx	kein Beispiel	kein Beispiel	kein Beispiel
Steuerroutine der Warteschlange für nicht zustellbare Nachrichten	Verzeichnis `./tools/c/Samples/dlq` („8” auf Seite 1130)	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
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. Die ausführbaren Versionen der IBM MQ MQI client-Beispielprogramme stehen unter IBM MQ for HP-UX nicht zur Verfügung.
4. Unterstützung nur unter IBM MQ for AIX, IBM MQ for HP-UX und IBM MQ for Solaris.
5. In IBM MQ for AIX, IBM MQ for HP-UX und IBM MQ for Solaris heißt dieses Programm `amqsvfc0.c`.
6. CICS wird nur von IBM MQ for AIX und IBM MQ for HP-UX unterstützt.

7. TUXEDO wird nicht von IBM MQ for Linux on System p unterstützt.
8. 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 UNIX 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 146 auf Seite 1130 wird aufgelistet, welche C- bzw. COBOL-Quellendateien bereitgestellt werden und ob eine ausführbare Server- oder Clientdatei enthalten ist.

Tabelle 146. 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
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

Tabelle 146. 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)
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
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 amqsx-ab0.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 1132)	kein Beispiel	amqsdlq	kein Beispiel
In einem IBM MQ MQI client eine Nachricht einreihen	kein Beispiel	kein Beispiel	kein Beispiel	amqsputc

Tabelle 146. 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)
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	amqsssslc	kein Beispiel	Nicht zutreffend	amqsssslc

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 147 auf Seite 1132 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 1084.

Tabelle 147. 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

Tabelle 147. IBM MQ for Windows-Beispielprogramme zur Veranschaulichung der Verwendung der MQI (Visual Basic) (Forts.)	
Verfahren	**Projektdateiname**
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 148 auf Seite 1133 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 148. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter IBM i				
Verfahren	**C (Quelle) („1" auf Seite 1135)**	**COBOL (Quelle) („2" auf Seite 1135)**	**RPG (Quelle) („3" auf Seite 1135)**	**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
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

Tabelle 148. Beispielprogramme zur Veranschaulichung der Verwendung der MQI (C und COBOL) unter IBM i (Forts.)

Verfahren	C (Quelle) („1” auf Seite 1135)	COBOL (Quelle) („2” auf Seite 1135)	RPG (Quelle) („3” auf Seite 1135)	Client (ausführbar in C) (4)
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
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

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_THREADED auf 'Y' und führen Sie die Anwendung aus 'qsh' aus.

   Weitere Informationen hierzu finden Sie im Abschnitt IBM MQ mit Java und JMS konfigurieren.

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.

Windows IBM i UNIX

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:

- „Beispielprogramme unter IBM i vorbereiten und ausführen” auf Seite 1138
- „Beispielprogramme unter UNIX and Linux vorbereiten und ausführen” auf Seite 1139
- „Beispielprogramme unter Windows vorbereiten und ausführen” auf Seite 1140

Windows IBM i UNIX *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 1136 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) Windows UNIX
   Für UNIX- und Windows-Systeme gilt:

   ```
   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')
   ```

   Kanalname ist 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 ')
   ```

   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 1136

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')
```

   Warteschlangenname ist der Name Ihrer Warteschlange.

6. Erteilen Sie die erforderliche Berechtigung für die Verbindung mit dem Warteschlangenmanager und die Abfrage desselben:

   a) Windows IBM i UNIX

   Geben Sie für IBM i IBM i-, UNIX- und Windows-Systeme folgende MQSC-Befehle aus:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
AUTHADD(CONNECT, INQ)
```

   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1" auf Seite 1136

7. Handelt es sich bei Ihrer Anwendung um eine Punkt-zu-Punkt-Anwendung, d. h., sie nutzt Warteschlangen, so 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:

   a) Windows IBM i UNIX

   Geben Sie für IBM i IBM i-, UNIX- und Windows-Systeme folgende MQSC-Befehle aus:

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

   Warteschlangenname ist der Name Ihrer Warteschlange.
   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1" auf Seite 1136

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:

   a) Windows IBM i UNIX

   Geben Sie für IBM i IBM i-, UNIX- und Windows-Systeme folgende MQSC-Befehle aus:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1" auf Seite 1136
   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 Informationen

SET CHLAUTH

CHANNEL DEFINE CHANNEL

QLOCAL DEFINIER

SET AUTHREC

IBM i IBM MQ-Berechtigungen unter IBM i

Zugriff auf ein IBM MQ-Objekt unter UNIX- oder Linux-Systemen und unter Windows erteilen

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.

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 1135.
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 1133
In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit IBM i veranschaulicht werden.

UNIX *Beispielprogramme unter UNIX and Linux vorbereiten und ausführen*

Bevor Sie die Beispielprogramme unter UNIX 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 UNIX and Linux-Systemen finden Sie in den unter Tabelle 149 auf Seite 1139 aufgeführten Verzeichnissen, wenn die Standardeinstellungen installiert wurden.

Tabelle 149. Verzeichnisse der Beispielprogramme für IBM MQ auf UNIX 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.

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 im Abschnitt 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 1135.

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 UNIX 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 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 Systeme mit UNIX and Linux” auf Seite 1127
In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit UNIX and 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 150 auf Seite 1140 aufgeführten Verzeichnissen, sofern zur Installationszeit die Standardeinstellungen verwendet wurden. Als Installationslaufwerk ist standardmäßig <c:> eingestellt.

Tabelle 150. Verzeichnisse der Beispielprogramme für IBM MQ for Windows

Inhalt	Directory
C-Quellcode	*MQ_INSTALLATION_PATH*\Tools\C\Samples
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\Samples

Tabelle 150. Verzeichnisse der Beispielprogramme für IBM MQ for Windows (Forts.)	
Inhalt	**Directory**
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\Samples
Visual Basic-Quellcode	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET-Beispielprogramme	*MQ_INSTALLATION_PATH*\Tools\dotnet\Samples

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.

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 im Abschnitt 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 1135.
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.

   Anmerkung:
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 1130
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 1132
In diesem Abschnitt werden die Verfahren erläutert, die von den Beispielprogrammen für IBM MQ auf Systemen mit Windows veranschaulicht werden.

API Exit-Beispielprogramm

Das API Exit-Beispielprogramm generiert einen MQI-Trace zu einer benutzerdefinierten Datei mit einem in der Umgebungsvariable MQAPI_TRACE_LOGFILE definierten Präfix.

Weitere Informationen zu API-Exits finden Sie im Abschnitt „API-Exits schreiben und kompilieren" auf Seite 1000.

Source
amqsaxe0.c

Binär
amqsaxe

Konfiguration für den Beispielexit

1. Fügen Sie der Datei 'qm.ini' Folgendes 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 fest:

   ```
   MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
   ```

3. Führen Sie Ihre Anwendung aus.

   Die Ausgabedateien werden im Verzeichnis '/tmp' mit Namen ähnlich den folgenden erstellt: `MqiTrace.pid.tid.log`

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 Windows-, UNIX and Linux-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 1127.

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 1127.

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 UNIX, Linux, and Windows
Dieser Abschnitt enthält Informationen zu den Beispielprogrammen für die die Durchsuchen-Funktion unter UNIX, 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. Beispiel:

```
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.

ULW *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 1127.

Das Programm verwendet folgende positionsgebundene Parameter:

1. Name der Quellenwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (erforderlich)
3. Optionaler Parameter für Eigenschaften (optional)

Diese Programme verwenden auch die Umgebungsvariable **MQSAMP_USER_ID**, die auf die für die Verbindungsauthentifizierung zu verwendende Benutzer-ID gesetzt werden sollte. Wenn diese festgelegt ist, fragt das Programm nach dem zugehörigen Kennwort für diese Benutzer-ID.

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 151. Zulässige Werte für den Eigenschaftenparameter

Wert	Verhalten
0	Standardverhalten, wie in IBM WebSphere MQ 6. 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. Beispiel:
	`****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 WebSphere MQ 6-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 .

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 1051.

Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE im Standardwarteschlangenmanager und reiht sie in die lokale Warteschlange ein, deren Name im Übertragungsheader der Nachricht enthalten ist. Mögliche Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICS.DLQ gesendet.

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 1127.

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 140 auf Seite 1150 zeigt, wie Sie mit den bereitgestellten Beispielen Datenbanken aktualisieren:

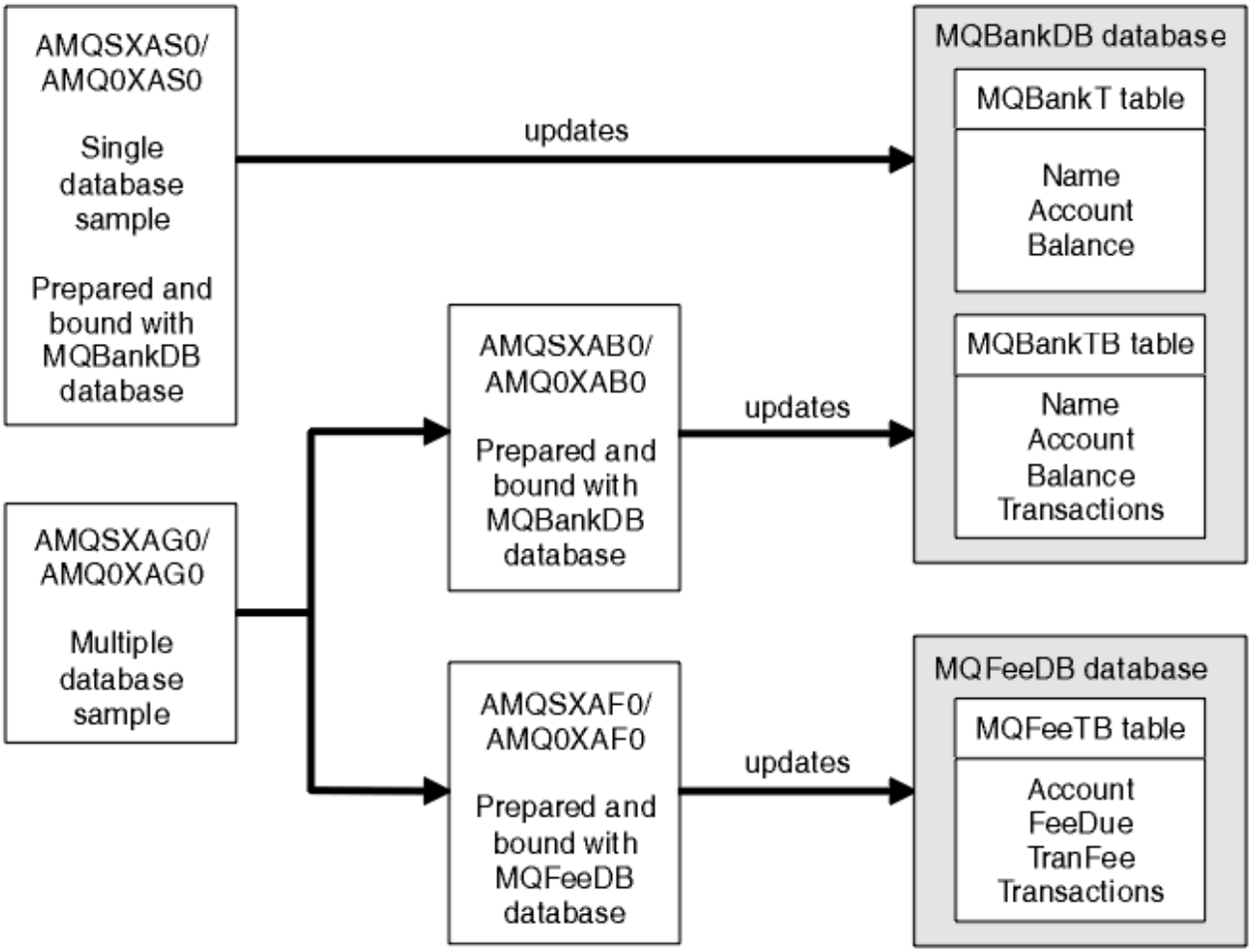


Abbildung 140. Beispiele zur Datenbankkoordination

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 Instan-

zen 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.

Kompilieren Sie die .SQC-Dateien (in C) und die .SQB-Dateien (in COBOL) vor und binden Sie sie an die entsprechende Datenbank, um die .C- bzw. .CBL-Dateien 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.

- Unter AIX lautet der Verzeichnispfad:

```
/usr/lpp/db2_05_00
```

- Unter HP-UX und Solaris lautet der Verzeichnispfad:

```
/opt/IBMdb2/V5.0
```

- Auf Windows-Systemen richtet sich der Verzeichnispfad nach dem für die Installation des Produkts ausgewählten Pfad. Wenn Sie die Standardeinstellungen wählen, lautet der Pfad:

```
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:

- Die Datei `amqsxag0.c` aus Ihrer IBM MQ-Installation

  Anmerkung: Diese Datei finden Sie in folgenden Verzeichnissen:

  – Auf UNIX and Linux-Systemen:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Auf Windows-Systemen:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Die `.c` -Dateien, die Sie durch Vorkompilieren der `.sqc` -Quellendateien `amqsxas0.sqc, amqsxaf0.sqc`und `amqsxab0.sqc` erhalten haben
- Die 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 jede `.c`-Datei. Verwenden Sie dazu folgenden Compilerbefehl für Ihre entsprechende Plattform:

- AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- HP-UX

```
cc -Aa +z -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Solaris

```
cc -Aa -KPIC -mt -I MQ_INSTALLATION_PATH
```

```
/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Systeme mit Windows

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

Erstellen Sie die ausführbaren amqsxag0-Dateien. Verwenden Sie dazu folgenden Verknüpfungsbefehl für Ihre entsprechende Plattform:

- 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
```

- HP-UX Revision 11i

```
ld -E -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread -lcl
/lib/crt0.o util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Solaris

```
cc -mt -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm -lthread -lsocket -lc -lnsl -ldl util.o
amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Systeme mit Windows

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

Erstellen Sie die ausführbaren amqsxas0-Dateien. Verwenden Sie dazu die folgenden Kompilierungs- und Verknüpfungsbefehle für die entsprechende Plattform:

- AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- HP-UX Revision 11i

```
ld -E -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread
-lcl /lib/crt0.o util.o amqsxas0.o -o amqsxas0
```

- Solaris

```
cc -mt -L DB2TOP/lib -ldb2-L MQ_INSTALLATION_PATH/lib
-lqm -lthread -lsocket -lc -lnsl -ldl util.o
amqsxas0.o -o amqsxas0
```

- Systeme mit Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Weitere Informationen

Wenn Sie unter AIX oder HP-UX 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 1155
- „COBOL-Beispiele” auf Seite 1156

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

Ein Beispielprogramm für eine Steuerroutine der Warteschlange für nicht zustellbare Nachrichten wird bereitgestellt. Der Name der ausführbaren Version lautet 'amqsdlq'. Wenn Sie statt RUNMQDLQ eine

andere Steuerroutine der Warteschlange für nicht zustellbare Nachrichten verwenden möchten, steht die Quelle des Beispielprogramms für Sie als Grundlage zur Verfügung.

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
Auf YES oder 'yes' setzen, um die Tracefunktion zu aktivieren

ODQ_MSG
Auf den Namen der Datei setzen, die Fehler- und Informationsnachrichten enthält. Die bereitgestellte Datei hat die Bezeichnung 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' ändern und an Ihre eigenen Anforderungen anpassen. Das Beispielprogramm reiht Nachrichten **nicht** in die IBM MQ-Fehlerprotokolldatei ein, sondern in 'stdout'.

Im Handbuch Verwalten bzw. *System Management Guide* für Ihre Plattform finden Sie Informationen zur Funktionsweise und Ausführung der Steuerroutine der Warteschlange für nicht zustellbare Nachrichten.

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. Beispiel:

```
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 1127.

Die Programme wurden als Auslöserprogramme konzipiert.

Auf IBM i-, UNIX, 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 i funktioniert, stellen Sie sicher, dass das gewünschte Echo-Beispielprogramm durch Nachrichten ausgelöst wird, die in der Warteschlange SYSTEM.SAMPLE.ECHO eintreffen. 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 1127.

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 gibt durch die Option MQGMO_WAIT ein *Warteintervall* von 15 Sekunden vor. Wenn die Warteschlange leer ist, wartet das Programm also 15 Sekunden lang auf eingehende Nachrichten. 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 ein Warteschlangenmanager nicht angegeben wird, stellt 'amqsget' eine Verbindung zum Standardwarteschlangenmanager her und 'amqsgetc' stellt eine Verbindung zu dem Warteschlangenmanager her, der durch eine Umgebungsvariable oder in der Definitionsdatei des Clientkanals 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).

Diese Programme verwenden auch die Umgebungsvariable **MQSAMP_USER_ID**, die auf die für die Verbindungsauthentifizierung zu verwendende Benutzer-ID gesetzt werden sollte. Nach dieser Festlegung fragt das Programm nach dem zugehörigen Kennwort für diese Benutzer-ID.

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 mithilfe der Umgebungsvariablen MQSERVER oder durch Definition von Verbindungsdetails mithilfe einer Definitionstabelle für den Clientkanal, die auch für **amqsgetc** unter Verwendung von Umgebungsvariablen bereitgestellt werden kann; siehe beispielsweise MQCCDTURL.

Ein Beispiel, in dem MQSERVER verwendet und eine lokale Verbindung zu einem Warteschlangenmanager hergestellt wird, der über einen Listener verfügt, der auf Port 1414 ausgeführt wird und den Standardserver-Verbindungskanal verwendet, lautet wie folgt:

```
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 1163).

Verwenden Sie **amqsfhac** parallel zu **amqmfsck**, um die Integrität des Dateisystems zu prüfen. Weitere Informationen finden Sie unter **amqmfsck** (Dateisystemprüfung) und Verhalten des gemeinsam genutzten Dateisystems überprüfen .

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. Befolgen Sie die Schritte unter Verhalten eines gemeinsam genutzten Dateisystems überprüfen zum Ausführen von **amqsfhac**.
- 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 141 auf Seite 1162.

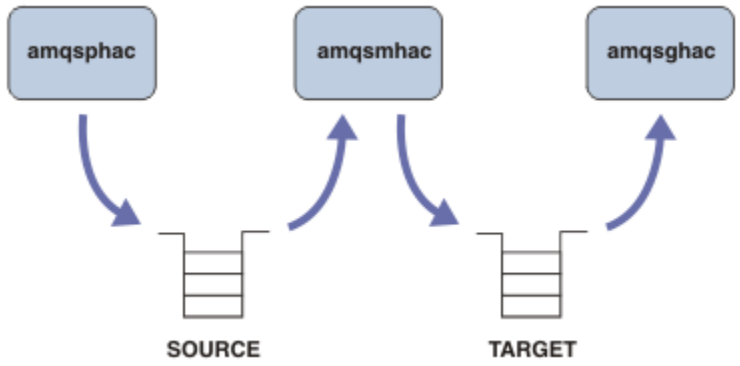


Abbildung 141. Beispiele zu wiederverbindbaren Clients

Gehen Sie zum Ausführen der Beispielprogramme wie folgt vor:

1. Erstellen Sie eine Datei namens `hasamples.tst`, die folgende Befehle enthält:
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 Clientkanal-Definitionsdatei `AMQCLCHL.TAB`; 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 1163 in „Warteschlangenmanager konfigurieren und steuern” auf Seite 1163.)
5. Geben Sie den Befehl `endmqm -r -p QM1` ein, um den Warteschlangenmanager zu stoppen, und gestatten Sie dann den Clients, die Verbindung wiederherzustellen.
6. Geben Sie den Befehl `strmqm QM1` ein, um den Warteschlangenmanager erneut 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.
```

   Notieren Sie sich das Datenverzeichnis, um die Variable **MQCHLLIB** zu einem späteren Zeitpunkt festlegen zu können.

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.
4. Machen Sie den Clients die Clientkanaltabelle bekannt.

   Verwenden Sie das Datenverzeichnis, das in Schritt „1" auf Seite 1163 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 Informationen

Verhalten eines gemeinsam genutzten Dateisystems überprüfen

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 1127.

Diese Programme sollen als Auslöserprogramme ausgeführt werden, sodass ihre einzige Eingabe eine MQTMC2-Struktur (Auslösenachricht) für Systeme unter IBM i, Windows, UNIX and Linux ist. 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:

- ULW Starten Sie unter UNIX, 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 (Veröffentlichen und Subskribieren) 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 1135.

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 anderes Befehlsfenster im selben 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 152 auf Seite 1167 sind sechs Beispielprogrammtypen aufgeführt.

Tabelle 152. 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.
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.

Java

Das Java-Beispielprogramm `MQPubSubApiSample.java` kombiniert Publisher und Subskribenten in einem Programm. Seine Quell- 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 1135.

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 1168, 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 1135 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...** > **General** > **File System** > **Next** > **Browse...** (Importieren > Allgemein > Dateisystem > Weiter > Durchsuchen) aus und navigieren Sie zum Pfad *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` (hierbei steht *MQ_INSTALLATION_PATH* für das IBM MQ-Installationsverzeichnis).
   k) Klicken Sie in der Anzeige **Import** (siehe Abbildung 142 auf Seite 1169) 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**.

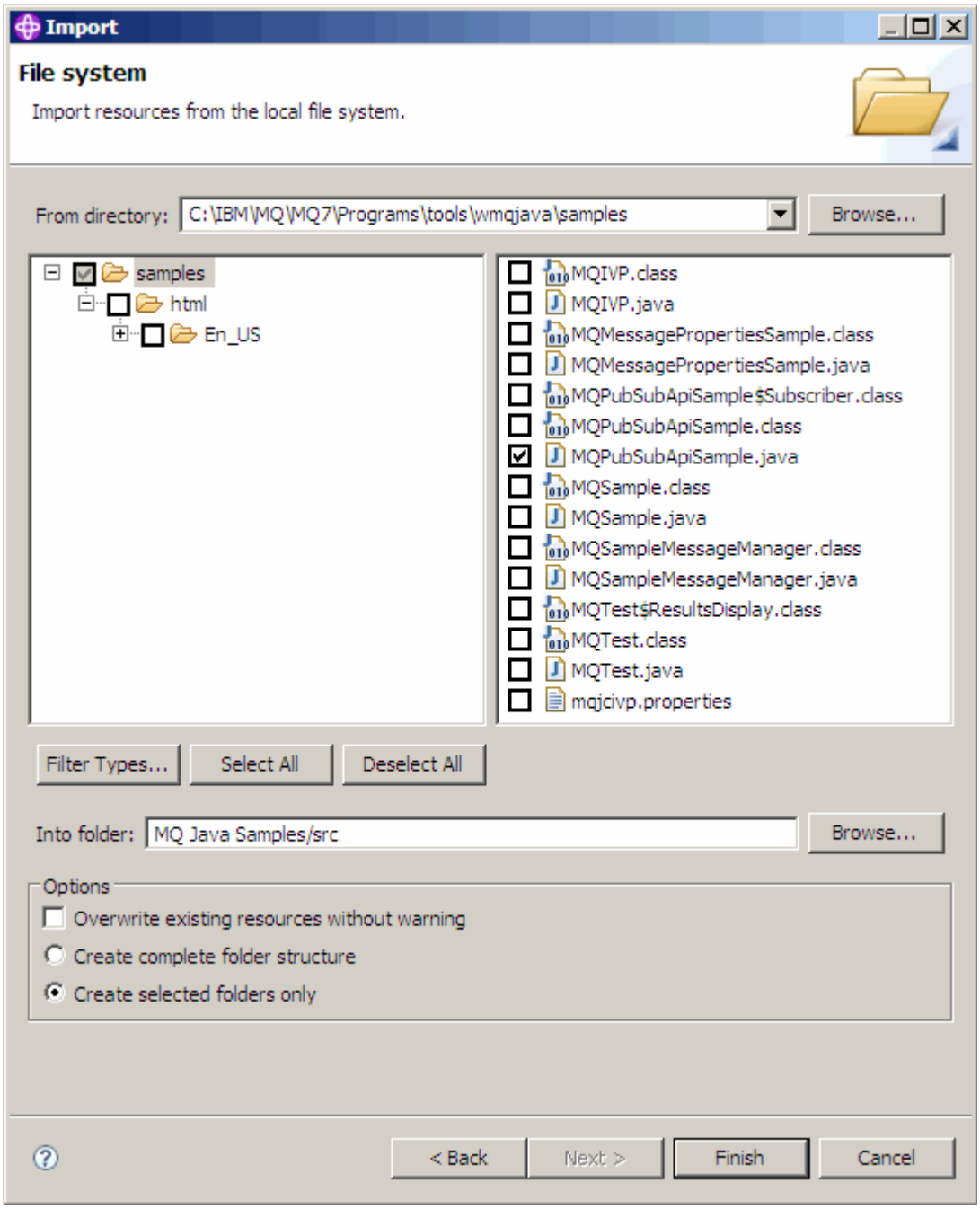


Abbildung 142. Dateisystemimport

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** > **1Java Application**.
   - 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 (Subskribentenzähler)

  Diese Attribute können nur durch Bearbeiten des Quellcodes im Konstruktor geändert werden.

  - hostname
  - Port
  - channel

  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). Doppelklicken Sie in **Create, Manage and Run applications** (Anwendungen erstellen, verwalten und ausführen) auf Java Application (Java-Anwendung) und klicken Sie dann auf die Registerkarte **(x) = Arguments** ((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.

   Beispiele: `-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 UNIX Fügen Sie auf UNIX 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. Beispiel:
   - Linux UNIX Stellen Sie auf UNIX and Linux-Systemen sicher, dass das Verzeichnis `/var/mqm/trace` vorhanden ist und exportieren Sie folgende Umgebungsvariable:

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows Stellen Sie unter Windows sicher, dass das Verzeichnis `C:\temp` vorhanden ist und legen Sie folgende Umgebungsvariable 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 1127.

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

ULW

Mit dem Beispielprogramm 'amqqsput' werden Nachrichten mithilfe lokaler Bindungen eingereiht, während mit dem Beispielprogramm 'amqsputc' Nachrichten mithilfe von Clientbindungen eingereiht werden. Jedes dieser Programme verwendet folgende positionsgebundene Parameter:

1. Name der Zielwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (optional)

   Wenn kein Warteschlangenmanager angegeben wird, stellt 'amqsput' eine Verbindung zum Standardwarteschlangenmanager her und 'amqsputc' stellt eine Verbindung zu dem Warteschlangenmanager her, der durch die Umgebungsvariable MQSERVER oder in der Definitionsdatei des Clientkanals festgelegt 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.

Diese Programme verwenden auch die Umgebungsvariable **MQSAMP_USER_ID**, die auf die für die Verbindungsauthentifizierung zu verwendende Benutzer-ID gesetzt werden sollte. Nach dieser Festlegung fragt das Programm nach dem zugehörigen Kennwort für diese Benutzer-ID.

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

ULW

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. Beispiel:

```
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 Reference Message-Beispielanwendung AMQSPRM bzw. AMQSPRMA unter IBM i verwendet 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:

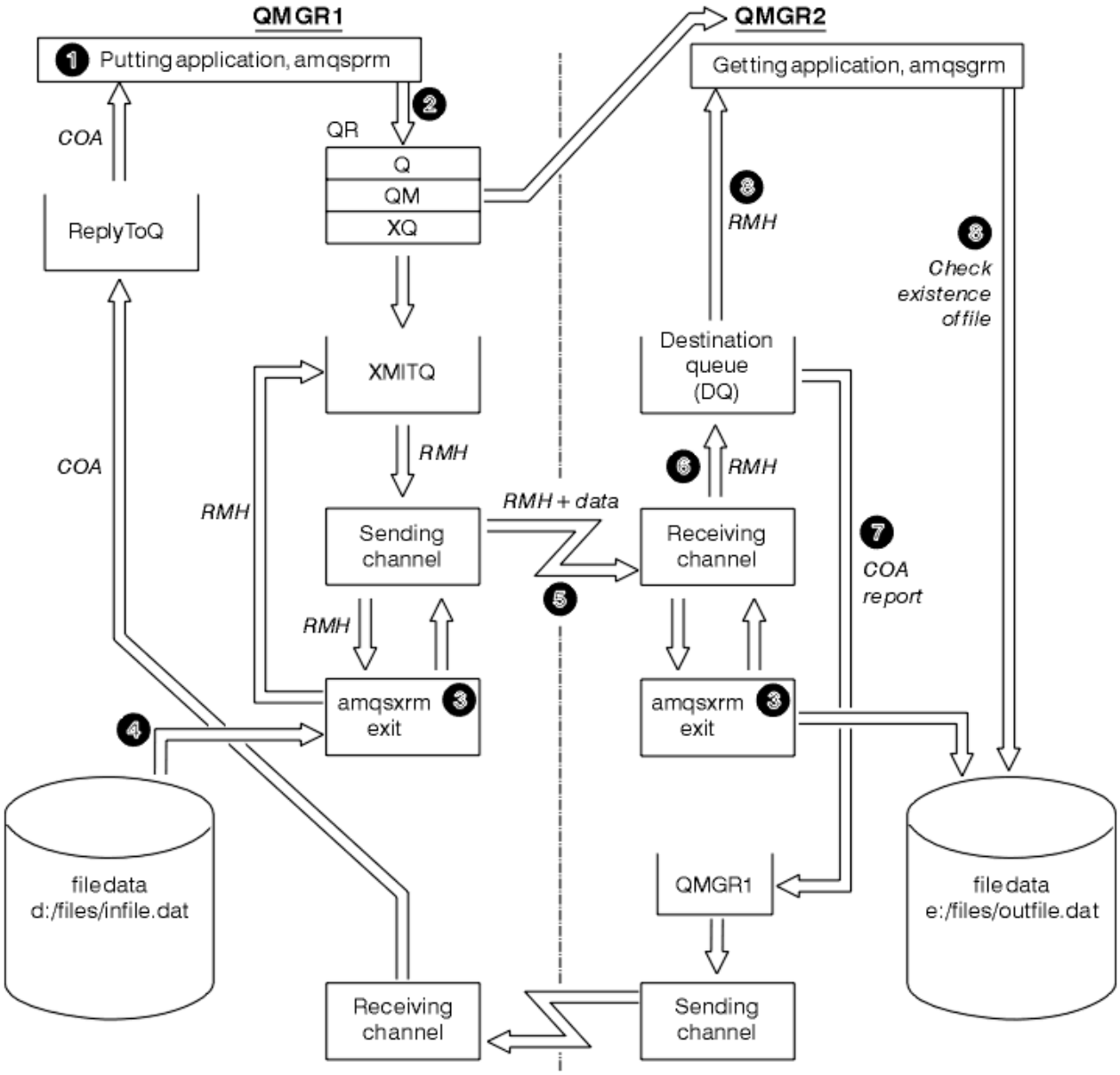


Abbildung 143. Referenznachrichtenbeispiel ausführen

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: Die folgenden Definitionen ermöglichen das Erstellen einer Referenznachricht, mit der eine Datei mit dem Objekttyp FLATFILE vom Warteschlangenmanager QMGR1 an den Warteschlangenmanager QMGR2 gesendet wird, und das erneute Erstellen der Datei gemäß Definition im AMQSPRM-Aufruf (bzw. AMQSPRMA unter IBM i). 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.

   In den MQSC-Definitionen wurde ein AIX-Stil zum Angeben der Exits verwendet; wenn Sie also 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-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-Befehlssyntax

Anmerkung: IBM i Wenn Sie 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: IBM i

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 Zur Angabe von Parametern können Sie entweder einen Schrägstrich (/) oder einen Gedankenstrich (-) verwenden.

IBM i Beispiel:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Anmerkung: Auf UNIX 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)')
   ```

   Solaris HP-UX AIX Dies wird unter AIX, HP-UX und Solaris 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.

   Solaris HP-UX AIX Ändern Sie unter AIX, HP-UX und Solaris den Gruppeninhaber 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 UNIX 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* am Requester- oder Empfängerkanal den Wert Null hat und die angegebene Datei nicht existiert, 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 bei Sender- und Serverkanälen das Feld *DataLogicalLength* in der Eingangsreferenznachricht den Wert Null hat, wird der übrige Teil der Datei, von *DataLogicalOffset* bis zum Dateiende, über den Kanal 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* auf MQXCC_SUPPRESS_FUNCTION gesetzt, sodass die Nachricht, die verarbeitet wird, in die Warteschlange für nicht zustellbare Nachrichten eingereiht und nicht an die Zielwarteschlange übermittelt wird. In MQCXP wird ein Rückkopplungscode zurückgegeben. *Feedback* wird an die Anwendung zurückgegeben, die die Nachricht in das Feld *Feedback* des Nachrichtendeskriptors einer Berichtsnachricht gestellt hat. Grund hierfür ist, dass die einreihende Anwendung durch das Festlegen von MQRO_EXCEPTION im Feld *Report* des MQMD Ausnahmeberichte angefordert 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 HP-UX

HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o amqsxrma.o amqsqrma.c -I MQ_INSTALLATION_PATH/inc
$ ld -b amqsxrma.o -o /var/mqm/exits64/amqsxrma -L MQ_INSTALLATION_PATH/lib64
-L/usr/lib/pa20_64 -lmqm_r -lpthread
```

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

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 Solaris

Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -I  MQ_INSTALLATION_PATH/inc
-L  MQ_INSTALLATION_PATH/lib64 -R  MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm


-lsocket
-lnsl -ldl
```

Unter Windows

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 1012
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 1127 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.

ULW *Request-Beispielprogramm mittels Triggering unter UNIX, Linux, and Windows ausführen*

Starten Sie unter UNIX, 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 144 auf Seite 1184 veranschaulicht die gemeinsame Verwendung der Request- und Inquire-Beispielprogramme.

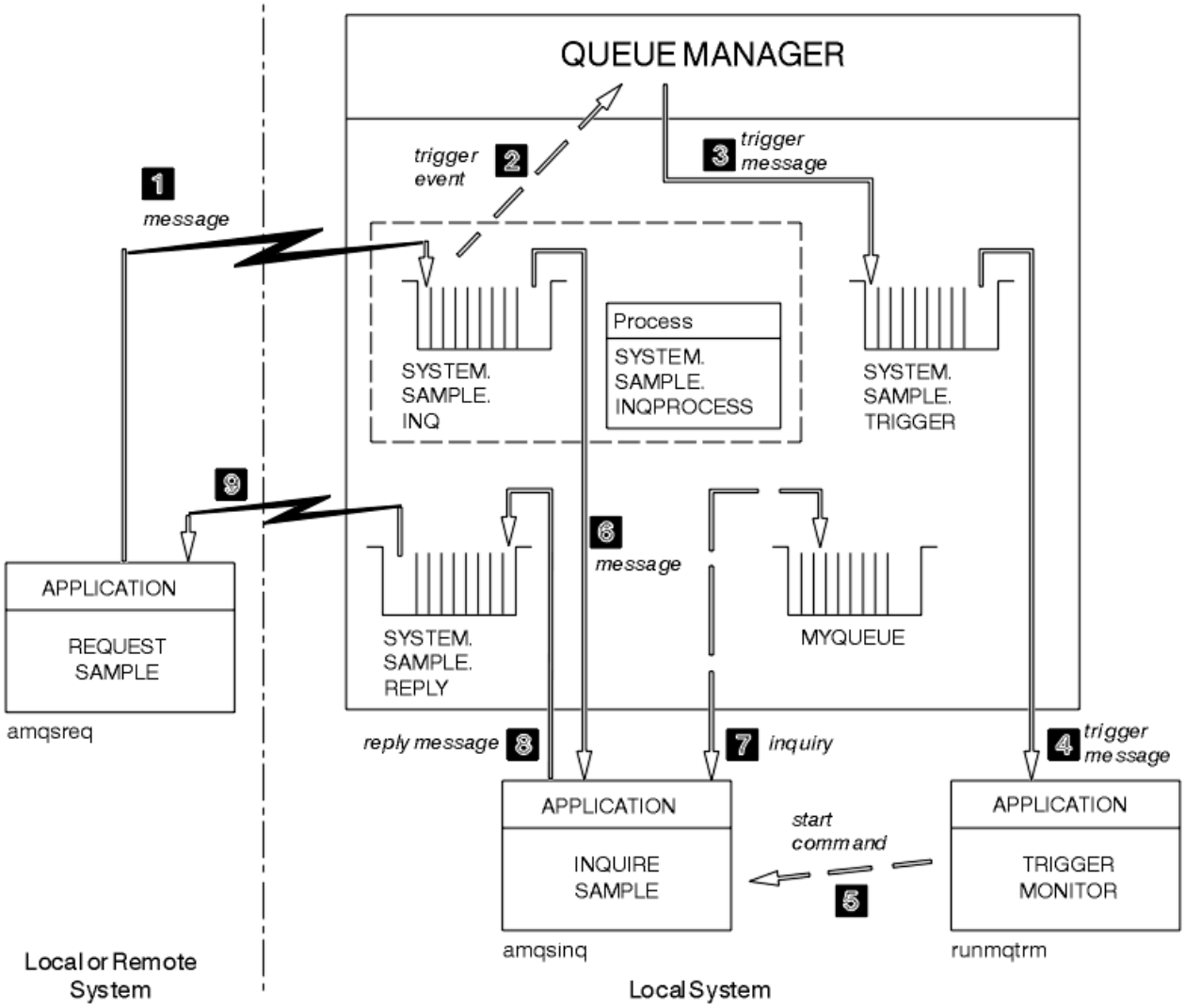


Abbildung 144. Request- und Inquire-Beispielprogramme mittels Triggering ausführen

In Abbildung 144 auf Seite 1184 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 144 auf Seite 1184 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 1185 ein.

Weitere Informationen zum Triggering finden Sie unter „IBM MQ-Anwendungen durch Auslöser starten" auf Seite 901.

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 145 auf Seite 1187 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.

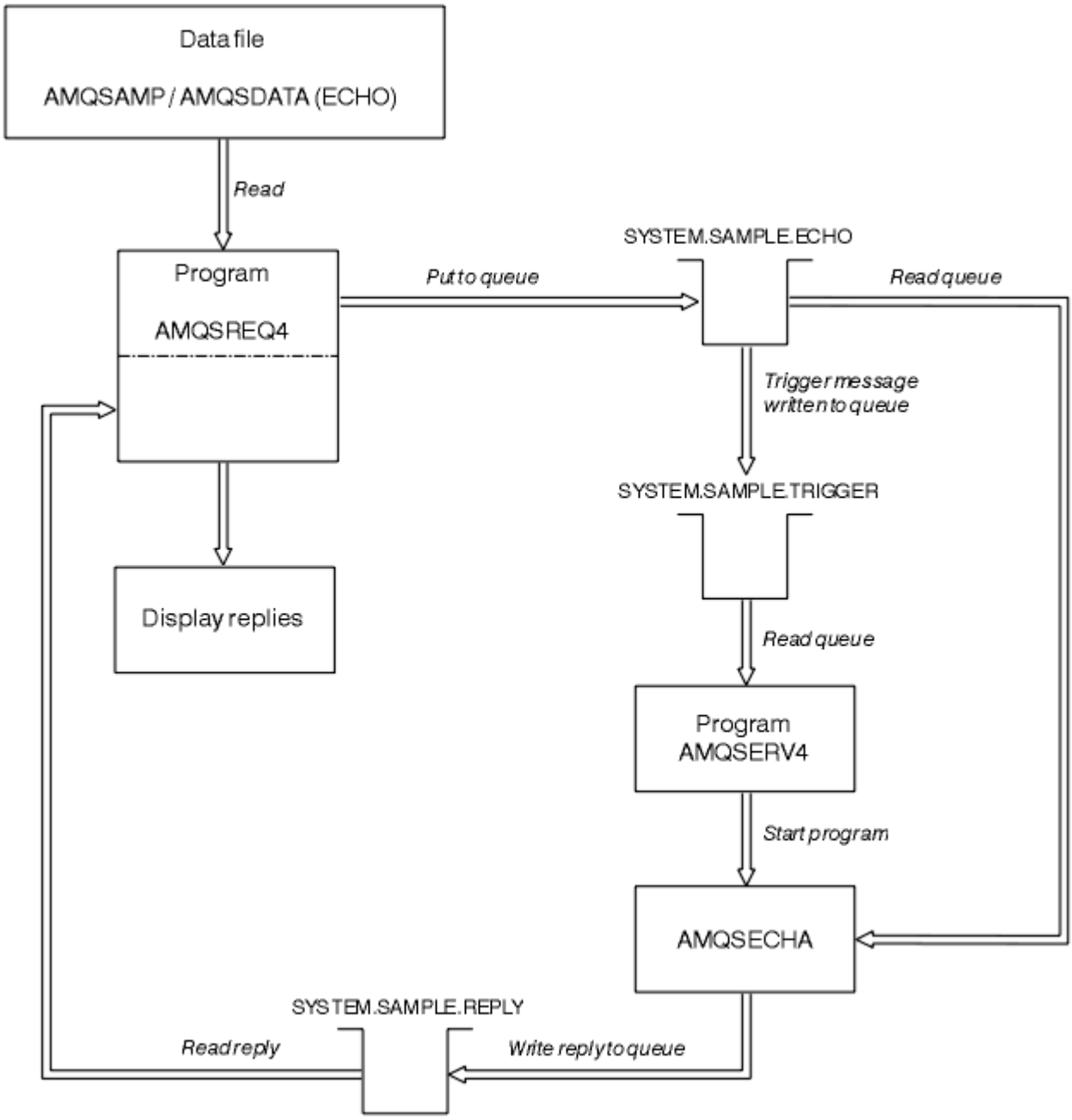


Abbildung 145. Ablaufdiagramm des IBM i Client/Server (Echo)-Beispielprogramms

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 1127.

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:

- ULW Starten Sie unter UNIX, 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

AMQSSLC ist ein C-Beispielprogramm, das veranschaulicht, wie die MQCNO-und MQSCO-Strukturen verwendet werden, um TLS-Clientverbindungsinformationen im MQCONNX-Aufruf bereitzustellen. 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. Es kann jedoch auch als reguläre MQI-Anwendung verknüpft werden. In diesem Fall stellt es lediglich eine Verbindung zu einem lokalen Warteschlangenmanager her; die Clientverbindungsinformationen werden ignoriert.

AMQSSLC 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 KeyReposStem
Der Stammname der Schlüsselrepositorydatei. Dies ist der vollständige Dateipfad ohne das Suffix '.kdb'. Beispiel:

```
/home/user/client
C:\User\client
```

-s CipherSpec
Die CipherSpec-Zeichenfolge für den TLS-Kanal, die dem Attribut SSLCIPH in der SVRCONN-Kanaldefinition im 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. Die Bedeutung dieser Werte entspricht den Werten für die Umgebungsvariable MQSUITEB sowie der entsprechenden EncryptionPolicySuiteB-Einstellung 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 Zertifikaten, 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.

Parameter für den OCSP-Zertifikatswiderruf:

-o URL
Die URL des OCSP-Responders.

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 UNIX, Linux, and Windows einrichten.
3. Erstellen Sie für den Client ein Schlüsselrepository und weisen Sie diesem den Namen *clientkey.kdb* zu.
4. Erstellen Sie für den Warteschlangenmanager ein persönliches Zertifikat. Weitere Informationen finden Sie im Abschnitt Selbst signiertes persönliches Zertifikat unter UNIX, 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 in den Abschnitten Öffentlichen Teil eines selbst signierten Zertifikats aus einem Schlüsselrepository unter UNIX, Linux, and Windows extrahieren und CA-Zertifikat (oder den öffentlichen Teil eines selbst signierten Zertifikats) zu einem Schlüsselrepository unter UNIX, Linux oder Windows 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_SHA)
   ```

   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" -s TLS_RSA_WITH_AES_128_CBC_SHA
    -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_SHA
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 1127.

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-, UNIX, 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).

- ULW Unter UNIX, 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.

Auslösebeispielprogramme in IBM i beenden

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 Er übergibt keinen IBM i-Job, sondern ruft IBM i-Anwendungen im eigenen Job auf (oder startet CICS-Anwendungen mit STRCICSUSR).
- Bei CICS -Anwendungen wird die *EnvData*beispielsweise durch die Angabe der CICS -Region aus der Auslösenachricht im Befehl STRCICSUSR ersetzt.
- Er öffnet die Initialisierungswarteschlange für die gemeinsam genutzte Eingabe, sodass viele Auslöse-server 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.

Windows UNIX *TUXEDO-Beispiele unter UNIX und 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.

Windows UNIX *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; dabei ist TUXDIR das Stammverzeichnis für TUXEDO und *MQ_INSTALLATION_PATH* das übergeordnete Verzeichnis, 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 der TUXEDO-Datei udataobj/RM 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; dabei ist TUXDIR das Stammverzeichnis für TUXEDO und *MQ_INSTALLATION_PATH* das übergeordnete Verzeichnis, 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 der TUXEDO-Datei udataobj/RM 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.

HP-UX *Serverumgebung für HP-UX (32-Bit) erstellen*
Einrichtung der Serverumgebung für IBM MQ for HP-UX (32 Bit).

Informationen zu diesem Vorgang

Anmerkung: Die 32-Bit-Version der TUXEDO-Serverumgebung kann nur auf Itanium-Plattformen erstellt werden.

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; dabei ist TUXDIR das Stammverzeichnis für TUXEDO:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin: MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Fügen Sie der TUXEDO-Datei udataobj/RM folgende Zeile hinzu:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.so MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

4. Führen Sie die folgenden Befehle aus:

```
$ mkfldhdr MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc    MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

   Nach Ausführung der Befehle 'mkfldhdr' und 'viewc' wird im TUXEDO-Anwendungsverzeichnis die Headerdatei amqstxvx.h erstellt. Kopieren Sie diese Datei aus dem TUXEDO-Anwendungsverzeichnis in das TUXEDO-Verzeichnis 'include' und führen Sie anschließend die folgenden Befehle aus:

```
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -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.so \
       -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.so
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. Bearbeiten Sie die Datei ubbstxcx.cfg. 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.

HP-UX *Serverumgebung für HP-UX (64-Bit) erstellen*
Einrichtung der Serverumgebung für IBM MQ for HP-UX (64 Bit).

Informationen zu diesem Vorgang

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

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; dabei ist TUXDIR das Stammverzeichnis für TUXEDO:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin: MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib64:/lib64
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Fügen Sie auf der HP-UX IA64 (IPF)-Plattform die folgende Zeile zur TUXEDO-Datei udataobj/RM hinzu:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.so MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

   Anmerkung: Die auf der HP-UX IA64 -Plattform (IPF) bereitgestellten IBM MQ -Bibliotheken haben die Dateinamenerweiterung .so.

4. Führen Sie die folgenden Befehle aus:

```
$ mkfldhdr MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc    MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

Nach Ausführung der Befehle 'mkfldhdr' und 'viewc' wird im TUXEDO-Anwendungsverzeichnis die Headerdatei amqstxvx.h erstellt. Kopieren Sie diese Datei aus dem TUXEDO-Anwendungsverzeichnis in das TUXEDO-Verzeichnis 'include' und führen Sie anschließend die folgenden Befehle aus:

```
$ buildtms  -o MQXA -r MQSERIES_XA_RMI
```

Auf der HP-UX IA64-Plattform (IPF):

```
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -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.so \
       -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.so
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

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.

Solaris *Serverumgebung für Solaris (32-Bit) erstellen*

Einrichtung der Serverumgebung für IBM MQ for Solaris (32 Bit).

Informationen zu diesem Vorgang

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

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; dabei ist TUXDIR das Stammverzeichnis für TUXEDO:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export LD_LIBRARY_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
```

3. Fügen Sie Folgendes zur TUXEDO-Datei udataobj/RM hinzu (RM muss *MQ_INSTALLATION_PATH*/lib/libmqmcs und *MQ_INSTALLATION_PATH*/lib/libmqmzse enthalten).

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a MQ_INSTALLATION_PATH/lib/libmqmcs.so \
MQ_INSTALLATION_PATH/lib/libmqmzse.so
```

4. Führen Sie die folgenden Befehle aus:

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
```

5. Bearbeiten Sie die Datei ubbstxcx.cfg. Fügen Sie ihr bei Bedarf Informationen wie den Maschinennamen, die Arbeitsverzeichnisse und den Warteschlangenmanager hinzu:

```
$ tmloadcf  -y 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.

Solaris *Serverumgebung für Solaris (64-Bit) erstellen*
Einrichtung der Serverumgebung für IBM MQ for Solaris (64 Bit).

Informationen zu diesem Vorgang

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

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; dabei ist TUXDIR das Stammverzeichnis für TUXEDO:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib64
$ export LD_LIBRARY_PATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Fügen Sie Folgendes zur TUXEDO-Datei udataobj/RM hinzu (RM muss *MQ_INSTALLATION_PATH*/lib/libmqmcs und *MQ_INSTALLATION_PATH*/lib/libmqmzse enthalten).

```
MQSERIES_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a MQ_INSTALLATION_PATH/lib64/libmqmcs.so \
MQ_INSTALLATION_PATH/lib64/libmqmzse.so
```

4. Führen Sie die folgenden Befehle aus:

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
```

5. Bearbeiten Sie die Datei ubbstxcx.cfg. Fügen Sie ihr bei Bedarf Informationen wie den Maschinennamen, die Arbeitsverzeichnisse und den Warteschlangenmanager hinzu:

```
$ tmloadcf  -y 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 153. 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 bei der Installation von TUXEDO angegebene Verzeichnispfad. Beispiel: `f:\tuxedo`.
APPDIR	Der für die Beispielanwendung zu verwendende Verzeichnispfad. Beispiel: `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 146. 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 146 auf Seite 1202 aufgeführt. Sie wird im IBM MQ-Verzeichnis 'samples' als `ubbstxcn.cfg` bereitgestellt.

Die für IBM MQ for Windows bereitgestellte Beispiel-Makefile (siehe Abbildung 147 auf Seite 1203) 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 147. 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 146 auf Seite 1202), indem Sie den Maschinennamen und den Warteschlangenmanager, zu dem eine Verbindung hergestellt werden soll, hinzufügen.
5. Fügen Sie der TUXEDO-Datei *TUXDIR*`udataobj\rm` 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. Setzen Sie das aktuelle Verzeichnis auf *APPDIR* 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.

Windows *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 154. 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 bei der Installation von TUXEDO angegebene Verzeichnispfad. Beispiel: `f:\tuxedo`.
APPDIR	Der für die Beispielanwendung zu verwendende Verzeichnispfad. Beispiel: `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 148. 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 148 auf Seite 1205 aufgeführt. Sie wird im IBM MQ-Verzeichnis 'samples' als `ubbstxcn.cfg` bereitgestellt.

Die für IBM MQ for Windows bereitgestellte Beispiel-Makefile (siehe Abbildung 149 auf Seite 1206) 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 149. 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 148 auf Seite 1205), 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. Setzen Sie das aktuelle Verzeichnis auf *APPDIR* 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.

Windows UNIX *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.

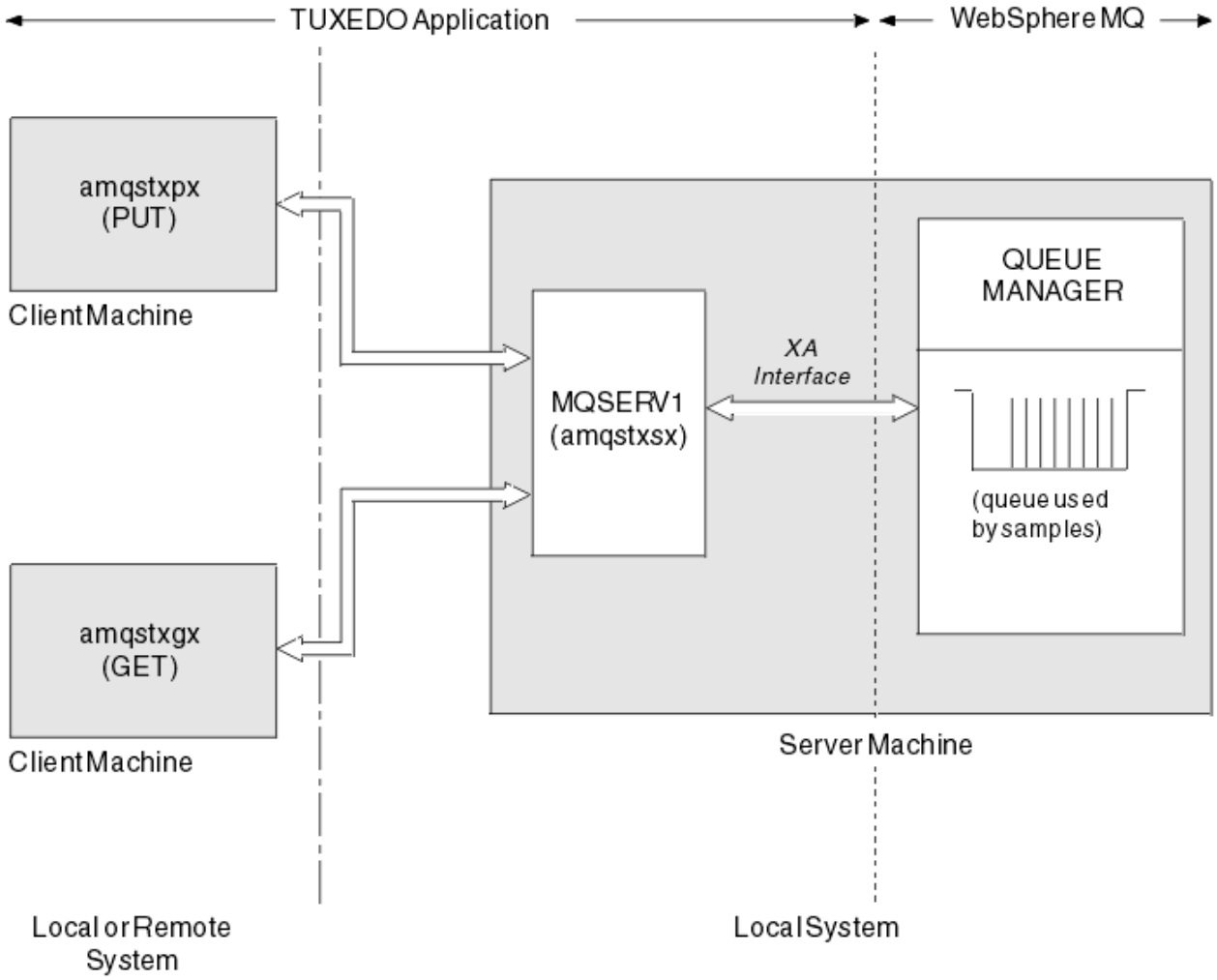


Abbildung 150. Zusammenspiel der TUXEDO-Beispiele

Windows UNIX *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`

Beispiel:

- `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.

Windows UNIX *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`

Beispiel:

- `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 hat den Namen `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 hat den Namen `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 enthalten sind. Beispiel:

```
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.[10]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.

[10] 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 RPT01-Datei erstellt. Die ursprüngliche RPT01-Datei wird in RPT02 umbenannt. RPT02 wird in RPT03 umbenannt und die ursprüngliche 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.

ULW *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.

Tivoli Directory Server 6.3 Client muss installiert sein, um CEPL verwenden zu können

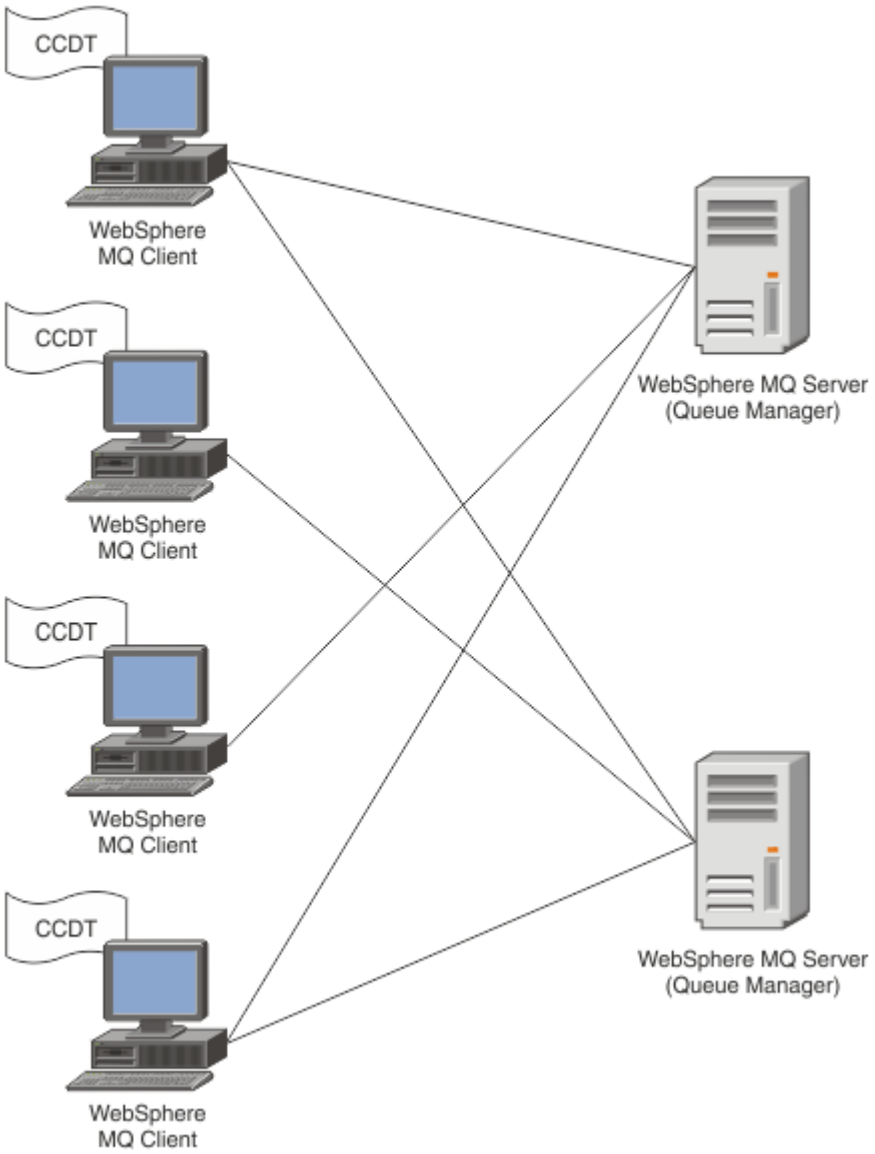
Für die Verwendung dieses Beispielprogramms sind praktische Erfahrungen mit der Verwaltung von IBM MQ auf den unterstützten Plattformen erforderlich.

Windows Solaris 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



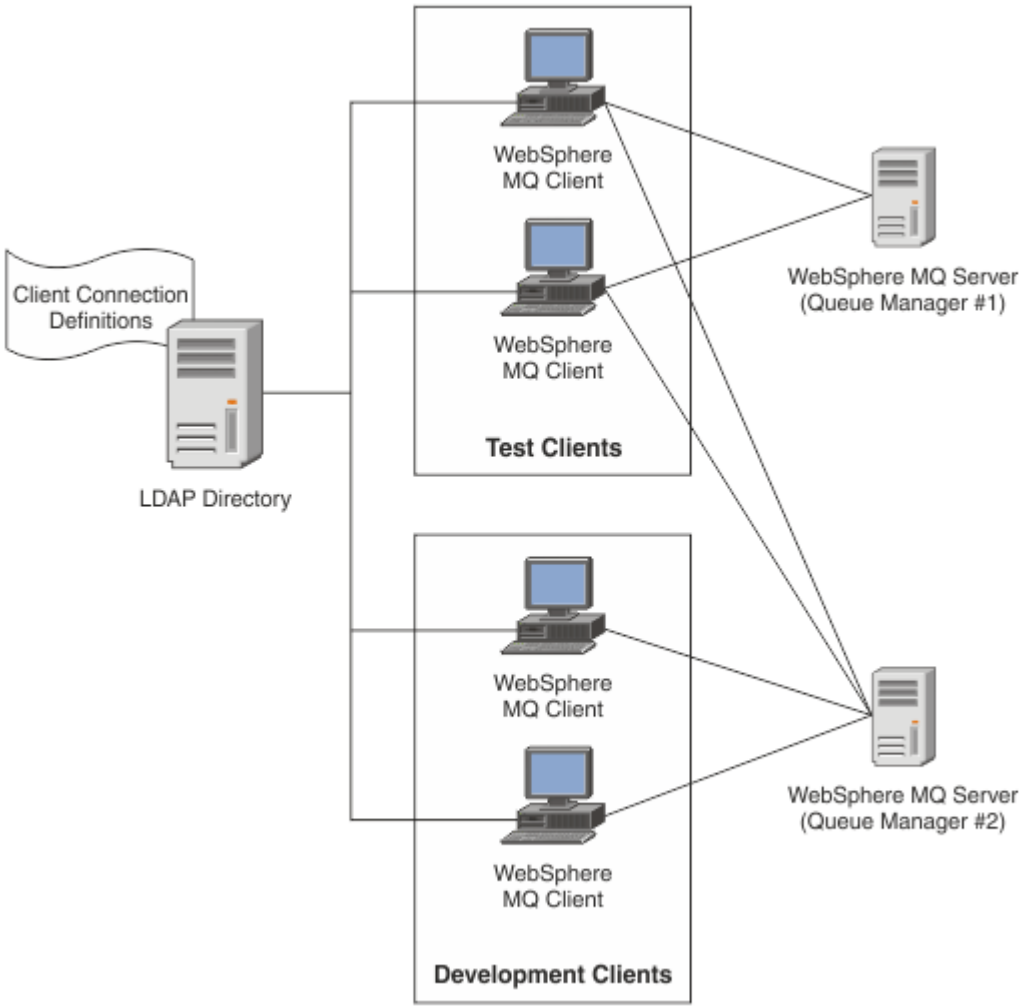
verteilt werden.

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.

ULW *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 6.3 Client oder höher

Unterstützte Betriebssysteme:

1. Windows Windows (7/8/2008/2012)
2. Solaris Solaris (SPARC und x86-64)
3. AIX AIX
4. Linux Linux
   - RHEL V4 und V5 unter System p
   - SUSE V9 und V10 unter System p
   - RHEL V4 und V5 unter System x 32 Bit und 64 Bit
   - SUSE V9 und V10 unter System x 32 Bit und 64 Bit
5. HP-UX HP IA64

Anmerkung: Das Beispielprogramm ist für die folgenden Plattformen nicht verfügbar:

- z/OS z/OS
- IBM i IBM i
- HP-UX HP PA-RISC

ULW *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

Ein neuer Abschnitt mit dem Namen 'PreConnect' muss zur Clientkonfigurationsdatei, z. B. `mqclient.ini`, hinzugefügt werden. Der Abschnitt 'PreConnect' enthält die folgenden Schlüsselwörter:

Module : Der Name des Moduls, das den API-Exit-Code enthält. Wenn dieses Feld den vollständigen Pfad des Moduls enthält, wird es unverändert übernommen. Andernfalls wird der Ordner `exit` oder `exit64` in der IBM MQ-Installation durchsucht.

Function : Der Name des funktionalen Eingangspunkts in die Bibliothek, die den PreConnect-Exit-Code enthält. Die Funktionsdefinition basiert auf dem MQ_PRECONNECT_EXIT-Prototyp.

Data : URI des LDAP-Repositorys, das die Kanaldefinitionen enthält.

Das folgende Snippet ist ein Beispiel für die erforderlichen Änderungen in der Datei `mqclient.ini`:

```
PreConnect:
Module=amqlcelp
Function=PreConnectExit
Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ULW *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 9.0 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 einer Eingabe handelt es sich bei diesem Parameter um die Filterzeichenfolge, die für den MQCONN API-Aufruf über den Parameter **QMgrName** bereitgestellt 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.

ULW *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 Solaris Linux AIX *Beispielcode zum Erstellen des Exits für die Suche nach Verbindungsendpunkten*

Mit den Beispiel-Code-Snippets können Sie die Quelle unter AIX, Linux, Solaris und Windows kompilieren.

Quelle kompilieren

Sie können die Quelle mit beliebigen LDAP-Clientbibliotheken kompilieren, z. B. IBM Tivoli Directory Server 6.3 Clientbibliotheken. In dieser Dokumentation wird vorausgesetzt, dass Sie Tivoli Directory Server 6.3 Clientbibliotheken verwenden.

Anmerkung: Die Preconnect-Exitbibliothek wird von den folgenden LDAP-Servern unterstützt:

- IBM Tivoli Directory Server 6.3
- Novell eDirectory 8.2

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 6.3 verwenden, die mit dem Microsoft Visual Studio 2003 -Compiler kompiliert werden, erhalten Sie möglicherweise Warnungen, während Sie die Clientbibliotheken von IBM Tivoli Directory Server 6.3 mit dem Microsoft Visual Studio 2012-Compiler oder einem höheren Compiler kompilieren.

Solaris Linux AIX **Exit unter AIX, Linux oder Solaris kompilieren**

Mit dem folgenden Snippet können Sie die Exitquelle unter Linux kompilieren. Einige Compileroptionen können sich unter AIX und Solaris 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)
```

ULW *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 wird mit dem Attribut 'Data' aus der PreConnect-Zeilengruppe in der Datei `mqclient.ini` (d. h. dem LDAP) aufgefüllt.

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 **MQNLDAPCTX**-Parameter eine Verbindung und Bindung mit dem 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. Enthält der **QMgrNameparameter** den Namen eines bestimmten Warteschlangenmanagers, werden bei der Suche alle Kanaldefinitionen zurückgegeben, bei denen der Wert des LDAP-Attributs **ibm-amqQueueManagerName** mit dem angegebenen Namen des Warteschlangenmanagers übereinstimmt.

Hat der Parameter **QMgrName** den Wert '*' oder ' '(leer), werden bei der Suche alle Kanaldefinitionen zurückgegeben, bei denen das Endpunktattribut **ibm-amqIsClientDefault Connection** auf TRUE gesetzt 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 bei einem Exit mit dem Ursachencode MQXR_TERM nicht verwendet und können NULL sein.

Zugehörige Informationen
Zeilengruppe für PreConnect der Clientkonfigurationsdatei

ULW *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.

ULW *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 155. Verzeichniszeichenfolgeattribute für IBM MQ-Clientkanäle

LDAP-Attribut	Beschreibung	IBM MQ-Eigenschaft
CN	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

Tabelle 155. Verzeichniszeichenfolgeattribute für IBM MQ-Clientkanäle (Forts.)		
LDAP-Attribut	**Beschreibung**	**IBM MQ-Eigenschaft**
ibm-amqLocalAddress	Die lokale Kommunikationsadresse des Kanals.	LOCLADDR
ibm-amqModeName	Der Name des LU 6.2-Modus.	MODENAME
ibm-amqPassword	Das Kennwort, das verwendet werden kann.	KENNWORT
ibm-amqQueueManagerName	Der Name des Warteschlangenmanagers bzw. der Gruppe von Warteschlangenmanagern, zu denen eine IBM MQ-Clientanwendung eine Verbindung anfordern kann.	QMNAME
ibm-amqSecurityExitUserData, LDAP-Attribut	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 156. Verzeichnisganzzahlattribute für IBM MQ-Clientkanäle		
LDAP-Attribut	**Beschreibung**	**IBM MQ-Eigenschaft**
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

Tabelle 156. Verzeichnisganzzahlattribute für IBM MQ-Clientkanäle (Forts.)		
LDAP-Attribut	**Beschreibung**	**IBM MQ-Eigenschaft**
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
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 157. 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 158. Listenattribute für IBM MQ-Clientkanäle		
LDAP-Attribut	**Beschreibung**	**IBM MQ-Eigenschaft**
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

ULW *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)
```

Beispiel:

```
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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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 1234
Dieses boolesche Attribut ist für die Suche nach Einträgen gedacht, für die das Attribut 'ibm-amqQueueManagerName' nicht definiert ist.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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 1232

Dieses Attribut gibt den Namen des Warteschlangenmanagers bzw. der Gruppe von Warteschlangenmanagern an, zu denen eine IBM MQ-Clientanwendung eine Verbindung anfordern kann.

ULW *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.

ULW *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.

Dieses Listenattribut wird in Form von Verzeichniszeichenfolgen angegeben, die durch Kommas getrennt sind. Die Angabe von '1,2,4' für dieses Attribut beispielsweise entspricht der Komprimierungsreihenfolge RLE, ZLIBFAST und ZLIBHIGH.

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.

ULW *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.

ULW *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.

ULW *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.

ULW *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 Beispielprogramme für z/OS verwenden

Die prozeduralen Beispielanwendungen, die mit IBM MQ for z/OS bereitgestellt werden, veranschaulichen typische Verwendungsmöglichkeiten der MQI (Message Queue Interface).

Informationen zu diesem Vorgang

IBM MQ for z/OS stellt darüber hinaus auch Beispielexits für die Datenkonvertierung bereit; eine Beschreibung finden Sie unter „Datenkonvertierungsexits schreiben” auf Seite 1033.

Alle Beispielanwendungen werden im Quellenformat bereitgestellt, einige auch im ausführbaren Format. Die Quellenmodule enthalten Pseudocode, der die Programmlogik beschreibt.

Anmerkung: Auch wenn einige der Beispielanwendungen über einfache, anzeigengesteuerte Schnittstellen verfügen, sind sie keine Vorgabe dafür, wie das Look-and-feel Ihrer Anwendungen gestaltet sein sollte. Weitere Informationen zum Erstellen anzeigengesteuerter Schnittstellen für nicht programmierbare Terminals finden Sie im *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) und in dessen Anhang (GG22-9508). Diese Dokumente enthalten Richtlinien für die Erstellung von Anwendungen, die in sich und auch mit anderen Anwendungen konsistent sind.

Prozedur

- Weitere Informationen zu den Beispielprogrammen finden Sie unter den folgenden Links:
  - „In den Beispielanwendungen für z/OS veranschaulichte Funktionen” auf Seite 1237

- „Beispielanwendungen für die Stapelumgebung unter z/OS vorbereiten und ausführen” auf Seite 1244
- „Beispielanwendungen für die TSO-Umgebung unter z/OS vorbereiten” auf Seite 1248
- „Beispielanwendungen für die CICS-Umgebung unter z/OS vorbereiten” auf Seite 1250
- „Beispielanwendung für die IMS-Umgebung unter z/OS vorbereiten” auf Seite 1253
- „Put-Beispielprogramme unter z/OS” auf Seite 1254
- „Get-Beispielprogramme für z/OS” auf Seite 1257
- „Browse-Beispielprogramm unter z/OS” auf Seite 1260
- „Beispielprogramm für die Nachrichtenausgabe unter z/OS” auf Seite 1262
- „Beispielprogramm 'Queue Attributes' für z/OS” auf Seite 1266
- „Mail-Manager-Beispielprogramm unter z/OS” auf Seite 1267
- „Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS” auf Seite 1274
- „Beispielprogramm 'Message Handler' für z/OS” auf Seite 1287
- „Beispielprogramm für asynchrone Put-Operationen unter z/OS” auf Seite 1291
- „Beispielprogramm für die asynchrone Stapelverarbeitung unter z/OS” auf Seite 1292
- „CICS-Beispielprogramme für asynchrone Verarbeitung und für Publish/Subscribe unter z/OS” auf Seite 1294
- „Publish/Subscribe-Beispielprogramm unter z/OS” auf Seite 1297
- „Beispielprogramm zum Setzen und Abfragen von Nachrichteneigenschaften unter z/OS” auf Seite 1299

Zugehörige Tasks

„Beispielprogramme plattformübergreifend verwenden” auf Seite 1125
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 *In den Beispielanwendungen für z/OS veranschaulichte Funktionen*

Dieser Abschnitt vermittelt Ihnen einen Überblick über die in den Beispielanwendungen veranschaulichten MQI-Funktionen; außerdem enthält er Informationen zu den Programmiersprachen, in denen die Beispielprogramme jeweils geschrieben sind, sowie zu den Umgebungen, in denen die einzelnen Beispielprogramme ausgeführt werden.

z/OS *Put-Beispielprogramme unter z/OS*

Die Put-Beispielprogramme veranschaulichen das Einreihen von Nachrichten in eine Warteschlange mithilfe des MQPUT-Aufrufs.

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

Das Programm wird in COBOL und C bereitgestellt und kann in Stapelumgebungen und CICS-Umgebungen ausgeführt werden. Informationen zur Stapelanwendung finden Sie in Tabelle 161 auf Seite 1245, Informationen zur CICS-Anwendung finden Sie in Tabelle 168 auf Seite 1251.

z/OS *Get-Beispielprogramm unter z/OS*

Die Get-Beispielprogramme veranschaulichen den Abruf von Nachrichten aus einer Warteschlange mithilfe des MQGET-Aufrufs.

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

Das Programm wird in COBOL und C bereitgestellt und kann in Stapelumgebungen und CICS-Umgebungen ausgeführt werden. Informationen zur Stapelanwendung finden Sie in Tabelle 161 auf Seite 1245, Informationen zur CICS-Anwendung finden Sie in Tabelle 168 auf Seite 1251.

z/OS *Beispielprogramm 'Browse' für z/OS*

Das Beispielprogramm 'Browse' veranschaulicht, wie mit der Option 'Browse' (Durchsuchen) eine Nachricht gesucht und ausgegeben wird und wie dieser Vorgang danach für alle Nachrichten der Warteschlange wiederholt wird.

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQGET (zum Anzeigen von Nachrichten)
- MQCLOSE
- MQDISC

Das Programm wird in COBOL, Assembler, PL/I und C bereitgestellt. Die Anwendung wird in Stapelumgebungen ausgeführt. Informationen zur Stapelanwendung finden Sie in Tabelle 162 auf Seite 1245.

z/OS *Beispielprogramm für die Nachrichtenausgabe unter z/OS*

Das Beispielprogramm für die Nachrichtenausgabe veranschaulicht, wie eine Nachricht aus einer Warteschlange entfernt wird und die Daten in der Nachricht sowie die Felder des Nachrichtendeskriptors der Nachricht ausgegeben werden. Es kann optional alle Nachrichteneigenschaften anzeigen, die jeder Nachricht zugeordnet sind.

Durch Entfernen von Kommentarzeichen aus zwei Zeilen des Quellenmoduls können Sie das Programm dergestalt ändern, dass die Nachrichten in einer Warteschlange nur angezeigt und nicht entfernt werden. Dieses Programm ist hilfreich für die Fehlerdiagnose bei Anwendungen, die Nachrichten in eine Warteschlange einreihen.

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQGET (zum Abrufen von Nachrichten aus einer Warteschlange, optional kann er auch zum Anzeigen von Nachrichten verwendet werden)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

Das Programm wird in C bereitgestellt. Die Anwendung wird in Stapelumgebungen ausgeführt. Informationen zur Stapelanwendung finden Sie in Tabelle 163 auf Seite 1246.

z/OS *Beispielprogramm für Warteschlangenattribute unter z/OS*

Das Beispielprogramm für Warteschlangenattribute veranschaulicht, wie IBM MQ for z/OS-Objektattribute abgefragt und ihre Werte festgelegt werden.

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

Das Programm wird in COBOL, Assembler und C bereitgestellt. Die Anwendung wird in CICS-Umgebungen ausgeführt. Informationen zur CICS-Anwendung finden Sie in Tabelle 169 auf Seite 1251.

z/OS *Mail-Manager-Beispielprogramm unter z/OS*

Dieser Abschnitt enthält Informationen zum Mail-Manager-Beispielprogramm.

Das Mail-Manager-Beispielprogramm veranschaulicht Folgendes:

- Die Verwendung von Aliaswarteschlangen
- Die Erstellung einer temporären dynamischen Warteschlange mithilfe einer Modellwarteschlange
- Die Verwendung von Empfangswarteschlangen für Antworten
- Die Verwendung von Synchronisationspunkten in CICS-Umgebungen und Stapelumgebungen
- Das Senden von Befehlen an die Eingabewarteschlange für Systembefehle
- Das Testen von Rückgabecodes
- Das Senden von Nachrichten an ferne Warteschlangenmanager (sowohl unter Verwendung der lokalen Definition eines fernen Warteschlangenmanagers als auch durch direktes Einreihen von Nachrichten in eine angegebene Warteschlange auf einm fernen Warteschlangenmanager)

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

Die Anwendung wird in drei Versionen bereitgestellt:

- Als CICS-Anwendung in COBOL
- Als TSO-Anwendung in COBOL
- Als TSO-Anwendung in C

Die TSO-Anwendungen verwenden den IBM MQ for z/OS-Stapeladapter und enthalten einige ISPF-Anzeigen.

Informationen zur TSO-Anwendung finden Sie in Tabelle 166 auf Seite 1248, Informationen zur CICS-Anwendung finden Sie in Tabelle 170 auf Seite 1251.

z/OS *Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS*

Dieser Abschnitt enthält Informationen zur Verwendung der Beispielanwendung zur Prüfung der Kreditwürdigkeit.

Bei der Beispielanwendung zur Prüfung der Kreditwürdigkeit handelt es sich um eine Reihe von Programmen, die Folgendes veranschaulichen:

- Die Entwicklung einer Anwendung, die in mehreren Umgebungen ausgeführt werden kann
- Die Erstellung einer temporären dynamischen Warteschlange mithilfe einer Modellwarteschlange
- Die Verwendung einer Korrelations-ID
- Das Definieren und Übergeben von Kontextinformationen
- Die Verwendung von Nachrichtenprioritäten und der Nachrichtenpersistenz
- Das Starten von Programmen über Auslöser
- Die Verwendung von Empfangswarteschlangen für Antworten
- Die Verwendung von Aliaswarteschlangen
- Die Verwendung einer Warteschlange für nicht zustellbare Nachrichten
- Die Verwendung von Namenslisten
- Das Testen von Rückgabecodes

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET (zum Anzeigen und Abrufen von Nachrichten unter Verwendung von Warte- und Signaloptionen und zum Abrufen einer bestimmten Nachricht)
- MQINQ
- MQSET
- MQCLOSE

Die Beispielanwendung kann als eigenständige CICS-Anwendung ausgeführt werden. Um jedoch die Erstellung einer Anwendung zum Einreihen von Nachrichten zu veranschaulichen, die sowohl die in CICS- als auch in IMS-Umgebungen bereitgestellten Funktionen nutzt, wird ein Modul auch als IMS-Programm für die Verarbeitung von Nachrichten im Stapelbetrieb bereitgestellt.

Die CICS-Programme werden in C und COBOL bereitgestellt. Das IMS-Programm wird in C bereitgestellt.

Informationen zur CICS-Anwendung finden Sie in Tabelle 171 auf Seite 1252, Informationen zur IMS-Anwendung finden Sie in Tabelle 173 auf Seite 1254.

z/OS *Beispielprogramm 'Message Handler' für z/OS*

Mit der Beispielanwendung für die Nachrichtenbehandlung können Sie nach Nachrichten in einer Warteschlange suchen sowie Nachrichten weiterleiten und löschen.

Die Anwendung verwendet die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

Die Anwendung wird in C und COBOL bereitgestellt. Die Anwendung wird unter TSO ausgeführt. Informationen zur TSO-Anwendung finden Sie in Tabelle 167 auf Seite 1249.

z/OS *Beispiele für Exits für die verteilte Steuerung von Warteschlangen unter z/OS*

Dieser Abschnitt enthält eine Tabelle mit den Quellenprogrammen der Exitbeispielprogramme für die verteilte Steuerung von Warteschlangen.

Die folgende Tabelle enthält die Namen der Quellenprogramme der Exitbeispielprogramme für die verteilte Steuerung von Warteschlangen:

Tabelle 159. Quelle für die Exitbeispiele für die verteilte Steuerung von Warteschlangen

Mitgliedsname	Für Programmiersprache	Beschreibung	Bereitgestellt in Bibliothek
CSQ4BAX0	Assembler	Quellenprogramm	SCSQASMS
CSQ4BCX1	C	Quellenprogramm	SCSQC37S
CSQ4BCX2	C	Quellenprogramm	SCSQC37S
CSQ4BCX4	C	Quellenprogramm	SCSQC37S

Anmerkung: Für die Quellenprogramme ist eine Programmverknüpfung mit CSQXSTUB hergestellt.

z/OS *Beispielprogramme für Datenkonvertierungsexits für z/OS*

Es wird ein Entwurf für eine Exitroutine für die Datenkonvertierung bereitgestellt; außerdem veranschaulicht ein in IBM MQ bereitgestelltes Beispielprogramm den MQXCNVC-Aufruf.

Die folgende Tabelle enthält die Namen der Quellenprogramme der Beispielexitprogramme für die Datenkonvertierung:

Tabelle 160. Quellenprogramme der Exitbeispiele für die Datenkonvertierung (nur für Assembler)

Mitgliedsname	Beschreibung	Bereitgestellt in Bibliothek
CSQ4BAX8	Quellenprogramm	SCSQASMS
CSQ4BAX9	Quellenprogramm	SCSQASMS
CSQ4CAX9	Quellenprogramm	SCSQASMS

Anmerkung: Für die Quellenprogramme ist eine Programmverknüpfung mit CSQASTUB hergestellt.

Weitere Informationen hierzu finden Sie unter „Datenkonvertierungsexits schreiben” auf Seite 1033.

z/OS *Publish/Subscribe-Beispielprogramme unter z/OS*

Die Publish/Subscribe-Beispielprogramme veranschaulichen die Verwendung der Publish- und Subscribe-Funktionen in IBM MQ.

Vier Beispielprogramme in C und zwei Beispielprogramme in COBOL veranschaulichen die Programmierung der Publish/Subscribe-Schnittstelle von IBM MQ.

Die Anwendungen verwenden die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH

- MQINQMP

Die Public/Subscribe-Beispielprogramme werden in C und COBOL bereitgestellt. Die Beispielanwendungen werden in der Stapelumgebung ausgeführt. Informationen zu den Stapelanwendungen finden Sie unter Publish/Subscribe-Beispielprogramme.

z/OS *Warteschlangenmanager für das Akzeptieren von Clientverbindungen unter z/OS konfigurieren*

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. Verwendet Ihr Warteschlangenmanager keine Kanalauthentifizierungsdatensätze, 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 1242 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. Diese Benutzer-ID legt die Berechtigung fest, unter der die Clientverbindung auf dem Warteschlangenmanager ausgeführt wird.
2. Starten Sie ein Listenerprogramm.
   a) Stellen Sie sicher, dass Ihr Kanalinitiator gestartet wurde. Gegebenenfalls müssen Sie ihn mit dem Befehl **START CHINIT** starten.
   b) Starten Sie das Listenerprogramm mit folgendem Befehl:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   Dabei ist *nnnn* die von Ihnen angegebene Portnummer.
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')
```

Kanalname ist 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 1242

5. Wenn 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 die erforderliche Berechtigung für die Verbindung mit dem Warteschlangenmanager und die Abfrage desselben:
   a) Stellen Sie sicher, dass Ihr Kanalinitiator gestartet wurde. Gegebenenfalls müssen Sie ihn mit dem Befehl START CHINIT starten.
   b) Starten Sie einen TCP-Listener beispielsweise durch Ausgabe des folgenden Befehls:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

Dabei ist *nnnn* die von Ihnen angegebene Portnummer.

7. Handelt es sich bei Ihrer Anwendung um eine Punkt-zu-Punkt-Anwendung, d. h., sie nutzt Warteschlangen, so 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:
   Geben Sie die RACF-Befehle aus:

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

Dabei gilt Folgendes:

Warteschlangenmanagername ist der Name Ihres Warteschlangenmanagers.
Warteschlangenname ist der Name Ihrer Warteschlange.
Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1” auf Seite 1242

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 über Ihr Thema einräumen. Geben Sie hierzu die folgenden RACF-Befehle aus:

```
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)
```

Dabei gilt Folgendes:

Warteschlangenmanagername ist der Name Ihres Warteschlangenmanagers.
Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID aus dem Schritt „1” auf Seite 1242
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. Weitere Informationen finden Sie im Abschnitt Benutzerzugriff auf Themen steuern.

Nächste Schritte

Ihre Clientanwendung kann nun eine Verbindung zum Warteschlangenmanager herstellen und Nachrichten über die Warteschlange einreihen oder abrufen.

Zugehörige Informationen

SET CHLAUTH
CHANNEL DEFINE CHANNEL
QLOCAL DEFINIER
SET AUTHREC
z/OS Berechtigung zum Arbeiten mit IBM MQ-Objekten in z/OS

z/OS *Beispielanwendungen für die Stapelumgebung unter z/OS vorbereiten und ausführen*

Zum Vorbereiten einer Beispielanwendung, die in der Stapelumgebung ausgeführt wird, werden dieselben Schritte wie beim Erstellen jeder anderen IBM MQ for z/OS-Stapelanwendung ausgeführt.

Diese Schritte sind unter „z/OS-Batchanwendungen erstellen” auf Seite 1089 aufgeführt.

Wenn ein Beispielprogramm in einem ausführbaren Format bereitgestellt wird, können Sie es auch aus der Ladebibliothek 'thlqual.SCSQLOAD' ausführen.

Anmerkung: Das Browse-Beispielprogramm in der Assemblersprache verwendet Datensteuerblöcke (DCB, Data Control Block), daher muss die Programmverknüpfung unter Verwendung von RMODE(24) erfolgen.

Die Bibliotheksmember, die verwendet werden sollen, sind in Tabelle 161 auf Seite 1245, Tabelle 162 auf Seite 1245, Tabelle 163 auf Seite 1246 und Tabelle 164 auf Seite 1246 aufgeführt.

Sie müssen für die Beispielprogramme, die Sie verwenden möchten, die bereitgestellte JCL verwenden (siehe Tabelle 161 auf Seite 1245, Tabelle 162 auf Seite 1245, Tabelle 163 auf Seite 1246 und Tabelle 164 auf Seite 1246).

Die Anweisung PARM in der bereitgestellten JCL enthält eine Reihe von Parametern, die Sie ändern müssen. Für die Ausführung der C-Beispielprogramme müssen die Parameter durch Leerzeichen getrennt werden, für die Beispielprogramme in der Assemblersprache sowie in COBOL und PL/I müssen sie durch Kommas getrennt werden. Hat Ihr Warteschlangenmanager beispielsweise den Namen CSQ1 und soll die Anwendung mit einer Warteschlange namens LOCALQ1 ausgeführt werden, muss die Anweisung PARM in der JCL für COBOL, PL/I und die Assemblersprache das folgende Format haben:

```
PARM=(CSQ1,LOCALQ1)
```

In der JCL für C muss die Anweisung PARM das folgende Format haben:

```
PARM=('CSQ1 LOCALQ1')
```

Sie können jetzt die Jobs übergeben.

z/OS *Namen der Beispielstapelanwendungen unter z/OS*

Dieser Abschnitt enthält eine Übersicht über die Programme, die für Beispielstapelanwendungen bereitgestellt werden.

Eine Übersicht über die Stapelanwendungsprogramme ist in den folgenden Tabellen enthalten:

- Tabelle 161 auf Seite 1245-Beispielprogramme für Put- und Get-Operationen
- Tabelle 162 auf Seite 1245: Browse-Beispielprogramm
- Tabelle 163 auf Seite 1246: Beispielprogramm für die Nachrichtenausgabe
- Tabelle 164 auf Seite 1246: Publish/Subscribe-Beispielprogramme
- Tabelle 165 auf Seite 1247: Weitere Beispielprogramme

Tabelle 161. Put- und Get-Stapelbeispielprogramme

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4BCJ1	C	Get-Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4BCK1	C	Put-Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4BCJR	C	Beispielausführungs-JCL für CSQ4BCJ1 und CSQBCK1	SCSQPROC	--
CSQ4BVJ1	COBOL	Get-Quellenprogramm	SCSQCOBS	SCSQLADEN
CSQ4BVK1	COBOL	Put-Quellenprogramm	SCSQCOBS	SCSQLADEN
CSQ4BVJR	COBOL	Beispielausführungs-JCL für CSQBVJ1 und CSQBVK1	SCSQPROC	--

Tabelle 162. Browse-Stapelbeispielprogramm

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4BVA1	COBOL	Quellenprogramm	SCSQCOBS	SCSQLADEN
CSQ4BVAR	COBOL	Beispielausführungs-JCL für CSQ4BVA1	SCSQPROC	--
CSQ4BAA1	Assembler	Quellenprogramm	SCSQASMS	SCSQLADEN

Tabelle 162. Browse-Stapelbeispielprogramm (Forts.)

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4BAAR	Assembler	Beispielausführungs-JCL für CSQ4BAA1	SCSQPROC	--
CSQ4BCA1	C	Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4BCAR	C	Beispielausführungs-JCL für CSQ4BCA1	SCSQPROC	--
CSQ4BPA1	PL/I	Quellenprogramm	SCSQPLIS	SCSQLADEN
CSQ4BPAR	PL/I	Beispielausführungs-JCL für CSQ4BPA1	SCSQPROC	--

Tabelle 163. Stapelbeispielprogramm für die Nachrichtenausgabe (nur in C)

Mitgliedsname	Beschreibung	Quellendatei bereitgestellt in Bibliothek	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4BCG1	Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4BCGR	Beispielausführungs-JCL für CSQ4BCG1	SCSQPROC	--
CSQ4BCL1	Browse-Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4BCLR	Beispielausführungs-JCL für CSQ4BCL1	SCSQPROC	--

Tabelle 164. Publish/Subscribe-Beispielprogramme

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	JCL in SCSQPROC	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4BCP1	C	Quellenprogramm zum Veröffentlichen von Nachrichten zu einem Thema	SCSQC37S	CSQ4BCPP	SCSQLADEN
CSQ4BCP2	C	Quellenprogramm zum Subskribieren eines Themas und zum Abrufen von Nachrichten	SCSQC37S	CSQ4BCPS	SCSQLADEN

Tabelle 164. Publish/Subscribe-Beispielprogramme (Forts.)

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	JCL in SCSQPROC	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4BCP3	C	Quellenprogramm zum Subskribieren eines Themas unter Verwendung einer vom Benutzer bereitgestellten Zieladresse und zum Abrufen von Nachrichten	SCSQC37S	CSQ4BCPD	SCSQLADEN
CSQ4BCP4	C	Quellenprogramm zum Subskribieren eines Themas unter Verwendung erweiterter Optionen und zum Abrufen von Nachrichten	SCSQC37S	CSQ4BCPE	SCSQLADEN
CSQ4BVP1	COBOL	Quellenprogramm zum Veröffentlichen von Nachrichten zu einem Thema	SCSQCOBS	CSQ4BVPP	SCSQLADEN
CSQ4BVP2	COBOL	Quellenprogramm zum Subskribieren eines Themas und zum Abrufen von Nachrichten	SCSQCOBS	CSQ4BVPS	SCSQLADEN

Tabelle 165. Weitere Beispielprogramme

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	JCL in SCSQPROC	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4BCS1	C	Quellenprogramm für die asynchrone Verarbeitung	SCSQC37S	CSQ4BCSC	SCSQLADEN
CSQ4BCS2	C	Quellenprogramm für asynchrone Put-Operationen und Statusüberprüfung	SCSQC37S	CSQ4BCSP	SCSQLADEN
CSQ4BCM1	C	Quellenprogramm zum Abfragen von Nachrichteneigenschaften	SCSQC37S	CSQ4BCMP	SCSQLADEN
CSQ4BCM2	C	Quellenprogramm zum Setzen von Nachrichteneigenschaften	SCSQC37S	CSQ4BCMP	SCSQLADEN

z/OS *Beispielanwendungen für die TSO-Umgebung unter z/OS vorbereiten*

Zum Vorbereiten einer Beispielanwendung, die in der TSO-Umgebung ausgeführt wird, werden dieselben Schritte wie beim Erstellen jeder anderen IBM MQ for z/OS-Stapelanwendung ausgeführt.

Diese Schritte sind unter „z/OS-Batchanwendungen erstellen” auf Seite 1089 aufgeführt. Die Bibliotheksmember, die verwendet werden, sind in Tabelle 166 auf Seite 1248 aufgeführt.

Wenn ein Beispielprogramm in einem ausführbaren Format bereitgestellt wird, können Sie es auch aus der Ladebibliothek 'thlqual.SCSQLOAD' ausführen.

Bei der Mail-Manager-Beispielanwendung müssen Sie sicherstellen, dass die von dieser Anwendung verwendeten Warteschlangen in Ihrem System verfügbar sind. Sie sind im Member '**thlqual**.SCSQPROC(CSQ4CVD)' definiert. Damit sichergestellt ist, dass diese Warteschlangen immer verfügbar sind, können Sie diese Member der Initialisierungseingabedatei CSQINP2 hinzufügen oder diese Warteschlangendefinitionen mithilfe des Programms CSQUTIL laden.

z/OS *Namen der TSO-Beispielanwendungen unter z/OS*

In diesem Abschnitt sind die Namen der Programme, die für die jeweiligen TSO-Beispielanwendungen bereitgestellt werden, sowie die Bibliotheken angegeben, in denen sich die entsprechenden Quellendateien, JCL-Dateien und ggf. ausführbaren Dateien (letzteres nur für die Beispielanwendung für die Nachrichtenbehandlung) befinden.

Eine Übersicht der TSO-Anwendungsprogramme finden Sie in folgenden Tabellen:

- Tabelle 166 auf Seite 1248: Mail-Manager-Beispiel
- Tabelle 167 auf Seite 1249: Beispielanwendung für die Nachrichtenbehandlung

Diese Beispielanwendungen verwenden ISPF-Anzeigen. Daher müssen Sie bei der Programmverknüpfung der Programme den ISPF-Stub (ISPLINK) einschließen.

Tabelle 166. Mail-Manager-TSO-Beispielprogramm

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek
CSQ4CVD	unabhängig	IBM MQ for z/OS-Objektdefinitionen	SCSQPROC
CSQ40	unabhängig	ISPF-Nachrichten	SCSQMSGE
CSQ4RVD1	COBOL	CLIST für die Initialisierung von CSQ4TVD1	SCSQCLST
CSQ4TVD1	COBOL	Quellenprogramm für das Menüprogramm	SCSQCOBS
CSQ4TVD2	COBOL	Quellenprogramm für das Programm zum Abrufen von Mails	SCSQCOBS
CSQ4TVD4	COBOL	Quellenprogramm für das Programm zum Senden von Mails	SCSQCOBS
CSQ4TVD5	COBOL	Quellenprogramm für das Programm für Kurznamen	SCSQCOBS
CSQ4VDP1-6	COBOL	Anzeigendefinitionen	SCSQPNLA
CSQ4VD0	COBOL	Datendefinition	SCSQCOBC
CSQ4VD1	COBOL	Datendefinition	SCSQCOBC

Tabelle 166. Mail-Manager-TSO-Beispielprogramm (Forts.)

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek
CSQ4VD2	COBOL	Datendefinition	SCSQCOBC
CSQ4VD4	COBOL	Datendefinition	SCSQCOBC
CSQ4RCD1	C	CLIST für die Initialisierung von CSQ4TCD1	SCSQCLST
CSQ4TCD1	C	Quellenprogramm für das Menüprogramm	SCSQC37S
CSQ4TCD2	C	Quellenprogramm für das Programm zum Abrufen von Mails	SCSQC37S
CSQ4TCD4	C	Quellenprogramm für das Programm zum Senden von Mails	SCSQC37S
CSQ4TCD5	C	Quellenprogramm für das Programm für Kurznamen	SCSQC37S
CSQ4CDP1-6	C	Anzeigendefinitionen	SCSQPNLA
CSQ4TC0	C	Include-Datei	SCSQC370

Tabelle 167. Die TSO-Beispielanwendung für die Nachrichtenbehandlung

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4TCH0	C	Datendefinition	SCSQC370	--
CSQ4TCH1	C	Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4TCH2	C	Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4TCH3	C	Quellenprogramm	SCSQC37S	SCSQLADEN
CSQ4RCH1	C und COBOL	CLIST für die Initialisierung von CSQ4TCH1 oder CSQ4TVH1	SCSQCLST	--
CSQ4CHP1	C und COBOL	Anzeigendefinition	SCSQPNLA	--
CSQ4CHP2	C und COBOL	Anzeigendefinition	SCSQPNLA	--
CSQ4CHP3	C und COBOL	Anzeigendefinition	SCSQPNLA	--
CSQ4CHP9	C und COBOL	Anzeigendefinition	SCSQPNLA	--
CSQ4TVH0	COBOL	Datendefinition	SCSQCOBC	--
CSQ4TVH1	COBOL	Quellenprogramm	SCSQCOBS	SCSQLADEN
CSQ4TVH2	COBOL	Quellenprogramm	SCSQCOBS	SCSQLADEN
CSQ4TVH3	COBOL	Quellenprogramm	SCSQCOBS	SCSQLADEN

z/OS *Beispielanwendungen für die CICS-Umgebung unter z/OS vorbereiten*

Melden Sie sich vor der Ausführung der CICS-Beispielprogramme bei CICS mit LOGMODE 32702 an. Dies ist erforderlich, da die Beispielprogramme für einen 3270-Bildschirm, Modell 2 geschrieben wurden.

Führen Sie zum Vorbereiten einer Beispielanwendung, die in der CICS-Umgebung ausgeführt werden soll, die folgenden Schritte aus:

1. Erstellen Sie die symbolische Maske und die physikalische Maske für das Beispielprogramm, indem Sie die BMS-Bildschirmdefinitionsquelle (in der Bibliothek '**thlqual**.SCSQMAPS', dabei ist **thlqual** das übergeordnete Qualifikationsmerkmal, das von Ihrer Installation verwendet wird) assemblieren. Verwenden Sie bei der Benennung der Masken den Namen der BMS-Bildschirmdefinitionsquelle (nicht verfügbar für Put- und Get-Beispielprogramme), lassen Sie jedoch das letzte Zeichen des Namens weg.
2. Führen Sie dieselben Schritten wie beim Erstellen einer beliebigen anderen IBM MQ for z/OS-Anwendung für die CICS-Umgebung aus. Diese Schritte sind unter „CICS-Anwendungen in z/OS erstellen" auf Seite 1092 aufgeführt. Die Bibliotheksmember, die verwendet werden sollen, sind in Tabelle 168 auf Seite 1251, Tabelle 169 auf Seite 1251, Tabelle 170 auf Seite 1251 und Tabelle 171 auf Seite 1252 aufgeführt.

   Wenn ein Beispielprogramm in einem ausführbaren Format bereitgestellt wird, können Sie es auch aus der CICS-Ladebibliothek laden.
3. Geben Sie für CICS die Maskengruppe, die Programme und die Transaktion an, indem Sie die CSD-Datei (CICS-Systemdefinition) entsprechend aktualisieren. Die Definitionen, die Sie benötigen, sind im Member '**thlqual**.SCSQPROC(CSQ4S100)' enthalten. Anleitungen hierzu finden Sie im Abschnitt *The CICS-IBM MQ Adapter* in der Produktdokumentation zu CICS Transaction Server 4.1 for z/OS unter CICS Transaction Server 4.1 for z/OS, The CICS-IBM MQ adapter.

   Anmerkung: Für die Beispielanwendung zur Prüfung der Kreditwürdigkeit werden Sie an diesem Punkt eine Fehlermeldung erhalten, wenn Sie die VSAM-Datei, die vom Beispiel verwendet wird, noch nicht erstellt haben.
4. Bei der Beispielanwendung zur Prüfung der Kreditwürdigkeit und bei der Mail-Manager-Beispielanwendung müssen Sie sicherstellen, dass die von diesen Anwendungen verwendeten Warteschlangen in Ihrem System verfügbar sind. Für das Beispiel zur Prüfung der Kreditwürdigkeit sind diese Warteschlangen im Member '**thlqual**.SCSQPROC(CSQ4CVB)' (für COBOL) und im Member '**thlqual**.SCSQPROC(CSQ4CCB)' (für C) definiert. Für das Mail-Manager-Beispiel sind sie im Member '**thlqual**.SCSQPROC (CSQ4CVD)' definiert. Damit sichergestellt ist, dass diese Warteschlangen immer verfügbar sind, können Sie diese Member der Initialisierungseingabedatei CSQINP2 hinzufügen oder diese Warteschlangendefinitionen mithilfe des Programms CSQUTIL laden.

   Für die Beispielanwendung für Warteschlangenattribute können Sie eine oder auch mehrere der Warteschlangen verwenden, die für die anderen Beispielanwendungen bereitgestellt werden. Ebenso können Sie auch Ihre eigenen Warteschlangen verwenden. Allerdings können für das Beispielprogramm in dem Format, in dem es bereitgestellt wird, nur Warteschlangen verwendet werden, in deren Name in den ersten 8 Bytes die Zeichen CSQ4SAMP enthalten sind.

z/OS *Namen der CICS-Beispielanwendungen unter z/OS*

Dieser Abschnitt enthält eine Übersicht über die Programme, die für CICS-Beispielanwendungen bereitgestellt werden.

Eine Übersicht über die CICS-Anwendungsprogramme ist in den folgenden Tabellen enthalten:

- Tabelle 168 auf Seite 1251-Beispielprogramme für Put- und Get-Operationen
- Das Tabelle 169 auf Seite 1251-Beispielprogramm für Warteschlangenattribute
- Das Tabelle 170 auf Seite 1251-Beispielprogramm für Mail-Manager (nur in COBOL)
- Tabelle 171 auf Seite 1252-Beispielanwendung zur Prüfung der Kreditwürdigkeit
- Tabelle 172 auf Seite 1253-Beispielprogramm für die asynchrone Verarbeitung und Publish/Subscribe-CICS-Programm

Tabelle 168. CICS-Beispielprogramme für Put- und Get-Operationen

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4CCK1	C	Put-Quellenprogramm	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Get-Quellenprogramm	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Get-Quellenprogramm	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Put-Quellenprogramm	SCSQCOBS	SCSQCICS
CSQ4S100	unabhängig	CICS-Systemdefinitionsdatei	SCSQPROC	--

Tabelle 169. Das CICS-Beispielprogramm für Warteschlangenattribute

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek	Ausführbare Datei bereitgestellt in Bibliothek
CSQ4CVC1	COBOL	Quellenprogramm	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Nachrichtendefinition	SCSQCOBC	--
CSQ4VCMS	COBOL	BMS-Bildschirmdefinition	SCSQMAPS	SCSQCICS (Name: CSQ4ACM)
CSQ4CAC1	Assembler	Quellenprogramm	SCSQASMS	SCSQCICS
CSQ4AMSG	Assembler	Nachrichtendefinition	SCSQMACS	--
CSQ4ACMS	Assembler	BMS-Bildschirmdefinition	SCSQMAPS	SCSQCICS (Name: CSQ4ACM)
CSQ4CCC1	C	Quellenprogramm	SCSQC37S	SCSQCICS
CSQ4CMSG	C	Nachrichtendefinition	SCSQC370	--
CSQ4CCMS	C	BMS-Bildschirmdefinition	SCSQMAPS	SCSQCICS (Name: CSQ4ACM)
CSQ4S100	unabhängig	CICS-Systemdefinitionsdatei	SCSQPROC	--

Tabelle 170. Das CICS-Beispielprogramm für Mail-Manager (nur in COBOL)

Mitgliedsname	Beschreibung	Quellendatei bereitgestellt in Bibliothek
CSQ4CVD	IBM MQ for z/OS-Objektdefinitionen	SCSQPROC
CSQ4CVD1	Quelle für das Menüprogramm	SCSQCOBS

Tabelle 170. Das CICS-Beispielprogramm für Mail-Manager (nur in COBOL) (Forts.)

Mitgliedsname	Beschreibung	Quellendatei bereitgestellt in Bibliothek
CSQ4CVD2	Quelle für das Programm zum Abrufen von Mails	SCSQCOBS
CSQ4CVD3	Quelle für das Programm zum Anzeigen von Nachrichten	SCSQCOBS
CSQ4CVD4	Quelle für das Programm zum Senden von Mails	SCSQCOBS
CSQ4CVD5	Quelle für das Programm für Kurznamen	SCSQCOBS
CSQ4VDMS	BMS-Bildschirmdefinitionsquelle	SCSQMAPS
CSQ4S100	CICS-Systemdefinitionsdatei	SCSQPROC
CSQ4VD0	Datendefinition	SCSQCOBC
CSQ4VD3	Datendefinition	SCSQCOBC
CSQ4VD4	Datendefinition	SCSQCOBC

Tabelle 171. CICS-Beispielanwendung zur Prüfung der Kreditwürdigkeit

Mitgliedsname	Für Programmiersprache	Beschreibung	Quellendatei bereitgestellt in Bibliothek
CSQ4CVB	unabhängig	IBM MQ-Objektdefinitionen	SCSQPROC
CSQ4CCB	unabhängig	IBM MQ-Objektdefinitionen	SCSQPROC
CSQ4CVB1	COBOL	Quelle für das Benutzerschnittstellenprogramm	SCSQCOBS
CSQ4CVB2	COBOL	Quelle für den Kreditantragsmanager	SCSQCOBS
CSQ4CVB3	COBOL	Quelle für das Girokontenprogramm	SCSQCOBS
CSQ4CVB4	COBOL	Quelle für das Verteilungsprogramm	SCSQCOBS
CSQ4CVB5	COBOL	Quelle für das Programm zur Abfrage der Kreditbürodatenbank	SCSQCOBS
CSQ4CCB1	C	Quelle für das Benutzerschnittstellenprogramm	SCSQC37S
CSQ4CCB2	C	Quelle für den Kreditantragsmanager	SCSQC37S
CSQ4CCB3	C	Quelle für das Girokontenprogramm	SCSQC37S
CSQ4CCB4	C	Quelle für das Verteilungsprogramm	SCSQC37S
CSQ4CCB5	C	Quelle für das Programm zur Abfrage der Kreditbürodatenbank	SCSQC37S
CSQ4CB0	C	Include-Datei	SCSQC370
CSQ4CBMS	C	BMS-Bildschirmdefinitionsquelle	SCSQMAPS
CSQ4VBMS	COBOL	BMS-Bildschirmdefinitionsquelle	SCSQMAPS
CSQ4VB0	COBOL	Datendefinition	SCSQCOBC

Tabelle 171. CICS-Beispielanwendung zur Prüfung der Kreditwürdigkeit (Forts.)			
Mitgliedsname	**Für Programmier-sprache**	**Beschreibung**	**Quellendatei bereitgestellt in Bibliothek**
CSQ4VB1	COBOL	Datendefinition	SCSQCOBC
CSQ4VB2	COBOL	Datendefinition	SCSQCOBC
CSQ4VB3	COBOL	Datendefinition	SCSQCOBC
CSQ4VB4	COBOL	Datendefinition	SCSQCOBC
CSQ4VB5	COBOL	Datendefinition	SCSQCOBC
CSQ4VB6	COBOL	Datendefinition	SCSQCOBC
CSQ4VB7	COBOL	Datendefinition	SCSQCOBC
CSQ4VB8	COBOL	Datendefinition	SCSQCOBC
CSQ4BAQ	unabhängig	Quelle für die VSAM-Datei	SCSQPROC
CSQ4FILE	unabhängig	JCL zum Erstellen der von CSQ4CVB3 verwendeten VSAM-Datei	SCSQPROC
CSQ4S100	unabhängig	CICS-Systemdefinitionsdatei	SCSQPROC

Tabelle 172. CICS-Beispielprogramm für die asynchrone Verarbeitung und Publish/Subscribe-CICS-Programm		
Mitgliedsname	**Beschreibung**	**Quellendatei bereitgestellt in Bibliothek**
CSQ4CVCN	Quelle für das Programm für einfache Nachrichtenverarbeitung	SCSQCOBS
CSQ4CVCT	Quelle für das Programm zur Steuerung der Nachrichtenverarbeitung	SCSQCOBS
CSQ4CVEV	Quelle für das Ereignishandlerprogramm	SCSQCOBS
CSQ4CVPT	Quelle für das Clientprogramm zum Einreihen von Nachrichten	SCSQCOBS
CSQ4CVRG	Quelle für das Registrierungsclientprogramm	SCSQCOBS
CSQ4S100	CICS-Systemdefinitionsdatei	SCSQPROC

z/OS *Beispielanwendung für die IMS-Umgebung unter z/OS vorbereiten*

Die Beispielanwendung zur Prüfung der Kreditwürdigkeit kann teilweise in der IMS-Umgebung ausgeführt werden.

Um diesen Teil der Anwendung für eine Ausführung zusammen mit dem CICS-Beispiel vorzubereiten, müssen Sie zunächst die unter „Beispielanwendungen für die CICS-Umgebung unter z/OS vorbereiten" auf Seite 1250 beschriebenen Schritte ausführen.

Gehen Sie anschließend wie folgt vor:

1. Führen Sie dieselben Schritten wie beim Erstellen einer beliebigen anderen IBM MQ for z/OS-Anwendung für die IMS-Umgebung aus. Diese Schritte sind unter „BMP- oder MPP-Anwendungen unter IMS

erstellen” auf Seite 1093 aufgeführt. Die Bibliotheksmember, die verwendet werden, sind in Tabelle 173 auf Seite 1254 aufgeführt.

2. Machen Sie das Anwendungsprogramm und die Datenbank in IMS bekannt. Um dies zu ermöglichen, werden die Beispiele mit PSBGEN-, DBDGEN-, IMSGEN- und IMSDALOC-Anweisungen sowie Anweisungen für die ACB-Definition bereitgestellt.
3. Laden Sie die Datenbank CSQ4CA, indem Sie die hierfür bereitgestellte Beispiel-JCL entsprechend anpassen und ausführen (CSQ4ILDB). Mit dieser JCL wird die Datenbank mit Daten aus der Datei CSQ4BAQ geladen. Aktualisieren Sie mit einer Datendefinitionsanweisung die IMS-Steuerregion für die Datenbank CSQ4CA.
4. Starten Sie das Girokontenprogramm als BMP-Programm, indem Sie die hierfür bereitgestellte Beispiel-JCL entsprechend anpassen und ausführen. Mit dieser JCL wird ein stapelorientiertes BMP-Programm gestartet. Soll das Programm als nachrichtenorientiertes BMP-Programm ausgeführt werden, müssen Sie die Kommentarzeichen aus der Zeile in der JCL entfernen, die die IN=-Anweisung enthält.

z/OS *Namen der IMS-Beispielanwendungen unter z/OS*

Dieser Abschnitt enthält eine Tabelle mit den Quellen und JCLs, die für die IMS-Beispielanwendung zur Prüfung der Kreditwürdigkeit bereitgestellt sind.

Tabelle 173. Quelle und JCL für die IMS-Beispielanwendung zur Prüfung der Kreditwürdigkeit (nur in C)

Mitgliedsname	Beschreibung	Bereitgestellt in Bibliothek
CSQ4CVB	IBM MQ-Objektdefinitionen	SCSQPROC
CSQ4ICB3	Quelle für das Girokontenprogramm	SCSQC37S
CSQ4ICBL	Quelle zum Laden der Girokontendatenbank	SCSQC37S
CSQ4CBI	Datendefinition	SCSQC370
CSQ4PSBL	PSBGEN-JCL für das Datenbankladeprogramm	SCSQPROC
CSQ4PSB3	PSBGEN-JCL für das Girokontenprogramm	SCSQPROC
CSQ4DBDS	DBDGEN-JCL für die Datenbank CSQ4CA	SCSQPROC
CSQ4GIMS	IMSGEN-Makrodefinitionen für CSQ4IVB3 und CSQ4CA	SCSQPROC
CSQ4ACBG	ACB-Definition (Anwendungssteuerblock) für CSQ4IVB3	SCSQPROC
CSQ4BAQ	Quelle für die Datenbank	SCSQPROC
CSQ4ILDB	Beispielausführungs-JCL für den Datenbankladejob	SCSQPROC
CSQ4ICBR	Beispielausführungs-JCL für das Girokontenprogramm	SCSQPROC
CSQ4DYNA	IMSDALOC-Makrodefinitionen für die Datenbank	SCSQPROC

z/OS *Put-Beispielprogramme unter z/OS*

Mit den Put-Beispielprogrammen werden Nachrichten mithilfe des MQPUT-Aufrufs in eine Warteschlange eingereiht.

Die Quellenprogramme werden in C und COBOL für Stapelumgebungen und CICS-Umgebungen bereitgestellt (siehe Tabelle 161 auf Seite 1245 und Tabelle 168 auf Seite 1251).

Design des Put-Beispielprogramms

Der Ablauf der Programmlogik ist wie folgt:

1. Mit dem MQCONN-Aufruf wird eine Verbindung zum Warteschlangenmanager hergestellt. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben und die Verarbeitung wird gestoppt.

   Anmerkung: Bei einer Ausführung des Beispielprogramms in einer CICS-Umgebung muss kein MQCONN-Aufruf ausgegeben werden; wird trotzdem ein solcher Aufruf ausgegeben, wird DEF_HCONN zurückgegeben. Sie können die Verbindungskennung MQHC_DEF_HCONN für die nachfolgenden MQI-Aufrufe verwenden.
2. Die Warteschlange wird mit dem MQOPEN-Aufruf unter Angabe der Option MQOO_OUTPUT geöffnet. Bei Eingaben für diesen Aufruf verwendet das Programm die unter Schritt „1" auf Seite 1257 zurückgegebene Verbindungskennung. Für die Objektdeskriptorstruktur (MQOD) verwendet es für alle Felder die Standardwerte, mit Ausnahme des Felds mit dem Namen des Warteschlangenmanagers, der als Parameter an das Programm übergeben wird. Schlägt der MQOPEN-Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben und die Verarbeitung wird gestoppt.
3. Im Programm wird eine Schleife erstellt, indem MQPUT-Aufrufe so lange ausgegeben werden, bis die erforderliche Anzahl an Nachrichten in die Warteschlange eingereiht wurden. Schlägt einer dieser MQPUT-Aufrufe fehl, wird die Schleife vorzeitig abgebrochen; es werden keine weiteren MQPUT-Aufrufe ausgegeben und die Beendigungs- und Ursachencodes werden zurückgegeben.
4. Die Warteschlange wird mit dem MQCLOSE-Aufruf unter Angabe der unter Schritt „2" auf Seite 1257 zurückgegebenen Objektkennung geschlossen. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben.
5. Die Verbindung zum Warteschlangenmanager wird mit dem MQDISC-Aufruf unter Angabe der unter Schritt „1" auf Seite 1257 zurückgegebenen Verbindungskennung getrennt. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben.

   Anmerkung: Bei einer Ausführung des Beispielprogramms in einer CICS-Umgebung muss kein MQDISC-Aufruf ausgegeben werden.

z/OS *Put-Beispielprogramme für die Stapelumgebung unter z/OS*

Lesen Sie diesen Abschnitt, wenn Put-Beispielprogramme in der Stapelumgebung eingesetzt werden sollen.

Um diese Beispielprogramme auszuführen, müssen Sie die Beispiel-JCL wie unter „Beispielanwendungen für die Stapelumgebung unter z/OS vorbereiten und ausführen" auf Seite 1244 beschrieben bearbeiten und ausführen.

Für das Programm können in einem EXEC PARM-Feld die folgenden Parameter angegeben werden; in C werden sie durch Leerstellen, in COBOL durch Kommas getrennt:

1. Der Name des Warteschlangenmanagers (4 Zeichen)
2. Der Name der Zielwarteschlange (48 Zeichen)
3. Die Anzahl an Nachrichten (bis zu 4 Ziffern)
4. Das Füllzeichen, das in die Nachricht eingefügt werden soll (1 Zeichen)
5. Die Anzahl der Zeichen, die in die Nachricht geschrieben werden sollen (bis zu 4 Ziffern)
6. Die Persistenz der Nachricht (1 Zeichen: P für 'persistent' oder N für 'nicht persistent)

Wird einer dieser Parameter falsch eingegeben, erhalten Sie entsprechende Fehlernachrichten.

Alle Nachrichten von diesen Beispielprogrammen werden in die SYSPRINT-Datei geschrieben.

Hinweise zur Verwendung

- Um die Beispielprogramme einfach zu halten, gibt es einige minimale funktionale Abweichungen zwischen den verschiedenen Sprachenversionen. Durch Verwendung des in der Beispielausführungs-JCL (CSQ4BCJR und CSQ4BVJR) veranschaulichten Layouts für die Parameter können diese Unterschiede jedoch minimiert werden. Keiner dieser Unterschiede hat Auswirkungen auf die MQI.
- Mit CSQ4BCK1 können Sie für die Anzahl der Nachrichten, die gesendet werden sollen, sowie für die Länge der Nachrichten mehr als 4 Ziffern eingeben.
- Geben Sie in den beiden numerischen Feldern eine Zahl zwischen 1 und 9999 ein. Hierbei sollte es sich um eine positive Zahl handeln. Soll beispielsweise nur eine Nachricht eingereiht werden, können Sie 1, 01, 001 oder 0001 eingeben. Bei Eingabe eines nicht numerischen oder negativen Werts erhalten Sie unter Umständen eine Fehlernachricht. Bei Eingabe von '-1' beispielsweise sendet das COBOL-Programm eine 1-Byte-Nachricht, das C-Programm hingegen erhält eine Fehlernachricht.
- Geben Sie bei beiden Programmen (CSQ4BCK1 und CSQ4BVK1) ein P für den Persistenzparameter ++PER++ ein, wenn die Nachricht persistent sein soll. Andernfalls ist die Nachricht nicht persistent.

z/OS *Put-Beispielprogramme für die CICS-Umgebung unter z/OS*

Lesen Sie diesen Abschnitt, wenn Put-Beispielprogramme in der CICS-Umgebung eingesetzt werden sollen.

Für die Transaktionen können die folgenden Parameter (durch Kommas getrennt) angegeben werden:

1. Die Anzahl an Nachrichten (bis zu 4 Ziffern)
2. Das Füllzeichen, das in die Nachricht eingefügt werden soll (1 Zeichen)
3. Die Anzahl der Zeichen, die in die Nachricht geschrieben werden sollen (bis zu 4 Ziffern)
4. Die Persistenz der Nachricht (1 Zeichen: P für 'persistent' oder N für 'nicht persistent)
5. Der Name der Zielwarteschlange (48 Zeichen)

Wird einer dieser Parameter falsch eingegeben, erhalten Sie entsprechende Fehlernachrichten.

Die COBOL-Version des Put-Beispielprogramms wird in der CICS-Umgebung durch die folgende Eingabe aufgerufen:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

Die C-Version des Put-Beispielprogramms wird in der CICS-Umgebung durch die folgende Eingabe aufgerufen:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Nachrichten von den Beispielprogrammen werden am Bildschirm angezeigt.

Hinweise zur Verwendung

- Um die Beispielprogramme einfach zu halten, gibt es einige minimale funktionale Abweichungen zwischen den verschiedenen Sprachenversionen. Keiner dieser Unterschiede hat Auswirkungen auf die MQI.
- Bei Eingabe eines Warteschlangennamens mit als als 48 Zeichen wird der Name bei 48 Zeichen abgeschnitten, es wird jedoch keine Fehlernachricht zurückgegeben.
- Drücken Sie vor Eingabe der Transaktion die Löschtaste.
- Geben Sie in den beiden numerischen Feldern eine Zahl zwischen 1 und 9999 ein. Hierbei sollte es sich um eine positive Zahl handeln. Soll beispielsweise nur eine Nachricht eingereiht werden, können Sie 1, 01, 001 oder 0001 eingeben. Bei Eingabe eines nicht numerischen oder negativen Werts erhalten Sie unter Umständen eine Fehlernachricht. Bei Eingabe von '-1' beispielsweise wird vom COBOL-Programm eine 1-Byte-Nachricht gesendet, das C-Programm hingegen wird mit einer Fehlernachricht von der Funktion 'malloc()' abgebrochen.

- Geben Sie bei beiden Programmen (CSQ4CCK1 und CSQ4CVK1) ein 'P' für den Persistenzparameter ein, wenn die Nachricht persistent sein soll. Sollen die Nachrichten nicht persistent sein, geben Sie ein 'N' an. Bei Eingabe eines anderen Werts erhalten Sie eine Fehlernachricht.
- Die Nachrichten werden synchronisationspunktgesteuert eingereiht, da für alle Parameter Standardwerte verwendet werden, ausgenommen der Parameter, die beim Programmaufruf gesetzt werden.

z/OS *Get-Beispielprogramme für z/OS*

Das Get-Beispielprogramm ruft Nachrichten aus einer Warteschlange mit dem MQGET-Aufruf ab.

Die Quellenprogramme werden in C und COBOL für Stapelumgebungen und CICS-Umgebungen bereitgestellt (siehe Tabelle 161 auf Seite 1245 und Tabelle 168 auf Seite 1251).

z/OS *Entwurf des Get-Beispielprogramms unter z/OS*

Dieser Abschnitt enthält Informationen zum Design des Get-Beispielprogramms sowie einige Hinweise zu seiner Verwendung.

Der Ablauf der Programmlogik ist wie folgt:

1. Mit dem MQCONN-Aufruf wird eine Verbindung zum Warteschlangenmanager hergestellt. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben und die Verarbeitung wird gestoppt.

   Anmerkung: Bei einer Ausführung des Beispielprogramms in einer CICS-Umgebung muss kein MQCONN-Aufruf ausgegeben werden; wird trotzdem ein solcher Aufruf ausgegeben, wird DEF_HCONN zurückgegeben. Sie können die Verbindungskennung MQHC_DEF_HCONN für die nachfolgenden MQI-Aufrufe verwenden.
2. Die Warteschlange wird mit dem MQOPEN-Aufruf unter Angabe der Optionen MQOO_INPUT_SHARED und MQOO_BROWSE geöffnet. Bei Eingaben für diesen Aufruf verwendet das Programm die unter Schritt „1" auf Seite 1257 zurückgegebene Verbindungskennung. Für die Objektdeskriptorstruktur (MQOD) verwendet es für alle Felder die Standardwerte, mit Ausnahme des Felds mit dem Namen des Warteschlangenmanagers, der als Parameter an das Programm übergeben wird. Schlägt der MQOPEN-Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben und die Verarbeitung wird gestoppt.
3. Im Programm wird eine Schleife erstellt, indem MQGET-Aufrufe so lange ausgegeben werden, bis die erforderliche Anzahl an Nachrichten aus der Warteschlange abgerufen wurden. Schlägt einer dieser MQGET-Aufrufe fehl, wird die Schleife vorzeitig abgebrochen; es werden keine weiteren MQGET-Aufrufe ausgegeben und die Beendigungs- und Ursachencodes werden zurückgegeben. Für den MQGET-Aufruf werden die folgenden Optionen angegeben:

   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT oder MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST und MQGMO_BROWSE_NEXT

   Eine Beschreibung dieser Optionen finden Sie unter MQGET. Für jede Nachricht wird die Nachrichtennummer ausgegeben, gefolgt von der Länge der Nachricht und den Nachrichtendaten.
4. Die Warteschlange wird mit dem MQCLOSE-Aufruf unter Angabe der unter Schritt „2" auf Seite 1257 zurückgegebenen Objektkennung geschlossen. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben.
5. Die Verbindung zum Warteschlangenmanager wird mit dem MQDISC-Aufruf unter Angabe der unter Schritt „1" auf Seite 1257 zurückgegebenen Verbindungskennung getrennt. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben.

   Anmerkung: Bei einer Ausführung des Beispielprogramms in einer CICS-Umgebung muss kein MQDISC-Aufruf ausgegeben werden.

Hinweise zur Verwendung

- Um die Beispielprogramme einfach zu halten, gibt es einige minimale funktionale Abweichungen zwischen den verschiedenen Sprachenversionen. Durch Verwendung des in der Beispielausführungs-JCL (CSQ4BCJR und CSQ4BVJR) veranschaulichten Layouts für die Parameter können diese Unterschiede jedoch minimiert werden. Keiner dieser Unterschiede hat Auswirkungen auf die MQI.
- Mit CSQ4BCJ1 können Sie für die Anzahl der Nachrichten, die abgerufen werden sollen, mehr als 4 Ziffern eingeben.
- Nachrichten mit einer Länge von mehr als 64 KB werden abgeschnitten.
- CSQ4BCJ1 kann nur Zeichennachrichten korrekt anzeigen, da es den Inhalt nur bis zum ersten Nullzeichen (\0) anzeigt.
- Geben Sie im numerischen Feld für die Anzahl der Nachrichten eine Zahl zwischen 1 und 9999 ein. Hierbei sollte es sich um eine positive Zahl handeln. Soll beispielsweise nur eine Nachricht abgerufen werden, können Sie 1, 01, 001 oder 0001 eingeben. Bei Eingabe eines nicht numerischen oder negativen Werts erhalten Sie unter Umständen eine Fehlernachricht. Bei Eingabe von '-1' beispielsweise ruft das COBOL-Programm genau eine Nachricht ab, das C-Programm hingegen ruft keine Nachrichten ab.
- Geben Sie für beide Programme (CSQ4BCJ1 und CSQ4BVJ1) ein 'B' für den Get-Parameter (++GET++) ein, wenn die Nachrichten angezeigt werden sollen.
- Geben Sie für beide Programme (CSQ4BCJ1 und CSQ4BVJ1) ein 'S' für den Synchronisationspunktparameter (++SYNC++) ein, wenn Nachrichten synchronisationspunktgesteuert abgerufen werden sollen.

z/OS *Get-Beispielprogramme für die Stapelumgebung unter z/OS*

Um diese Beispielprogramme auszuführen, müssen Sie die Beispiel-JCL wie unter „Beispielanwendungen für die Stapelumgebung unter z/OS vorbereiten und ausführen” auf Seite 1244 beschrieben bearbeiten und ausführen.

Für das Programm können in einem EXEC PARM-Feld die folgenden Parameter angegeben werden; in C werden sie durch Leerstellen, in COBOL durch Kommas getrennt:

1. Der Name des Warteschlangenmanagers (4 Zeichen)
2. Der Name der Zielwarteschlange (48 Zeichen)
3. Die Anzahl der Nachrichten, die abgerufen werden sollen (bis zu 4 Ziffern)
4. Die Option zum Anzeigen/Abrufen von Nachrichten (1 Zeichen: B für 'Anzeigen' oder D für 'Abruf mit Löschen)
5. Die Synchronisationspunktsteuerung (1 Zeichen: S für 'synchronisationspunktgesteuert' oder N für 'nicht synchronisationspunktgesteuert)

Wird einer dieser Parameter falsch eingegeben, erhalten Sie entsprechende Fehlernachrichten.

Die Ausgabe von den Beispielprogrammen wird in die Datei SYSPRINT-Datei geschrieben:

```
==================================
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 *Get-Beispielprogramme für die CICS-Umgebung unter z/OS*

Lesen Sie diesen Abschnitt, wenn die Get-Beispielprogramme in der CICS-Umgebung ausgeführt werden sollen.

Für die Transaktionen können die folgenden Parameter in einem EXEC PARM-Feld (durch Kommas getrennt) angegeben werden:

1. Die Anzahl der Nachrichten, die abgerufen werden sollen (bis zu vier Ziffern)
2. Die Option zum Anzeigen/Abrufen von Nachrichten (1 Zeichen: B für 'Anzeigen' oder D für 'Abruf mit Löschen)
3. Die Synchronisationspunktsteuerung (1 Zeichen: S für 'synchronisationspunktgesteuert' oder N für 'nicht synchronisationspunktgesteuert)
4. Der Name der Zielwarteschlange (48 Zeichen)

Wird einer dieser Parameter falsch eingegeben, erhalten Sie entsprechende Fehlernachrichten.

Die COBOL-Version des Get-Beispielprogramms wird in der CICS-Umgebung durch die folgende Eingabe aufgerufen:

```
MVGT,9999,B,S,QUEUE.NAME
```

Die C-Version des Get-Beispielprogramms wird in der CICS-Umgebung durch die folgende Eingabe aufgerufen:

```
MCGT,9999,B,S,QUEUE.NAME
```

Bei ihrem Abruf aus der Warteschlange werden die Nachrichten in eine temporäre CICS-Speicherwarteschlange mit demselben Namen wie die CICS-Transaktion eingereiht (für das C-Beispielprogramm beispielsweise 'MCGT').

Hier eine Beispielausgabe der Get-Beispielprogramme:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE **********************
```

Hinweise zur Verwendung

- Um die Beispielprogramme einfach zu halten, gibt es einige minimale funktionale Abweichungen zwischen den verschiedenen Sprachenversionen. Keiner dieser Unterschiede hat Auswirkungen auf die MQI.
- Bei Eingabe eines Warteschlangennamens mit als als 48 Zeichen wird der Name bei 48 Zeichen abgeschnitten, es wird jedoch keine Fehlernachricht zurückgegeben.
- Drücken Sie vor Eingabe der Transaktion die Löschtaste.
- CSQ4CCJ1 kann nur Zeichennachrichten korrekt anzeigen, da es den Inhalt nur bis zum ersten Nullzeichen (\0) anzeigt.
- Geben Sie in dem numerischen Feld eine Zahl zwischen 1 und 9999 ein. Hierbei sollte es sich um eine positive Zahl handeln. Soll beispielsweise nur eine Nachricht abgerufen werden, können Sie 1, 01, 001 oder 0001 eingeben. Bei Eingabe eines nicht numerischen oder negativen Werts erhalten Sie unter Umständen eine Fehlernachricht.
- Nachrichten mit einer Länge von mehr als 24 526 Bytes (in C) bzw. 9 950 Bytes (in COBOL) werden abgeschnitten. Dies hängt damit zusammen, wie die temporären CICS-Speicherwarteschlangen verwendet werden.
- Geben Sie bei beiden Programmen (CSQ4CCK1 und CSQ4CVK1) ein 'B' für den Get-Parameter ein, wenn die Nachrichten angezeigt werden sollen; geben Sie andernfalls 'D' ein. Dadurch werden MQGET-Aufrufe

ausgeführt, mit denen die Nachrichten beim Abruf gelöscht werden. Bei Eingabe eines anderen Werts erhalten Sie eine Fehlernachricht.

- Geben Sie für beide Programme (CSQ4CCJ1 und CSQ4CVJ1) ein 'S' für den Synchronisationspunktparameter ein, wenn Nachrichten synchronisationspunktgesteuert abgerufen werden sollen. Wenn Sie für den Synchronisationspunktparameter ein 'N' eingeben, werden die MQGET-Aufrufe nicht synchronisationspunktgesteuert ausgegeben. Bei Eingabe eines anderen Werts erhalten Sie eine Fehlernachricht.

z/OS *Browse-Beispielprogramm unter z/OS*

Das Browse-Beispielprogramm ist eine Stapelanwendung, die das Anzeigen von Nachrichten in einer Warteschlange mithilfe des MQGET-Aufrufs veranschaulicht.

Die Anwendung durchläuft alle Nachrichten in einer Warteschlange, wobei sie jeweils die ersten 80 Bytes der einzelnen Nachrichten ausgibt. Mit dieser Anwendung können Sie Nachrichten in einer Warteschlange anzeigen, ohne sie zu ändern.

Die Quellenprogramme und die Beispielausführungs-JCL sind in COBOL, Assembler, PL/I und C bereitgestellt (siehe Tabelle 162 auf Seite 1245).

Um die Anwendung auszuführen, müssen Sie die Beispielausführungs-JCL wie unter „Beispielanwendungen für die Stapelumgebung unter z/OS vorbereiten und ausführen” auf Seite 1244 beschrieben bearbeiten und ausführen. Sie können die Nachrichten in einer Ihrer eigenen Warteschlangen anzeigen, indem Sie in der Ausführungs-JCL den Namen der Warteschlange angeben.

Sind in der Warteschlange bei der Ausführung der Anwendung Nachrichten enthalten, erhalten Sie eine Ausgabedatei ähnlich der folgenden:

```
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 **********
```

Wenn keine Nachrichten in der Warteschlange vorhanden sind, enthält die Datei nur die Überschriften und die Nachricht `End of report` (Ende des Berichts). Sind bei einem der MQI-Aufrufe Fehler aufgetreten, enthält die Ausgabedatei auch die Beendigungs- und Ursachencodes.

z/OS *Entwurf des Browse-Beispielprogramms unter z/OS*

Die Browse-Beispielanwendung verwendet ein einziges Programmmodul. Für jede der unterstützten Programmiersprachen wird ein Modul bereitgestellt.

Der Ablauf der Programmlogik ist wie folgt:

1. Eine Ausgabedatei wird geöffnet und die Titelzeile des Berichts wird ausgegeben. Es wird überprüft, ob von der Ausführungs-JCL der Name des Warteschlangenmanagers und der der Warteschlange übergeben wurden. Wurden beide Namen übergeben, werden die Berichtszeilen mit den Namen ausgegeben. Ist dies nicht der Fall, wird eine Fehlernachricht ausgegeben, die Ausgabedatei geschlossen und die Verarbeitung gestoppt.

   Wie das Programm die von der JCL übergebenen Parameter überprüft, hängt von der Programmiersprache ab, in der das Programm erstellt wurde; weitere Informationen finden Sie unter „Sprachenabhängige Designhinweise unter z/OS” auf Seite 1262.

2. Mit dem MQCONN-Aufruf wird eine Verbindung zum Warteschlangenmanager hergestellt. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben, die Ausgabedatei wird geschlossen und die Verarbeitung wird gestoppt.
3. Die Warteschlange wird mit dem MQOPEN-Aufruf unter Angabe der Option MQOO_BROWSE geöffnet. Bei Eingaben für diesen Aufruf verwendet das Programm die unter Schritt „2" auf Seite 1261 zurückgegebene Verbindungskennung. Für die Objektdeskriptorstruktur (MQOD) verwendet es für alle Felder die Standardwerte, mit Ausnahme desWarteschlangennamens (der unter Schritt „1" auf Seite 1260 übergeben wurde). Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben, die Ausgabedatei wird geschlossen und die Verarbeitung wird gestoppt.
4. Mit dem MQGET-Aufruf wird die erste Nachricht in der Warteschlange angezeigt. Als Eingabe für diesen Aufruf gibt das Programm Folgendes ein:
   - Die Verbindungs- und Warteschlangenkennungen aus den Schritten „2" auf Seite 1261 und „3" auf Seite 1261
   - Eine MQMD-Struktur mit allen Feldern, die auf ihre Anfangswerte gesetzt sind
   - Zwei Optionen:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - Einen Puffer mit einer Länge von 80 Bytes für die aus der Nachricht kopierten Daten

   Bei Angabe der Option MQGMO_ACCEPT_TRUNCATED_MSG kann der Aufruf auch dann abgeschlossen werden, wenn die Nachricht länger als der im Aufruf angegebene 80-Byte-Puffer ist. Überschreitet die Länge der Nachricht die des Puffers, wird die Nachricht entsprechend der Länge des Puffers abgeschnitten; darauf wird in den Beendigungs- und Ursachencodes hingewiesen. Ziel war es, einen übersichtlichen Bericht zu erhalten, daher wurde das Beispielprogramm so konzipiert, dass Nachrichten mit einer Länge von mehr als 80 Zeichen abgeschnitten werden. Die Puffergröße wird mit der Anweisung `DEFINE` gesetzt, sie kann also jederzeit bei Bedarf geändert werden.
5. Die folgende Schleife wird ausgeführt, bis der MQGET-Aufruf fehlschlägt:
   a. Eine Berichtszeile mit den folgenden Informationen wird ausgegeben:
      - Der Folgenummer der Nachricht (ein Zähler für die Browse-Operationen)
      - Der tatsächlichen Länge der Nachricht (nicht die abgeschnittene Länge); dieser Wert wird im Feld `DataLength` des MQGET-Aufrufs zurückgegeben
      - Den ersten 80 Bytes der Nachrichtendaten

   b. Die Felder `MsqId` und `CorrelId` der MQMD-Struktur werden auf Nullwerte zurückgesetzt.

   c. Die nächste Nachricht wird mit dem MQGET-Aufruf unter Angabe der folgenden beiden Optionen angezeigt:
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. Schlägt der MQGET-Aufruf fehl, wird anhand des Ursachencodes überprüft, ob der Aufruf fehlgeschlagen ist, weil der Anzeigecurser am Ende der Warteschlange angekommen ist. In diesem Fall wird die Nachricht `End of report` (Ende des Berichts) ausgegeben und mit Schritt „7" auf Seite 1261 fortgefahren; andernfalls werden die Beendigungs- und Ursachencodes ausgegeben, die Ausgabedatei geschlossen und die Verarbeitung gestoppt.
7. Die Warteschlange wird mit dem MQCLOSE-Aufruf unter Angabe der unter Schritt „3" auf Seite 1261 zurückgegebenen Objektkennung geschlossen.
8. Die Verbindung zum Warteschlangenmanager wird mit dem MQDISC-Aufruf unter Angabe der unter Schritt „2" auf Seite 1261 zurückgegebenen Verbindungskennung getrennt.
9. Die Ausgabedatei wird geschlossen und die Verarbeitung wird gestoppt.

z/OS *Sprachenabhängige Designhinweise unter z/OS*

Die Quellenmodule für das Browse-Beispielprogramm werden für vier Programmiersprachen bereitgestellt.

Die beiden wichtigsten Unterschiede zwischen den Quellenmodulen sind folgende:

- Beim Überprüfen der von der Ausführungs-JCL übergebenen Parameter suchen die Module in COBOL, PL/I und Assembler nach Kommazeichen (,). Wird `PARM=(,LOCALQ1)` von der JCL übergeben, versucht die Anwendung, die Warteschlange LOCALQ1 auf dem Standardwarteschlangenmanager zu öffnen. Folgt auf das Komma kein Name (oder ist keine Komma vorhanden), wird von der Anwendung eine Fehlernachricht zurückgegeben. Das Modul in C sucht nicht nach Kommazeichen. Wird von der JCL ein einzelner Parameter (beispielsweise `PARM='LOCALQ1'`) übergeben, verwendet das C-Modul die Warteschlange mit diesem Namen auf dem Standardwarteschlangenmanager.
- Um die Assembler-Module einfach zu halten, wird beim Erstellen des Ausgabeberichts das Datumsformat *jj/ttt* (z. B. 05/116) verwendet. Die anderen Module verwenden das Kalenderdatum im Format *mm/tt/jj*.

z/OS *Beispielprogramm für die Nachrichtenausgabe unter z/OS*

Das Beispielprogramm für die Nachrichtenausgabe ist eine Stapelanwendung, die das Entfernen aller Nachrichten aus einer Warteschlange mithilfe des MQGET-Aufrufs veranschaulicht.

Das Beispielprogramm für die Nachrichtenausgabe verwendet drei Parameter:

1. Der Name des Warteschlangenmanagers.
2. Den Namen der Quellenwarteschlange
3. Einen optionalen Parameter für Eigenschaften

Darüber hinaus gibt es für jede Nachricht die Felder des Nachrichtendeskriptors aus, gefolgt von den Nachrichtendaten. Das Programm gibt die Daten sowohl im Hexadezimalformat als auch im Zeichenformat (wenn sie druckbar) aus. Ist ein Zeichen nicht druckbar, wird es vom Programm durch einen Punkt (.) ersetzt. Dieses Programm können Sie für die Fehlerdiagnose einer Anwendung verwenden, die Nachrichten in eine Warteschlange einreiht.

Folgende Werte sind für den Eigenschaftenparameter zulässig:

Tabelle 174. Zulässige Werte für den Eigenschaftenparameter

Wert	Verhalten
0	Standardverhalten, wie in IBM WebSphere MQ 6. 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. Beispiel:
	`****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.

Tabelle 174. Zulässige Werte für den Eigenschaftenparameter (Forts.)	
Wert	**Verhalten**
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 WebSphere MQ 6-Eigenschaft enthalten ist. Andernfalls werden die Eigenschaften gelöscht.

Sie können die Anwendung so ändern, dass Nachrichten in der Warteschlange nur angezeigt, nicht abgerufen werden. Dazu müssen Sie sie mit der Option '-DBROWSE' kompilieren, um wie unter „Entwurf des Beispielprogramms für die Nachrichtenausgabe unter z/OS” auf Seite 1264 beschrieben das BROWSE-Makro zu definieren. Ausführbarer Code ist in der Bibliothek SCSQLOAD bereitgestellt. Das Modul CSQ4BCG0 wird unter Angabe der Option -DBROWSE erstellt; das Modul CSQ4BCG1 liest die Nachrichten in der Warteschlange und löscht sie.

Die Anwendung hat ein einziges Quellenprogramm, das in C erstellt ist. Außerdem wird auch Code einer Beispielausführungs-JCL bereitgestellt (siehe Tabelle 163 auf Seite 1246).

Um die Anwendung auszuführen, müssen Sie die Beispielausführungs-JCL wie unter „Beispielanwendungen für die Stapelumgebung unter z/OS vorbereiten und ausführen” auf Seite 1244 beschrieben bearbeiten und ausführen. Sind in der Warteschlange bei der Ausführung der Anwendung Nachrichten enthalten, erhalten Sie eine Ausgabedatei ähnlich der in Abbildung 151 auf Seite 1264.

```
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
```

Abbildung 151. Beispiel für einen Bericht von der Beispielanwendung für die Nachrichtenausgabe

z/OS *Entwurf des Beispielprogramms für die Nachrichtenausgabe unter z/OS*

Die Beispielanwendung für die Nachrichtenausgabe verwendet ein einfaches, in der Programmiersprache C geschriebenes Programm.

Der Ablauf der Programmlogik ist wie folgt:

1. Es wird überprüft, ob von der Ausführungs-JCL der Name des Warteschlangenmanagers und der der Warteschlange übergeben wurden. Ist dies nicht der Fall, wird eine Fehlernachricht ausgegeben und die Verarbeitung wird gestoppt.
2. Mit dem MQCONN-Aufruf wird eine Verbindung zum Warteschlangenmanager hergestellt. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben und die Verarbeitung wird gestoppt; andernfalls wird der Name des Warteschlangenmanagers ausgegeben.
3. Die Warteschlange wird mit dem MQOPEN-Aufruf unter Angabe der Option MQOO_INPUT_SHARED geöffnet.

Anmerkung: Wenn die Anwendung die Nachrichten aus der Warteschlange nicht entfernen, sondern nur anzeigen soll, müssen Sie das Beispielprogramm mit '-DBROWSE' kompilieren oder `#define BROWSE` am Anfang der Quelle hinzufügen. Der Makrovorprozessor fügt diese Zeile dann dem Programm hinzu, das bei der Kompilierung die Option MQOO_BROWSE auswählt.

Bei Eingaben für diesen Aufruf verwendet das Programm die unter Schritt „2" auf Seite 1264 zurückgegebene Verbindungskennung. Für die Objektdeskriptorstruktur (MQOD) verwendet es für alle Felder die Standardwerte, mit Ausnahme des Warteschlangennamens (der unter Schritt „1" auf Seite 1264 übergeben wurde). Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben und die Verarbeitung wird gestoppt; andernfalls wird der Name der Warteschlange ausgegeben.

4. Wenn Sie die Nachrichteneigenschaften über eine Nachrichtenkennung abrufen, müssen Sie diese Kennung mit dem MQCRTMH-Aufruf erstellen; sie wird in den nachfolgenden MQGET-Aufrufen verwendet. Schlägt dieser Aufruf fehl, werden die Beendigungs- und Ursachencodes ausgegeben und de Verarbeitung wird gestoppt.
5. Die Optionen zum Abrufen von Nachrichten werden entsprechend der jeweiligen Anforderungsaktion für eine Nachrichteneigenschaft gesetzt.
6. Die folgende Schleife wird ausgeführt, bis der MQGET-Aufruf fehlschlägt:
   a. Die Puffer werden auf Leerzeichen gesetzt, damit die Nachrichtendaten nicht durch im Puffer eventuell bereits vorhandene Nachrichten beschädigt werden.
   b. Die Felder `MsgId` und `CorrelId` der MQMD-Struktur werden auf Nullwerte gesetzt, sodass der MQGET-Aufruf die erste Nachricht in der Warteschlange auswählt.
   c. Mit dem MQGET-Aufruf wird eine Nachricht aus der Warteschlange abgerufen. Als Eingabe für diesen Aufruf gibt das Programm Folgendes ein:
      - Die Verbindungs- und Warteschlangenkennungen aus den Schritten „2" auf Seite 1264 und „3" auf Seite 1264
      - Eine MQMD-Struktur, in der alle Felder auf ihre Anfangswerte gesetzt sind (`MsgId` und `CorrelId` werden bei jedem MQGET-Aufruf auf Nullwerte zurückgesetzt)
      - Die Option MQGMO_NO_WAIT

        Anmerkung: Wenn die Anwendung die Nachrichten aus der Warteschlange nicht entfernen, sondern nur anzeigen soll, müssen Sie das Beispielprogramm mit '-DBROWSE' kompilieren oder `#define BROWSE` am Anfang der Quelle hinzufügen. Der Makrovorprozessor fügt diese Zeile dann dem Programm hinzu, das bei der Kompilierung die Option MQGMO_BROWSE_NEXT auswählt. Wird diese Option in einem Aufruf für eine Warteschlange verwendet, für die zuvor kein Anzeigecursor mit der aktuellen Objektkennung verwendet wurde, wird der Anzeigecursor logisch vor die erste Nachricht positioniert.
      - Einen Puffer mit einer Länge von 64 KB für die aus der Nachricht kopierten Daten
   d. Die Subroutine 'printMD' wird aufgerufen. Sie gibt die Namen der einzelnen Felder im Nachrichtendeskriptor aus, gefolgt vom Inhalt dieser Felder.
   e. Wurde unter Schritt „4" auf Seite 1265 eine Nachrichtenkennung erstellt, wird die Subroutine 'printProperties' zur Anzeige der vorhandenen Nachrichteneigenschaften aufgerufen.
   f. Die Länge der Nachricht wird ausgegeben, gefolgt von den Nachrichtendaten. Die einzelnen Zeilen mit Nachrichtendaten haben das folgende Format:
      - Relative Position (im Hexadezimalformat) dieser Daten
      - 16 Bytes Hexadezimaldaten
      - Dieselben 16 Datenbytes im Zeichenformat, sofern die Zeichen druckbar sind (nicht druckbare Zeichen werden durch Punkte ersetzt)
7. Schlägt der MQGET-Aufruf fehl, wird anhand des Ursachencodes überprüft, ob der Aufruf fehlgeschlagen ist, weil in der Warteschlange keine Nachrichten mehr enthalten sind. In diesem Fall wird die Nachricht angezeigt, dass keine Nachrichten mehr vorhanden sind; andernfalls werden die Been-

digungs- und Ursachencodes ausgegeben. In beiden Fällen wird der Vorgang mit Schritt „9” auf Seite 1266 fortgesetzt.

Anmerkung: Der MQGET-Aufruf schlägt fehl, wenn eine Nachricht Daten mit einer Länge von mehr als 64 KB enthält. Wenn das Programm in der Lage sein soll, größere Nachrichten zu verarbeiten, haben Sie folgende Möglichkeiten:

- Fügen Sie dem MQGET-Aufruf die Option MQGMO_ACCEPT_TRUNCATED_MSG hinzu, sodass die ersten 64 KB an Daten abgerufen und die restlichen Daten verworfen werden
- Definieren Sie das Programm so, dass es Nachrichten mit der angegebenen Datenlänge in der Warteschlange belässt
- Erhöhen Sie den Längenwert des Puffers

8. Wenn unter Schritt „4” auf Seite 1265 eine Nachrichtenkennung erstellt wurde, wird MQDLTMH aufgerufen, um diese zu löschen.
9. Die Warteschlange wird mit dem MQCLOSE-Aufruf unter Angabe der unter Schritt „3” auf Seite 1264 zurückgegebenen Objektkennung geschlossen.
10. Die Verbindung zum Warteschlangenmanager wird mit dem MQDISC-Aufruf unter Angabe der unter Schritt „2” auf Seite 1264 zurückgegebenen Verbindungskennung getrennt.

z/OS *Beispielprogramm 'Queue Attributes' für z/OS*

Das Beispielprogramm für Warteschlangenattribute ist eine dialoggesteuerte CICS-Anwendung, die die Verwendung der MQINQ- und MQSET-Aufrufe veranschaulicht.

Dieses Beispielprogramm veranschaulicht, wie die Werte der Warteschlangenattribute **InhibitPut** und **InhibitGet** abgefragt und so geändert werden, dass Programme keine Nachrichten mehr in eine Warteschlangen einreihen bzw. aus ihr abrufen können. Dies ist eine Möglichkeit, eine Warteschlange zum Testen eines Programms zu *sperren*.

Um eine versehentliche Sperre Ihrer eigenen Warteschlangen zu verhindern, ist dieses Beispielprogramm für Warteschlangenobjekte konzipiert, deren ersten acht Byte im Namen aus den Zeichen CSQ4SAMP bestehen. Der Quellcode enthält jedoch Kommentare, die zeigen, wie diese Einschränkung aufgehoben werden kann.

Die Quellenprogramme werden in COBOL, Assembler und C bereitgestellt (siehe Tabelle 169 auf Seite 1251).

Die Assembler-Version des Beispielprogramms verwendet wiedereintrittsfähigen Code. Sie können dies daran feststellen, dass der Code der einzelnen MQI-Aufrufe in dieser Version des Beispielprogramms das Schlüsselwort MF enthält; Beispiel:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(Das Schlüsselwort VL gibt an, dass Sie die bereitgestellte CEDF-Transaktion (CICS Execution Diagnostic Facility) für ein Debugging des Programms verwenden können. Weitere Informationen zum Erstellen simultan verwendbarer Programme finden Sie unter Coderung in der System/390-Assemblersprache.

Um die Anwendung zu starten, starten Sie das CICS-System und verwenden Sie die folgenden CICS-Transaktionen:

- Für COBOL: MVC1
- Für Assembler: MAC1
- Für C: MCC1

Sie können die Namen dieser Transaktionen über die unter Schritt 3 aufgeführte CSD-Datei ändern.

Design des Beispielprogramms

Beim Start des Beispielprogramms wird eine Maske mit Feldern für die folgenden Eingaben geöffnet:

- Den Namen der Warteschlange

- Die Benutzeranforderung (gültige Aktionen sind 'inquire', 'allow' oder 'inhibit')
- Den aktuellen Status der Put-Operationen für die Warteschlange
- Den aktuellen Status der Get-Operationen für die Warteschlange

Die beiden ersten Felder werden vom Benutzer ausgefüllt, die beiden letzten Felder werden von Anwendung ausgefüllt und enthalten die Angabe `INHIBITED` oder `ALLOWED`.

Die Werte, die Sie in den ersten beiden Feldern angeben, werden von der Anwendung ausgewertet. Sie überprüft, ob der Name der Warteschlange mit den Zeichen `CSQ4SAMP` beginnt und ob Sie im Aktionsfeld eine der drei zulässigen Anforderungen eingegeben haben. Ebenso konvertiert die Anwendung Ihre Eingaben in Großbuchstaben; daher dürfen Sie keine Warteschlangen verwenden, deren Name Kleinbuchstaben enthält.

Bei Angabe von `inquire` im Feld **Action** ist der Ablauf der Programmlogik wie folgt:

1. Die Warteschlange wird mit dem MQOPEN-Aufruf unter Angabe der Option MQOO_INQUIRE geöffnet.
2. MQINQ wird mit den Selektoren MQIA_INHIBIT_GET und MQIA_INHIBIT_PUT aufgerufen.
3. Die Warteschlange wird mit dem MQCLOSE-Aufruf geschlossen.
4. Die im Parameter **`IntAttrs`** des MQINQ-Aufrufs zurückgegebenen Attribute werden analysiert und je nach Einstellung wird der Wert INHIBITED oder der Wert ALLOWED in die entsprechenden Maskenfelder eingefügt.

Bei Angabe von `inhibit` im Feld **Action** ist der Ablauf der Programmlogik wie folgt:

1. Die Warteschlange wird mit dem MQOPEN-Aufruf unter Angabe der Option MQOO_SET geöffnet.
2. MQSET wird mit den Selektoren MQIA_INHIBIT_GET und MQIA_INHIBIT_PUT und mit den Werten MQQA_GET_INHIBITED und MQQA_PUT_INHIBITED im Parameter **`IntAttrs`** aufgerufen.
3. Die Warteschlange wird mit dem MQCLOSE-Aufruf geschlossen.
4. Der Wert INHIBITED wird in die entsprechenden Maskenfelder eingefügt.

Bei Angabe von `allow` im Feld **Action** ist die Programmlogik ähnlich der einer 'inhibit'-Anforderung. Die einzigen Unterschiede sind die Einstellungen der Attribute und die in der Maske angezeigten Werte.

Beim Öffnen der Warteschlange verwendet die Anwendung die Standardverbindungskennung für den Warteschlangenmanager. (CICS stellt beim Start des CICS-Systems eine Verbindung zu dem Warteschlangenmanager her.) Die Anwendung kann an diesem Punkt die folgenden Fehler abfangen:

- Die Anwendung ist nicht mit dem Warteschlangenmanager verbunden.
- Die Warteschlange ist nicht vorhanden.
- Der Benutzer hat keine Zugriffsberechtigung für die Warteschlange.
- Die Anwendung ist nicht zum Öffnen der Warteschlange berechtigt.

Bei allen anderen MQI-Fehlern werden von der Anwendung die Beendigungs- und Ursachencodes angezeigt.

z/OS *Mail-Manager-Beispielprogramm unter z/OS*

Bei der Mail-Manager-Beispielanwendung handelt es sich um eine Reihe von Programmen, die das Senden und Empfangen von Nachrichten innerhalb einer Umgebung sowie über verschiedene Umgebungen hinweg veranschaulichen. Die Anwendung ist ein einfaches elektronische Mailsystem, mit dem Benutzer auch bei Verwendung unterschiedlicher Warteschlangenmanager Nachrichten austauschen können.

Die Anwendung zeigt, wie Warteschlangen mithilfe des MQOPEN-Aufrufs erstellt werden und indem IBM MQ for z/OS-Befehle in die Eingabewarteschlange für Systembefehle eingereiht werden.

Die Anwendung wird in drei Versionen bereitgestellt:

- Als CICS-Anwendung in COBOL
- Als TSO-Anwendung in COBOL
- Als TSO-Anwendung in C

z/OS *Beispielprogramm 'Mail Manager' für z/OS vorbereiten*

Das Mail-Manager-Beispielprogramm wird für die Ausführung in zwei verschiedenen Umgebungen bereitgestellt. Die Vorbereitungen vor Ausführung der Anwendung hängen von der Umgebung ab, die verwendet werden soll.

Benutzer können von TSO und CICS aus auf Mailwarteschlangen und Kurznamenwarteschlangen zugreifen, sofern auf beiden Systemen dieselben Benutzer-IDs verwendet werden.

Bevor Nachrichten an einen anderen Warteschlangenmanager gesendet werden können, müssen Sie zunächst einen Nachrichtenkanal zu diesem Warteschlangenmanager einrichten. Dies erfolgt mithilfe der unter Channel control function beschriebenen Kanalsteuerfunktion von IBM MQ.

Beispielprogramm für die TSO-Umgebung vorbereiten

Führen Sie folgende Schritte aus:

1. Bereiten Sie das Beispielprogramm wie unter „Beispielanwendungen für die TSO-Umgebung unter z/OS vorbereiten” auf Seite 1248 beschrieben vor.
2. Passen Sie die für das Beispielprogramm bereitgestellte CLIST an, um Folgendes zu definieren:
   - Den Pfad mit den Anzeigen
   - Den Pfad mit der Nachrichtendatei
   - Den Pfad mit den Lademodulen
   - Den Namen des Warteschlangenmanagers, der zusammen mit der Anwendung verwendet werden soll

   Für jede Version des Beispielprogramms wird eine eigene CLIST bereitgestellt.
   - Für die COBOL-Version: CSQ4RVD1
   - Für die C-Version: CSQ4RCD1
3. Stellen Sie sicher, dass die von der Anwendung verwendeten Warteschlangen auf dem Warteschlangenmanager verfügbar sind (die Warteschlangen sind in CSQ4CVD definiert).

Anmerkung: Multitasking mit ISPF wird von VS COBOL II nicht unterstützt. Sie können daher die Mail-Manager-Beispielanwendung nicht in beiden Seiten einer geteilten Anzeige verwenden, da dies ein unvorhersehbares Ergebnis zur Folge hätte.

z/OS *Mail-Manager-Beispielprogramm unter z/OS ausführen*

Um das Beispielprogramm in CICS Transaction Server for z/OS zu starten, müssen Sie die Transaktion MAIL ausführen. Wenn Sie noch nicht bei CICS angemeldet sind, werden Sie von derAnwendung zur Eingabe einer Benutzer-ID aufgefordert, an die sie Ihre Mails senden kann.

Beim Starten der Anwendung wird Ihre Mailwarteschlange geöffnet. Ist diese Warteschlange noch nicht vorhanden, wird sie für Sie von der Anwendung erstellt. Die Namen der Mailwarteschlangen haben das Format CSQ4SAMP.MAILMGR. *Benutzer-ID*, wobei der Wert *Benutzer-ID* von der jeweiligen Umgebung abhängt:

In TSO:
Die TSO-ID des Benutzers

In CICS
Die Anmelde-ID für CICS des Benutzers oder die Benutzer-ID, die vom Benutzer bei einer entsprechenden Aufforderung beim Start des Mail-Managers eingegeben wurde

Die Namen aller Warteschlangen, die vom Mail-Manager verwendet werden, müssen durchgängig aus Großbuchstaben bestehen.

Von der Anwendung wird daraufhin eine Menüanzeige mit den folgenden Optionen geöffnet:

- Read incoming mail (Eingehende Mail lesen)
- Mail senden

- Kurznamen erstellen

In der Menüanzeige wird außerdem angezeigt, wie viele Nachrichten in der Mailwarteschlange anstehen. Über jede dieser Menüoptionen wird eine weitere Anzeige geöffnet:

Read incoming mail (Eingehende Mail lesen)
Der Mail-Manager zeigt eine Liste der Nachrichten an, die in Ihrer Mailwarteschlange enthalten sind (es werden nur die ersten 99 Nachrichten in der Warteschlange angezeigt). Ein Beispiel dieser Anzeige finden Sie in Abbildung 154 auf Seite 1272. Bei Auswahl einer Nachricht in dieser Liste wird der Inhalt dieser Nachricht angezeigt (siehe Abbildung 155 auf Seite 1273).

Mail senden
In einer Anzeige werden Sie aufgefordert, Folgendes einzugeben:

- Den Namen des Benutzers, dem Sie eine Nachricht senden möchten
- Den Namen des Warteschlangenmanagers, zu dem die Mailwarteschlange des Benutzers gehört
- Den Nachrichtentext

Geben Sie im Benutzernamensfeld eine Benutzer-ID oder einen mit dem Mail-Manager erstellten Kurznamen ein. Das Feld für den Namen des Warteschlangenmanagers kann leer bleiben, wenn die Mailwarteschlange des Benutzers zu dem von Ihnen verwendeten Warteschlangenmanager gehört; wenn Sie im Benutzernamensfeld einen Kurznamen eingegeben haben, muss das Feld für den Namen des Warteschlangenmanagers leer bleiben:

- Wenn Sie nur einen Benutzernamen angeben, geht das Programm zunächst davon aus, dass es sich bei dem Namen um einen Kurznamen handelt, und sendet die Nachricht an das über den Namen definierte Objekt. Ist kein solcher Kurzname vorhanden, versucht das Programm, die Nachricht an eine lokale Warteschlange dieses Namens zu senden.
- Wenn Sie sowohl einen Benutzernamen als auch den Namen eines Warteschlangenmanagers angeben, sendet das Programm die Nachricht an die über diese beiden Namen angegebene Mailwarteschlange.

Soll beispielsweise eine Nachricht an den Benutzer JONESM auf dem fernen Warteschlangenmanager QM12 gesendet werden, haben Sie zwei Möglichkeiten:

- Sie können in beiden Feldern den Benutzer JONESM auf dem Warteschlangenmanager QM12 angeben.
- Sie können eine Kurznamen (beispielsweise MARY) für den Benutzer definieren und eine Nachricht an diesen Benutzer senden, indem Sie im Benutzernamensfeld MARY eingeben und das Feld für den Namen des Warteschlangenmanagers leer lassen.

Kurznamen erstellen
Sie können einen eingängigen Namen für einen Benutzer definieren, mit dem Sie häufig Nachrichten austauschen. Sie werden aufgefordert, die Benutzer-ID und den Warteschlangenmanager, zu dem die Mailwarteschlange gehört, einzugeben.

Kurznamen sind Warteschlangen, deren Namen das Format CSQ4SAMP.MAILMGR. *Benutzer-ID.Kurzname* haben, wobei *Benutzer-ID* Ihre eigene Benutzer-ID und *Kurzname* der von Ihnen gewünschte Kurzname ist. Mit diesem Namensformat kann jeder Benutzer seine eigenen Kurznamen erstellen.

Welcher Warteschlangentyp vom Programm geöffnet wird, hängt davon ab, wie Sie die Felder der Anzeige 'Create Nickname' (Kurznamen erstellen) ausfüllen:

- Wenn Sie nur einen Benutzernamen angeben oder wenn es sich bei dem Namen des Warteschlangenmanagers um denselben Warteschlangenmanager handelt, mit dem der Mail-Manager verbunden ist, erstellt das Programm eine Aliaswarteschlange.
- Wenn Sie sowohl einen Benutzernamen als auch den Namen eines Warteschlangenmanagers angeben (bei dem es sich nicht um den Warteschlangenmanager handelt, mit dem der Mail-Manager verbunden ist), erstellt das Programm eine lokale Definition einer fernen Warteschlange. Das Programm überprüft nicht, ob die Warteschlange, in die die Definition aufgelöst wird, auch tatsächlich vorhanden ist; es wird nicht einmal überprüft, ob der ferne Warteschlangenmanager vorhanden ist.

Wenn beispielsweise Ihre eigene Benutzer-ID SMITHK ist und Sie für den Benutzer JONESM (der den fernen Warteschlangenmanager QM12 verwendet) den Kurznamen MARY erstellen, dann erstellt das Kurznamenprogramm eine lokale Definition für eine ferne Warteschlange des Namens CSQ4SAMP.MAILMGR.SMITHK.MARY. Diese Definition wird in die Mailwarteschlange CSQ4SAMP.MAILMGR.JONESM (auf dem Warteschlangenmanager QM12) von Mary aufgelöst. Wenn auch Sie den Warteschlangenmanager QM12 verwenden, erstellt das Programm stattdessen eine Aliaswarteschlange desselben Namens (CSQ4SAMP.MAILMGR.SMITHK.MARY).

Die C-Version der TSO-Anwendung macht intensiveren Gebrauch von den Funktionen für die Nachrichtenbehandlung von ISPF als die COBOL-Version. Sie werden unter Umständen feststellen, dass von den C- und COBOL-Versionen unterschiedliche Fehlernachrichten angezeigt werden.

z/OS *Entwurf des Mail-Manager-Beispielprogramms unter z/OS*

In den folgenden Abschnitten sind alle Programme beschrieben, aus denen sich die Mail-Manager-Beispielanwendung zusammensetzt.

Das Zusammenspiel der Programme und die von der Anwendung verwendeten Anzeigen sind für die TSO-Version in Abbildung 152 auf Seite 1270, für die Version für CICS Transaction Server for z/OS in Abbildung 153 auf Seite 1271 veranschaulicht.

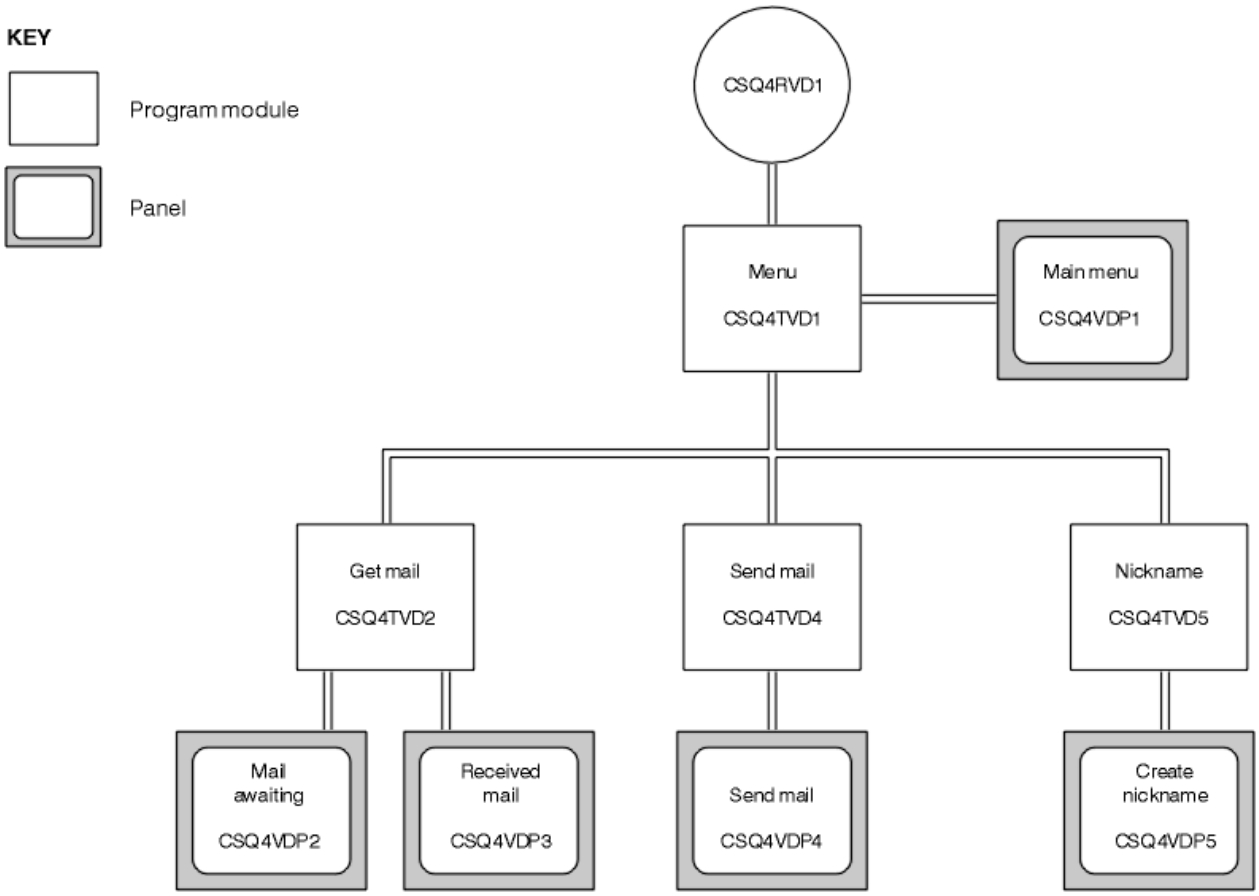


Abbildung 152. Programme und Anzeigen für die TSO-Versionen des Mail-Manager-Beispielprogramms

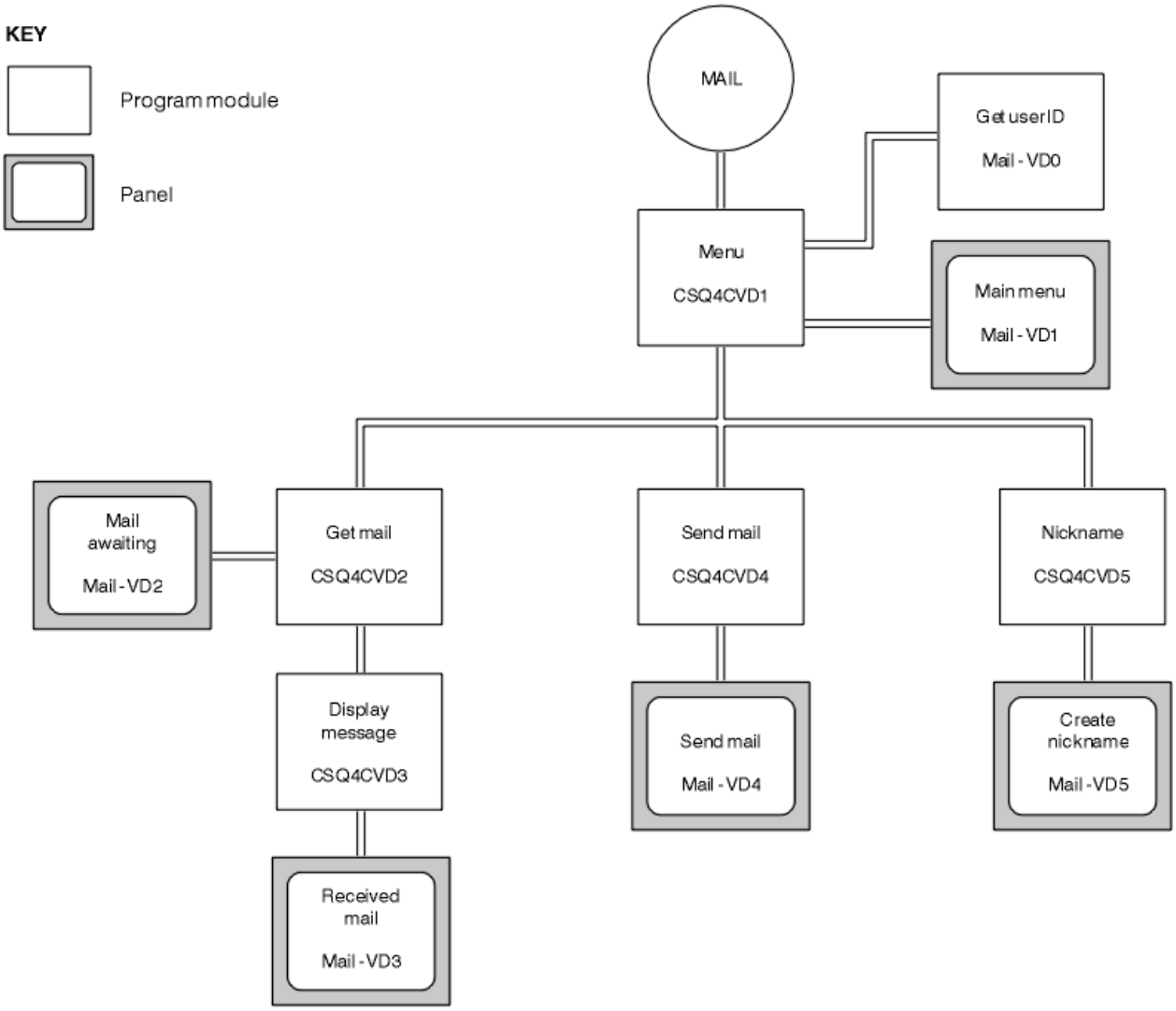


Abbildung 153. Programme und Anzeigen für die CICS-Version des Mail-Manager-Beispielprogramms

z/OS *Menüprogramm unter z/OS*

In der TSO-Umgebung wird das Menüprogramm von der CLIST aufgerufen, In der CICS-Umgebung wird das Programm durch die Transaktion MAIL aufgerufen.

Das Menüprogramm (CSQ4TVD1 für TSO, CSQ4CVD1 für CICS) ist das Startprogramm der Programmsuite. Es zeigt das Menü (CSQ4VDP1 für TSO, VD1 für CICS) an und ruft die anderen Programme auf, wenn sie im Menü ausgewählt werden.

Das Programm ruft zunächst die Benutzer-ID ab:

- In der CICS-Version des Programms wird die Benutzer-ID mit dem CICS-Befehl ASSIGN USERID abgerufen, wenn der Benutzer bei CICS angemeldet ist. Ist der Benutzer noch nicht angemeldet, wird vom Programm die Anmeldeanzeige (CSQ4VD0) angezeigt, in der der Benutzer zur Eingabe einer Benutzer-ID aufgefordert wird. In diesem Programm findet keine Sicherheitsverarbeitung statt, d. h., der Benutzer kann eine beliebige Benutzer-ID angeben.
- In der TSO-Version wird die ID des Benutzers aus TSO in der CLIST abgerufen und als eine Variable im gemeinsamen ISPF-Pool an das Menüprogramm übergeben.

Nachdem das Programm die Benutzer-ID abgerufen hat, überprüft es, ob der Benutzer über eine Mailwarteschlange (CSQ4SAMP.MAILMGR. *Benutzer-ID*) verfügt. Ist keine Mailwarteschlange vorhanden, wird vom Programm eine solche Warteschlange erstellt, indem eine Nachricht in die Eingabewarteschlange für Systembefehle eingereiht wird. Diese Nachricht enthält den IBM MQ for z/OS-Befehl DEFINE QLOCAL.

Die von diesem Befehl verwendete Objektdefinition legt die maximale Länge der Warteschlange auf 9999 Nachrichten fest.

Darüber hinaus erstellt das Programm auch eine temporäre dynamische Warteschlange für Antworten von der Eingabewarteschlange für Systembefehle. Dazu verwendet das Programm den MQOPEN-Aufruf unter Angabe von SYSTEM.DEFAULT.MODEL.QUEUE als Vorlage für die dynamische Warteschlange. Der Warteschlangenmanager erstellt die temporäre dynamische Warteschlange, deren Name das Präfix CSQ4SAMP hat; der Rest des Namens wird vom Warteschlangenmanager generiert.

Anschließend öffnet das Programm die Mailwarteschlange des Benutzers und ermittelt die Anzahl der in der Warteschlange enthaltenen Nachrichten, indem es die aktuelle Länge der Warteschlange abfragt. Dazu gibt das Programm den MQINQ-Aufruf unter Angabe des Selektors MQIA_CURRENT_Q_DEPTH aus.

Das Programm führt anschließend eine Schleife aus, mit der das Menü angezeigt und die vom Benutzer getroffene Auswahl verarbeitet wird. Die Schleife wird gestoppt, wenn der Benutzer die Taste PF3 drückt. Bei einer zulässigen Auswahl wird das entsprechende Programm gestartet, andernfalls wird eine Fehlernachricht angezeigt.

z/OS *Programme zum Abrufen von Mails und zum Anzeigen von Nachrichten unter z/OS*

In der TSO-Version der Anwendung werden die Funktionen zum Abrufen von Mails und zum Anzeigen von Nachrichten von demselben Programm (CSQ4TVD2) ausgeführt. In der CICS-Version der Anwendung werden diese Funktionen jeweils von einem eigenen Programm ausgeführt (CSQ4CVD2 und CSQ4CVD3).

In der Anzeige mit anstehenden Nachrichten (CSQ4VDP2 für TSO, VD2 für CICS; ein Beispiel sehen Sie in Abbildung 154 auf Seite 1272) werden alle Nachrichten angezeigt, die sich in der Mailwarteschlange des Benutzers befinden. Das Programm erstellt diese Liste mithilfe des MQGET-Aufrufs, mit dem alle Nachrichten in der Warteschlange durchsucht und Informationen zu den einzelnen Nachrichten gespeichert werden. Zusätzlich zu den angezeigten Informationen werden vom Programm die Nachrichten-ID (`MsgId`) und die Korrelations-ID (`CorrelId`) der einzelnen Nachrichten aufgezeichnet.

```
---------------------- 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
```

Abbildung 154. Beispiel einer Anzeige mit einer Liste anstehender Nachrichten

In der Anzeige für anstehende Nachrichten kann der Benutzer eine Nachricht auswählen und ihren Inhalt anzeigen (ein Beispiel sehen Sie in Abbildung 155 auf Seite 1273). Das Programm entfernt diese Nachricht mit dem MQGET-Aufruf aus der Warteschlange und verwendet dabei die Werte für `MsgId` und `CorrelId`, die es beim Durchsuchen aller Nachrichten aufgezeichnet hat. Dieser MQGET-Aufruf wird unter Angabe der Option MQGMO_SYNCPOINT ausgeführt. Das Programm zeigt den Inhalt der Nachricht an und erstellt dann einen Synchronisationspunkt; der MQGET-Aufruf wird damit festgeschrieben, sodass die Nachricht nicht mehr vorhanden ist.

```
--------------------- 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                          |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
'-------------------------------------------------------------------------'
```

Abbildung 155. Beispiel einer Anzeige mit dem Inhalt einer Nachricht

Die vom Mail-Manager bereitgestellte Funktion kann dahingehend erweitert werden, dass der Benutzer wählen kann, ob die Nachricht nach Anzeige ihres Inhalts weiterhin in der Warteschlange verbleiben soll. Dazu muss der MQGET-Aufruf, der die Nachricht aus der Warteschlange entfernt, nach Anzeige der Nachricht zurückgesetzt werden.

z/OS *Programm zum Senden von Mails unter z/OS*

Nachdem der Benutzer die Anzeige zum Senden von Mails (CSQ4VDP4 für TSO, VD4 für CICS) ausgefüllt hat, reiht das Programm zum Senden von Mails (CSQ4TVD4 für TSO, CSQ4CVD4 für CICS) die Nachricht in die Mailwarteschlange des Empfängers ein.

Dazu gibt das Programm den MQPUT1-Aufruf aus. Das Ziel der Nachricht hängt davon ab, wie der Benutzer die Felder in der Anzeige zum Senden von Mails ausgefüllt hat:

- Wenn der Benutzer nur einen Benutzernamen angegeben hat, geht das Programm zunächst davon aus, dass es sich bei dem Namen um einen Kurznamen handelt, und sendet die Nachricht an das über den Namen definierte Objekt. Ist kein solcher Kurzname vorhanden, versucht das Programm, die Nachricht an eine lokale Warteschlange dieses Namens zu senden.
- Wenn der Benutzer sowohl einen Benutzernamen als auch den Namen eines Warteschlangenmanagers angegeben hat, sendet das Programm die Nachricht an die über diese beiden Namen angegebene Mailwarteschlange.

Leere Nachrichten werden vom Programm nicht akzeptiert; es entfernt alle führenden Leerstellen aus den einzelnen Zeilen des Nachrichtentextes.

War der MQPUT1-Aufruf erfolgreich, zeigt das Programm eine Nachricht mit dem Benutzernamen und dem Namen des Warteschlangenmanagers an, in den die Nachricht eingereiht wurde. Ist der Aufruf fehlgeschlagen, prüft das Programm explizit, ob Ursachencodes vorhanden sind, die darauf hinweisen, dass die Warteschlange oder der Warteschlangenmanager nicht vorhanden sind (Ursachencodes MQRC_UNKNOWN_OBJECT_NAME und MQRC_UNKNOWN_OBJECT_Q_MGR). Das Programm zeigt für jeden dieser Fehler seine eigene Fehlernachricht an; bei anderen Fehlern zeigt das Programm die vom Aufruf zurückgegebenen Beendigungs- und Ursachencodes an.

z/OS *Programm für Kurznamen unter z/OS*

Wenn der Benutzer einen Kurznamen definiert, erstellt das Kurznamenprogramm (CSQ4TVD5 für TSO, CSQ4CVD5 für CICS) eine Warteschlange, deren Name als Bestandteil diesen Kurznamen enthält.

Dazu reiht das Programm eine Nachricht in die Eingabewarteschlange für Systembefehle ein. Diese Nachricht enthält den IBM MQ for z/OS-Befehl DEFINE QALIAS oder DEFINE QREMOTE. Welcher Warteschlangentyp vom Programm erstellt wird, hängt davon ab, wie der Benutzer die Felder der Anzeige 'Create Nickname' (Kurznamen erstellen) (CSQ4VDP5 für TSO, VD5 für CICS) ausgefüllt hat:

- Hat der Benutzer nur einen Benutzernamen angegeben oder handelt es sich bei dem Warteschlangenmanager um denselben Warteschlangenmanager, mit dem der Mail-Manager verbunden ist, erstellt das Programm eine Aliaswarteschlange.
- Hat der Benutzer sowohl einen Benutzernamen als auch den Namen eines Warteschlangenmanagers angegeben (bei dem es sich nicht um den Warteschlangenmanager handelt, mit dem der Mail-Manager verbunden ist), erstellt das Programm eine lokale Definition einer fernen Warteschlange. Das Programm überprüft nicht, ob die Warteschlange, in die die Definition aufgelöst wird, auch tatsächlich vorhanden ist; es wird nicht einmal überprüft, ob der ferne Warteschlangenmanager vorhanden ist.

Darüber hinaus erstellt das Programm auch eine temporäre dynamische Warteschlange für Antworten von der Eingabewarteschlange für Systembefehle.

Kann der Warteschlangenmanager aus einem vom Programm erwarteten Grund keine Kurznamenwarteschlange erstellen (weil die Warteschlange beispielsweise bereits vorhanden ist), wird vom Programm eine eigene Fehlernachricht angezeigt. Kann der Warteschlangenmanager aus einem vom Programm nicht erwarteten Grund keine Warteschlange erstellen, werden vom Programm bis zu zwei der vom Befehlsserver zurückgegebenen Fehlernachrichten angezeigt.

Anmerkung: Das Kurznamenprogramm erstellt für jeden Kurznamen nur eine Aliaswarteschlange oder eine lokale Definition einer fernen Warteschlange. Die lokalen Warteschlangen, in die diese Warteschlangennamen aufgelöst werden, werden nur erstellt, wenn die Mail-Manager-Anwendung mit der im Kurznamen enthaltenen Benutzer-ID gestartet wird.

z/OS *Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS*

Bei der Beispielanwendung zur Prüfung der Kreditwürdigkeit handelt es sich um eine Reihe von Programmen, die die Verwendung einer Vielzahl der von IBM MQ for z/OS bereitgestellten Funktionen veranschaulichen. Sie zeigt, auf welche Weise viele der Programmkomponenten einer Anwendung unter Verwendung von Message-Queuing-Verfahren Nachrichten austauschen können.

Die Beispielanwendung kann als eigenständige CICS-Anwendung ausgeführt werden. Um jedoch die Erstellung einer Anwendung zum Einreihen von Nachrichten zu veranschaulichen, die sowohl die in CICS- als auch in IMS-Umgebungen bereitgestellten Funktionen nutzt, wird ein Modul auch als IMS-Programm für die Verarbeitung von Nachrichten im Stapelbetrieb bereitgestellt. Diese Erweiterung der Beispielanwendung wird unter „IMS-Erweiterung zum Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS” auf Seite 1286 beschrieben.

Sie können die Beispielanwendung auch auf mehreren Warteschlangenmanagern ausführen und Nachrichten zwischen den einzelnen Anwendungsinstanzen austauschen. Hinweise hierzu finden Sie unter „Beispielprogramm 'Credit Check' mit mehreren Warteschlangenmanagern unter z/OS” auf Seite 1285.

Die CICS-Programme werden in C und COBOL bereitgestellt. Das einzige IMS-Programm wird nur in C geliefert. Die bereitgestellten Dateien sind in Tabelle 171 auf Seite 1252 und Tabelle 173 auf Seite 1254 aufgeführt.

Die Anwendung veranschaulicht ein Verfahren, mit dem eventuelle Risiken beurteilt werden können, wenn Bankkunden einen Kredit beantragen. Die Anwendung zeigt zwei Möglichkeiten für eine Bank, Kreditanträge zu bearbeiten:

- Wird persönlich mit dem Kunden verhandelt, benötigen die Bankangestellten unmittelbaren Zugriff auf Informationen zum Konto und zum Kreditrisiko.
- Bei schriftlichen Anträgen können die Bankangestellten eine Reihe von Anforderungen übergeben, in denen Informationen zum Konto und zum Kreditrisiko angefragt werden, und die Antworten dann zu einem späteren Zeitpunkt bearbeiten.

Die finanz- und sicherheitspezifischen Details wurden in der Anwendung einfach gehalten, um die Message-Queuing-Verfahren besser darstellen zu können.

z/OS *Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS vorbereiten und ausführen*

Um die Beispielanwendung zur Prüfung der Kreditwürdigkeit vorzubereiten und auszuführen, führen Sie die folgenden Schritte aus:

1. Erstellen Sie die VSAM-Datei, die Informationen zu einigen Beispielkonten enthält. Dazu wird die in der Datei CSQ4FILE bereitgestellte JCL bearbeitet und ausgeführt.
2. Führen Sie die unter „Beispielanwendungen für die CICS-Umgebung unter z/OS vorbereiten" auf Seite 1250 beschriebenen Schritte aus (die zusätzlich erforderlichen Schritte bei einer Verwendung der IMS-Erweiterung für die Beispielanwendung sind unter „IMS-Erweiterung zum Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS" auf Seite 1286 beschrieben).
3. Starten Sie unter Verwendung der CICS-Transaktion CKQC den Auslösemonitor CKTI (wird in IBM MQ for z/OS bereitgestellt) für die Warteschlange CSQ4SAMP.INITIATION.QUEUE.
4. Um die Anwendung zu starten, starten Sie das CICS-System und verwenden Sie die Transaktion MVB1.
5. Wählen Sie in der ersten Anzeige für die Abfrage **Immediate** (Direkt) oder **Batch** (Stapel) aus.

   Die Anzeigen für eine Direktbnfrage und eine Stapelabfrage sind ähnlich; Abbildung 156 auf Seite 1275 zeigt die Anzeige 'Immediate Inquiry' (Direktabfrage).

```
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
```

   Abbildung 156. Anzeige 'Immediate Inquiry' (Direktabfrage) für die Beispielanwendung zur Prüfung der Kreditwürdigkeit
6. Geben Sie eine Kontonummer und die beantragte Kreditsumme in die entsprechenden Felder ein (Hinweise zu den Informationen, die in diesen Felder eingegeben werden müssen finden Sie unter „Informationen in die Abfrageanzeigen eingeben" auf Seite 1275).

Informationen in die Abfrageanzeigen eingeben

Die Beispielanwendung zur Prüfung der Kreditwürdigkeit überprüft, ob die von Ihnen im Feld **Amount requested** (Beantragte Kreditsumme) der Abfrageanzeigen eingegebenen Daten in Form von Ganzzahlen eingegeben wurde.

Bei Eingabe einer der folgenden Kontonummern sucht die Anwendung in der VSAM-Datei CSQ4BAQ nach dem betreffenden Kontonamen, dem durchschnittlichen Kontostand und dem Kreditwürdigkeitsindex:

- 2222222222
- 3111234329
- 3256478962

- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

In die anderen Feldern können Sie beliebige Informationen eintragen oder die Felder auch leer lassen. Die Anwendung behält alle von Ihnen eingegebenen Daten bei und gibt sie in den von ihr generierten Berichten zurück.

z/OS *Entwurf des Beispielprogramms zur Prüfung der Kreditwürdigkeit unter z/OS*

In diesem Abschnitt ist das Design der einzelnen Programme beschrieben, aus denen sich die Beispielanwendung zur Prüfung der Kreditwürdigkeit zusammensetzt.

Weitere Informationen zu einigen der Verfahren, die in das Design der Anwendung eingingen, finden Sie unter „Entwurfsaspekte zum Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS" auf Seite 1283.

In Abbildung 157 auf Seite 1277 sind die Programme zu sehen, aus denen sich die Anwendung zusammensetzt, sowie die Warteschlangen, die von diesen Programmen verwendet werden. In dieser Abbildung wurde das Präfix CSQ4SAMP in den Warteschlangennamen weggelassen, um die Abbildung übersichtlicher zu gestalten.

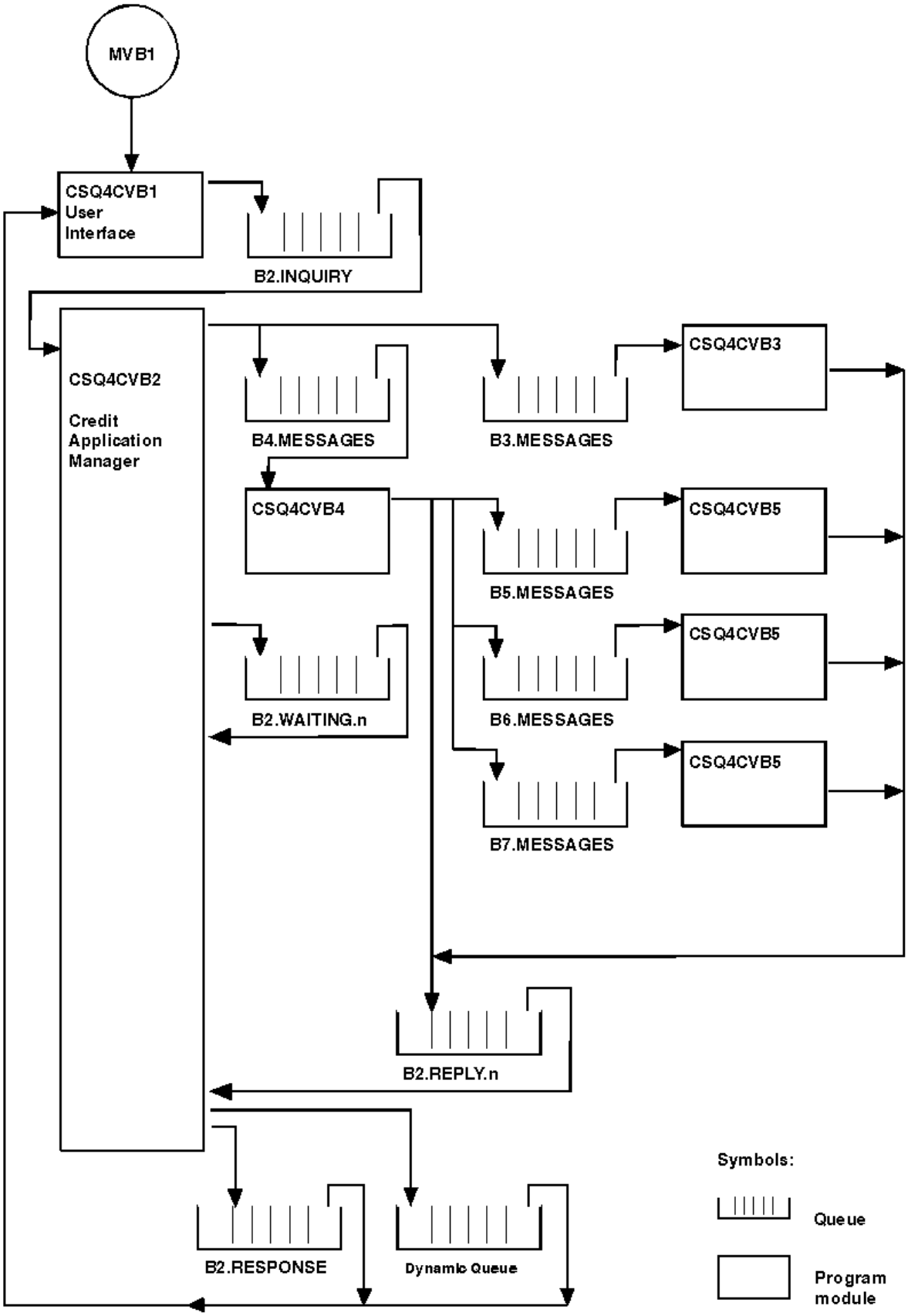


Abbildung 157. Programme und Warteschlangen für die Beispielanwendung zur Prüfung der Kreditwürdigkeit (nur COBOL-Programme)

z/OS *Benutzerschnittstellenprogramm (CSQ4CVB1) unter z/OS*

Beim Start der dialoggesteuerten CICS-Transaktion MVB1 wird die Benutzerschnittstelle für die Anwendung gestartet.

Dieses Programm reiht Anfragenachrichten in die Warteschlange CSQ4SAMP.B2.INQUIRY ein und ruft Antworten zu diesen Anfragen aus einer Empfangswarteschlange für Antworten ab, die bei der Abfrage angegeben wird. Sie können über die Benutzerschnittstelle Direkt- oder Stapelabfragen übergeben:

- Bei Direktabfragen erstellt das Programm eine temporäre dynamische Warteschlange, die es als Empfangswarteschlange für Antworten verwendet; das bedeutet, dass für jede Abfrage eine eigene Empfangswarteschlange für Antworten vorhanden ist.
- Bei Stapelabfragen ruft das Benutzerschnittstellenprogramm die Antworten aus der Warteschlange CSQ4SAMP.B2.RESPONSE ab. Um das Programm übersichtlich zu gestalten, ruft es die Antworten auf alle Abfragen nur aus dieser einen Empfangswarteschlange für Antworten ab. Es ist verständlich, dass eine Bank hingegen für jeden Benutzer von MVB1 eine eigene Empfangswarteschlange für Antworten bevorzugt, damit den Benutzern nur die Antworten auf die von ihnen eingeleiteten Abfragen angezeigt werden.

Wichtige Unterschiede zwischen den Eigenschaften der in der Anwendung verwendeten Nachrichten im Stapel- oder Direktmodus:

- Im Stapelmodus haben die Nachrichten eine niedrige Priorität, daher werden sie erst nach den im Direktmodus eingegebenen Kreditanträgen bearbeitet. Darüber hinaus sind die Nachrichten persistent, d. h., sie werden bei einem Neustart der Anwendung oder des Warteschlangenmanagers wiederhergestellt.
- Im Direktmodus haben die Nachrichten eine hohe Priorität und werden daher vor allen im Stapelmodus eingegebenen Kreditanträgen verarbeitet. Darüber hinaus sind diese Nachrichten nicht persistent, werden also bei einem Neustart der Anwendung oder des Warteschlangenmanagers verworfen.

In allen Fällen jedoch werden die Eigenschaften der Nachrichten mit Kreditanträgen in der Anwendung weitergegeben. Dies bedeutet, dass alle Nachrichten in Zusammenhang mit einer Anforderung von hoher Priorität ebenfalls eine hohe Priorität haben.

z/OS *Kreditantragsmanager (CSQ4CVB2) für z/OS*

Das CAM-Programm (Credit Application Manager; Kreditantragsmanager) führt den Großteil der Verarbeitung für die Anwendung zu Prüfung der Kreditwürdigkeit aus.

Der Kreditantragsmanager wird bei Auftreten eines Auslöserereignisses in der Warteschlange CSQ4SAMP.B2.INQUIRY oder CSQ4SAMP.B2.REPLY.*n* vom Auslösemonitor CKTI gestartet (der Monitor ist in IBM MQ for z/OS enthalten). Dabei ist *n* eine ganze Zahl, die in einem Satz von Antwortwarteschlangen eine bestimmte Warteschlange bezeichnet. Die Auslösenachricht enthält Daten wie beispielsweise den Namen der Warteschlange, in der das Auslöserereignis aufgetreten ist.

Der Kreditantragsmanager speichert Informationen zu Anfragen, die von ihm verarbeitet werden, in Warteschlangen, deren Namen das Format 'CSQ4SAMP.B2.WAITING.n' haben. Diese Warteschlangen werden so benannt, dass sie jeweils einer Empfangswarteschlange für Antworten zugeordnet sind. So enthält beispielsweise die Warteschlange CSQ4SAMP.B2.WAITING.3 Eingabedaten für eine bestimmte Anfrage und die Warteschlange CSQ4SAMP.B2.REPLY.3 enthält eine Reihe von Antwortnachrichten (aus Programmen, die Datenbankenabfragen durchführen), die alle mit dieser Anfrage in Verbindung stehen. Hintergrundinformationen zu diesem Design finden Sie unter „Separate Abfrage- und Antwortwarteschlangen im Kreditantragsmanager (CAM)” auf Seite 1283.

Startlogik

Wenn das Auslöserereignis für die Warteschlange CSQ4SAMP.B2.INQUIRY auftritt, öffnet der Kreditantragsmanager die Warteschlange für den gemeinsamen Zugriff. Anschließend versucht er, die Antwortwarteschlangen nacheinander zu öffnen, bis er eine freie Warteschlange findet. Findet er keine freie Antwortwarteschlange, protokolliert er dies, anschließend wird er normal beendet.

Tritt das Auslöserereignis für die Warteschlange CSQ4SAMP.B2.REPLY.n auf, öffnet der Kreditantragsmanager diese Warteschlange für den exklusiven Zugriff. Geht aus dem Rückgabecode hervor, dass das Objekt in Gebrauch ist, wird der Kreditantragsmanager normal beendet. Wird ein anderer Fehler gemeldet, wird er vom Kreditantragsmanager protokolliert und der Kreditantragsmanager wird beendet. Der Kreditantragsmanager öffnet die entsprechende Warteschlange für anstehende Anfragen und die Anfragewarteschlange und beginnt dann mit dem Abrufen und Verarbeiten von Nachrichten. Aus der Warteschlange für anstehende Anfragen stellt der Kreditantragsmanager Details zu teilweise verarbeiteten Anfragen wieder her.

Um das Beispiel möglichst einfach zu halten, sind die Namen der Warteschlangen im Programm enthalten. In einer Geschäftsumgebung befinden sich die Namen der Warteschlangen wahrscheinlich in einer Datei, auf die das Programm zugreift.

Nachricht aus der Anfragewarteschlange abrufen

Der Kreditantragsmanager versucht zunächst, mit dem MQGET-Aufruf unter Angabe der Option MQGMO_SET_SIGNAL eine Nachrichten aus der Anfragewarteschlange abzurufen. Ist die Nachricht sofort verfügbar, wird sie verarbeitet; ist keine Nachricht verfügbar, wird ein Signal gesetzt.

Der Kreditantragsmanager versucht, mit dem MQGET-Aufruf unter Angabe derselben Option eine Nachricht aus der Antwortschlange abzurufen. Ist eine Nachricht sofort verfügbar, wird sie verarbeitet, andernfalls wird ein Signal gesetzt.

Sind beide Signale gesetzt, wartet das Programm, bis eines der beiden Signale übertragen wird. Wird ein Signal übertragen und zeigt damit an, dass eine Nachricht verfügbar ist, wird die Nachricht abgerufen und verarbeitet. Läuft das Signal ab oder wird der Warteschlangenmanager beendet, wird das Programm beendet.

Vom Kreditantragsmanager abgerufene Nachricht verarbeiten

Bei den vom Kreditantragsmanager abgerufenen Nachrichten kann es sich um einen der folgenden vier Nachrichtentypen handeln:

- Eine Anfragenachricht
- Eine Antwortnachricht
- Eine Weiterleitungsnachricht
- Eine nicht erwartete oder unerwünschte Nachricht

Der Kreditantragsmanager verarbeitet diese Nachrichten wie unter „Vom Kreditantragsmanager abgerufene Nachricht verarbeiten (unter z/OS)” auf Seite 1280 beschrieben.

Antwort senden

Nachdem der Kreditantragsmanager alle für eine Anfrage erwarteten Antworten erhalten hat, verarbeitet er die Antworten und erstellt eine Antwortnachricht. In dieser einen Nachricht werden sämtliche Daten aus allen Antwortnachrichten mit derselben Korrelations-ID (`CorrelId`) zusammengefasst. Diese Antwortnachricht wird in die im ursprünglichen Kreditantrag angegebene Empfangswarteschlange für Antworten eingereiht. Die Antwortnachricht wird innerhalb der Arbeitseinheit eingereiht, die den Abruf der letzten Antwortnachricht enthält. Ziel ist es, die Wiederherstellung zu erleichtern, indem sichergestellt wird, dass die Warteschlange CSQ4SAMP.B2.WAITING.n niemals eine abgeschlossene Nachricht enthält.

Teilweise abgeschlossene Anfragen wiederherstellen

Der Kreditantragsmanager kopiert alle Nachrichten, die er erhält, in die Warteschlange CSQ4SAMP.B2.WAITING.n. Dabei setzt er die Felder des Nachrichtendeskriptors wie folgt:

- *`Priority`* (Priorität) wird durch den Typ der Nachricht bestimmt:
  - Anforderungsnachrichten erhalten die Priorität '3'
  - Datagramme erhalten die Priorität '2'

  - Antwortnachrichten erhalten die Priorität '1'
- Das Feld *CorrelId* wird auf die Nachrichten-ID (*MsgId*) der Kreditantragsnachricht gesetzt.
- Andere Felder der MQMD-Struktur werden aus den Feldern der empfangenen Nachrichten kopiert.

Wenn eine Anfrage abgeschlossen ist, werden die Nachrichten für eine bestimmte Anfrage bei der Verarbeitung der Antworten aus der Warteschlange für anstehende Anfragen entfernt. Die Warteschlange für anstehende Anfragen enthält daher immer alle Nachrichten in Zusammenhang mit Anfragen, die gerade verarbeitet werden. Anhand dieser Nachrichten werden im Falle eine Programmneustarts Details zu Anfragen wiederhergestellt, die gerade in Bearbeitung waren. Die unterschiedlichen Prioritäten werden so gesetzt, dass Anfragenachrichten vor Weiterleitungs- oder Antwortfragen wiederhergestellt werden.

z/OS *Vom Kreditantragsmanager abgerufene Nachricht verarbeiten (unter z/OS)*

Eine von CAM (Credit Application Manager; Kreditantragsmanager) abgerufene Nachricht kann einen von vier Typen aufweisen. Von diesem Typ ist es abhängig, wie eine Nachricht vom Kreditantragsmanager verarbeitet wird.

Bei den vom Kreditantragsmanager abgerufenen Nachrichten kann es sich um einen der folgenden vier Nachrichtentypen handeln:

- Eine Anfragenachricht
- Eine Antwortnachricht
- Eine Weiterleitungsnachricht
- Eine nicht erwartete oder unerwünschte Nachricht

Der Kreditantragsmanager verarbeitet diese Nachrichten wie folgt:

Anfragenachricht

Anfragenachrichten stammen vom Benutzerschnittstellenprogramm. Dieses erstellt für jeden Kreditantrag eine Anfragenachricht.

Für alle Kreditanträge wird vom Kreditantragsmanager der Kontostand des Kundengirokontos angefordert. Dazu reiht er eine Anforderungsnachricht in die Aliaswarteschlange CSQ4SAMP.B2.OUTPUT.ALIAS ein. Dieser Warteschlangenname wird in die Warteschlange CSQ4SAMP.B3.MESSAGES aufgelöst, die vom Girokontenprogramm CSQ4CVB3 verarbeitet wird. Wenn der Kreditantragsmanager eine Nachricht in diese Aliaswarteschlange einreiht, gibt er die entsprechende CSQ4SAMP.B2.REPLY.n-Warteschlange für die Empfangswarteschlange für Antworten an. Eine Aliaswarteschlange wird hier verwendet, damit das Programm CSQ4CVB3 jederzeit durch ein anderes Programm ersetzt werden kann, das eine Basiswarteschlange mit einem anderen Namen verarbeitet. Dazu müssen Sie die Aliaswarteschlange neu definieren, sodass ihr Name in diese neue Warteschlange aufgelöst wird. Ebenso können Sie der Aliaswarteschlange und der Basiswarteschlange unterschiedliche Zugriffsberechtigungen zuordnen.

Wird von einem Benutzer ein Kredit von mehr als 10.000 Währungseinheiten beantragt, überprüft der Kreditantragsmanager auch andere Datenbanken. Dazu reiht er eine entsprechende Anforderungsnachricht in die Warteschlange CSQ4SAMP.B4.MESSAGES ein, die vom Verteilungsprogramm CSQ4CVB4 verarbeitet wird. Der Prozess, der diese Warteschlange bedient, leitet die Nachricht an Warteschlangen weiter, die von Programmen bedient werden, die Zugriff auf weitere Datensätze wie beispielsweise Kreditkartenprotokoll, Sparkonten und Hypothekenraten haben. Die Daten von diesen Programmen werden an die in der Put-Operation angegebene Empfangswarteschlange für Antworten zurückgegeben. Darüber hinaus wird von diesem Programm eine Weiterleitungsnachricht an die Empfangswarteschlange für Antworten gesendet, um anzugeben, wie viele Weiterleitungsnachrichten gesendet wurden.

In einer Geschäftsumgebung wird das Verteilungsprogramm die bereitgestellten Daten wahrscheinlich neu formatieren, damit sie den für die verschiedenen Bankkontotypen erforderlichen Formaten entsprechen.

Alle Warteschlangen, auf die verwiesen wird, können sich auf einem fernen System befinden.

Für jede Anfragenachricht initialisiert der Kreditantragsmanager einen Eintrag in der speicherresidenten Anfragedatensatztabelle (IRT, Inquiry Record Table). Dieser Datensatz enthält Folgendes:

- Das Feld `MsgId` mit der Nachrichten-ID der Anfragenachricht
- Das Feld 'ReplyExp' mit der Anzahl der erwarteten Antworten (die der Anzahl der gesendeten Nachrichten entspricht)
- Das Feld 'ReplyRec' mit der Anzahl der empfangenen Antworten (an diesem Punkt null)
- Das Feld 'PropsOut' mit der Angabe, ob eine Weiterleitungsnachricht erwartet wird

Der Kreditantragsmanager kopiert die Anfragenachricht mit den folgenden Werten in die Warteschlange für anstehende Anfragen:

- Das Feld `Priority` ist auf '3' gesetzt
- Das Feld `CorrelId` ist auf die `MsgId` der Anfragenachricht gesetzt
- Die anderen Felder des Nachrichtendeskriptors sind auf die Werte der Anfragenachricht gesetzt

Weiterleitungsnachricht
Eine Weiterleitungsnachricht enthält die Anzahl der Warteschlangen, an die das Programm die Anfrage weitergeleitet hat. Die Nachricht wird wie folgt verarbeitet:

1. Im Feld 'ReplyExp' des entsprechenden Datensatzes in der Anfragedatensatztabelle (IRT) wird die Anzahl der gesendeten Nachrichten hinzugefügt. Diese Information ist in der Nachricht enthalten.
2. Der Wert des Felds 'ReplyRec' des Datensatzes in der Anfragedatensatztabelle (IRT) wird um 1 erhöht.
3. Der Wert des Felds 'PropsOut' des Datensatzes in der Anfragedatensatztabelle (IRT) wird um 1 verringert.
4. Die Nachricht wird in die Warteschlange für anstehende Anfragen kopiert. Dabei setzt der Kreditantragsmanager das Feld `Priority` auf '2' und die anderen Felder des Nachrichtendeskriptors auf die Werte der Weiterleitungsnachricht.

Antwort- nachricht
Eine Antwortnachricht enthält eine Antwort auf eine der Anforderungen, die an das Girokontenprogramm oder an die Programme zur Abfrage der Kreditbürodatenbank gingen. Antwortnachrichten werden wie folgt verarbeitet:

1. Der Wert des Felds 'ReplyRec' des Datensatzes in der Anfragedatensatztabelle (IRT) wird um 1 erhöht.
2. Die Nachricht wird in die Warteschlange für anstehende Anfragen kopiert. Dabei setzt der Kreditantragsmanager das Feld `Priority` auf '1' und die anderen Felder des Nachrichtendeskriptors auf die Werte der Antwortnachricht.
3. Wenn ReplyRec=ReplyExp ist und PropsOut auf '0' gesetzt ist, wird das Flag 'MsgComplete' gesetzt.

Andere Nachrichten
Die Anwendung erwartet keine anderen Nachrichten. Unter Umständen erhält die Anwendung aber vom System generierte Nachrichten oder Antwortnachrichten mit einer unbekannten Korrelations-ID (`CorrelIds`).

Der Kreditantragsmanager reiht diese Nachrichten in die Warteschlange CSQ4SAMP.DEAD.QUEUE ein, wo sie geprüft werden können. Schlägt dieser Vorgang fehl, geht die Nachricht verloren und das Programm wird fortgesetzt. Weitere Informationen zum Design dieses Programmteils finden Sie unter „Handhabung unerwarteter Nachrichten in der Beispielanwendung” auf Seite 1284.

z/OS *Girokontenprogramm (CSQ4CVB3) für z/OS*

Das Girokontenprogramm wird über ein Auslöserereignis für die Warteschlange CSQ4SAMP.B3.MESSAGES gestartet. Nachdem es die Warteschlange geöffnet hat, ruft dieses Programm mit dem MQGET-Aufruf unter Angabe der (auf ein Warteintervall von 30 Sekunden gesetzten) Warteoption eine Nachricht ab.

Das Programm durchsucht die VSAM-Datei CSQ4BAQ nach der in der Kreditantragsnachricht angegebenen Kontonummer. Es ruft den entsprechenden Kontonamen, den durchschnittlichen Kontostand und den Kreditwürdigkeitsindex ab oder hält fest, dass die Kontonummer nicht in der Datei enthalten ist.

Anschließend reiht das Programm mit dem MQPUT1-Aufruf eine Antwortnachricht in die in der Kreditantragsnachricht angegebene Empfangswarteschlange für Antworten ein. Für diese Antwortnachricht geht das Programm wie folgt vor:

- Es kopiert die Korrelations-ID (`CorrelId`) der Kreditantragsnachricht
- Es verwendet die Option MQPMO_PASS_IDENTITY_CONTEXT

Das Programm ruft so lange Nachrichten aus der Warteschlange ab, bis das Warteintervall abläuft.

z/OS *Verteilungsprogramm (CSQ4CVB4) unter z/OS*

Das Verteilungsprogramm wird über ein Auslöserereignis für die Warteschlange CSQ4SAMP.B4.MESSAGES gestartet.

Um die Verteilung des Kreditantrags an andere Kreditbüros zu simulieren, die Zugriff auf Datensätze wie Kreditkartenprotokoll, Sparkonten und Hypothekenraten haben, reiht das Programm in alle in der Namensliste CSQ4SAMP.B4.NAMELIST aufgeführten Warteschlangen eine Kopie derselben Nachricht ein. Es gibt drei dieser Warteschlangen mit dem Namen CSQ4SAMP.B*n*.MESSAGES, wobei *n* 5, 6 oder 7 ist. In einer Geschäftsanwendung können sich die Kreditbüros an verschiedenen Standardorten befinden, daher kann es sich bei diesen Warteschlangen auch um ferne Warteschlangen handeln. Hinweise, wie Sie die Beispielanwendung hierfür entsprechend ändern können, finden Sie unter „Beispielprogramm 'Credit Check' mit mehreren Warteschlangenmanagern unter z/OS” auf Seite 1285.

Das Verteilungsprogramm führt die folgenden Schritte aus:

1. Es ruft aus der Namensliste die Namen der Warteschlangen ab, die es verwenden soll. Dazu ruft das Programm mit dem MQINQ-Aufruf die Attribute des Namenslistenobjekts ab.
2. Es öffnet diese Warteschlangen und außerdem die Warteschlange CSQ4SAMP.B4.MESSAGES.
3. Es führt die folgende Schleife aus, bis in der Warteschlange CSQ4SAMP.B4.MESSAGES keine Nachrichten mehr enthalten sind:
   a. Es ruft mit dem MQGET-Aufruf unter Angabe der (auf ein Warteintervall von 30 Sekunden gesetzten) Warteoption eine Nachricht ab.
   b. Es reiht in jede der in der Namensliste aufgeführten Warteschlangen eine Nachricht ein, wobei es den Namen der entsprechenden CSQ4SAMP.B2.REPLY.n-Warteschlange für die Empfangswarteschlange für Antworten angibt. Das Programm kopiert die Korrelations-ID (*`CorrelId`*) der Kreditantragsnachricht in diese Nachrichtenkopien und verwendet den MQPUT-Aufruf unter Angabe der Option MQPMO_PASS_IDENTITY_CONTEXT.
   c. Es sendet eine Datagrammnachricht an die Warteschlange CSQ4SAMP.B2.REPLY.n und zeigt damit an, wie viele Nachrichten erfolgreich eingereiht wurden.
   d. Es erstellt einen Synchronisationspunkt.

z/OS *Programm zur Abfrage der Kreditbürodatenbank (CSQ4CVB5/CSQ4CCB5) für z/OS*

Das Programm zur Abfrage der Kreditbürodatenbank wird sowohl als COBOL- als auch als C-Version bereitgestellt. Beide Programme haben dasselbe Design. Dies zeigt, dass Programme unterschiedlichen Typs problemlos in einer IBM MQ-Anwendung koexistieren können und dass die Programmmodule, aus denen sich eine solche Anwendung zusammensetzt, ohne Weiteres ausgetauscht werden können.

Eine Instanz des Programms wird bei einem Auslöserereignis für eine der folgenden Warteschlangen gestartet:

- Für die COBOL-Version (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- Für die C-Version (CSQ4CCB5): Warteschlange CSQ4SAMP.B8.MESSAGES

Anmerkung: Wenn Sie die C-Version des Programms verwenden möchten, müssen Sie die Definition der Namensliste CSQ4SAMP.B4.NAMELIST ändern, indem die Warteschlange CSQ4SAMP.B7.MESSAGES durch CSQ4SAMP.B8.MESSAGES ersetzt wird. Folgende Möglichkeiten stehen hierzu zur Auswahl:

- Die IBM MQ for z/OS-Operationen und -Anzeigefelder
- Der Befehl ALTER NAMELIST
- Das Dienstprogramm CSQUTIL

Nachdem es die entsprechende Warteschlange geöffnet hat, ruft dieses Programm mit dem MQGET-Aufruf unter Angabe der (auf ein Warteintervall von 30 Sekunden gesetzten) Warteoption eine Nachricht ab.

Das Programm simuliert die Suche in der Datenbank eines Kreditbüros, indem es die VSAM-Datei CSQ4BAQ nach der Kontonummer durchsucht, die in der Kreditantragsnachricht übergeben wurde. Anschließend erstellt es eine Antwortnachricht, die den Namen der Warteschlange enthält, die es bedient, und einen Kreditwürdigkeitsindex. Um die Verarbeitung einfach zu halten, wird ein beliebiger Kreditwürdigkeitsindex ausgewählt.

Das Programm reiht die Antwortnachricht mit dem MQPUT1-Aufruf ein und geht dazu wie folgt vor:

- Es kopiert die Korrelations-ID (`CorrelId`) der Kreditantragsnachricht
- Es verwendet die Option MQPMO_PASS_IDENTITY_CONTEXT

Das Programm sendet die Antwortnachricht an die in der Kreditanfragenachricht angegebene Empfangswarteschlange für Antworten (der Name des Warteschlangenmanagers, der der Eigner der Empfangswarteschlange für Antworten ist, ist ebenfalls in der Kreditantragsnachricht angegeben).

z/OS *Entwurfsaspekte zum Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS*

Dieser Abschnitt enthält Hinweise zum Design der Beispielanwendung zur Prüfung der Kreditwürdigkeit

In diesem Abschnitt wird Folgendes erläutert:

- „Separate Abfrage- und Antwortwarteschlangen im Kreditantragsmanager (CAM)" auf Seite 1283
- „Fehlerbehandlung in der Beispielanwendung" auf Seite 1284
- „Handhabung unerwarteter Nachrichten in der Beispielanwendung" auf Seite 1284
- „Verwendung von Synchronisationspunkten in der Beispielanwendung" auf Seite 1284
- „Verwendung von Nachrichtenkontextinformationen in der Beispielanwendung" auf Seite 1285
- „Verwendung von Nachrichten- und Korrelations-IDs im Kreditantragsmanager" auf Seite 1285

Separate Abfrage- und Antwortwarteschlangen im Kreditantragsmanager (CAM)

Für die Anwendung könnte zwar ein und dieselbe Warteschlange für Abfragen und Antworten verwendet werden, aus den folgenden Gründen wurde sie jedoch für die Verwendung separater Warteschlangen konzipiert:

- Bei der Verarbeitung der maximal möglichen Anzahl an Abfragen können alle darüber hinausgehenden Abfragen in der Warteschlange verbleiben. Bei Verwendung nur einer Warteschlange müssten diese Abfragen aus der Warteschlange entfernt und anderweitig gespeichert werden.
- Bei einer entsprechend großen Anzahl an Nachrichtenübertragungen werden unter Umständen automatisch weitere Instanzen des Kreditantragsmanagers gestartet, um dieselbe Anfragewarteschlange zu bedienen. Die Anwendung muss jedoch verfolgen können, welche Abfragen gerade verarbeitet werden, und muss dazu alle Antworten auf Abfragen erhalten, die von ihr initiiert wurden. Bei Verwendung nur einer Warteschlange müsste die Anwendung die Nachrichten durchsuchen, um festzustellen, ob sie für diese Programminstanz oder eine andere gelten. Dies würde zu einem ineffizienten Betrieb führen.

  Die Anwendung kann mehrere Kreditantragsmanager unterstützen und ist in der Lage, Abfragen, die gerade verarbeitet werden, wiederherzustellen, indem sie paarweise zugeordnete Empfangswarteschlangen für Antworten und Warteschlangen für anstehende Abfragen verwendet.
- Die Anwendung kann unter Verwendung von Signalen mehrere Warteschlangen überwachen.

Fehlerbehandlung in der Beispielanwendung

Das Benutzerschnittstellenprogramm handhabt Fehler, indem es sie direkt an den Benutzer meldet.

Die anderen Programme verfügen über keine Benutzerschnittstelle und handhaben Fehler daher auf andere Weise. In manchen Situationen (wenn beispielsweise ein MQGET-Aufruf fehlschlägt) ist den Programmen die Identität des Benutzers der Anwendung nicht bekannt.

Die anderen Programme reihen Fehlernachrichten in die temporäre CICS-Speicherwarteschlange CSQ4SAMP ein. Sie können diese Warteschlange mithilfe der von CICS bereitgestellten Transaktion CEBR durchsuchen. Außerdem schreiben die Programme Fehlernachrichten in das CSML-Protokoll von CICS.

Handhabung unerwarteter Nachrichten in der Beispielanwendung

Beim Erstellen einer Message-Queuing-Anwendung müssen Sie sich überlegen, wie mit Nachrichten umgegangen werden soll, die unerwarteterweise in einer Warteschlange ankommen.

Es gibt zwei grundlegende Lösungen:

- Die Anwendung stellt die Verarbeitung ein, bis die unerwartete Nachricht verarbeitet wurde. In diesem Fall wird die Anwendung wahrscheinlich einen Bediener benachrichtigen und die Anwendung selbst beendet, wobei sichergestellt wird, dass sie nicht automatisch erneut gestartet werden kann (dies wird erreicht, indem die Auslösefunktion inaktiviert wird). Bei dieser Option kann die gesamte Verarbeitung für die Anwendung wegen einer einzigen unerwarteten Nachricht zum Stillstand kommen und für den Neustart der Anwendung ist ein Eingriff des Bedieners erforderlich.
- Die Anwendung entfernt die Nachricht aus der Warteschlange, die sie bedient, speichert die Nachricht an anderer Stelle und setzt die Verarbeitung fort. Am besten wird die Nachricht in die Systemwarteschlange für nicht zustellbare Nachrichten eingereiht.

Die Auswahl der zweiten Option bedeutet Folgendes:

- Ein Bediener oder ein anderes Programm muss die Nachrichten untersuchen, die in die Warteschlange für nicht zustellbare Nachrichten eingereiht wurden, und muss feststellen, woher diese Nachrichten stammen.
- Eine unerwartete Nachricht geht verloren, wenn sie nicht in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden kann.
- Eine lange unerwartete Nachricht wird abgeschnitten, wenn sie die maximal zulässige Länge für Nachrichten in der Warteschlange für nicht zustellbare Nachrichten oder die maximal zulässige Pufferlänge der Anwendung überschreitet.

Um sicherzustellen, dass die Anwendung alle Abfragen problemlos und möglichst unbeeinflusst von externen Aktivitäten verarbeiten kann, wird für die Beispielanwendung zur Prüfung der Kreditwürdigkeit die zweite Option verwendet. Damit die Beispielanwendung separat von anderen Anwendungen eingesetzt werden kann, die denselben Warteschlangenmanager verwenden, verwendet die Beispielanwendung zur Prüfung der Kreditwürdigkeit nicht die Systemwarteschlange für nicht zustellbare Nachrichten, sondern eine eigene Warteschlange für nicht zustellbare Nachrichten, die Warteschlange CSQ4SAMP.DEAD.QUEUE. Die Beispielanwendung schneidet alle Nachrichten ab, die die Länge des für die Beispielanwendung bereitgestellten Pufferbereichs überschreiten. Sie können die Nachrichten in dieser Warteschlange mithilfe der Browse-Beispielanwendung anzeigen oder mithilfe der Beispielanwendung zum Ausgeben von Nachrichten zusammen mit den Nachrichtendeskriptoren ausgeben.

Wenn Sie die Beispielanwendung für eine Ausführung über mehrere Warteschlangenmanager hinweg erweitern, können unerwartete Nachrichten oder Nachrichten, die nicht zugestellt werden können, vom Warteschlangenmanager in die Systemwarteschlange für nicht zustellbare Nachrichten eingereiht werden.

Verwendung von Synchronisationspunkten in der Beispielanwendung

Die Programme in der Beispielanwendung zur Prüfung der Kreditwürdigkeit erstellen Synchronisationspunkte, um Folgendes sicherzustellen:

- Nur eine Antwortnachricht wird als Antwort auf eine erwartete Nachricht gesendet
- Es werden niemals mehrere Kopien einer unerwarteten Nachricht in die Systemwarteschlange für nicht zustellbare Nachrichten eingereiht
- Der Kreditantragsmanager kann den Status aller teilweise abgeschlossenen Abfragen wiederherstellen, indem er persistente Nachrichten aus der Warteschlange für anstehende Abfragen abruft.

Dazu erfolgen der Abruf einer Nachricht, ihre Verarbeitung und alle anschließenden Put-Operationen innerhalb einer Arbeitseinheit.

Verwendung von Nachrichtenkontextinformationen in der Beispielanwendung

Das Benutzerschnittstellenprogramm (CSQ4CVB1) verwendet beim Senden von Nachrichten die Option MQPMO_DEFAULT_CONTEXT. Dies bedeutet, dass der Warteschlangenmanager sowohl die Identitätskontextinformationen als auch die Ursprungskontextinformationen generiert. Diese Informationen ruft der Warteschlangenmanager aus der Transaktion (MVB1) ab, die das Programm gestartet hat, und aus der Benutzer-ID, von der die Transaktion gestartet wurde.

Beim Senden von Anfragenachrichten verwendet der Kreditantragsmanager die Option MQPMO_PASS_IDENTITY_CONTEXT. Dadurch werden die Identitätskontextinformationen der Nachricht, die eingereiht wird, aus dem Identitätskontext der ursprünglichen Anfragenachricht kopiert. Mit dieser Option werden vom Warteschlangenmanager Ursprungskontextinformationen generiert.

Beim Senden von Antwortnachrichten verwendet der Kreditantragsmanager die Option MQPMO_ALTERNATE_USER_AUTHORITY. Dadurch verwendet der Warteschlangenmanager eine alternative Benutzer-ID für die Sicherheitsprüfung, wenn der Kreditantragsmanager eine Empfangswarteschlange für Antworten öffnet. Der Kreditantragsmanager verwendet die Benutzer-ID des Benutzers, von dem die ursprüngliche Anfragenachricht übergeben wurde. Benutzer können dadurch Antworten nur auf die Abfragen anzeigen, die von ihnen stammen. Die alternative Benutzer-ID wird aus den Identitätskontextinformationen im Nachrichtendeskriptor der ursprünglichen Anfragenachricht abgerufen.

Beim Senden von Antwortnachrichten verwenden die Abfrageprogramme (CSQ4CVB3/4/5) die Option MQPMO_PASS_IDENTITY_CONTEXT. Dadurch werden die Identitätskontextinformationen der Nachricht, die eingereiht wird, aus dem Identitätskontext der ursprünglichen Anfragenachricht kopiert. Mit dieser Option werden vom Warteschlangenmanager Ursprungskontextinformationen generiert.

Anmerkung: Für die den Transaktionen MVB3/4/5 zugeordnete Benutzer-ID ist ein Zugriff auf die B2.REPLY.n-Warteschlangen erforderlich. Diese Benutzer-IDs sind unter Umständen nicht dieselben wie die, die der Anforderung zugeordnet sind, die verarbeitet wird. Um diese möglichen Sicherheitslücken zu vermeiden, können die Abfrageprogramme beim Einreihen ihrer Antworten die Option MQPMO_ALTERNATE_USER_AUTHORITY verwenden. Damit benötigt jeder Benutzer der Transaktion MVB1 eine Berechtigung zum Öffnen der B2.REPLY.n-Warteschlangen.

Verwendung von Nachrichten- und Korrelations-IDs im Kreditantragsmanager

Die Anwendung muss jederzeit den Fortschritt aller aktiven Abfragen überwachen, die sie gerade verarbeitet. Dazu ordnet sie alle Informationen, die zu den einzelnen Abfragen vorhanden sind, mithilfe der eindeutigen Nachrichten-ID der einzelnen Kreditantragsnachrichten zu.

Der Kreditantragsmanager kopiert den Wert des Felds `MsgId` der Anfragenachricht in das Feld `CorrelId` aller Anforderungsnachrichten, die er für diese Abfrage sendet. Die anderen Programme in der Beispielanwendung (CSQ4CVB3-5) kopieren den Wert des Felds `CorrelId` jeder Nachricht, die sie erhalten, in das Feld `CorrelId` der Antwortnachricht.

z/OS *Beispielprogramm 'Credit Check' mit mehreren Warteschlangenmanagern unter z/OS*

Sie können mit der Beispielanwendung zur Prüfung der Kreditwürdigkeit die verteilte Steuerung von Warteschlangen veranschaulichen, indem Sie die Beispielanwendung auf zwei Warteschlangenmanagern und zwei CICS-Systemen installieren (dabei ist jeder Warteschlangenmanager mit einem anderen CICS-System verbunden).

Nach der Installation der Beispielanwendung und wenn der Auslösemonitor (CKTI) auf jedem System aktiv ist, müssen Sie Folgendes ausführen:

1. Richten Sie die Kommunikationsverbindungen zwischen den beiden Warteschlangenmanagern ein. Informationen hierzu finden Sie unter Configuring distributed queuing.
2. Erstellen Sie auf einem Warteschlangenmanager eine logische Definition für jede der fernen Warteschlangen (auf dem anderen Warteschlangenmanager), die verwendet werden sollen. Es kann sich hierbei um jede der Warteschlangen CSQ4SAMP.B*n*.MESSAGES handeln, wobei *n* 3, 5, 6 oder 7 sein kann. Es handelt sich hier um die Warteschlangen, die vom Girokontenprogramm und vom Programm zur Abfrage der Kreditbürodatenbank bedient werden. Informationen hierzu finden Sie unter DEFINE QREMOTE und DEFINE queues.
3. Ändern Sie die Definition der Namensliste (CSQ4SAMP.B4.NAMELIST) so, dass sie die Namen der fernen Warteschlangen enthält, die verwendet werden sollen. Informationen hierzu finden Sie unter DEFINE NAMELIST.

z/OS *IMS-Erweiterung zum Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS*

Eine Version des Girokontenprogramms wird als BMP-Programm (Batch Message Processing) der IMS-Umgebung bereitgestellt. Es ist in der Programmiersprache C erstellt.

Das Programm hat dieselbe Funktion wie die CICS-Version, nur dass es die Kontoinformationen nicht aus einer VSAM-Datei, sondern aus einer IMS-Datenbank abruft. Wenn Sie die CICS-Version des Girokontenprogramms durch die IMS-Version ersetzen, werden Sie keinen Unterschied in der Verwendung der Anwendung feststellen.

Um die IMS-Version vorzubereiten und auszuführen, müssen Sie wie folgt vorgehen:

1. Führen Sie die unter „Beispielprogramm zur Prüfung der Kreditwürdigkeit unter z/OS vorbereiten und ausführen” auf Seite 1275 beschriebenen Schritte aus.
2. Führen Sie die unter „Beispielanwendung für die IMS-Umgebung unter z/OS vorbereiten” auf Seite 1253 beschriebenen Schritte aus.
3. Ändern Sie die Definition der Aliaswarteschlange CSQ4SAMP.B2.OUTPUT.ALIAS so, dass sie in die Warteschlange CSQ4SAMP.B3.IMS.MESSAGES (anstatt in CSQ4SAMP.B3.MESSAGES) aufgelöst wird. Folgende Möglichkeiten stehen hierzu zur Auswahl:
   - Die IBM MQ for z/OS-Operationen und -Anzeigefelder
   - Der Befehl ALTER QALIAS

Das IMS-Girokontenprogramm kann auch so eingesetzt werden, dass es eine der Warteschlangen bedient, die Nachrichten vom Verteilungsprogramm empfangen. In der Form, in der die Beispielanwendung zur Prüfung der Kreditwürdigkeit bereitgestellt wird, stehen drei dieser Warteschlangen(B5/6/7.MESSAGES) zur Verfügung, die alle vom Programm zur Abfrage der Kreditbürodatenbank bedient werden. Dieses Programm durchsucht eine VSAM-Datei. Damit Sie einen Vergleich zwischen der Verwendung der VSAM-Datei und der Verwendung der IMS-Datenbank haben, können Sie das IMS-Girokontenprogramm so definieren, dass es stattdessen eine dieser Warteschlangen bedient. Dazu müsse Sie die Definition der Namensliste CSQ4SAMP.B4.NAMELIST so ändern, dass eine der CSQ4SAMP.B*n*.MESSAGES-Warteschlangen durch die Warteschlange CSQ4SAMP.B3.IMS.MESSAGES ersetzt wird. Folgende Möglichkeiten stehen hierzu zur Auswahl:

- Die IBM MQ for z/OS-Operationen und -Anzeigefelder
- Der Befehl ALTER NAMELIST

Anschließend können Sie das Beispiel über die CICS-Transaktion MVB1 ausführen. Der Benutzer wird keinen Unterschied in der Ausführung oder in den Antworten erkennen. Das IMS-Programm BMP wird entweder nach Empfang einer Stoppnachricht gestoppt oder wenn es fünf Minuten lang inaktiv war.

Design des IMS-Girokontenprogramms (CSQ4ICB3)

Dieses Programm wird als BMP-Programm ausgeführt. Bevor IBM MQ-Nachrichten an dieses Programm gesendet werden, wird es zunächst mit der zugehörigen JCL gestartet.

Das Programm durchsucht eine IMS-Datenbank nach der Kontonummer in der Kreditantragsnachricht. Es ruft den entsprechenden Kontonamen, den durchschnittlichen Kontostand und den Kreditwürdigkeitsindex ab.

Das Programm sendet das Ergebnis der Datenbanksuche an die Empfangswarteschlange für Antworten, die in der IBM MQ-Nachricht, die gerade verarbeitet wird, angegeben ist. Die zurückgegebene Nachricht hängt den Kontotyp und die Suchergebnisse an die empfangene Nachricht an, sodass die Transaktion, die die Antwort erstellt, feststellen kann, dass die richtige Abfrage verarbeitet wird. Die Nachricht besteht aus drei Blöcken mit je 79 Zeichen:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

Bei der Ausführung als nachrichtenorientiertes BMP-Programm leert das Programm die IMS-Nachrichtenwarteschlange, liest dann Nachrichten aus der IBM MQ for z/OS-Warteschlange und verarbeitet sie. Es werden keine Informationen aus der IMS-Nachrichtenwarteschlange empfangen. Das Programm trennt nach jedem Prüfpunkt die Verbindung zum Warteschlangenmanager, da die Kennungen geschlossen wurden.

Bei der Ausführung als stapelorientiertes BMP-Programm bleibt die Verbindung des Programms zum Warteschlangenmanager nach jedem Prüfpunkt weiterhin bestehen, da die Kennungen nicht geschlossen werden.

z/OS *Beispielprogramm 'Message Handler' für z/OS*

Mit der TSO-Beispielanwendung für die Nachrichtenbehandlung können Sie nach Nachrichten in einer Warteschlange suchen sowie Nachrichten weiterleiten und löschen. Die Beispielanwendung wird in C und COBOL bereitgestellt.

Beispielanwendung vorbereiten und ausführen

Gehen Sie wie folgt vor:

1. Bereiten Sie das Beispielprogramm wie unter „Beispielanwendungen für die TSO-Umgebung unter z/OS vorbereiten” auf Seite 1248 beschrieben vor.
2. Passen Sie die für die Beispielanwendung bereitgestellte CLIST (CSQ4RCH1) entsprechend an und geben Sie an, wo sich die Anzeigen, die Nachrichtendatei und die Lademodule befinden.

Sie können die CLIST (CSQ4RCH1) sowohl in der C- als auch in der COBOL-Version der Beispielanwendung ausführen. Die bereitgestellte CSQ4RCH1-Version wird in C ausgeführt und enthält Anweisungen zu den notwendigen Anweisungen für eine Anpassung für die COBOL-Version.

Anmerkung:

1. Für diese Beispielanwendung werden keine Beispielwarteschlangendefinitionen bereitgestellt.
2. Multitasking mit ISPF wird von VS COBOL II nicht unterstützt; Sie sollten daher die Beispielanwendung für die Nachrichtenbehandlung nicht in beiden Seiten einer geteilten Anzeige verwenden, da dies ein unvorhersehbares Ergebnis zur Folge hätte.

z/OS *Beispielanwendung für die Nachrichtenbehandlung unter z/OS verwenden*

Nachdem die Beispielanwendung installiert und über die angepasste CLIST CSQ4RCH1 aufgerufen wurde, wird die in Abbildung 158 auf Seite 1288 dargestellte Anzeige angezeigt.

```
---------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND == = >
User Id : JOHNJ



Enter information. Press ENTER :


Name des WS-Managers: ________________________________________________:

Warteschlangenname: _________________________________________________:









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP    F8=DOWN   F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

Abbildung 158. Eingangsanzeige der Beispielanwendung für die Nachrichtenbehandlung

Geben Sie den Namen des Warteschlngenmanagers und der Warteschlange ein (die Groß-/Kleinschreibung muss bei der Eingabe beachtet werden), die angezeigt werden sollen; daraufhin wird die Anzeige mit der Nachrichtenliste angezeigt (siehe Abbildung 159 auf Seite 1288).

```
---------------------- 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 ******************************
```

Abbildung 159. Anzeige mit der Nachrichtenliste für die Beispielanwendung für die Nachrichtenbehandlung

In dieser Anzeige werden die ersten 99 der in der Warteschlange enthaltenen Nachrichten angezeigt, wobei für jede Nachricht die folgenden Felder aufgeführt sind:

Msg No
Nachrichtennummer

Put Date MM/DD/YYYY
Das Datum, an dem die Nachricht in die Warteschlange eingereiht wurde (mittlere Greenwich-Zeit)

Put Time HH:MM:SS
Die Uhrzeit, zu der die Nachricht in die Warteschlange in die Warteschlange eingereiht wurde (mittlere Greenwich-Zeit)

Format Name
Das Feld 'MQMD.Format'

Benutzer-ID
Das Feld 'MQMD.UserIdentifier'

Put Application Type
Das Feld 'MQMD.PutApplType'

Put Application Name
Das Feld 'MQMD.PutApplName'

Die Gesamtzahl der in der Warteschlange enthaltenen Nachrichten wird ebenfalls angezeigt.

In dieser Anzeige kann eine Nachrichten anhand ihrer Nummer (nicht über die Cursorposition) ausgewählt und angezeigt werden. Ein Beispiel hierzu finden Sie im Thema Abbildung 160 auf Seite 1289.

```
---------------------- 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 :
------------------------------------------------------------------------------
Nachrichtendeskriptor
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 ********************************
```

Abbildung 160. Anzeige der ausgewählten Nachricht

Nachdem die Nachricht angezeigt wurde, kann sie gelöscht, in der Warteschlange belassen oder an eine andere Warteschlange weitergeleitet werden. Die Felder `Forward to Q Mgr` und `Forward to Queue` werden mit Werten aus dem MQMD-Struktur initialisiert; diese Werte können vor einer Weiterleitung der Nachricht geändert werden.

Aufgrund des Designs der Beispielanwendung können nur Nachrichten mit eindeutigen `MsgId/CorrelId`-Kombinationen ausgewählt und angezeigt werden, da die Auswahl der Nachricht über die Nachrichten-ID (`MsgId`) und die Korrelations-ID (`CorrelId` als Schlüssel erfolgt. Ist dieser Schlüssel nicht eindeutig, kann die Beispielanwendung die ausgewählte Nachricht nicht gezielt abrufen.

Anmerkung: Wenn Sie Nachrichten mithilfe von SCSQCLST(CSQ4RCH1) durchsuchen, wird der Rücksetzungszähler der Nachricht bei jedem Aufruf erhöht. Soll das Verhalten dieser Beispielanwendung geändert werden, müssen Sie das Beispiel kopieren und den Inhalt entsprechend ändern. Allerdings sollten Sie dabei berücksichtigen, dass diese Erhöhung des Zählers Auswirkungen auf andere Anwendungen haben kann, die von diesem Rücksetzungszähler abhängig sind.

z/OS *Entwurf der Beispielanwendung für die Nachrichtenbehandlung unter z/OS*

In diesem Abschnitt wird das Design der einzelnen Programme beschrieben, aus denen sich die Beispielanwendung für die Nachrichtenbehandlung zusammensetzt.

Objektprüfungsprogramm

Für dieses Programm ist ein gültiger Warteschlangenname und der gültige Name eines Warteschlangenmanagers erforderlich.

Wird kein Warteschlangenmanager angegeben, wird der Standardwarteschlangenmanager verwendet (sofern vorhanden). Nur lokale Warteschlangen können verwendet werden; mit einem MQINQ-Aufruf wird der Warteschlangentyp überprüft; handelt es sich nicht um eine lokale Warteschlange, wird ein Fehler gemeldet. Konnte die Warteschlange nicht erfolgreich geöffnet werden oder wird der MQGET-Aufruf für die Warteschlange unterdrückt, werden Fehlernachrichten zurückgegeben und der Beendigungscode und der Ursachencode werden zurückgegeben.

Nachrichtenlistenprogramm

Dieses Programm zeigt eine Liste von Nachrichten an, die sich in der Warteschlange befinden, sowie Informationen zu den einzelnen Nachrichten, wie beispielsweise Datum und Uhrzeit des Einreihens sowie das Nachrichtenformat.

In der Liste können maximal 99 Nachrichten gespeichert werden. Enthält die Warteschlange mehr Nachrichten, wird zusätzlich noch die aktuelle Länge der Warteschlange angezeigt. Um eine Nachricht anzuzeigen, geben Sie im Eingabefeld die Nummer der Nachricht ein (der Standardwert ist 01). Bei einer ungültigen Eingabe erhalten Sie eine entsprechende Fehlernachricht.

Nachrichteninhaltsprogramm

Dieses Programm zeigt den Nachrichteninhalt an.

Der Inhalt wird formatiert und in zwei Teile aufgetrennt:

1. Nachrichtendeskriptor
2. Nachrichtenpuffer

Der Nachrichtendeskriptor zeigt den Inhalt der einzelnen Felder in jeweils einer eigenen Zeile an.

Die Formatierung des Nachrichtenpuffers hängt von seinem Inhalt ab. Enthält der Puffer eine MQDLH-Struktur (Header einer nicht zustellbaren Nachricht) oder eine MQXQH-Struktur (Header der Übertragungswarteschlange), wird dies formatiert und vor dem Puffer selbst angezeigt.

Vor der Formatierung der Pufferdaten gibt eine Titelzeile die Pufferlänge der Nachricht in Bytes an. Die maximale Puffergröße liegt bei 32768 Bytes; Nachrichten, die diese Länge überschreiten, werden abgeschnitten. Die vollständige Größe des Puffers wird zusammen mit der Nachricht angezeigt, dass nur die ersten 32768 Bytes der Nachricht angezeigt werden.

Die Pufferdaten werden wie folgt formatiert:

1. Nachdem der Offset im Puffer ausgegeben wurde, werden die Pufferdaten im Hexadezimalformat angezeigt.
2. Anschließend werden die Pufferdaten erneut als EBCDIC-Werte angezeigt. EBCDIC-Werte, die nicht ausgegeben werden können, werden als Punkt (.) ausgegeben.

Im Aktionsfeld können Sie 'D' (für Delete = Löschen) oder 'F' (für Forward = Weiterleiten) eingeben. Wenn die Nachricht weitergeleitet werden soll, müssen die Felder `forward-to queue` und `queue manager name` korrekt gesetzt werden. Die Standardwerte für diese Felder werden aus den Feldern 'ReplyToQ' und 'ReplyToQMgr' des Nachrichtendeskriptors ausgelesen.

Wenn Sie eine Nachricht weiterleiten, werden alle eventuell im Puffer gespeicherten Headerblöcke entfernt. Bei einer erfolgreichen Weiterleitung wird die Nachricht aus der ursprünglichen Warteschlange entfernt. Bei Eingabe einer ungültigen Aktion werden Fehlernachrichten angezeigt.

Darüber hinaus steht eine Hilfetextanzeige (CSQ4CHP9) zur Verfügung.

z/OS *Beispielprogramm für asynchrone Put-Operationen unter z/OS*

Das Beispielprogramm für asynchrone Put-Operationen reiht Nachrichten unter Verwendung des asynchronen MQPUT-Aufrufs in eine Warteschlange ein. Darüber hinaus ruft das Beispielprogramm mit dem MQSTAT-Aufruf Statusinformationen ab.

Die Anwendungen für asynchrone Put-Operationen verwenden die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

Die Beispielprogramme werden in C bereitgestellt.

Die Anwendungen für asynchrone Put-Operationen werden in der Stapelumgebung ausgeführt; Informationen zu den Stapelanwendungen finden Sie unter Weitere Beispielprogramme.

Dieser Abschnitt enthält außerdem Informationen zum Design des Programms zur asynchronen Verarbeitung und zur Ausführung des Beispielprogramms CSQ4BCS2.

- „Beispielprogramm CSQ4BCS2 ausführen” auf Seite 1291
- „Design des Beispielprogramms für asynchrone Put-Operationen” auf Seite 1291

Beispielprogramm CSQ4BCS2 ausführen

Für dieses Beispielprogramm können bis zu sechs Parameter angegeben werden:

1. Der Name der Zielwarteschlange (erforderlich)
2. Der Name des Warteschlangenmanagers (optional)
3. Optionen für das Öffnen (optional)
4. Optionen für das Schließen (optional)
5. Der Name des Zielwarteschlangenmanagers (optional)
6. Der Name der dynamischen Warteschlange (optional)

Wird kein Warteschlangenmanager angegeben, stellt CSQ4BCS2 eine Verbindung zum Standardwarteschlangenmanager her. Der Nachrichteninhalt wird über die Standardeingabe (**`SYSIN DD`**) bereitgestellt.

Für die Ausführung des Programms steht in CSQ4BCSP eine Beispiel-JCL zur Verfügung.

Design des Beispielprogramms für asynchrone Put-Operationen

Das Programm öffnet die Zielwarteschlange zum Einreihen von Nachrichten mithilfe des MQOPEN-Aufrufs unter Verwendung der bereitgestellten Ausgabeoptionen oder unter Verwendung der Optionen MQOO_OUTPUT und MQOO_FAIL_IF_QUIESCING.

Kann das Programm die Warteschlange nicht öffnen, gibt es eine Fehlernachricht mit dem vom MQOPEN-Aufruf zurückgegebenen Ursachencode aus. Um das Programm für diesen und nachfolgende MQI-Aufrufe möglichst einfach zu halten, werden für viele der Optionen Standardwerte verwendet.

Für jede Ausgabezeile liest das Programm den Text in einen Puffer, erstellt mit dem MQPUT-Aufruf unter Verwendung der Option MQPMO_ASYNC_RESPONSE eine Datagrammnachricht mit dem Text dieser Zeile und reiht die Nachricht asynchron in die Zielwarteschlange ein. Das Programm wird so lange fortgesetzt, bis es das Ende der Warteschlange erreicht oder bis der MQPUT-Aufruf fehlschlägt. Wenn das Programm das Ende der Warteschlange erreicht, schließt es die Warteschlange unter Verwendung des MQCLOSE-Aufrufs.

Anschließend gibt das Programm einen MQSTAT-Aufruf aus, der eine MQSTS-Struktur zurückgibt, und zeigt Nachrichten mit der Anzahl der erfolgreich eingereihten Nachrichten, der Anzahl der Nachrichten, die mit einer Warnung eingereiht wurden sowie die Anzahl der Nachrichten, die nicht eingereiht werden konnten, an.

Anmerkung: Wenn Sie sehen möchten, was passiert, wenn der MQSTAT-Aufruf einen MQPUT-Fehler feststellt, setzen Sie das Attribut MAXDEPTH der Zielwarteschlange auf einen niedrigen Wert.

z/OS *Beispielprogramm für die asynchrone Stapelverarbeitung unter z/OS*

Das in der Programmiersprache C bereitgestellte Beispielprogramm CSQ4BCS1 veranschaulicht die Verwendung der Aufrufe MQCB und MQCTL für die asynchrone Verarbeitung von Nachrichten aus mehreren Warteschlangen.

Die Beispielprogramme für die asynchrone Verarbeitung werden in der Stapelumgebung ausgeführt; Informationen zu den Stapelanwendungen finden Sie unter Weitere Beispielprogramme.

Es steht auch ein Beispielprogramm in COBOL bereit, das in der CICS-Umgebung ausgeführt wird (siehe „CICS-Beispielprogramme für asynchrone Verarbeitung und für Publish/Subscribe unter z/OS" auf Seite 1294).

Die Anwendungen verwenden die folgenden MQI-Aufrufe:

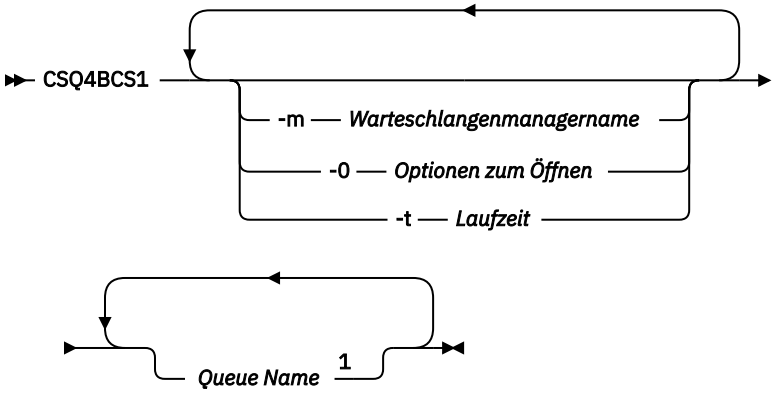
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

Dieser Abschnitt enthält außerdem noch die folgenden Informationen:

- „Beispielprogramm CSQ4BCS1 ausführen" auf Seite 1292
- „Design des Beispielprogramms für die asynchrone Stapelverarbeitung" auf Seite 1293

Beispielprogramm CSQ4BCS1 ausführen

Für dieses Beispielprogramm gilt die folgende Syntax:



Anmerkungen:

[1] Dieses Beispielprogramm unterstützt bis zu 10 Warteschlangen. Geben Sie mehrere Warteschlangennamen an, damit das Beispielprogramm Nachrichten aus mehreren Warteschlangen liest.

Für die Ausführung dieses Programms steht in CSQ4BCSC eine Beispiel-JCL zur Verfügung.

Design des Beispielprogramms für die asynchrone Stapelverarbeitung

Das Beispielprogramm veranschaulicht, wie Nachrichten aus mehreren Warteschlangen in der Reihenfolge ihres Eintreffens gelesen werden. Bei Verwendung eines synchronen MQGET-Aufrufs wäre hierzu mehr Code erforderlich. Bei der asynchronen Verarbeitung hingegen ist kein Polling erforderlich und die Thread- und Speicherverwaltung werden von IBM MQ übernommen. In diesem Beispielprogramm werden Fehler in die Konsole geschrieben.

Der Beispielcode setzt sich aus den folgenden Schritten zusammen:

1. Callback-Funktion für die Verarbeitung einer einzigen Nachricht definieren

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Verbindung zum Warteschlangenmanager herstellen

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Eingabewarteschlangen öffnen und jeder Warteschlange der Callback-Funktion `MessageConsumer` zuordnen.

```
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 können jeder Warteschlange eine eigene Callback-Funktion zuordnen.

4. Verarbeitung der Nachrichten starten

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Warten, bis der Benutzer die Eingabetaste drückt, dann die Nachrichtenverarbeitung stoppen

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Die Verbindung zum Warteschlangenmanager trennen

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *CICS-Beispielprogramme für asynchrone Verarbeitung und für Publish/Subscribe unter z/OS*

Die Beispielprogramme für die asynchrone Verarbeitung und für Publish/Subscribe veranschaulichen die Verwendung der asynchronen Verarbeitung sowie der Funktionen zum Veröffentlichen und Subskribieren (Publish/Subscribe) in CICS.

Ein *Registrierungsclient*programm registriert drei Callback-Handler (einen Ereignishandler sowie zwei Nachrichtenkonsumenten) und startet die asynchrone Verarbeitung. Ein *Messaging-Client*-Programm reiht Nachrichten in eine Warteschlange ein oder veröffentlicht über eine CICS-Konsole Nachrichten, die zur Verarbeitung geeignet sind, über die beiden Nachrichtenkonsumenten (CSQ4CVCN und CSQ4CVCT).

Damit eine Steuerung des Beispielprogramms während der Laufzeit möglich ist, kann einer der Nachrichtenkonsumenten über die Nachrichten, die er empfängt, bei Bedarf angewiesen werden, die Callback-Handler auszusetzen (SUSPEND), fortzusetzen (RESUME) oder ihre Registrierung zurückzunehmen (DEREGISTER). Über ihn kann auch der Befehl MQCTL STOP ausgegeben werden, um die asynchrone Verarbeitung kontrolliert zu beenden. Der andere Nachrichtenkonsument wird registriert, um ein Thema zu subskribieren.

Jedes Programm gibt an entsprechenden Punkten COBOL DISPLAY-Anweisungen aus, um das Verhalten des Beispielprogramms anzuzeigen.

Die Anwendungen verwenden die folgenden MQI-Aufrufe:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

Die Programme werden in COBOL bereitgestellt; Informationen zu den CICS -Anwendungen finden Sie im Abschnitt CICS -Beispielprogramme für asynchrone Verarbeitung und Publish/Subscribe .

Dieser Abschnitt enthält außerdem folgende Informationen:

- „Einrichtung” auf Seite 1294
- „Registrierungsclient (CSQ4CVRG)” auf Seite 1295
- „Ereignishandler (CSQ4CVEV)” auf Seite 1295
- „Konsument für einfache Nachrichten (CSQ4CVCN)” auf Seite 1295
- „Konsument für Steuernachrichten (CSQ4CVCT)” auf Seite 1295
- „Messaging-Client (CSQ4CVPT)” auf Seite 1295

Einrichtung

Der Name der Warteschlange und der Name des Themas, die von den Nachrichtenkonsumenten verwendet werden, sind im Registrierungsclientprogramm und im Messaging-Client-Programm fest codiert.

Die Warteschlange (**SAMPLE.CONTROL.QUEUE**) sollte vor der Ausführung des Beispielprogramms in dem Warteschlangenmanager definiert werden, der der CICS-Region zugeordnet ist. Das Thema (**News/**

Media/Movies) kann bei Bedarf definiert werden oder während der Laufzeit unter dem standardmäßigen Verwaltungsobjekt erstellt werden, falls es noch nicht vorhanden ist.

CICS-Programme und -Transaktionsdefinitionen können über die Installation einer Gruppe (CSQ4SAMP) installiert werden.

Registrierungsclient (CSQ4CVRG)

Das Registrierungsclientprogramm muss unter Steuerung der CICS-Transaktion MVRG gestartet werden. Es erfolgen keine Eingaben für dieses Programm.

Beim Start registriert der Registrierungsclient mit dem MQCB-Aufruf die folgenden Callback-Handler:

- CSQ4CVEV als Ereignishandler
- CSQ4CVCN als Nachrichtenkonsument für ein Thema (**News/Media/Movies**)
- CSQ4CVCT als Nachrichtenkonsument für eine Warteschlange (**SAMPLE.CONTROL.QUEUE**)

Der Registrierungsclient übergibt eine Datenstruktur mit den Namen aller drei registrierten Callback-Handler sowie die Objektkennungen der beiden Nachrichtenkonsumenten an CSQ4CVCT.

Nach der Registrierung der Callback-Handler gibt der Registrierungsclient den Befehl MQCTL START_WAIT aus, um die asynchrone Verarbeitung zu starten und sich selbst auszusetzen, bis die Steuerung wieder an ihn zurückgegeben wird (indem beispielsweise einer der Callback-Handler den Befehl MQCTL STOP ausgibt).

Ereignishandler (CSQ4CVEV)

Wenn der Ereignishandler ausgelöst wird, zeigt er eine Nachricht an, in der der Aufruftyp (beispielsweise START) angegeben ist. Wird der Ereignishandler für den IBM MQ-Ursachencodes CONNECTION_QUIESCING ausgelöst, gibt er den Befehl MQCTL STOP aus, um die asynchrone Verarbeitung zu beenden, und gibt die Steuerung an den Registrierungsclient zurück.

Konsument für einfache Nachrichten (CSQ4CVCN)

Wenn dieser Nachrichtenkonsument ausgelöst wird, zeigt er eine Nachricht an, in der der Aufruftyp (beispielsweise REGISTER) angegeben ist. Wird der Nachrichtenkonsument für den Aufruftyp MSG_REMOVED ausgelöst, ruft er die ankommende Nachricht ab und gibt sie im CICS-Jobprotokoll aus.

Konsument für Steuernachrichten (CSQ4CVCT)

Wenn dieser Nachrichtenkonsument ausgelöst wird, zeigt er eine Nachricht an, in der der Aufruftyp (beispielsweise START) angegeben ist. Wird der Nachrichtenkonsument für den Aufruftyp MSG_REMOVED ausgelöst, ruft er die ankommende Nachricht und die vom Registrierungsclient übergebene Datenstruktur ab. Anschließend gibt er abhängig vom Nachrichteninhalt entsprechende MQCB- oder MQCTL-Befehle aus, um Folgendes auszuführen:

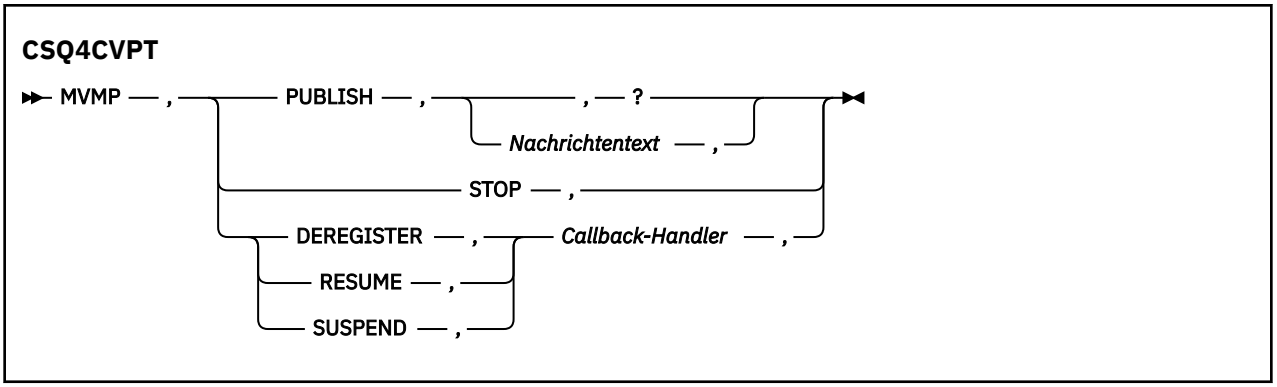
- Asynchrone Verarbeitung stoppen (STOP); die Steuerung wird an den Registrierungsclient zurückgegeben
- Einen angegebenen Callback-Handler (einschließlich sich selbst) auszusetzen (SUSPEND), fortzusetzen (RESUME) oder die Registrierung des Callback-Handlers zurückzunehmen (DEREGISTER)

Messaging-Client (CSQ4CVPT)

Der Messaging-Client hat zwei Funktionen:

- Er veröffentlicht eine Nachricht zu einem Thema, die vom Nachrichtenkonsumenten CSQ4CVCN verarbeitet wird.
- Er reiht Steuernachrichten in eine Warteschlange ein, die vom Konsumenten für Steuernachrichten (CSQ4CVCT) verarbeitet werden, was unter Umständen zu einer Änderung im Verhalten des Beispielprogramms führen kann.

Das Messaging-Client-Programm muss über eine CICS-Konsole unter Steuerung einer CICS-Transaktion gestartet werden; für dieses Programm werden in der Befehlszeile Befehle mit der folgenden Syntax eingegeben:



PUBLISH
Nachrichtentext (oder eine Standardnachricht) als ständige Nachricht zur Verarbeitung durch den Konsumenten für einfache Nachrichten veröffentlichen

STOPP
Asynchrone Verarbeitung stoppen

DEREGISTER
Registrierung des angegebenen Callback-Handlers zurücknehmen

RESUME
Angegebenen Callback-Handler fortsetzen

SUSPEND
Angegebenen Callback-Handler aussetzen

Die Eingabefelder sind positionsgebunden und durch Kommas getrennt. Die Groß-/Kleinschreibung muss bei den Schlüsselwörtern und bei den Namen der Callback-Handler nicht beachtet werden.

Beispiele:

Tabelle 175. Eingabebeispiele

Beispiel	**Beschreibung**
MVMP,PUBLISH,,	Standardnachricht veröffentlichen
MVMP, veröffentlichen, `A short message`,	Angegebenen Text veröffentlichen
MVMP,STOP,	Asynchrone Verarbeitung stoppen
MVMP,DEREGISTER,CSQ4CVEV,	Registrierung des Ereignishandlers zurücknehmen
MVMP,resume,csq4cvcn,	Konsumenten für einfache Nachrichten fortsetzen
MVMP,SUSPEND,CSQ4CVEV,	Ereignishandler aussetzen

Dabei ist MVMP die dem Messaging-Client-Programm CSQ4CVPT zugeordnete CICS-Transaktion.

Anmerkung:

- Wenn alle Callback-Handler ausgesetzt werden oder die Registrierung aller Callback-Handler zurückgenommen wird, wird der vom Registrierungsclient ausgegebene START_WAIT-Befehl beendet, die Steuerung an den Registrierungsclient zurückgegeben und die Task beendet.
- Das Aussetzen des Callback-Handlers für die Steuerung bzw. die Rücknahme der Registrierung dieses Callback-Handlers wurde bewusst nicht verhindert, verhindert jedoch die Möglichkeit, das Verhalten des Beispielprogramms weitergehend zu steuern.

z/OS *Publish/Subscribe-Beispielprogramm unter z/OS*

Die Publish/Subscribe-Beispielprogramme veranschaulichen die Verwendung der Publish- und Subscribe-Funktionen in IBM MQ.

Vier Beispielprogramme in C und zwei Beispielprogramme in COBOL veranschaulichen die Programmierung der Publish/Subscribe-Schnittstelle von IBM MQ. Die Programme werden in C und COBOL bereitgestellt. Die Anwendungen werden in der Stapelumgebung ausgeführt; Informationen zu den Stapelanwendungen finden Sie unter Publish/Subscribe-Beispielprogramme.

Es werden auch COBOL-Beispielprogramme bereitgestellt, die in der CICS-Umgebung ausgeführt werden (siehe „CICS-Beispielprogramme für asynchrone Verarbeitung und für Publish/Subscribe unter z/OS" auf Seite 1294).

Dieser Abschnitt enthält außerdem Informationen zur Ausführung der Publish/Subscribe-Beispielprogramme. Es handelt sich um die folgenden Beispielprogramme:

- „Beispielprogramm CSQ4BCP1 ausführen" auf Seite 1297
- „Beispielprogramm CSQ4BCP2 ausführen" auf Seite 1297
- „Beispielprogramm CSQ4BCP3 ausführen" auf Seite 1298
- „Beispielprogramm CSQ4BCP4 ausführen" auf Seite 1298
- „Beispielprogramm CSQ4BVP1 ausführen" auf Seite 1298
- „Beispielprogramm CSQ4BVP2 ausführen" auf Seite 1299

Beispielprogramm CSQ4BCP1 ausführen

Dieses Programm ist in C geschrieben und veröffentlicht Nachrichten zu einem Thema. Vor Ausführung dieses Programms müssen Sie eines der Subskribentenbeispiele starten.

Für dieses Programm können bis zu vier Parameter angegeben werden:

1. Der Name der Zielthemazeichenfolge (erforderlich).
2. Der Name des Warteschlangenmanagers (optional)
3. Optionen für das Öffnen (optional)
4. Optionen für das Schließen (optional)

Wird kein Warteschlangenmanager angegeben, stellt CSQ4BCP1 eine Verbindung zum Standardwarteschlangenmanager her. Für die Ausführung des Programms steht in CSQ4BCPP eine Beispiel-JCL zur Verfügung.

Der Nachrichteninhalt wird über die Standardeingabe (**SYSIN DD**) bereitgestellt.

Beispielprogramm CSQ4BCP2 ausführen

Das Programm ist in C geschrieben; es subskribiert ein Thema und gibt die Nachrichten aus, die es empfängt.

Für dieses Programm können bis zu drei Parameter angegeben werden:

1. Der Name der Zielthemazeichenfolge (erforderlich).
2. Der Name des Warteschlangenmanagers (optional)
3. Die MQSD-Subskriptionsoptionen (optional)

Wird kein Warteschlangenmanager angegeben, stellt CSQ4BCP2 eine Verbindung zum Standardwarteschlangenmanager her. Für die Ausführung des Programms steht in CSQ4BCPS eine Beispiel-JCL zur Verfügung.

Beispielprogramm CSQ4BCP3 ausführen

Dieses Programm ist in C geschrieben; es subskribiert unter Verwendung einer benutzerdefinierten Zielwarteschlange ein Thema und gibt die Nachrichten aus, die es empfängt.

Für dieses Programm können bis zu vier Parameter angegeben werden:

1. Der Name der Zielthemazeichenfolge (erforderlich).
2. Der Name der Zielwarteschlange (erforderlich)
3. Der Name des Warteschlangenmanagers (optional)
4. Die MQSD-Subskriptionsoptionen (optional)

Wird kein Warteschlangenmanager angegeben, stellt CSQ4BCP3 eine Verbindung zum Standardwarteschlangenmanager her. Für die Ausführung des Programms steht in CSQ4BCPD eine Beispiel-JCL zur Verfügung.

Beispielprogramm CSQ4BCP4 ausführen

Dieses Programm ist in C geschrieben; es subskribiert ein Thema und ruft Nachrichten zu diesem Thema ab; dabei können im MQSUB-Aufruf erweiterte Optionen verwendet werden, die über die im einfacheren MQSUB-Beispielprogramm CSQ4BCP2 enthaltenen Optionen hinausgehen. Zusätzlich zu den Nachrichtennutzdaten werden für jede Nachricht Nachrichteneigenschaften empfangen und angezeigt.

Für dieses Programm kann eine variable Gruppe von Parametern angegeben werden:

- **-t** *Themazeichenfolge* (erforderlich).
- **-o** *Themaobjektname* (erforderlich).
- **-m** *Warteschlangenmanagername* (optional).
- **-q** *Zielwarteschlangenname* (optional).
- **-w** *Warteintervall für MQGET in Sekunden* (optional), wobei für *seconds* folgende Werte zulässig sind:
  - unlimited: MQWI_UNLIMITED
  - none: Kein Wartezyklus
  - *n*: Warteintervall in Sekunden
  - Kein Wert angegeben: Wird kein Wert angegeben, gilt der Standardwert von 30 Sekunden.
- **-d** *Subskriptionsname* (optional). Erstellt die genannte permanente Subskription oder setzt sie fort.
- **-k** (optional). Behält die permanente Subskription für MQCLOSE bei.

Wird kein Warteschlangenmanager angegeben, stellt CSQ4BCP4 eine Verbindung zum Standardwarteschlangenmanager her. Für die Ausführung des Programms steht in CSQ4BCPE eine Beispiel-JCL-zur Verfügung.

Beispielprogramm CSQ4BVP1 ausführen

Dieses Programm wurde in COBOL erstellt und veröffentlicht Nachrichten zu einem Thema. Vor Ausführung dieses Programms müssen Sie eines der Subskribentenbeispiele starten.

Dieses Programm nimmt keine Parameter an. Der Name des Eingabethemas, der Name des Warteschlangenmanagers und der Nachrichteninhalt werden über die Standardeingabe (**SYSIN DD**) bereitgestellt.

Wird kein Warteschlangenmanager angegeben, stellt CSQ4BVP1 eine Verbindung zum Standardwarteschlangenmanager her. Für die Ausführung des Programms steht in CSQ4BVPP eine Beispiel-JCL-zur Verfügung.

Beispielprogramm CSQ4BVP2 ausführen

Dieses Programm wurde in COBOL erstellt; es subskribiert ein Thema und gibt die Nachrichten aus, die es empfängt.

Dieses Programm nimmt keine Parameter an. Der Name des Eingabethemas und der Name des Warteschlangenmanagers werden über die Standardeingabe (**SYSIN DD**) bereitgestellt.

Wird kein Warteschlangenmanager angegeben, stellt CSQ4BVP1 eine Verbindung zum Standardwarteschlangenmanager her. Für die Ausführung des Programms steht in CSQ4BVPP eine Beispiel-JCL-zur Verfügung.

z/OS *Beispielprogramm zum Setzen und Abfragen von Nachrichteneigenschaften unter z/OS*

Die Beispielprogramme für Nachrichteneigenschaften veranschaulichen, wie einer Nachrichtenkennung benutzerdefinierte Eigenschaften hinzugefügt und die Eigenschaften dieser Nachricht abgefragt werden.

Die Anwendungen verwenden die folgenden MQI-Aufrufe:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

Die Programme werden in C bereitgestellt. Die Anwendungen werden in der Stapelumgebung ausgeführt; Informationen zu den Stapelanwendungen finden Sie unter Weitere Beispielprogramme.

Mit dem Programm CSQ4BCM1 werden die Eigenschaften einer Nachrichtenkennung aus einer Nachrichtenwarteschlange abgefragt; es veranschaulicht die Verwendung des API-Aufrufs MQINQMP. Das Beispielprogramm ruft eine Nachricht aus einer Warteschlange ab und gibt sämtliche Eigenschaften der Nachrichtenkennung aus.

Mit dem Programm CSQ4BCM2 werden die Eigenschaften einer Nachrichtenkennung in einer Nachrichtenwarteschlange gesetzt; es veranschaulicht die Verwendung des API-Aufrufs MQSETMP. Dieses Beispiel erstellt eine Nachrichtenkennung und gibt sie im Feld `MsgHandle` der MQGMO-Struktur an. Anschließend reiht es die Nachricht in eine Warteschlange ein.

Weitere Beispiele zum Abfragen und Ausgeben von Nachrichteneigenschaften sind in den Beispielprogrammen CSQ4BCG1 und CSQ4BCP4 enthalten.

Dieser Abschnitt enthält auch Informationen zur Ausführung der Beispielprogramme zum Setzen und Abfragen von Nachrichteneigenschaften:

- „Beispielprogramm CSQ4BCM1 ausführen” auf Seite 1299
- „Beispielprogramm CSQ4BCM2 ausführen” auf Seite 1300

Beispielprogramm CSQ4BCM1 ausführen

Für dieses Programm können bis zu vier Parameter angegeben werden:

1. Der Name der Zielwarteschlange (erforderlich)
2. Der Name des Warteschlangenmanagers (optional)
3. Optionen für das Öffnen (optional)

4. Optionen für das Schließen (optional)

Beispielprogramm CSQ4BCM2 ausführen

Für dieses Programm können bis zu sechs Parameter angegeben werden:

1. Der Name der Zielwarteschlange (erforderlich)
2. Der Name des Warteschlangenmanagers (optional)
3. Optionen für das Öffnen (optional)
4. Optionen für das Schließen (optional)
5. Der Name des Zielwarteschlangenmanagers (optional)
6. Der Name der dynamischen Warteschlange (optional)

Die Namen und Werte der Eigenschaften sowie der Nachrichteninhalt werden über die Standardeingabe (**`SYSIN DD`**) bereitgestellt. Für die Ausführung des Programms steht in CSQ4BCMP eine Beispiel-JCL zur Verfügung.

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 Informationen

MQ Telemetry

Telemetrieanwendungsfälle

MQ Telemetry installieren

MQ Telemetry verwalten

MQ Telemetry-Referenz

MQ Telemetry-Fehlerbehebung

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.

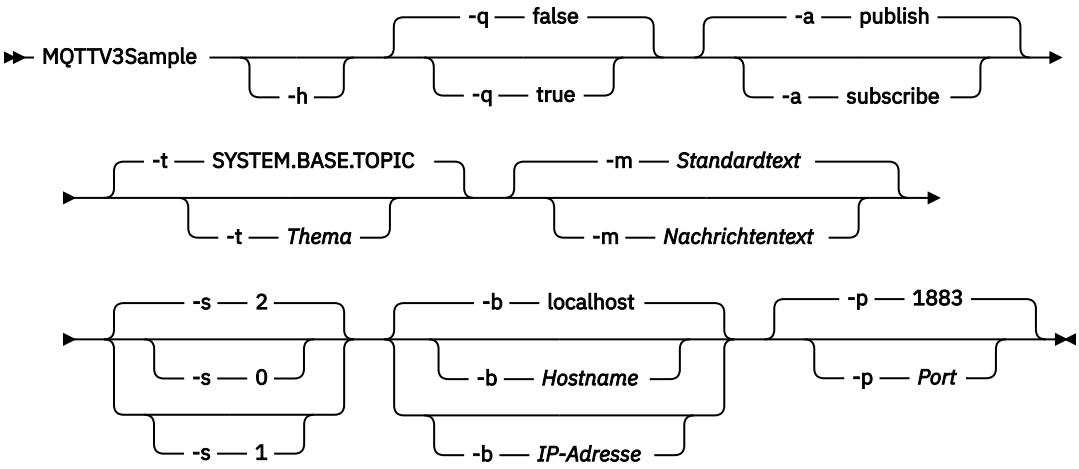
Programm 'MQTTV3Sample'

In diesem Abschnitt finden Sie Referenzinformationen zur Beispielsyntax und den Parametern für das Programm 'MQTTV3Sample'.

Verwendungszweck

Mit dem Programm 'MQTTV3Sample' kann eine Nachricht veröffentlicht und ein Thema subskribiert werden.

Syntax von 'MQTTV3Sample'



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:

```
runMQTTV3Sample -a subscribe
```

Veröffentlichen Sie eine Nachricht unter Windows mit folgendem Befehl:

```
runMQTTV3Sample
```

Weitere Informationen zur Ausführung des bereitgestellten Beispielscripts finden Sie im Abschnitt „IBM MQ Telemetry Transport-Beispielprogramme” auf Seite 1300.

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 Verbindungsverlustereignisse werden den Methoden in einer Callback-Klasse zugestellt, die `MqttCallback`implementiert.

Callbacks

Anmerkung: Die Website zu Eclipse Paho gibt Aufschluss über die neuesten Änderungen für `MqttCallback`. `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 zurück 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 dereferenziert das `MqttMessage` -Objekt, wenn die Zustellung abgeschlossen ist, und das Zustellungstokenobjekt, wenn die Clientverbindung getrennt wird. 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 für die vorherige Sitzung `MqttClient.CleanSession` auf `false` und in der neuen Sitzung auf `false` setzen. Der MQTT-Client erstellt neue Zustellungstokens und Nachrichtenobjekte in der neuen Sitzung für anstehende Übergaben. Er stellt die Objekte mithilfe der Klasse `MqttClientPersistence` wieder her. Falls der Anwendungsclient noch Referenzen auf die alten Zustellungstokens und Nachrichten enthält, löschen Sie diese Referenzen. Der Anwendungscallback wird in der neuen Sitzung für alle Übergaben aufgerufen, die in der früheren Sitzung eingeleitet wurden und in dieser Sitzung abgeschlossen werden.

Der Anwendungscallback wird nach Herstellung der Anwendungsclientverbindung aufgerufen, wenn eine anstehende Übergabe abgeschlossen ist. Bevor der Anwendungsclient die Verbindung herstellt, kann er anstehende Übergaben mit der Methode `MqttClient.getPendingDeliveryTokens` abrufen.

Beachten Sie, dass das Nachrichtenobjekt, das veröffentlicht wird, und dessen Bytefeldgruppe mit Nutzdaten ursprünglich von der 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 QoS 1 oder 2gesendet wird, wird die Nachricht 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 Referenzen auf das Thema und die Nachricht, wenn `messageArrived` zum MQTT -Client zurückkehrt. 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 176 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 176. 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`		
`MqttClient.Publish`	Kehrt nach der Übergabe der Anforderung an den MQTT-Client unverzüglich zum Anwendungsthread zurück.	Keine.

Tabelle 176. Synchronisationsverhalten von Methoden, die zu Anforderungen an den Server führen (Forts.)		
Methode	**Synchronisation**	**Zeitlimitintervall**
`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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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 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.

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 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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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 wird die Clientanwendung von `MqttCallback.deliveryComplete` nicht regulär an den MQTT-Client zurückgegeben. 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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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 beispielsweise `MQTT-Verwaltung/Verbindungen/`*`Server-URI`*`/`*`Client-ID`*`/Verloren` 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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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 MessageSight und 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 MessageSight und IBM MQ eingereiht. Wenn sie als `QoS=0` markiert ist, wird sie als nicht persistente Nachricht in IBM MessageSight und 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 für Java beschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference. Sie können die Schnittstellen auf ein Gerät angepasst implementieren. Der MQTT-Client, der 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` im Konstruktor `MqttClient` nicht angeben, speichert der MQTT -Client persistente Nachrichten unter Verwendung 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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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.

Veröffentlichung

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 oder IBM MessageSight ist für das Erstellen von Veröffentlichungen verantwortlich, die an einen MQTT-Subskribenten gesendet werden. Der Warteschlangenmanager überprüft, ob der von einem MQTT-Client erstellte Themenfilter in einer Subskription mit der Themenzeichenfolge in einer Veröffentlichung übereinstimmt. Es kann sich dabei um eine exakte Übereinstimmung oder um eine Übereinstimmung mit Platzhalterzeichen handeln. Vor dem Weiterleiten der Veröffentlichung an den Subskribenten durch den Warteschlangenmanager überprüft der Warteschlangenmanager die Themenattribute, die der Veröffentlichung zugeordnet sind. Er folgt dabei der im Abschnitt Subskription mithilfe einer Themenzeichenfolge mit Platzhalterzeichen beschriebenen Suchprozedur, um zu ermitteln, ob ein administratives Themenobjekt dem Benutzer die Berechtigung zur Subskription erteilt.

Wenn der MQTT-Client eine Veröffentlichung mit der Servicequalität "at least once" empfängt, ruft er zur Verarbeitung der Veröffentlichung die Methode `MqttCallback.messageArrived` auf. Wenn die Servicequalität der Veröffentlichung "exactly once" (QoS=2) ist, ruft der MQTT-Client die Schnittstelle `MqttClientPersistence` auf, um die Nachricht nach dem Empfang zu speichern. Anschließend wird die Methode `MqttCallback.messageArrived` aufgerufen.

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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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.

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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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:

- Ausführung des MQSC-Befehls **CLEAR TOPICSTR**.

- 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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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.Listen Sie Subskriptionen mit IBM MQ Exploreroder mit **runmqsc** -oder PCF-Befehlen auf. Alle MQTT-Clientsubskriptionen haben einen Namen. Sie erhalten einen Namen im folgenden Format: *`ClientIdentifier`*`:`*`Topic name`*

Wenn Sie den Standardwert `MqttConnectOptions`verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor der Client verbunden wird, werden alle alten Subskriptionen für den Client entfernt, wenn der Client eine Verbindung herstellt. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung auf `false` setzen, werden alle Subskriptionen, die der Client erstellt, zu allen Subskriptionen hinzugefügt, die für den Client vor dem Herstellen der Verbindung vorhanden waren. Alle Subskriptionen bleiben aktiv, wenn die Clientverbindung getrennt wird.

Eine andere Möglichkeit, um zu verstehen, wie sich das Attribut `cleanSession` auf Subskriptionen auswirkt, besteht darin, es sich als modales Attribut vorzustellen. Der Standardmodus (`cleanSession=true`) bedeutet, dass der Client nur im Rahmen der Sitzung Subskriptionen erstellt und Veröffentlichungen empfängt. Im alternativen Modus (`cleanSession=false`) sind Subskriptionen permanent. Der Client kann Verbindungen herstellen und trennen, seine Subskriptionen bleiben aktiv. Bei der Wiederherstellung einer Verbindung empfängt der Client alle nicht zugestellten Veröffentlichungen. Solange die Verbindung besteht, kann er die Gruppe der Subskriptionen, die in seinem Auftrag aktiv sind, ändern.

Sie müssen den Modus `cleanSession` festlegen, bevor Sie eine Verbindung herstellen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie die Modi von `cleanSession=false` in `cleanSession=true`ändern, werden alle vorherigen Subskriptionen für den Client und alle Veröffentlichungen, die nicht empfangen wurden, verworfen.

Veröffentlichungen, die aktiven Subskriptionen entsprechen, werden unmittelbar bei ihrer Veröffentlichung an den Client gesendet. Veröffentlichungen, die aktiven Subskriptionen entsprechen, werden unmittelbar bei ihrer Veröffentlichung an den Client gesendet. Wenn der Client nicht verbunden ist, werden sie an den Client gesendet, sobald dieser die Verbindung zu demselben Server wiederherstellt. Dabei

muss die Client-ID identisch sein und `MqttConnectOptions.cleanSession` muss auf `false` gesetzt sein.

Subskriptionen für einen bestimmten Client werden über die Client-ID ermittelt. Sie können den Client von einem anderen Clientgerät aus erneut mit demselben Server verbinden und weiterhin dieselben Subskriptionen nutzen und nicht zugestellte Veröffentlichungen empfangen.

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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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 wird in der Klasse `MqttClient` in der API-Dokumentation für den MQTT-Client für Java beschrieben.Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

Die Syntax ist bei allen Themenzeichenfolgen nahezu identisch. Es gibt vier geringfügige Unterschiede:

1. Bei Themenzeichenfolgen, die von MQTT-Clients an IBM MQ gesendet werden, muss die Konvention für Warteschlangenmanagernamen eingehalten werden.
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` nicht das Konzept eines verwalteten Themenobjekts. Es kann keine Themenzeichenfolge aus einem Themenobjekt und einer Themenzeichenfolge erstellt werden. Eine Themenzeichenfolge wird jedoch einem Verwaltungsthema in 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 wird im Abschnitt Themenbasiertes Platzhalterschema beschrieben. Die Syntax der Themenfilter, die Sie mit einem MQTT-Client erstellen können, wird in der Klasse `MqttClient` in der API-Dokumentation für den MQTT-Client für Java beschrieben.Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

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 Sitzungsbereinigungsmodus, indem Sie `MqttConnectOptions.cleanSession` festlegen, bevor Sie eine Verbindung herstellen.

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 (WCF = Microsoft Windows Communication Foundation) für IBM MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

Zugehörige Konzepte

„Einführung in die Verwendung des angepassten IBM MQ-Kanals für WCF mit .NET 3” auf Seite 1327

Übersicht über die Informationen, die für Programmierer verfügbar sind, die den angepassten IBM MQ-Kanal für Windows Communication Foundation (WCF) mit .NET 3 verwenden.

„Angepasste IBM MQ-Kanäle für WCF verwenden" auf Seite 1333
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 1354
Das Beispiel für Windows Communication Foundation (WCF) stellt einige einfache Beispiele zur Verwendung des angepassten IBM MQ-Kanals bereit.

„Problembestimmung im angepassten WCF-Kanal für IBM MQ" auf Seite 1360
Mit dem IBM MQ-Trace können Sie ausführliche Informationen zu den Funktionen der verschiedenen Teile des IBM MQ-Codes erfassen. Bei der Verwendung von Windows Communication Foundation (WCF) wird für den Trace des angepassten WCF-Kanals, der in den Trace der Microsoft WCF-Infrastruktur integriert ist, eine separate Traceausgabe erstellt.

Einführung in die Verwendung des angepassten IBM MQ-Kanals für WCF mit .NET 3

Übersicht über die Informationen, die für Programmierer verfügbar sind, die den angepassten IBM MQ-Kanal für Windows Communication Foundation (WCF) mit .NET 3 verwenden.

Was ist der angepasste IBM MQ-Kanal für WCF?

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 Microsoft Windows Communication Foundation-Framework ermöglicht .NET-Anwendungen und -Services, unabhängig vom Transport und den Protokollen entwickelt zu werden, die zu deren Verbindung verwendet werden, und es ermöglicht die Verwendung alternativer Transporte und Konfigurationen gemäß der Umgebung, in der dieser Service oder diese Anwendung implementiert ist.

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 1333.

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 Transport-

protokolle. Anwendungen erhalten somit Zugriff auf die Funktionen von IBM MQ innerhalb des einheitlichen WCF-Programmiermodells.

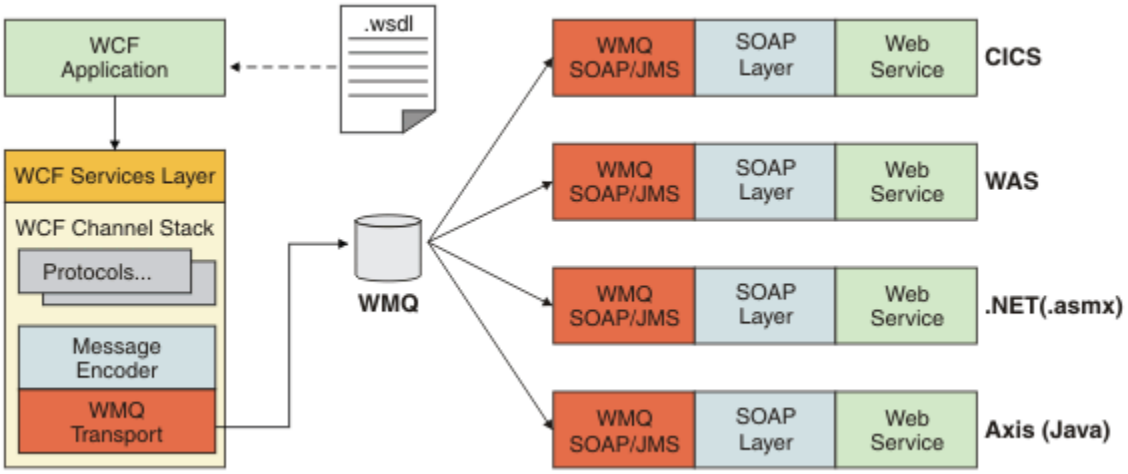
Typische Verwendungsmusterszenarien für den angepassten IBM MQ-Kanal für WCF:

- Als Schnittstelle zu Web-Services, die über IBM MQ gehostet werden (SOAP over JMS).
- Als Nicht-SOAP-Schnittstelle für die Übertragung nativer IBM MQ-Nachrichten.

Nachrichten, die mit dem SOAP over JMS-Format übertragen werden

Ein typisches Szenario für das Verwendungsmusters des angepassten IBM MQ-Kanals für WCF ist die Verwendung als Schnittstelle für Web-Services, die über IBM MQ (SOAP/JMS) bereitgestellt werden.

Nachrichten werden mit dem SOAP over JMS-Nachrichtenformat von IBM MQ übertragen. Dadurch können WCF-Clients und -Services auch andere WebSphere Application Server-Anwendungen oder -Hostumgebungen aufrufen oder von diesen aufgerufen werden, die mit diesem Format kompatibel sind. Dazu gehören Web-Services und Clients, die in CICS, Axis v1 (Java) und .asmx (.NET) ausgeführt werden, wie im folgenden Diagramm gezeigt wird:



Details zu SOAP over JMSfinden Sie unter „Web-Services mit IBM MQ-Transport für SOAP entwickeln” auf Seite 1369

Folgendes Beispiel ist ein typische Szenario aus dem Diagramm:

1. Ein Web-Service, der in WebSphere Application Server gehostet und über IBM MQ unter Verwendung der Unterstützung für SOAP over JMS in WebSphere Application Serververfügbar gemacht wird.
2. Das WSDL-Dokument, in dem der Service beschrieben wird, kann anschließend vom WCF-Tool zum Generieren eines Client-Proxys und einer Konfiguration verwendet werden, mit der anschließend ein entsprechender WCF-Kanal-Stack einschließlich des angepassten Kanals erstellt wird.
3. Die Clientanwendung verwendet daraufhin den Proxy, um den Web-Service auf die gleiche Weise wie jeden anderen Web-Service zu starten.

Der Kanal wird üblicherweise mit einem Encoder für WCF-Texte und SOAP-Nachrichten verwendet, aber der Kanal kann bei Bedarf auch mit anderen WCF-Nachrichtenencodern paarweise verbunden werden. Durch die Verwendung von alternativen Encodern kann die Integration in native IBM MQ-Anwendungen eingeschränkt sein, die SOAP over JMS nicht unterstützen, aber dies ist nicht die primäre Rolle des Kanals.

Die Verwendung des angepassten Kanals in einer WCF-Umgebung hat folgende wesentlichen Vorteile:

- Asynchroner Aufruf: Unterstützung von Fire-and-Forget-Clientoperationen, bei denen der Client von der Verfügbarkeit des Service und der Funktionen entkoppelt wird, z. B. bei der Weiterleitung von Antworten und dem Ansteuern über Multihopping.
- Zuverlässige Merkmale für die Skalierung: Die auf Warteschlangen basierende Nachrichtenübermittlung ermöglicht das vorhersehbare Hinzufügen von Kapazität zu einem System.

- Servicequalität: Nachrichten sind konkret und tracefähig und können ohne großen Aufwand gesteuert und verwaltet werden.

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 Windows, UNIX oder z/OS ausgeführt werden.

Softwarevoraussetzungen und Installationsanweisungen für den angepassten IBM MQ-Kanal für WCF

In diesem Abschnitt werden die Softwarevoraussetzungen und Installationsinformationen 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.

Softwarevoraussetzungen für den angepassten IBM MQ-Kanal für WCF

Diese Informationen enthalten die Softwarevoraussetzungen für den angepassten WCF-Kanal für IBM MQ.

Laufzeitumgebung

- Auf der Hostmaschine muss Microsoft.NET Framework V3.5 oder höher installiert sein.
- *Java and .NET Messaging and Web Services* wird standardmäßig als Teil des IBM MQ 8.0-Installationsprogramms installiert. Damit werden die für den angepassten Kanal erforderlichen .NET-Assemblys im Global Assembly Cache installiert.

Anmerkung: Wenn Microsoft.NET Framework V3.5 oder höher vor der Installation von IBM MQ 8.0 oder höher noch nicht installiert wurde, lässt sich das IBM MQ-Produkt zwar fehlerfrei installieren, jedoch steht der angepasste IBM MQ-Kanal danach nicht zur Verfügung. Wenn .NET Framework nach der Installation von IBM MQ 8.0 oder höher installiert wird, muss der angepasste Kanal IBM MQ aktiviert werden, indem das Script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd ausgeführt wird. Dabei ist *WMQInstallDir* das Verzeichnis, in dem IBM MQ 8.0 oder höher installiert ist. Mit diesem Script werden die erforderlichen Assemblys im Global Assembly Cache (GAC) installiert. Im Verzeichnis %TEMP% wird eine Gruppe von amqi*.log-Dateien erstellt, in denen die ausgeführten Aktionen dokumentiert werden. Es ist nicht erforderlich, das Script amqiRegisterdotNet.cmd erneut auszuführen, wenn für .NET ein Upgrade von einer früheren Version auf v3.5 oder höher durchgeführt wird, z. B. von .NET v2.0.

Entwicklungsumgebung

- Microsoft Visual Studio 2008 oder Windows Software Development Kit for .NET 3.5 oder höher.
- Zur Erstellung der Beispiellösungsdateien muss Microsoft.NET Framework V3.5 oder höher auf der Hostmaschine installiert sein.

Anmerkung: Wenn Microsoft.NET Framework V3.5 oder höher vor der Installation von IBM MQ 8.0 oder höher noch nicht installiert wurde, lässt sich das IBM MQ-Produkt zwar fehlerfrei installieren, jedoch steht der angepasste IBM MQ-Kanal danach nicht zur Verfügung. Wenn .NET Framework nach der Installation von IBM MQ 8.0 oder höher installiert wird, muss der angepasste Kanal IBM MQ aktiviert werden, indem das Script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd ausgeführt wird. Dabei ist *WMQInstallDir* das Verzeichnis, in dem IBM MQ 8.0 oder höher installiert ist. Mit diesem Script werden die erforderlichen

Assemblys im Global Assembly Cache (GAC) installiert. Im Verzeichnis %TEMP% wird eine Gruppe von `amqi*.log`-Dateien erstellt, in denen die ausgeführten Aktionen dokumentiert werden. Es ist nicht erforderlich, das Script `amqiRegisterdotNet.cmd` erneut auszuführen, wenn für .NET ein Upgrade von einer früheren Version auf v3.5 oder höher durchgeführt wird, z. B. von .NET v2.0.

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 von IBM MQ 8.0 oder höher installiert.

Angepasster IBM MQ-Kanal für WCF

Der angepasste IBM MQ-Kanal für WCF wird standardmäßig im Rahmen der Installation von IBM MQ 8.0 installiert. 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 Upgrade auf IBM MQ 8.0 aus einer früheren Version 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 177. 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

V 9.0.5

Wenn die Anforderungsnachricht den Typ MQSTR hat, können Sie ab IBM MQ 9.0.5 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). Beispiel:

```
"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). Beispiel:

```
"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 für IBM MQ. Weitere Informationen zum Beispiel für den angepassten IBM MQ-Kanal finden Sie unter „WCF-Beispiele verwenden” auf Seite 1354.

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 für 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 1351.

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:

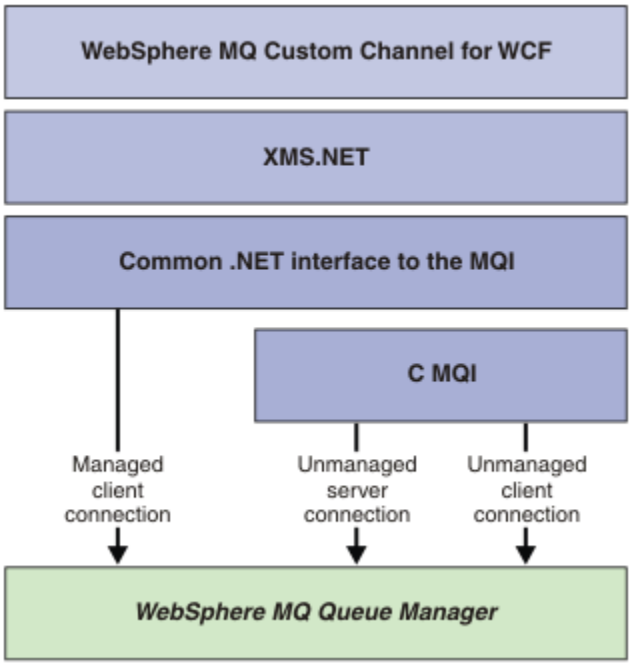


Abbildung 161. WCF-Architektur für die SOAP-/JMS-Schnittstelle

Für IBM WebSphere MQ 7.0.1 und höher sind alle erforderlichen Komponenten standardmäßig mit der Produktinstallation 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 1340.

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:

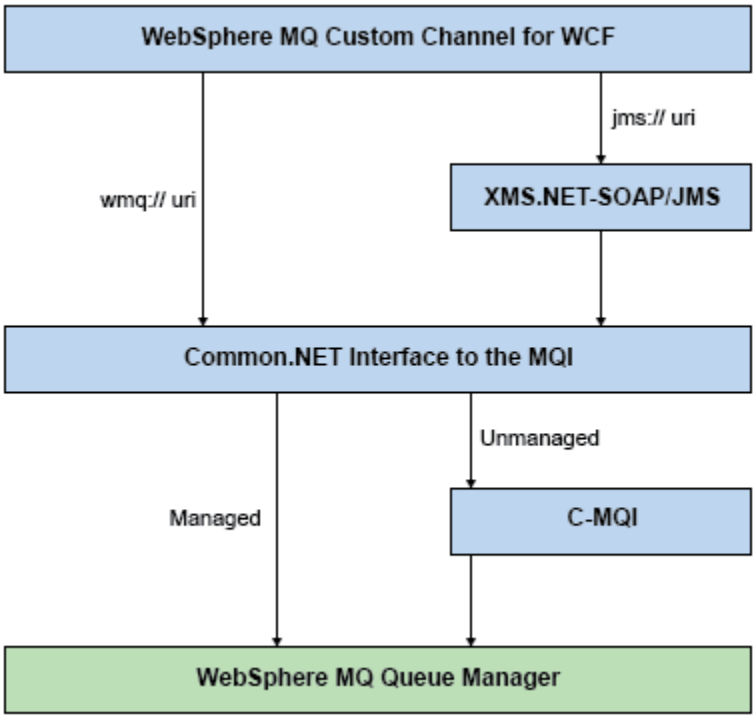


Abbildung 162. WCF-Architektur für eine andere Schnittstelle als eine SOAP-/JMS-Schnittstelle

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 WebSphere MQ 7.0 oder höher kann in WCF in .NET Framework 3 auf dieselbe Weise wie die integrierten Kanäle von Microsoftverwendet werden.

Nachrichten, die im angepassten Kanal übertragen werden, sind gemäß der SOAP over JMS-Implementierung von IBM WebSphere MQ 7.0 oder höher formatiert. Anwendungen können anschließend mit Services kommunizieren, die von WCF oder von der Serviceinfrastruktur von WebSphere SOAP over JMS gehostet werden. Weitere Informationen zu SOAP over JMS finden Sie unter „Web-Services mit IBM MQ-Transport für SOAP entwickeln” auf Seite 1369.

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. Beispiel:

```
    [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. Informationen zur Syntax dieses Parameters finden Sie im Abschnitt URI-Syntax und -Parameter für Web-Service-Bereitstellung.

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 (siehe „Service für WebSphere Application Server zur Verwendung von WebSphere Transport for SOAP implementieren” auf Seite 1424).

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

Beispiel:

```
"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.

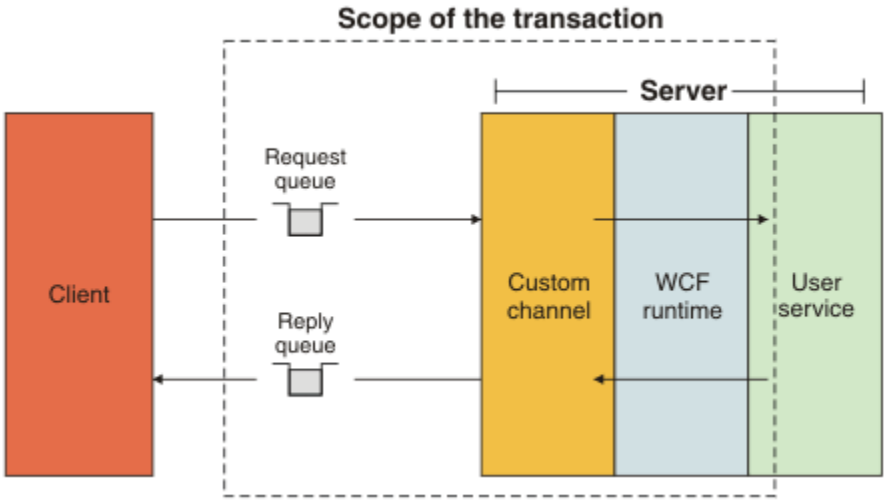
Eine Anforderungsnachricht kann unter einem Synchronisationspunkt für eine lokale Transaktion empfangen werden und von dort können alle Antwortnachrichten gesendet werden. Diese Nachrichten können im Falle eines Laufzeitfehlers rückgängig gemacht werden. Diese Fehler sind beispielsweise eine nicht behandelte 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. Weitere

Informationen zum Festlegen der URI-Eigenschaften finden Sie im Abschnitt URI-Syntax und -Parameter für die Web-Service-Bereitstellung.

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 1337.
- 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 1346.
- 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 Feh-

ler 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.

Es gibt zwei Möglichkeiten, TLS anzugeben:

- Geben Sie TLS direkt in der URI für SOAP over JMS an. Eine vollständige Beschreibung der TLS-Optionen finden Sie im Abschnitt TLS und der IBM MQ-Transport für SOAP.
- Geben Sie TLS über einen Eintrag in der Clientkanaldefinitionstabelle (CCDT) an. Weitere Informationen zur Definitionstabelle für den Clientkanal finden Sie in 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.

Sie können die Tabelle für die Kanaldefinition nicht direkt in der URI von SOAP over JMS angeben. 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.

Zugehörige Informationen

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 1335
- 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:

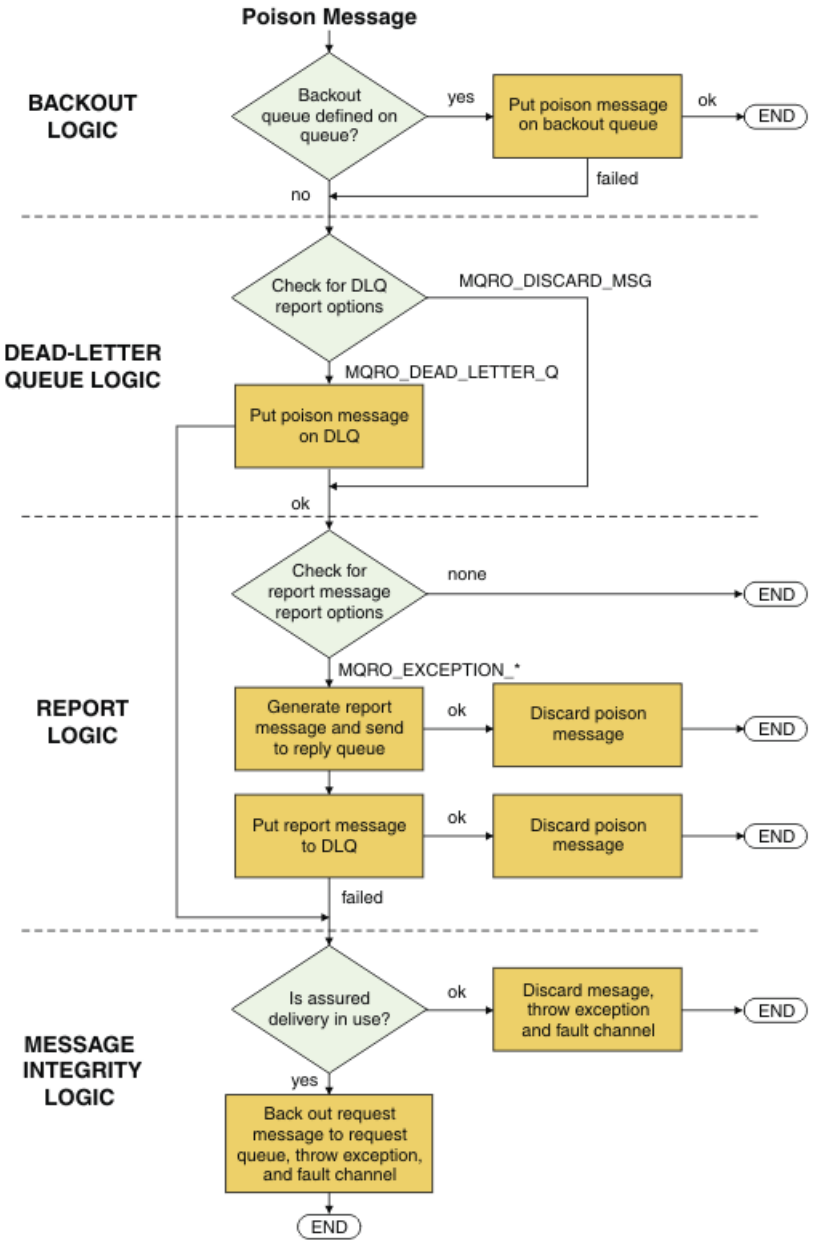
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

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 Verarbeitung von falsch formatierten 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 574

Weitere Informationen zur WCF-Architektur finden Sie unter „WCF-Architektur" auf Seite 1331

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 574.

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 1343).
- 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 178. 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 179. 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 1342 erstellt haben.
2. Nehmen Sie alle erforderlichen Änderungen an den Standardwerten der Eigenschaften wie in „Optionen zur WCF-Bindungskonfiguration" auf Seite 1346 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 1342 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 180. 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 1345

Tabelle 180. 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 181. 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 1345
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 1354.

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 1341).
- 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 1345.

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.

Weitere Informationen zum Festlegen der URI-Eigenschaften finden Sie im Abschnitt Syntax und Parameter der URI für die Implementierung des Web-Service.

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 entspricht der Spezifikation von SupportPac MA93. Unter SupportPac MA93: IBM MQ -Servicedefinition finden Sie weitere Details zu den IBM MQ IRI-Spezifikationen.

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 182. Werte von Bindungseigenschaften beim administrativen oder programmgesteuerten Festlegen

Eigenschaftsname	Client- oder Service-anwendung	Administrativer Wert	Programmgesteuerter Wert	Beschreibung
maxBufferPoolSize	Beide	Ganze Zahl mit Vorzeichen mit 0 bis 64 Bit	Ganze Zahl mit Vorzeichen mit 0 bis 64 Bit	Gibt die maximale Größe des Speichers an, der zum Speichern von WCF-Nachrichtenpuffern für eine Instanz des Kanals verwendet werden kann.
maxMessageSize	Beide	Ganze Zahl mit Vorzeichen mit 1 bis 32 Bit	Ganze Zahl mit Vorzeichen mit 1 bis 32 Bit	Gibt die maximale Speicherkapazität an, die für einzelne WCF-Nachrichten verwendet werden kann

Tabelle 182. 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 182. 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 1335). 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 1341 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 dem Namensbereich `IBM.XMS.WCF` eine Anweisung *using* hinzu, beispielsweise `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 1343 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 folgender Adresse verfügbar: `http://localhost:8000/MyService`

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 1352

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.

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:ExactlyOne>
                <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 1341 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 dem Namensbereich `IBM.XMS.WCF` eine Anweisung *using* hinzu, beispielsweise `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 1343 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 1351).

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 in der IBM MQ-Produktdokumentation unter *IBM MQ Transport for SOAP*. 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 1351).

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 in der IBM MQ-Produktdokumentation unter *IBM MQ Transport for SOAP*. 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. Informationen zur Installation und Konfiguration der Umgebung finden Sie unter „IBM MQ Web Transport for SOAP installieren” auf Seite 1380

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, beispielsweise `C:\Programme\IBM\MQ`.
2. Stellen Sie sicher, dass der Java-Compiler `javac` im Pfad verfügbar ist.
3. Kopieren Sie die Datei `axis.jar` aus dem Verzeichnis `prereqs/axis` der WebSphere-Installations-CD in das IBM MQ-Produktionsverzeichnis, beispielsweise `C:\Programme\IBM\MQ\java\lib\soap`.
4. Fügen Sie dem Pfad folgende Information hinzu: *MQ_INSTALLATION_PATH*`\Java\lib`. Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem IBM MQ installiert ist, beispielsweise `C:\Programme\IBM\MQ`.
5. Stellen Sie sicher, dass die Position von .NET in *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` korrekt angegeben ist. Dabei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem IBM MQ installiert ist, beispielsweise `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 in Ihr Arbeitsverzeichnis von SOAP over JMS.
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

In diesem Beispiel wird eine Verbindung von einem WCF-Client zu dem in IBM MQ bereitgestellten Beispiel für den .NET SOAP over JMS-Service in Form eines Einwegkanals gezeigt. 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 1352 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 in der Clientanwendung in der Datei *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\app.config` und in 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 für 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. Informationen zur Installation und Konfiguration der Umgebung finden Sie unter „IBM MQ Web Transport for SOAP installieren” auf Seite 1380

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, beispielsweise `C:\Programme\IBM\MQ`.
2. Stellen Sie sicher, dass der Java-Compiler `javac` im Pfad verfügbar ist.

3. Kopieren Sie die Datei `axis.jar` aus dem Verzeichnis `prereqs/axis` der Installations-CD von WebSphere in das Installationsverzeichnis von IBM MQ.
4. Fügen Sie dem Pfad folgende Information hinzu: *MQ_INSTALLATION_PATH*`\Java\lib`. Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem IBM MQ installiert ist, beispielsweise `C:\Programme\IBM\MQ`.
5. Stellen Sie sicher, dass die Position von .NET in *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` korrekt angegeben ist. Dabei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem IBM MQ installiert ist, beispielsweise `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 in Ihr Arbeitsverzeichnis von SOAP over JMS.
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

Im Beispiel wird eine Verbindung von einem WCF-Client zu dem in IBM MQ bereitgestellten Beispiel für den Axis Java-Service für SOAP over JMS in Form eines Einwegkanals gezeigt. 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 1352 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.

Zugehörige Verweise

„Handhabung unterschiedlicher Namen für SOAP-Antwortelemente” auf Seite 1367
In WCF wird erwartet, dass der Name eines zurückgegebenen Werts standardmäßig ein bestimmtes Format hat, aber ein Service gibt möglicherweise kein Element mit einem Namen im erwarteten Format zurück.

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 183. 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 in WebSphere Application Server einen Nachrichten-Listener-Port mit der Bezeichnung `HelloWorldPort` mit folgender 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 `HelloWorldEJBEAR.ear`-Anwendung 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`. Datei 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 1351).

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\wcf\samples\WAS\default\client\app.config` und in 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. Weitere Informationen zum Format des JMS-Endpunkt-URI finden Sie im Abschnitt URI-Syntax und -Parameter für die Web-Service-Bereitstellung.

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.

Problembestimmung im angepassten WCF-Kanal für IBM MQ

Mit dem IBM MQ-Trace können Sie ausführliche Informationen zu den Funktionen der verschiedenen Teile des IBM MQ-Codes erfassen. Bei der Verwendung von Windows Communication Foundation (WCF) wird für den Trace des angepassten WCF-Kanals, der in den Trace der Microsoft WCF-Infrastruktur integriert ist, eine separate Traceausgabe erstellt.

Durch das vollständige Aktivieren des Trace für den angepassten WCF-Kanal werden zwei Ausgabedateien erzeugt:

1. Der Trace für den angepassten WCF-Kanal, der in den Trace der Microsoft WCF-Infrastruktur integriert ist.
2. Der Trace für den angepassten WCF-Kanal, der in XMS .NET integriert ist.

Da zwei Traceausgaben vorhanden sind, können Probleme mithilfe der entsprechenden Tools an jeder Schnittstelle überwacht werden. Beispiel:

- WCF-Problembestimmung mit geeigneten Microsoft-Tools.
- Ausgaben aus dem IBM MQ MQI client im XMS-Traceformat.

Zur Vereinfachung der Traceaktivierung werden die .NET 3-Tracequelle und der XMS .NET-Trace-Stack über eine einzelne Schnittstelle gesteuert. Informationen dazu finden Sie unter „WCF-Tracekonfiguration und Namen von Tracedateien: SOAP-/JMS-Schnittstelle” auf Seite 1362.

Hierarchie der Ausnahmen im angepassten WCF-Kanal

Die vom angepassten Kanal ausgelösten Ausnahmetypen sind mit WCF konsistent und sind üblicherweise eine Ausnahme vom Typ 'TimeoutException' oder 'CommunicationException' (oder eine Unterklasse von 'CommunicationException'). Weitere Details der Fehlerbedingung (soweit verfügbar) werden mithilfe von verknüpften Ausnahme oder mit der Ausnahme 'inner exception' bereitgestellt.

SOAP/JMS-Schnittstelle

Die folgenden Ausnahmen sind typische Beispiele für Ausnahmen. Jede Ebene in der Architektur des Kanals ergänzt eine zusätzliche verknüpfte Ausnahme; so enthält 'CommunicationsException' beispielsweise eine verknüpfte XMSException-Ausnahme, die eine verknüpfte MQException-Ausnahme enthält:

1. System.serviceModel.CommunicationsExceptions
2. IBM.XMS.XMSException
3. IBM.WMQ.MQException

Wichtige Informationen werden erfasst und in der Datensammlung der höchsten CommunicationException-Ausnahme in der Hierarchie bereitgestellt. Durch diese Erfassung und Bereitstellung müssen die Anwendungen keine Verknüpfung zu jeder Ebene in der Architektur des Kanals herstellen, um die verknüpften Ausnahmen und weitere zusätzliche Informationen abzufragen, die möglicherweise darin enthalten sind. Die folgenden Schlüsselnamen sind definiert:

- IBM.XMS.WCF.ErrorCode: Der Fehlernachrichtencode der aktuellen Ausnahme des angepassten Kanals.
- IBM.XMS.ErrorCode: Die Fehlernachricht der ersten XMS-Ausnahme im Stack.
- IBM.WMQ.ReasonCode: Der zugrunde liegende IBM MQ-Ursachencode.
- IBM.WMQ.CompletionCode: Der zugrunde liegende IBM MQ-Beendigungscode.

Andere Schnittstelle als SOAP-/JMS-Schnittstellen

Die folgenden Ausnahmen sind typische Beispiele für Ausnahmen. Jede Ebene in der Architektur des Kanals ergänzt eine zusätzliche verknüpfte Ausnahme; beispielsweise enthält die CommunicationsException-Ausnahme eine verknüpfte MQException-Ausnahme:

1. System.ServiceModel.CommunicationsExceptions
2. IBM.WMQ.MQException

Wichtige Informationen werden erfasst und in der Datensammlung der höchsten CommunicationException-Ausnahme in der Hierarchie bereitgestellt. Durch diese Erfassung und Bereitstellung müssen die Anwendungen keine Verknüpfung zu jeder Ebene in der Architektur des Kanals herstellen, um die verknüpften Ausnahmen und weitere zusätzliche Informationen abzufragen, die möglicherweise darin enthalten sind. Die folgenden Schlüsselnamen sind definiert:

- IBM.WMQ.WCF.ErrorCode: Der Fehlernachrichtencode der aktuellen Ausnahme des angepassten Kanals.

- IBM.WMQ.ReasonCode: Der zugrunde liegende IBM MQ-Ursachencode.
- IBM.WMQ.CompletionCode: Der zugrunde liegende IBM MQ-Beendigungscode.

WCF-Tracekonfiguration

Es gibt zwei Möglichkeiten der WCF-Tracekonfiguration. Sie können den Trace programmgesteuert oder über eine Umgebungsvariable konfigurieren.

WCF-Tracekonfiguration und Namen von Tracedateien: SOAP-/JMS-Schnittstelle

Wenn die Tracefunktion vollständig aktiviert ist, werden zwei Ausgabedateien erstellt: Eine Datei zur Diagnose von WCF-Problemen und eine ausführliche Datei für interne Diagnoseinformationen zum Trace. Zum Vereinfachung der Traceaktivierung verwenden die Tracequelle von .NET 3 und die XMS .NET-Trace-Stacks eine einzige Schnittstelle.

Für den angepassten WCF-Kanal sind zwei unterschiedliche Tracemethoden verfügbar. Die beiden Tracemethoden können unabhängig voneinander oder gemeinsam aktiviert werden. Jedes Verfahren liefert eine eigene Tracedatei, sodass zwei Traceausgabedateien erstellt werden, wenn beide Traceverfahren aktiviert wurden.

Um die Konfiguration und Aktivierung so einfach wie möglich zu gestalten, wird zum Steuern beider Tracemethoden die gleiche Schnittstelle verwendet. Die Datei `app.config` muss bearbeiten werden, damit die relevante Tracekonfiguration wie im folgenden Abschnitt beschrieben eingeschlossen werden kann. Benutzer können anschließend eigene entsprechende Abschnitte hinzufügen, um die Ausgabe mit dem Trace aus ihrer eigenen Anwendung zu kombinieren.

Die Tracefunktion für den angepassten WCF-Kanal ist nicht standardmäßig aktiviert. Sie müssen zuerst einen Trace-Listener erstellen und anschließend die erforderliche Tracestufe für die ausgewählte Tracequelle in der Datei `app.config` festlegen.

Angepassten WCF-Kanal mit WCF-Infrastrukturtrace konfigurieren

Fügen Sie den folgenden Codeabschnitt dem Abschnitt `<system.diagnostics><sources>` in der Datei `app.config` hinzu:

```
<source name="IBM.XMS.WCF" switchValue="Verbose,ActivityTracing">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

Durch den vorherigen Codeteil wird erreicht, dass der Kanaltrace die Tracequelle von .NET 3 verwendet. Alle Aufrufe von Konfigurationsdateien, die den ausführbaren Dateien zugeordnet sind, werden von diesem Codeteil gesteuert.

Angepassten WCF-Kanal mit XMS .NET-Trace konfigurieren

Bei der Konfiguration des XMS .NET-Trace müssen Sie dem Abschnitt `<system.diagnostics><sources>` in der Datei `app.config` einen Codeabschnitt hinzufügen. Der Codeteil wird allerdings dem erweiterbaren Element `<source>` hinzugefügt, wie im Abschnitt Angepassten WCF-Kanal mit WCF-Infrastrukturtrace konfigurieren beschrieben ist. Obwohl der Trace-Code für die WCF-Infrastruktur zur Ausführung des XMS .NET-Trace vorhanden sein muss, kann also der WCF-Infrastrukturtrace inaktiviert werden, falls er nicht erforderlich ist. Dies wird im Abschnitt WCF-Trace aktivieren beschrieben.

```
<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing"
 xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path"
 xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
```

```
    </listeners>
</source>
```

Konfigurationsvariablen für den WCF-Trace

Tabelle 184. Konfigurationsvariablen für den WCF-Trace

Variable	Beschreibung
Name	Geben Sie folgenden Namen an: `IBM.XMS.WCF`
switchValue	Die Variable 'switchValue' steuert die Tracestufe. Wenn 'switchValue' auf `Off` gesetzt ist, wird die WCF-Infrastruktur 'TraceSource' nicht generiert. 'TraceSource' wird von einem anderen Wert (z. B. `Verbose`' generiert. Ausführliche Informationen zu den Tracestufen von Microsoft finden Sie in Ihrer WCF-Dokumentation oder auf der Webseite für Microsoft WCF Tracing unter https://msdn.microsoft.com/en-us/library/ms733025(vs.85).aspx

xmsTraceSpecification= *Komponentenname* = *Typ* = *Status*	*Komponentenname* ist der Name der Klasse, für die Sie einen Trace erstellen möchten. In diesem Namen können Sie ein Platzhalterzeichen (*) verwenden. Beispiel:
	`*=all=enabled`
	Gibt an, dass für alle Klassen ein Trace erstellt werden soll.
	`IBM.XMS.impl.*=all=enabled`
	Gibt an, dass nur der API-Trace erforderlich ist.
	Typ kann einer der folgenden Tracetypen sein:
	• alle
	• Debug
	• Ereignis
	• EntryExit
	Status kann aktiviert oder inaktiviert sein.

xmsTraceFilePath=" *Dateiname* "	Wenn Sie die Variable 'xmsTraceFilePath' nicht angeben ist oder wenn 'xmsTraceFilePath' vorhanden ist, aber eine leere Zeichenfolge enthält, wird der Trace im aktuellen Verzeichnis gespeichert. Zum Speichern der Tracedatei in einem benannten Verzeichnis geben Sie den Verzeichnisnamen in 'xmsTraceFilePath'; Beispiel: `xmsTraceFilePath="c:\somepath"`

Tabelle 184. Konfigurationsvariablen für den WCF-Trace (Forts.)	
Variable	**Beschreibung**
xmsTraceFileSize=" *Größe* "	Die maximal zulässige Größe der Tracedatei. Wenn eine Datei diese Größe erreicht, wird sie archiviert und umbenannt. Die standardmäßig festgelegte maximale Größe liegt bei 20 KB und wird folgendermaßen angegeben: `xmsTraceFileSize="20000000".`
xmsTraceFileNumber=" *Anzahl* "	Die Anzahl der Tracedateien, die aufbewahrt werden sollen. Der Standardwert ist 4 (eine aktive Datei und drei Archivdateien). Die zulässige Mindestanzahl ist zwei.

xmsTraceFormat=" *Format* "	Die Variable 'xmsTraceFormat' liegt in zwei Ebenen vor: `basic` (Basis) und `advanced` (Erweitert). Das standardmäßige Traceformat ist 'basic', wenn Sie keine xmsTraceFormat-Variable angeben oder wenn die xmsTraceFormat-Variable vorhanden ist, aber eine leere Zeichenfolge enthält>. Tracedateien werden in diesem Format erzeugt, wenn Sie Folgendes angeben:
	`xmsTraceFormat="basic"`
	Wenn ein Trace erforderlich ist, der mit den Tools für die Traceanalysefunktion kompatibel ist, müssen Sie Folgendes angeben:
	`traceFormat="advanced"`

WCF-Trace aktivieren

Es gibt vier mögliche Kombinationen für das Aktivieren und Inaktivieren der beiden Traceverfahren. Für die vier Kombinationen müssen die Werte der in den vorherigen Abschnitten beschriebenen Code-Abschnitten bearbeitet werden.

Es ist auch eine Umgebungsvariable vorhanden, die bearbeitet werden kann. Weitere Informationen dazu finden Sie unter „WCF-Trace mit der Umgebungsvariablen WCF_TRACE_ON aktivieren” auf Seite 1365.

Diese Tabelle und die gezeigten Werte sind davon abhängig, ob die zuvor gezeigten Codeteile der Datei `app.config` bereits hinzugefügt wurden.

Tabelle 185. Kombinationen zur Aktivierung des WCF-Trace

Tracetyp	Geänderter Wert	Beispiel

XMS-Trace ist aktiviert. WCF-Tracequelle ist aktiviert.	Die Variable `switchValue` ist nicht auf `Off` gesetzt.	`<source name="IBM.XMS.WCF" switchValue=" Verbose, Activi tyTracing"`
		`xmsTraceSpecification="*=all=enabled" xmsTraceFile Path="path"`
		`xmsTraceFileSize="2000000" xmsTraceFileNumber="4"`
		`xmsTraceFormat="advanced">`
		`<listeners>`
		`<remove name="Default"/>`
		`<add name="NewListener"/>`
		`</listeners>`
		`</source>`

Tabelle 185. Kombinationen zur Aktivierung des WCF-Trace (Forts.)

Tracetyp	Geänderter Wert	Beispiel
XMS-Trace ist aktiviert. WCF-Tracequelle ist inaktiviert.	Der Wert `switchValue` ist auf `Off` gesetzt und `xmsTraceSpecification` wurde angegeben.	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
XMS-Trace ist inaktiviert. WCF-Tracequelle ist aktiviert.	Dieses Ergebnis kann auf zwei Arten erreicht werden: • Die Variable `switchValue` ist nicht auf `Off` gesetzt und es wurde keine `xmsTraceSpecification`-Variable hinzugefügt. • Die Variable `switchValue` ist nicht auf `Off` gesetzt und die Variable `xmsTraceSpecification` ist auf `inaktiviert` gesetzt.	`<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
XMS-Trace ist inaktiviert. WCF-Tracequelle ist inaktiviert.	Dieses Ergebnis kann auf drei Arten erreicht werden: • In der Datei `app.config` befindet sich kein `<source>`-Element. • Die Variable `switchValue` ist auf `Off` gesetzt und es wurde keine `xmsTraceSpecification`-Variable hinzugefügt. • Die Variable `switchValue` ist auf `Off` gesetzt und die Variable `xmsTraceSpecification` ist auf `inaktiviert` gesetzt.	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

WCF-Trace mit der Umgebungsvariablen WCF_TRACE_ON aktivieren

Wie auch in den zuvor beschriebenen Methoden zum Aktivieren des WCF-Trace kann auch der XMS .NET-Trace mithilfe der Umgebungsvariablen WCF_TRACE_ON aktiviert werden.

Das Festlegen der Umgebungsvariable WCF_TRACE_ON auf einen Wert ungleich null entspricht dem Festlegen von `xmstraceSpecification` auf den Wert `*=all=enabled`, Beispiel: " `set WCF_TRACE_ON=true` "

Wenn allerdings `xmstraceSpecification` in der Datei `app.config` explizit festgelegt ist, wird die Umgebungsvariable WCF_TRACE_ON überschrieben.

Ausgabedateien und Dateinamen für den WCF-Trace

XMS -Tracedateien werden traditionell mit dem Basisnamen und dem Prozess-ID-Format `xms_trace_`*`pid`*`.log`benannt, wobei *pid* die Prozess-ID ist.

Da die XMS -Tracedateien weiterhin parallel zu den Tracedateien des angepassten WCF-Kanals erstellt werden können, hat der Trace des angepassten WCF-Kanals, der in XMS .NET -Traceausgabedateien integriert ist, das folgende Format, um Unklarheiten zu vermeiden: `wcf xms_trace_`*`pid`*`.log`, wobei *pid* die Prozess-ID ist.

Die Traceausgabedatei wird standardmäßig im aktuellen Arbeitsverzeichnis erstellt, aber dieses Ziel kann bei Bedarf neu definiert werden.

WCF-Tracekonfiguration: Nicht-SOAP/Nicht-JMS-Schnittstelle

Für die Nicht-SOAP/Nicht-JMS-Schnittstelle können Sie den Trace entweder über eine Umgebungsvariable oder programmgesteuert konfigurieren.

Es gibt zwei Möglichkeiten, den Trace für die Nicht-SOAP-/Nicht-JMS-Schnittstelle zu aktivieren:

- Durch Festlegung von WMQ_TRACE_ON als Umgebungsvariable
- Durch Aufnahme des folgenden Codeabschnitts in den Abschnitt `<system.diagnostics><sources>` der Datei 'app.config'

```
<source name="IBM.WMQ.WCF" switchValue="Verbose, ActivityTracing"
xmsTraceSpecification="*=all=enabled"
xmsTraceFileSize="2000000" xmsTraceFileNumber="4"
xmsTraceFormat="advanced">
</source>
```

WCF XMS First Failure Support Technology (FFST)

Mit dem IBM MQ-Trace können Sie ausführliche Informationen zu den Funktionen der verschiedenen Teile des IBM MQ-Codes erfassen. XMS FFST verfügt über eigene Konfigurations- und Ausgabedateien für den angepassten WCF-Kanal.

XMS-FFST-Tracedateien werden üblicherweise mithilfe des Basisnamens und der Prozess-ID mit folgendem Format benannt: `xmsffdc` *`pid_date`*`.txt`. Dabei steht *PID* für die Prozess-ID und *Datum* für die Uhrzeit und das Datum.

Da XMS-FFST-Tracedateien weiterhin parallel zu den FFST-Dateien für den angepassten WCF-Kanal erstellt werden können, haben die FFST-Ausgabedateien des angepassten WCF-Kanals das folgende Format, um Verwechslungen zu vermeiden: `wcf ffdc` *`pid_date`*`.txt`. Dabei steht *PID* für die Prozess-ID und *Datum* für die Uhrzeit und das Datum.

Diese Traceausgabedatei wird standardmäßig im aktuellen Arbeitsverzeichnis erstellt, aber dieses Ziel kann bei Bedarf neu definiert werden.

Der Header für den angepassten WCF-Kanal mit dem XMS .NET-Trace entspricht weitgehend dem folgenden Beispiel:

```
************* Start Display XMS WCF Environment *************
Product Name :- value
WCF Version  :- value
Level  :- value
************* End Display XMS WCF Environment *************
```

Die FFST-Tracedateien sind standardmäßig formatiert, ohne spezielle Formatierung für den angepassten Kanal.

WCF-Versionsinformationen

WCF-Versionsinformationen unterstützen Sie bei der Problembestimmung. Sie sind in die Assembly-Metadaten des angepassten Kanals integriert.

Der angepasste IBM MQ-Kanal für Metadaten der WCF-Version kann auf eine von drei Arten abgerufen werden:

- Mit dem IBM MQ-Dienstprogramm 'dspmqver'. Informationen zur Verwendung von 'dspmqver' finden Sie unter dspmqver.
- Mit dem Dialog 'Eigenschaften' in Windows Explorer: Klicken Sie in Windows mit der rechten Maustaste auf **IBM.XMS.WCF.dll** > **Eigenschaften** > **Version**.

- Aus den Headerinformationen einer FFST-Datei des Kanals oder aus Tracedateien. Weitere Informationen zu den FFST-Headerinformationen finden Sie unter „WCF XMS First Failure Support Technology (FFST)" auf Seite 1366.

Hinweise und Tipps zu WCF

Die folgenden Hinweise und Tipps sind in keiner maßgeblichen Reihenfolge und werden bei einem Release neuer Versionen der Dokumentation möglicherweise hinzugefügt. Es handelt sich dabei um Themen, mit denen Sie Zeit sparen können, wenn sie für die von Ihnen ausgeführte Arbeit relevant sind.

Ausnahmebedingungen aus dem WCF-Diensthost auslagern

Bei Services, die mithilfe des WCF-Diensthosts bereitgestellt werden, werden die vom Service, internen WCF-Daten oder Channel-Stacks ausgelösten nicht behandelten Ausnahmen nicht standardmäßig ausgelagert. Damit Sie über diese Ausnahmen informiert werden, muss eine Fehlerbehandlungsroutine registriert sein.

Im folgenden Code wird ein Beispiel für die Definition des Serviceverhaltens der Fehlerbehandlungsroutine bereitgestellt, die als Attribut eines Service angewendet werden kann:

```
using System.ServiceModel.Dispatcher;
using System.Collections.ObjectModel;
....
    public class ErrorHandlerBehaviorAttribute : Attribute, IServiceBehavior, IErrorHandler
    {
        //
        // IServiceBehavior Interface
        //
        public void AddBindingParameters(ServiceDescription serviceDescription,
           ServiceHostBase serviceHostBase, CollectionServiceEndpoint endpoints,
           BindingParameterCollection bindingParameters)
        {
        }
        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
            ServiceHostBase serviceHostBase)
        {
            foreach (ChannelDispatcher channelDispatcher in serviceHostBase.ChannelDispatchers)
{
                channelDispatcher.ErrorHandlers.Add(this);
            }
        }
        public void Validate(ServiceDescription serviceDescription, ServiceHostBase serviceHost
Base)
        {
        }

        //
        // IErrorHandler Interface
        //
        public bool HandleError(Exception e)
        {
            // Process the exception in the required way, in this case just outputting to the
console
            Console.Out.WriteLine(e);

            // Always return false to allow any other error handlers to run
            return false;
        }
        public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
        {
        }
    }
```

Handhabung unterschiedlicher Namen für SOAP-Antwortelemente

In WCF wird erwartet, dass der Name eines zurückgegebenen Werts standardmäßig ein bestimmtes Format hat, aber ein Service gibt möglicherweise kein Element mit einem Namen im erwarteten Format zurück.

In WCF gilt die Konvention, dass für den Namen des zurückgegebenen Werts das folgende Format erwartet wird: *Methodenname* `Result`. Dabei ist *Methodenname* der Name der Serviceoperation. Für

einen Service mit der Bezeichnung `getQuote` erwartet WCF eine Antwort mit der Bezeichnung `getQuoteResult`.

Der Service kann allerdings ein Element mit einem Namen zurückgeben, der diesem Format nicht entspricht.

Wenn das Tool 'scvutil' zum Generieren eines Proxy-Clients ausgeführt wird und die WSDL einen anderen Namen angibt, fügt die Proxy-Schnittstelle Parameter hinzu, um WCF über den Namen zu informieren, der gesucht wird. Beispiel:

```
[System.ServiceModel.OperationContractAttribute(Action = "", ReplyAction = "*")]
[System.ServiceModel.XmlSerializerFormatAttribute(Style = System.ServiceModel.OperationFormat
Style.Rpc,
                                                  Use = System.ServiceModel.OperationFormatU
se.Encoded)]
[return: System.ServiceModel.MessageParameterAttribute(Name = "getQuoteReturn")]
float getQuote(string in0);
```

Wenn Sie Ihre eigene Schnittstelle erstellen (beispielsweise durch Hinzufügen einer Anforderungs-/Antwortmethode zu einer vorhandenen Proxy-Schnittstelle), müssen Sie sicherstellen, dass Sie der Schnittstelle die gleichen Parameter hinzufügen, wenn der Service einen anderen Namen zurückgibt. Wenn Sie dies nicht tun, wird als häufigster Fehler beim Aufrufen einer Servicemethode ein Nullwert zurückgegeben; wenn ein Objekt zurückgegeben wird, gibt die Methode null zurück, aber wenn ein numerischer Wert wie beispielsweise eine ganze Zahl zurückgegeben wird, gibt die Methode den Wert 0 zurück.

Web-Services mit IBM MQ entwickeln

Sie können IBM MQ-Anwendungen für Web-Services mithilfe des IBM MQ-Transports für SOAP entwickeln.

Informationen zu diesem Vorgang

Anmerkung: Ab IBM MQ 9.0 wird IBM MQ Transport for SOAP nicht mehr verwendet. Dies schließt folgende Produktfunktionen ein:

- IBM MQ Java-Listener
- IBM MQ .NET 1 und 2 Listener
- IBM MQ Java Axis2-Client
- IBM MQ Java-Client (Einstellung der Unterstützung angekündigt in IBM MQ 8.0)
- IBM MQ .NET 1- und 2-Clients (Einstellung der Unterstützung angekündigt in IBM MQ 8.0)
- IBM MQ bridge for HTTP (Einstellung der Unterstützung angekündigt in IBM MQ 8.0)

Der IBM MQ-Transport für SOAP stellt einen JMS-Transport für SOAP zur Verfügung. Der IBM MQ-Transport für SOAP wird auch in andere Umgebungen wie Microsoft Windows Communication Foundation, WebSphere Application Server und CICS Transaction Server integriert.

Sie finden weitere Informationen zum IBM MQ-Transport für SOAP finden Sie im Abschnitt „Web-Services mit IBM MQ-Transport für SOAP entwickeln” auf Seite 1369.

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 6
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM MQ-Anwendungen zu schreiben. Über die in diesem Abschnitt enthaltenen Links erhalten Sie Informationen über IBM MQ-Konzepte, die für Anwendungsentwickler hilfreich sind.

„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 48

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 743
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 953
Informationen zum Schreiben von Clientanwendungen unter IBM MQ in einer prozeduralen Programmiersprache.

„Publish/Subscribe-Anwendungen schreiben" auf Seite 840
Beginnen Sie nun, Ihre eigenen IBM MQ-Publish/Subscribe-Anwendungen zu schreiben.

„Prozedurale Anwendung erstellen" auf Seite 1051
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 1103
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 1124
Diese Beispielprogramme sind in prozeduralen Programmiersprachen geschrieben und veranschaulichen typische Verwendungen der Message Queue Interface (MQI). Es gibt IBM MQ-Programme für verschiedene Plattformen.

Web-Services mit IBM MQ-Transport für SOAP entwickeln

IBM MQ-Transport für SOAP stellt einen JMS-Transport für SOAP bereit. Der IBM MQ-Transport für SOAP wird auch in andere Umgebungen wie Microsoft Windows Communication Foundation, WebSphere Application Server und CICS Transaction Server integriert.

Informationen zu diesem Vorgang

Anmerkung: Ab IBM MQ 9.0 wird IBM MQ Transport for SOAP nicht mehr verwendet. Dies schließt folgende Produktfunktionen ein:

- IBM MQ Java-Listener
- IBM MQ .NET 1 und 2 Listener
- IBM MQ Java Axis2-Client
- IBM MQ Java-Client (Einstellung der Unterstützung angekündigt in IBM MQ 8.0)
- IBM MQ .NET 1- und 2-Clients (Einstellung der Unterstützung angekündigt in IBM MQ 8.0)
- IBM MQ bridge for HTTP (Einstellung der Unterstützung angekündigt in IBM MQ 8.0)

Einführung in IBM MQ Transport for SOAP

Der IBM MQ-Transport für SOAP stellt einen JMS-Transport für SOAP zur Verfügung. Der IBM MQ-SOAP-Sender und -Listener bieten ein Verfahren für den Aufruf von Web-Services.

Der IBM MQ-SOAP-Listener unterstützt Services, die von .NET Framework 1, .NET Framework 2 und Axis 1.4 gehostet werden. Der IBM MQ-SOAP-Sender unterstützt Web-Service-Clients, die in .NET Framework 1, .NET Framework 2 und Axis 1.4 ausgeführt werden. Bei den Clients kann es sich um eine IBM MQ-Server- oder -Clientanwendung handeln. Der IBM MQ-Transport für SOAP wird auch in andere Umgebungen wie Microsoft Windows Communication Foundation, WebSphere Application Server und CICS Transaction Server integriert.

Die Integration in Microsoft Windows Communication Foundation ist Bestandteil der IBM MQ-Unterstützung für .NET Framework 3.

IBM MQ Transport for SOAP ist eine Gruppe von Protokollen und Tools für den Transport von SOAP-Nachrichten mithilfe von JMS über IBM MQ. Er wird auf verschiedene Arten für unterschiedliche Anwendungsumgebungen gepackt (siehe Tabelle 186 auf Seite 1370).

Tabelle 186. Anwendungsumgebungen für IBM MQ Transport for SOAP

	In zusätzliche IBM MQ-Komponenten integriert	**In ein Framework integriert**
Bereitstellung als Bestandteil der IBM MQ-Installation	.NET Framework 1 .NET Framework 2 Achse 1.4	Windows Communication Foundation (.NET Framework 3) Axis2 (nur Client)
Bereitstellung in einem anderen Softwarepaket		WebSphere Application Server CICS Transaction Server 4.1 WebSphere ESB WebSphere Process Server for Multiplatforms

Die Integration des IBM MQ-Transports für SOAP in ein Anwendungsframework vereinfacht die Entwicklung und Bereitstellung von Web-Services für IBM MQ.

Mit zusätzlichen IBM MQ-SOAP-Komponenten können Sie direkt mit den IBM MQ-SOAP-Komponenten interagieren, um Services zu entwickeln und zu implementieren. Verwenden Sie die IBM MQ-SOAP-Tools für die Konfiguration und Implementierung der Web-Services und Web-Service-Clients in IBM MQ.

In den integrierten Umgebungen sind die Entwicklung und Bereitstellung einfacher. Sie verwenden dieselben Tools für die Entwicklung und Bereitstellung wie bei der Entwicklung und Bereitstellung eines SOAP-HTTP-Web-Service. Sie müssen nach wie vor mit den IBM MQ-Tools die IBM MQ-Warteschlangen, -Kanäle und -Warteschlangenmanager konfigurieren, die Sie benötigen.

Sie können IBM MQ-SOAP-Clients und -Server aus allen diesen Umgebungen beliebig kombinieren.

Leistungen

IBM MQ Transport for SOAP bietet den IBM MQ-Bestandsbenutzern die folgenden Hauptvorteile:

Verwendung Ihres IBM MQ-Netzes für die Verbindung bereits vorhandener Web-Services

Bei den Services kann es sich um von Ihnen geschriebene Services oder um Services handeln, die als Schnittstellen zu anderen Standardsoftwareanwendungen bereitgestellt werden, die Sie implementiert haben.

Sie profitieren davon, dass Sie Ihr bestehendes IBM MQ-Netz für die Verbindung von Web-Services verwenden können. Der IBM MQ-Transport bietet den Vorteil, dass es sich bei ihm um einen verwalteten und zuverlässigen Service für Nachrichten in der Warteschlange handelt.

Schreiben neuer Anwendungen oder Konvertierung bereits vorhandener Anwendungen für die Verwendung von SOAP anstelle von IBM MQ-Schnittstellen

Normalerweise muss für Anwendungen ein bestimmter IBM MQ-Adapter entwickelt werden, damit eine Integration in eine andere Anwendung möglich ist. Adapter bestehen aus zwei Komponenten: dem Anschlussteil, mit dem Nachrichten in den Transport gestellt und aus diesem abgerufen werden, und dem Adapterteil, das Daten in anwendungsspezifische Formate konvertiert und umgekehrt. Die Integration der einzelnen Anwendungspaare ist eine neue Herausforderung.

Der Vorteil von SOAP besteht in der SOAP-Standardisierung für die Definition von Anwendungsschnittstellen und der Wahlmöglichkeiten des Transports. Sie müssen keine anwendungsspezifischen Adapter schreiben und können wählen, ob Sie IBM MQ oder HTTP als Verbindung verwenden möchten. Der von Ihnen gewählte Transport hängt davon ab, welche Servicequalität und Konnektivität Sie benötigen.

Für Bestandsbenutzer von SOAP over HTTP bietet IBM MQ Transport for SOAP den Vorteil, dass ein verwalteter und zuverlässiger asynchroner Transport verwendet wird. Das hat zwei Vorteile:

Ein vollständig asynchrones Programmiermodell für Verfügbarkeit und Leistung
Da eine asynchrone Clientschnittstelle verwendet wird, müssen die Client- und Serviceanwendungen nicht gleichzeitig verfügbar sein. Die vom Client gesendeten Anforderungen werden gespeichert, bis der Service für deren Verarbeitung verfügbar ist.

Ein sofort einsatzfähiges verwaltetes Netz mit einem Design für hohe Zuverlässigkeit und Verfügbarkeit

Wenn Sie IBM MQ als Transport wählen, profitieren Sie von einem verwalteten Netz, das ein zuverlässiges Messaging (Reliable Messaging) bietet.

Im Gegensatz dazu werden Transporte wie HTTP und FTP über TCP/IP nicht verwaltet. Ein nicht verwaltetes Netz eignet sich ideal für unberechenbare Verbindungen: Es sind weniger Management-Tasks auszuführen.

Zusammenfassung

IBM MQ Transport for SOAP stellt die folgenden Komponenten bereit:

- Die SOAP-/JMS-Transportbindung wird in WSDL-Dokumenten für die Bindung eines SOAP-Service an einen JMS-Transport verwendet. Die IBM MQ-Implementierung der SOAP-/JMS-Bindung verwendet einen URI, der eines der folgenden beiden Formate hat:

  IBM MQ-Transport für SOAP

  ```
  jms:/queue? &Name=Value&Name=Value...
  ```

  IBM MQ-Sendeformat für W3C-Kandidatenempfehlung

  ```
  jms:queue: qName ?connectionFactory=connectQueueManager (qMgrName)&Name=Value&Name=Va
  lue...
  ```

- Die Zuordnung einer SOAP-Nachricht zu einer IBM MQ-Nachricht.
- Zwei IBM MQ -SOAP-Listener zum Empfangen von SOAP-Anforderungen, einer für Java und einer für .NET Framework 1 oder .NET Framework 2. Die Listener verwenden .NET oder Axis 1.4 , um die SOAP-Anforderung zu verarbeiten.
- Zwei IBM MQ-SOAP-Sender für die Erstellung von IBM MQ-SOAP-Anforderungen. Web-Service-Clients registrieren sich bei einem Sender, um `jms:`-SOAP-Anforderungen zu verarbeiten.
- Integration in Windows Communication Foundation (WCF), gelegentlich auch als '.NET 3' bezeichnet, für das Senden und Empfangen von Nachrichten des IBM MQ-Transports für SOAP.
- Integration des Clients in Axis2 (gelegentlich auch als 'JAX-WS' bezeichnet) für das Senden von Nachrichten des IBM MQ-Transports für SOAP oder von W3C-SOAP-JMS- Nachrichten.
- Den Befehl **amqwdeployWMQService**, der Entwicklungs- und Laufzeitkomponenten und -scripts erstellt, um einen Web-Service mithilfe des IBM MQ-Transports für SOAP zu implementieren.
- Java- und .NET-Client- und -Servicebeispielcode.
- Ein Script für die Festlegung des Klassenpfads sowie weitere Dienstprogrammscripts.

In den integrierten Umgebungen werden der Sender und Listener als Erweiterungen zu den Entwicklungs- und Bereitstellungstools in jede Umgebung integriert.

Integration von SOAP und IBM MQ

Der IBM MQ-Transport für SOAP ist eine Erweiterung von SOAP sowie der Web-Service-Tools und Laufzeit mit IBM MQ als Transportalternative zu HTTP für SOAP. Sie müssen bestehende Web-Services nicht für die Verwendung des IBM MQ-Transports für SOAP als Transport ändern. Der Transport verwendet ein angepasstes URI-Format für SOAP/JMS. Das W3C-URI-Format für SOAP/JMS wird in beschränktem Maß von Axis2-Clients unterstützt.

Den Clients in Umgebungen mit .NET Framework 1, .NET Framework 2 und Axis 1.4 muss eine zusätzliche Codezeile hinzugefügt werden. Bei Clients in Axis 2 und Windows Communication Foundation (WCF) ist kein zusätzlicher Code erforderlich. Der IBM MQ-SOAP-Listener führt Services in den Umgebungen mit .NET Framework 1, .NET Framework 2 und Axis 1.4 aus. Der IBM MQ-Transport für SOAP ist in einige andere Anwendungsserverumgebungen integriert, darunter WCF, CICS und WebSphere Application Server.

Was ist SOAP?

SOAP[11]beschreibt das Standardformat der Nachrichten und Interaktionsprotokolle, die Anwendungen für den Austausch von Anforderungen, Antworten und Datagrammen verwenden. SOAP ist unabhängig von dem Transport, mit dem die Nachrichten übertragen werden, und auch von der Anwendungsumgebung, die die Nachrichten sendet und empfängt. Das W3C definiert SOAP 1.2 prägnant:

SOAP 1.2 enthält die Definition der XML-basierten Informationen, die für den Austausch strukturierter und typisierter Informationen zwischen Peers in einer dezentralen, verteilten Umgebung verwendet werden können.[12].

Damit SOAP verwendet werden kann, muss es an einen Transport wie HTTP, E-Mail oder IBM MQ gebunden werden.

Ein SOAP-Protokollbindungsframework ist die Gruppe von Regeln für den Übertrag einer SOAP-Nachricht, die auf einem anderen Protokoll aufsetzt (beispielsweise HTTP). SOAP 1.2 Teil 2: Adjuncts (Second Edition) beschreibt die SOAP-HTTP-Bindung.

In der W3C-Kandidatenempfehlung (Candidate Recommendation) vom 4. Juni 2009 (SOAP over Java Message Service 1.0) wird die Empfehlung für die SOAP-JMS-Bindung beschrieben. Da es sich bei JMS um eine API-Spezifikation und nicht um ein Transportprotokoll handelt, wird in der JMS-SOAP-Empfehlung das Sendeformat von SOAP-JMS-Nachrichten nicht beschrieben. Stattdessen werden die SOAP-Interaktionsprotokolle und die JMS-API-Bindung beschrieben. Wenn Sie also die JMS-SOAP-Empfehlung verwenden, müssen Sie nach wie vor dieselbe JMS-Implementierung für den SOAP-Client und den SOAP-Server verwenden. Die Ausführung einer SOAP-JMS-Anwendung ist in jeder beliebigen Implementierung von JMS möglich. Eine JMS-Implementierung kann als Plug-in in einem J2EE-Anwendungsserver verwendet werden, sofern sowohl der Server als auch die JMS-Implementierung die JCA-Spezifikation einhalten.IBM MQ JMS entspricht der JCA-Spezifikation und kann in einen kompatiblen Anwendungsserver integriert werden.

Der IBM MQ-Transport für die SOAP-Bindung ähnelt dem vorgeschlagenen W3C-Standard, ist jedoch nicht damit identisch. Seine Nutzung wird im Abschnitt MQRFH2-SOAP-Einstellungen beschrieben. Im Gegensatz zur W3C-Kandidatenempfehlung wird die SOAP-Bindung nicht formal angegeben. Effektiv handelt es sich um die HTTP-Bindung und die Serviceadresse hat das Format `jms:/queue?name=value&name=value...` statt `http://authority/path?query#fragment`. `jms:` ist kein offiziell registriertes IANA-URI-Schema.

Was ist ein Web-Service?

SOAP ermöglicht Programmen, die in verschiedenen Sprachen geschrieben wurden, die Ausführung auf unterschiedlichen Plattformen zur Kommunikation über verschiedene Transportprotokolle. SOAP ist die Protokollspezifikation. Ein Web-Service ist eine Anwendung, die einen Service über eine SOAP-Schnittstelle bereitstellt, auf den über Internetprotokolle zugegriffen wird.

Ein wichtiges Ziel von SOAP ist die Erbringung von Services, die von Clients ohne großen Aufwand verwendet werden können. Sobald Sie einen Client für die Verwendung eines Service entworfen haben, können Sie den Aufruf programmieren, um den Service ohne externe Dokumentationsreferenz aufrufen zu können. Serviceschnittstellen werden in XML in einem WSDL-Dokument beschrieben. Die Abfrage `http://authority/path?wsdl` gibt die WSDL-Beschreibung eines SOAP-Service zurück.

---

11 Früher stand das Akronym für Simple Object Access Protocol.

12 W3C: SOAP 1.2 Teil 0

Tipp: Wenn Sie einen Web-Service für die Verwendung von IBM MQ implementieren, müssen Sie auch den Service für HTTP implementieren, damit die WSDL-Standardabfrage funktioniert.

Web-Services entwickeln

Web-Services haben einen Clientbereich und einen Servicebereich. Zunächst wird der Service geschrieben, wobei entweder die Schnittstellenbeschreibung in WSDL als Ausgangspunkt verwendet wird oder die Regeln für das Schreiben der Serviceklasse eingehalten werden. Web-Service-Toolkits verfügen über Dienstprogramme zur Generierung von WSDL auf Basis der Schnittstellendefinition einer Klasse; Beispiele: **`java2wsdl`** oder **`disco`**. Zudem verfügen Sie über Tools für die Generierung oder Klassifizierung von Gerüsten auf Basis der WSDL-Schnittstellenbeschreibungen; Beispiele: **`wsdl2java`**, **`wsimport`** oder **`wsdl`**. Die erste Methode wird als Entwicklung von unten nach oben bezeichnet, während die letztere als Entwicklung von oben nach unten bekannt ist.

Der Befehl **`amqwdeployWMQService`** im IBM MQ-Transport für SOAP verwendet diese Tools, um WSDL, Client-Stubs und Client-Proxys zu generieren.

Web-Services werden normalerweise mithilfe einer integrierten Entwicklungsumgebung geschrieben, die sich an eine bestimmte Anwendungsserverumgebung richtet:

Eclipse IDE for Java EE Developers
Erstellt Web-Services für Axis 2. Unterstützt JAX-RPC und JAX-WS

Rational Anwendungsentwickler 7.5
Erstellt Web-Services für WebSphere Application Server V7 und Vorgängerversionen sowie für Axis. Unterstützt JAX-RPC und JAX-WS.

WebSphere Integrationsentwickler 6.2
Erstellt Web-Services für WebSphere Process Server und WebSphere ESB. Unterstützt JAX-RPC und JAX-WS.

Visual Studio 2012
Erstellt Web-Services für .NET Framework 3.5 und früher (Windows Communication Foundation)

Sie können jedes dieser Tools in Kombination mit dem IBM MQ-Transport für SOAP verwenden. Sobald Sie einen Service für die Verwendung mit HTTP entwickelt haben, verwenden Sie das Tool **`amqwdeployWMQService`**, um die Services zu implementieren, damit IBM MQ als Transport verwendet werden kann. Sie können einen neuen Client mithilfe der Ausgabe des Tools schreiben oder Ihre bereits vorhandenen Clients ändern, damit sie den IBM MQ-Transport für SOAP verwenden.

Wenn der IBM MQ-Transport für SOAP in die Anwendungsumgebung integriert wird, müssen Sie weder das Tool **`amqwdeployWMQService`** verwenden noch den Client-Code ändern. Die Client-SOAP-Schicht leitet Clientanforderungen, die einen URI mit dem Präfix `jms:` haben, an den IBM MQ-Transport für SOAP. Die Server-SOAP-Schicht ruft den IBM MQ-Transport für SOAP auf, um auf `jms:`-SOAP-Anforderungen zu warten, und gibt Antworten an den IBM MQ-Transport für SOAP zurück.

.NET-Services wurden bislang in der Regel von unten nach oben mit Web-Service-Anmerkungen im Code entwickelt, während Java-Services mit WSDL-Schnittstellendefinitionen von oben nach unten entwickelt wurden. Der Unterschied bei den Ansätzen ist die Eingrenzung, da Java Standard Edition 6 JAX-WS 2.0unterstützt und Annotationen verwendet, um die Definition von Serviceschnittstellen zu qualifizieren. Die Java-Services können jetzt genauso einfach von unten nach oben wie von oben nach unten entwickelt werden. Das von Ihnen gewählte Konzept ist eine Frage des Entwicklungsverfahrens.

Der Web-Service-Client wird nach der Entwicklung des Service geschrieben. Hierzu werden die WSDL-Servicedefinition und die generierten Client-Stubs und -Proxys verwendet. In einigen Anwendungen ist die Servicedefinition nicht bekannt, wenn der Client geschrieben wird. Der Client ruft den Service WSDL ab und erstellt Serviceanforderungen dynamisch. Häufiger kommt es jedoch vor, dass zwar die Servicedefinition bekannt ist, die Adresse, unter der Service implementiert wird, jedoch nicht. Das Web-Service-Toolkit generiert Schnittstellen, die vom Client für Serviceanforderungen verwendet werden. Der Client stellt die Serviceadresse zur Verfügung, falls sie benötigt wird. Im dritten Fall enthält die WSDL sämtliche Informationen, die von einem Client benötigt werden. Die WSDL enthält sowohl die Schnittstelle als auch die Adresse des Service. Der vom Web-Service-Toolkit generierte Code enthält alle Informationen, die der Client für Serviceanforderungen benötigt.

Sie können jede dieser drei Methoden zusammen mit dem IBM MQ-Transport für SOAP verwenden.

Web-Service-Anwendungsumgebungen

Web-Service-Toolkits benötigen eine Zuordnung von der WSDL-Definition eines Service zu den Bytesströmen, die in SOAP-Anforderungen und -Antworten übertragen werden. Der Bytestrom wird durch die SOAP-Spezifikation definiert und ist in der SOAP-Rahmenanweisung enthalten. Die SOAP-Rahmenanweisung ist in Abbildung 163 auf Seite 1374 dargestellt.

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->

<!-- headers... -->

</soap:Header>
<soap:Body>
<!-- payload or fault message -->

</soap:Body>
</soap:Envelope>
```

Abbildung 163. SOAP-Rahmenanweisung

Die Zuordnung von der SOAP-Rahmenanweisung zum programmiersprachenbezogenen Binden und umgekehrt ist teils standardisiert, teils proprietär. Die Zuordnung ist für die .NET-Architektur sehr wichtig und wird als Teil der Common Language Runtime (CLR) bereitgestellt. Die Zuordnung ist in Java durch JAX-Spezifikationen standardisiert. Da die Java-Zuordnungen standardisiert sind, können Java-Web-Service-Clients und -Services beliebig zwischen verschiedenen Java-basierten Anwendungsumgebungen portiert werden. JAX-RPC (gelegentlich auch als 'JAX-WS 1.0' bezeichnet) ist die derzeit am häufigsten eingesetzte Zuordnung. Sie wird von Axis 1.4 unterstützt. JAX-WS (gelegentlich auch als 'JAX-WS 2.0' bezeichnet) ist ein hochgradig verbesserter Standard und wird JAX-RPC wahrscheinlich bald ersetzen. JAX-WS wird von Axis 2.0 unterstützt. IBM MQ 7.0.1 unterstützt JAX-WS und Axis 2 nicht.

Der IBM MQ-Transport für SOAP ändert den Inhalt der SOAP-Rahmenanweisung nicht und der Inhalt hat keine Auswirkung auf den Transport. Die programmiersprachenbezogenen Bindungen wirken sich hingegen auf den IBM MQ-Transport für SOAP aus. IBM MQ 7.0.1 unterstützt .NET Framework 1, .NET Framework 2 und Axis 1.4 unter Verwendung des Codes und der Dienstprogramme, die mit dem IBM MQ-Transport für SOAP ausgeliefert werden. Die Unterstützung des WebSphere-Transports für SOAP in .NET Framework 3 und 3.5 wird mit dem angepassten IBM MQ-Kanal für Windows Communication Foundation implementiert.

Andere SOAP-Entwicklungs- und -Laufzeitumgebungen beinhalten möglicherweise eine Unterstützung des IBM MQ-Transports für SOAP und unterstützen verschiedene Sprachen. Die in CICS ausgeführten Web-Services unterstützen beispielsweise Sprachen wie COBOL und PL/1.

Anmerkung: Die verwendete Zuordnung spielt bei der Interoperabilität der Web-Services keine Rolle. Sie können geschriebene Clients und Services mit .NET-, JAX-RPC- und JAX-WS-Zuordnungen beliebig kombinieren.

Was ist der IBM MQ-Transport für SOAP?

Der IBM MQ-Transport für SOAP ist eine SOAP-Bindung und ein Web-Service-Toolkit. Gemeinsam ermöglichen sie Anwendungen den Austausch von SOAP-Nachrichten mit IBM MQ statt HTTP. In Abbildung 164 auf Seite 1375 ist IBM MQ als Alternative zu HTTP als SOAP-Transport dargestellt.

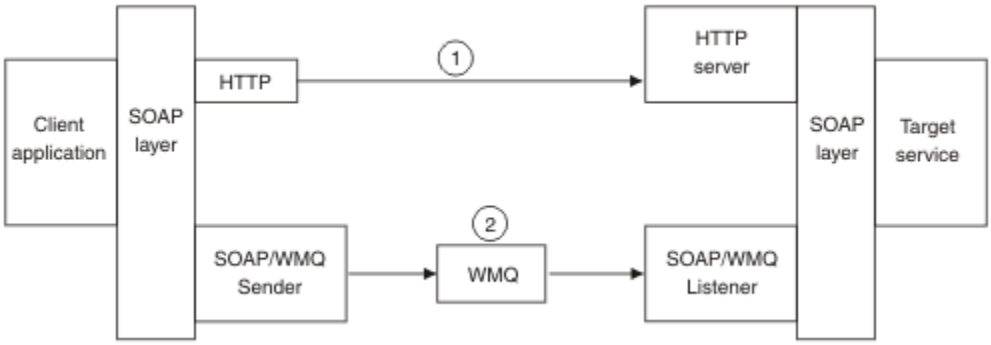


Abbildung 164. Übersicht über den IBM MQ-Transport für SOAP

SOAP over HTTP ist im Diagramm als (1) dargestellt. Die Client-SOAP-Schicht konvertiert eine Anforderung in eine SOAP-Nachricht und die HTTP-Komponente sendet Daten über TCP/IP. Die HTTP-Serverkomponente ist für HTTP-Anforderungen empfangsbereit (in der Regel am TCP/IP-Port 80). Wenn die Anforderung einen SOAP-Service betrifft, ruft die HTTP-Serverkomponente die SOAP-Schicht auf, um die SOAP-Anforderung in einen Methodenaufruf zu konvertieren. Anschließend gibt sie die Antwort zurück.

SOAP over IBM MQ wird als (2) dargestellt. Die Clientanwendung registriert die IBM MQ-SOAP-Senderkomponente als Handler für das `jms:`-Protokoll mit der SOAP-Schicht. Die SOAP-Schicht übergibt an `jms:` adressierte SOAP-Nachrichten an den IBM MQ-SOAP-Sender. Der Sender nutzt den URI in der Nachricht, um die Nachricht mit der erforderlichen Servicequalität in die Anforderungswarteschlange zu stellen. Der entsprechende IBM MQ-SOAP-Listener wartet in seiner Anforderungswarteschlange auf Nachrichten und ruft die SOAP-Schicht auf, um Anforderungen zu verarbeiten und Antworten zurückzugeben.

Beim SOAP-Sender und -Listener handelt es sich um normale IBM MQ-Programme. Sie können wie in Abbildung 165 auf Seite 1376 mit demselben Warteschlangenmanager oder wie in Abbildung 166 auf Seite 1377 mit unterschiedlichen Warteschlangenmanagern verbunden werden. Der Client kann über eine Clientverbindung verbunden werden.

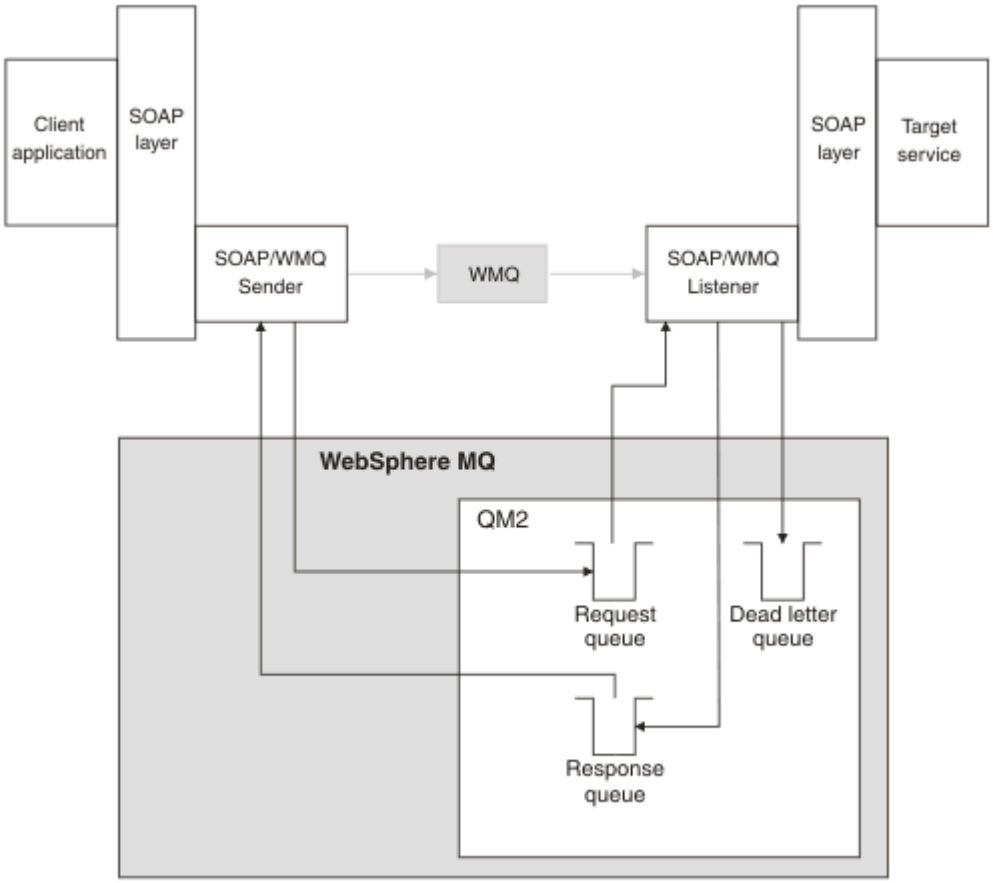


Abbildung 165. Von SOAP/IBM MQ verwendete Warteschlangen (einzelner Warteschlangenmanager)

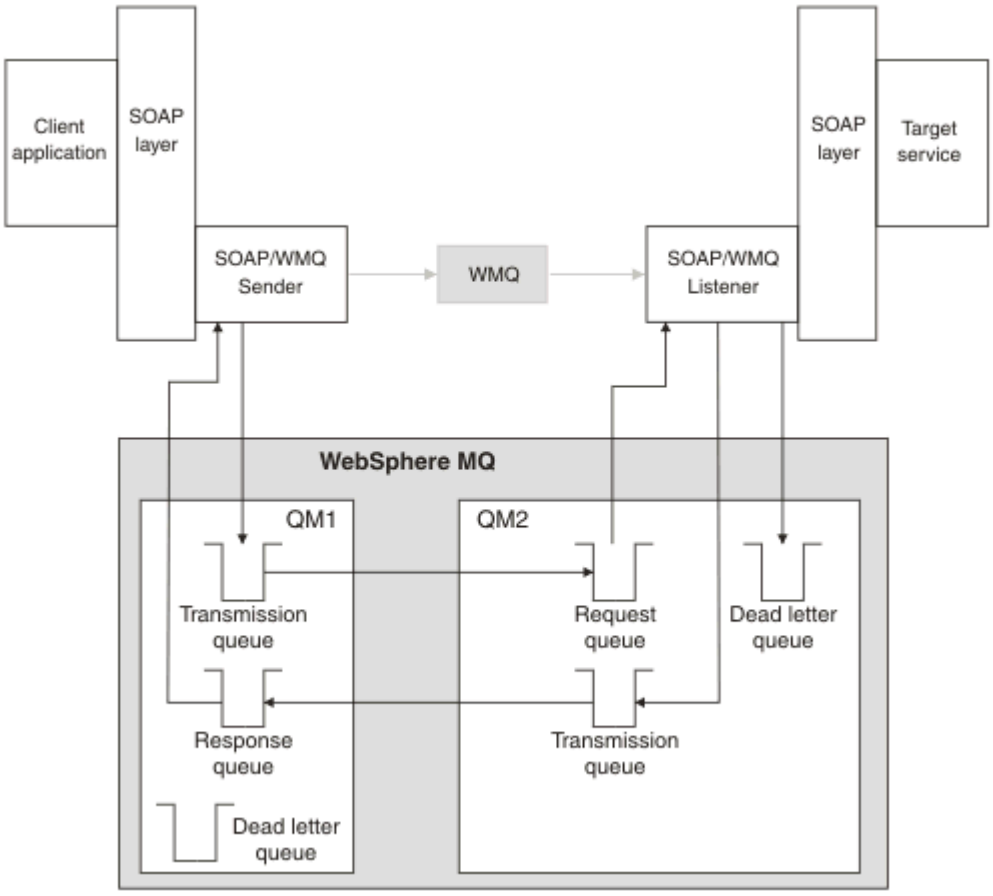


Abbildung 166. Von SOAP/IBM MQ verwendete Warteschlangen (separate Warteschlangenmanager)

W3C-Kandidatenempfehlung für die Bindung von SOAP an JMS

Der W3C-Empfehlungskandidat (Candidate Recommendation) definiert die SOAP over JMS-Bindung; SOAP over Java Message Service 1.0. Ebenfalls nützlich aufgrund der enthaltenen Beispiele ist das Dokument URI Scheme for Java(tm) Message Service 1.0 .[13].

Einige Anwendungsframeworks wie WebSphere Application Server v7 unterstützen die W3C-Kandidatenempfehlung. Senden Sie SOAP-Anforderungen, die mit einem URI formatiert sind, welcher mit der W3C-Kandidatenempfehlung kompatibel ist, unter Verwendung des Axis2-Clients; siehe W3C-SOAP over JMS-URI für den IBM MQ Axis 2-Client. Der Axis2-Client sendet eine SOAP-Anforderung, die entweder mit einem W3C- oder einem IBM MQ-Transport für SOAP formatiert ist, auf Basis eines URI in der SOAP-Anforderung.

Die Axis2-Clientunterstützung für die W3C-Empfehlung steht ab Fixpack 7.0.1.3 zur Verfügung. Es wird keine Unterstützung für sonstige Clients und die von IBM MQ bereitgestellten SOAP-Listener angeboten.

Zugehörige Konzepte

Implementierung des WebSphere-Transports für SOAP in .NET Framework 1, .NET 2 und Axis 1.4
Möglicherweise möchten Sie Ihren eigenen IBM MQ-SOAP-Sender und -Listener schreiben. Verwenden Sie hierfür die Implementierung des IBM MQ-Transports für SOAP in .NET Framework 1, .NET Framework 2 und Axis 1.4 als Leitfaden.

IBM MQ-Transport für SOAP und Web Services Reliable Messaging
Web Services Reliable Messaging (WSRM) ist ein Protokoll für den zuverlässigen Austausch von Web-Service-Anforderungen und -Antworten über eine störanfällige Verbindung. Es eignet sich am besten für die Problemlösung aufgrund von kurzen Verbindungsunterbrechungen.

[13] Den aktuellen Entwurf finden Sie in den W3C-Spezifikationsreferenzen unter *URI Scheme for JMS*.

Implementierung des WebSphere-Transports für SOAP in .NET Framework 1, .NET 2 und Axis 1.4

Möglicherweise möchten Sie Ihren eigenen IBM MQ-SOAP-Sender und -Listener schreiben. Verwenden Sie hierfür die Implementierung des IBM MQ-Transports für SOAP in .NET Framework 1, .NET Framework 2 und Axis 1.4 als Leitfaden.

1. Ein Clientprogramm verwendet das entsprechende Web-Service-Framework auf dieselbe Weise wie beim HTTP-Transport. Es muss ebenfalls das Präfix `jms:` registrieren. Das Präfix wird entweder mit der Java-Methode `com.ibm.mq.soap.Register.extension()` oder der CLR-Methode `IBM.WMQSOAP.Register.Extension()` registriert.
2. Das Framework von Axis 1.4 bzw. das .NET Framework 1 oder 2 setzen den Aufruf genau wie bei SOAP/HTTP in einer SOAP-Anforderungsnachricht zusammen.
3. Ein IBM MQ-Service wird über einen URI angegeben, der das Präfix `jms:` hat. Sobald das Framework den `jms:`-URI ermittelt, ruft es den IBM MQ-Transportsendercode auf: `com.ibm.mq.soap.transport.jms.WMQSender` (bei Axis 1.4) oder `IBM.WMQSOAP.MQWebRequest` (bei .NET1 und 2). Wenn das Framework einen URI mit einem `http:`-Präfix findet, ruft es den Standardserver von SOAP over HTTP auf.
4. Die SOAP-Nachricht wird mithilfe der Anforderungswarteschlange vom IBM MQ-SOAP-Sender transportiert. **`SimpleJavaListener`** (bei Java) oder **`amqwSOAPNETListener`** (bei .NET) empfängt die Anforderungsnachricht.

   Die IBM MQ-SOAP-Listener sind eigenständige Prozesse und werden als Multithread-Prozesse mit einer anpassbaren Anzahl von Threads ausgeführt.
5. Der IBM MQ-SOAP-Listener liest die eingehende SOAP-Anforderung und übergibt diese an die entsprechende Web-Service-Infrastruktur.
6. Die Web-Service-Infrastruktur analysiert die SOAP-Anforderungsnachricht und ruft den Service auf. Die Vorgehensweise ist mit derjenigen bei einer Nachricht identisch, die über einen HTTP-Transport eingetroffen ist.
7. Die Infrastruktur formatiert die Antwort als SOAP-Antwortnachricht und gibt sie an den IBM MQ-SOAP-Listener zurück.
8. Der Listener stellt die Nachricht in die Antwortwarteschlange und die Nachricht wird an den IBM MQ-SOAP-Sender übertragen. Der Sender übergibt sie an die Web-Service-Infrastruktur des Clients.
9. Die Clientinfrastruktur analysiert die SOAP-Antwortnachricht und meldet das Ergebnis zurück an die Clientanwendung.

Jeder Anwendungskontext wird von einer separaten IBM MQ-Anforderungswarteschlange bedient.

Der Anwendungskontext wird in Axis 1.4 gesteuert, indem sichergestellt wird, dass der IBM MQ-SOAP-Listener und -Service im richtigen Verzeichnis ausgeführt werden. Axis 1.4 legt den richtigen CLASSPATH für das Verzeichnis fest.

In .NET wird der Anwendungskontext durch den IBM MQ-SOAP-Listener gesteuert, der den Service in einem Kontext ausführt, der durch einen Aufruf an `ApplicationHost.CreateApplicationHost` erstellt wird. Der Aufruf gibt das Zielausführungsverzeichnis an. Jeder Service wird dann in dem Verzeichnis betrieben, in dem er implementiert wurde.

`amqwdeployWMQService` generiert die Anforderungs- und Antwortwarteschlangen. Darüber hinaus generiert er die Infrastruktur, die für die Handhabung der Warteschlangen und die Implementierung von Services für Axis 1.4 erforderlich ist.

Zugehörige Konzepte

Integration von SOAP und IBM MQ

IBM MQ-Transport für SOAP und Web Services Reliable Messaging

Web Services Reliable Messaging (WSRM) ist ein Protokoll für den zuverlässigen Austausch von Web-Service-Anforderungen und -Antworten über eine störanfällige Verbindung. Es eignet sich am besten für die Problemlösung aufgrund von kurzen Verbindungsunterbrechungen.

IBM MQ-Transport für SOAP und Web Services Reliable Messaging

Web Services Reliable Messaging (WSRM) ist ein Protokoll für den zuverlässigen Austausch von Web-Service-Anforderungen und -Antworten über eine störanfällige Verbindung. Es eignet sich am besten für die Problemlösung aufgrund von kurzen Verbindungsunterbrechungen.

IBM MQ für SOAP nutzt die Vorteile des verwalteten und zuverlässigen IBM MQ-Netzes für die Übergabe von SOAP-Nachrichten. Transporte wie HTTP und FTP werden nicht verwaltet. Nicht verwaltete Netze eignen sich ideal für unberechenbare Verbindungen, bei denen die Schwierigkeiten und der Aufwand für die Verbindungsverwaltung die Vorteile ständig verfügbarer Anforderungen und Antworten überwiegen.

Zur Umgehung des Problems verlorener Dateien durch Verbindungsunterbrechungen in nicht verwalteten Netzen setzen Services wie das verwaltete FTP eine Verwaltungsschicht auf FTP auf. Die Verwaltungsschicht übernimmt die Aufgabe der Prüfung, ob Dateien erfolgreich von Benutzern übertragen wurden, und überträgt fehlende Dateien bei Bedarf erneut. Wenn Sie das verwaltete FTP verwenden möchten, müssen Sie die Management-Software auf beiden Verbindungsseiten installiert haben.

Web Services Reliable Messaging (WSRM) löst das Problem störanfälliger Verbindungen mit einer anderen Strategie. Das Ziel besteht darin, Web-Service-Anforderungen und -Antworten zuverlässig zu übertragen, ohne dass beide Seiten der Verbindung dieselbe Software verwenden müssen. Durch die Implementierung des Web Services Reliable Messaging-Protokolls kann jede beliebige Software Nachrichten zuverlässig mit einer anderen Software austauschen.

Wenn eine Verbindung ausfällt, müssen ein Sender und Empfänger den Kontext der WSRM-Nachrichtenübertragung mithilfe eines generierten URI als Schlüssel beibehalten. Der Sender und Empfänger versuchen fortlaufend, eine neue Verbindung herzustellen. Sobald eine neue Verbindung hergestellt wurde, wird die Übertragung abgeschlossen. Die WSRM-Spezifikation gibt nicht an, wie Kontext beibehalten werden soll. Sie gibt auch den Zeitpunkt für einen neuen Verbindungsversuch nicht vor.

In Ihrem Fall sind möglicherweise nur kurzzeitige Ausfälle von Interesse. Bei längeren Ausfällen sind Sie unter Umständen bereit, Übertragungen zu verwerfen, die nach einer gewissen Zeit nicht erneut gestartet werden konnten. Auf ähnliche Weise sind Sie möglicherweise bereit, Übertragungen zu verwerfen, wenn entweder der Client oder der Service fehlschlägt. Der Benutzer ist für die Sicherstellung von Übertragungen verantwortlich und der Bedarf an Koordinierungsverwaltung des Clients und Service wird verringert.

Bei Netzausfällen, die länger andauern (länger als etwa 30 Minuten), oder bei einem Ausfall des Clients oder Servers besteht eine höhere Wahrscheinlichkeit, dass manche Verbindungen nie erneut aufgebaut werden. Sie können sich nicht mehr darauf verlassen, dass WSRM die Nachrichtenübertragung automatisch auf nicht verwaltete Weise wiederherstellt. Sie müssen die Verwaltung der fehlgeschlagenen WSRM-Verbindungen erwägen. Dies schließt die Entwicklung von Software ein, mit der das Netz der Clients und Services verwaltet wird.

Die Verwendung von WSRM zur Bewältigung von kurzen Ausfallzeiten kann die Probleme aufgrund verlorener Nachrichten in einem mobilen Netz beträchtlich verringern. Wenn Sie keine Nachrichtenübermittlung zusichern müssen, können die Vorteile von weniger Nachrichtenverlusten die zusätzlichen Kosten für die Entwicklung einer WSRM-Implementierung wettmachen.

SOAP over JMS bietet eine zuverlässige Nachrichtenübermittlung und befasst sich mit längeren Ausfallzeiten des Clients, Servers und Netzes. Falls Sie eine zuverlässigere Servicequalität für SOAP als HTTP wünschen, stehen Sie vor der Frage, welche Lösung gewählt werden soll: IBM MQ Transport for SOAP oder WSRM? Die Antwort hängt von vielen Faktoren ab. Es folgt eine Auflistung einiger der Faktoren, die es zu berücksichtigen gilt:

1. Ist die Unzuverlässigkeit auf einen Verbindungsfehler zurückzuführen?
2. Wie lange dauern Verbindungsfehler an?
3. Können Sie sowohl die Client- als auch die Serverseite der Verbindung verwalten?
4. Ist der Benutzer oder ein Administrator letztendlich für die Nachrichtenübermittlung verantwortlich?

Zugehörige Konzepte

Integration von SOAP und IBM MQ

Implementierung des WebSphere-Transports für SOAP in .NET Framework 1, .NET 2 und Axis 1.4

Möglicherweise möchten Sie Ihren eigenen IBM MQ-SOAP-Sender und -Listener schreiben. Verwenden Sie hierfür die Implementierung des IBM MQ-Transports für SOAP in .NET Framework 1, .NET Framework 2 und Axis 1.4 als Leitfaden.

IBM MQ-Web-Services installieren und überprüfen

Mithilfe der Anweisungen in diesen Abschnitten können Sie den IBM MQ-Transport für SOAP installieren und überprüfen.

IBM MQ Web Transport for SOAP installieren

Mithilfe dieser Anweisungen können Sie IBM MQ Web Transport for SOAP installieren. Die Installation erstellt Tools für die Ausführung der Web-Service-Clients oder Services, die IBM MQ als SOAP-Transport verwenden. Die Tools werden in den SOAP-Umgebungen mit .NET Framework 1, .NET 2, Axis 1.4 oder Axis2 verwendet.

Vorbereitende Schritte

Überprüfen Sie die vorausgesetzten Produkte im Abschnitt Systemvoraussetzungen für IBM MQ. Der Installationsprozess überprüft weder das Vorhandensein noch die Verfügbarkeit der Softwarevoraussetzungen. Sie müssen überprüfen, ob die Voraussetzungen installiert sind.

IBM MQ stellt eine Kopie der Laufzeit von Axis 1.4 zur Verfügung. Verwenden Sie diese Version zusammen mit IBM MQ anstelle einer anderen möglicherweise installierten Version. IBM stellt für Apache Axis keine technische Unterstützung bereit. Wenden Sie sich an Apache Software Foundation, falls technische Probleme damit auftreten.

Zum Ausführen von Web-Services in der SOAP-Umgebung mit .NET Framework 3 verwendet IBM MQ Windows Communication Foundation. Der angepasste IBM MQ-Kanal für Windows Communication Foundation führt Web-Service-Clients und Services unter Verwendung von IBM MQ als Transport für SOAP-Nachrichten aus.

Informationen zu diesem Vorgang

Sie können IBM MQ Web Transport for SOAP entweder als IBM MQ MQI client- oder als Serveranwendung installieren. Wenn Sie IBM MQ bereits als Client oder Server auf Ihrem Computer installiert haben, prüfen Sie, ob Sie die aufgelisteten Komponenten installiert haben.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM MQ installiert ist.

Führen Sie die folgenden Installationsschritte aus.

Vorgehensweise

1. Wählen Sie die `Java and .Net Messaging and web services` -Komponente für die Installation aus.
2. Wählen Sie unter Solaris und HP-UX die Komponente `Java Runtime environment` für die Installation aus.
3. Wählen Sie das Entwicklungstoolkit für die Installation aus.
4. Installieren und überprüfen Sie IBM MQ wie im jeweiligen Handbuch zum Schnelleinstieg für Ihre Plattform beschrieben.
5. Kopieren Sie die Laufzeit von Apache Axis 1.4 (`axis.jar`) aus dem Verzeichnis `prereqs/axis` auf dem Installationsmedium von IBM MQ. Kopieren Sie sie in das Installationsverzeichnis, das in Tabelle 187 auf Seite 1381, Tabelle 188 auf Seite 1381 oder Tabelle 189 auf Seite 1381 beschrieben ist.

   Windows

   ```
   Copy D:\PreReqs\axis\axis.jar MQ_INSTALLATION_PATH\java\lib\soap
   ```

AIX

```
cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
```

Installationsverzeichnisse für HP-UX, Solaris und Linux (alle Plattformen)

```
cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
```

6. Führen Sie unter Windows 2003 den Befehl **Aspnet_regiis.exe** aus, um die Scriptzuordnungen zu aktualisieren, damit sie auf die von Ihnen verwendete Version der Common Language Runtime verweisen.

   Suchen Sie in %SystemRoot%Microsoft.NET/Framework/*version-number*nach dem Dienstprogramm **Aspnet_regiis.exe** .
7. Legen Sie die Umgebungsvariable WMQSOAP_HOME so fest, dass sie auf das Installationsverzeichnis von IBM MQ verweist.

Ergebnisse

Tabelle 187. Installationsverzeichnisse von Windows

Position	Inhalt
MQ_INSTALLATION_PATH \programs\bin	Binärdateien, Befehlsdateien, DLL-Dateien und ausführbare Dateien
MQ_INSTALLATION_PATH \programs\java\lib	.jar files
MQ_INSTALLATION_PATH \programs\java\lib\soap	SOAP .jar files
MQ_INSTALLATION_PATH \programs\soap\samples	Beispiele und Installationsprüftest

Tabelle 188. Installationsverzeichnisse von AIX

Position	Inhalt
MQ_INSTALLATION_PATH/bin	Shell-Scripts
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar und sonstige .jar-Dateien für Java API for XML-based RPC (JAX-RPC)
MQ_INSTALLATION_PATH/samp/soap	Beispiele und Installationsprüftest

Tabelle 189. Installationsverzeichnisse für HP-UX, Solaris und Linux (alle Plattformen)

Position	Inhalt
MQ_INSTALLATION_PATH/bin	Shell-Scripts
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar und sonstige .jar-Dateien für Java API for XML-based RPC (JAX-RPC)
MQ_INSTALLATION_PATH/samp/soap	Beispiele und Installationsprüftest

Nächste Schritte

1. Gilt nur für .NET: Sie müssen die Dateien des IBM MQ-Transports für SOAP beim Global Assembly Cache registrieren. Wenn .NET bei der Installation von IBM MQ bereits installiert ist, wird die Registrierung bei der Installation automatisch ausgeführt. Wenn Sie .NET nach IBM MQ installieren, wird die Registrierung automatisch bei der ersten Ausführung des Installationsprüftests ausgeführt.

   Sie können **amqiregisterdotnet.cmd** ausführen, um die Registrierung der .NET-Assemblys vorzunehmen. Mit **amqiregisterdotnet.cmd** können Sie außerdem in jeder Phase eine erneute Registrierung erzwingen. Sobald die Registrierung vorgenommen wurden, bleibt sie auch nach Systemwiederanläufen gültig und normalerweise ist keine erneute Registrierung erforderlich.
2. Führen Sie den Installationsprüftest wie im Abschnitt „IBM MQ Transport for SOAP überprüfen” auf Seite 1382 beschrieben aus.
3. Wenn Sie einen Axis2-Client entwickeln möchten, müssen Sie Axis2 1.4.1 aus Apache herunterladen; siehe „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln” auf Seite 1401.

IBM MQ Transport for SOAP überprüfen

Die Installation des IBM MQ-Transports für SOAP können Sie mit dem Befehl **runivt** prüfen. Dieser Befehl führt eine Reihe von Demo-Anwendungen aus und stellt sicher, dass die Umgebung nach der Installation ordnungsgemäß konfiguriert ist. Beachten Sie, dass der Web-Service-Teil des Transports für SOAP nicht weiter unterstützt wird. Als neuer Benutzer sollten Sie ihn deshalb nicht verwenden.

Vorbereitende Schritte

Stellen Sie vor Ausführung des Befehls **runivt** sicher, dass Sie über folgende Laufzeitumgebungen verfügen:

- Testausführung nur für Axis: Sie benötigen auf Ihrem System ein Java SDK (in SOE). Außerdem müssen Sie der Umgebungsvariablen **PATH** des Systems die Speicherposition der Befehle `java.exe` und `javac.exe` hinzufügen.
- Testausführung nur für .NET (nur unter Windows unterstützt): Sie benötigen auf Ihrem System das Java SDK und die Compiler und Tools für .NET. Zur Ausführung des Tests öffnen Sie entweder eine Visual Studio-Eingabeaufforderung oder die Eingabeaufforderung des Microsoft Windows SDK. Fügen Sie dann der Umgebungsvariablen **PATH** die Speicherposition der Dateien `java.exe` und `javac.exe` hinzu.
- Ausführung aller verfügbaren Tests: Auf Windows-Systemen muss die Umgebung konfiguriert sein, wie für die Ausführung des Tests für .NET beschrieben. Auf UNIX and Linux-Systemen muss die Umgebung konfiguriert sein, wie für die Ausführung des Tests für Axis beschrieben.

Informationen zu diesem Vorgang

Vielleicht möchten Sie den Test nicht für .NET und Axis ausführen, sondern nur für Axis oder nur für .NET.

Wenn Sie den Test nach einem Problem erneut starten wollen:

1. Stoppen Sie den Warteschlangenmanager `WMQSOAP.DEMO.QM` mit der Option `immediate`.
2. Stoppen Sie das Empfangsprogramm, das in einem separaten Fenster gestartet wurde.
3. Löschen Sie den Warteschlangenmanager.
4. Löschen Sie das von Ihnen erstellte temporäre Verzeichnis "samples" und beginnen Sie noch einmal von vorne.

Unter UNIX and Linux müssen Sie den Befehl über eine X Windows System-Sitzung ausführen.

Der Befehl **runivt** ändert den Inhalt des Verzeichnisses `soap/samples`. Wenn das Installationsimage unverändert bleiben soll, kopieren Sie das Verzeichnis "samples" in ein temporäres Verzeichnis und führen Sie die Prüfung in diesem Verzeichnis aus.

Sie können die Installationsprüfung beliebig oft ausführen.

Zur Prüfung der Installation des IBM MQ-Transports für SOAP unter .NET Framework 1, .NET Framework 2 und Axis 1.4 führen Sie die folgenden Schritte aus:

Vorgehensweise

1. Kopieren Sie die Verzeichnisbaumstruktur `./tools/soap/samples` in ein temporäres Verzeichnis.
2. Starten Sie ein Befehlsfenster mit dem temporären Verzeichnis als aktuelles Verzeichnis.
3. Verwenden Sie den Befehl **runivt**, um die Installationsprüfung zu starten. Das Script `runivt` kompiliert eine Testklasse, den Beispielclient und die zugehörigen Services vor deren Implementierung und Ausführung. Führen Sie zur Ausführung der Testklasse, des Beispielclients und der Services die im Abschnitt WebSphere(r) MQ Web Transport for SOAP installieren beschriebenen Installationsschritte aus, und stellen Sie dabei bei der Ausführung des Befehls `runivt` sicher, dass Sie an der Eingabeaufforderung die richtige Laufzeitumgebung angeben. Führen Sie den Befehl **runivt** mit einer der folgenden Methoden aus:

   - Testausführung nur für Axis: `runivt Axis`.
   - Testausführung nur für .NET (nur unter Windows unterstützt)): `runivt DotNet`.
   - Ausführung aller verfügbaren Tests: `runivt`.

   Weitere Informationen zur Syntax und den Parametern des Befehls `runivt` finden Sie im Abschnitt **runivt**: Funktionstest für die Installation des IBM MQ-Transports für SOAP. Die Tests, die Sie ausführen können, sind in der Datei `ivttests.txt` (für Windows) bzw. in der Datei `ivttests_unix.txt` (für UNIX and Linux) aufgeführt.

Zugehörige Informationen

runivt: Funktionstest für die Installation des IBM MQ-Transports für SOAP

Web-Services für IBM MQ Transport for SOAP entwickeln

Zur Entwicklung von Services für IBM MQ Transport for SOAP verwenden Sie Ihre ganz normale Entwicklungsumgebung für Web-Services.

Vorbereitende Schritte

1. Wenn Sie die mit IBM MQ Transport for SOAP bereitgestellten Befehlszeilentools verwenden möchten:
   a. Erstellen Sie ein Implementierungsverzeichnis für den Service.
   b. Starten Sie in diesem Verzeichnis ein Befehlsfenster.
   c. Für .NET müssen sich die Dateien csc.exe und wsdl.exe im Pfad befinden und die gleiche .NET Framework-Version aufweisen.
   d. Für Java:
      i) Führen Sie den Befehl **amqwsetcp** zur Festlegung des Klassenpfads aus.
      ii) Im aktuellen Pfad müssen sich eine IBM JRE und ein JDK derselben Version befinden. Die Version muss mindestens 5.0 sein.
      iii) Fügen Sie dem Klassenpfad die Pfade aller `.jar`-Bibliotheken und -Verzeichnisse hinzu, die .java-Pakete enthalten, auch diejenigen des Service, den Sie gerade entwickeln. Stellen Sie das aktuelle Verzeichnis "`.`" in den Klassenpfad.
      iv) Erstellen Sie relativ zum aktuellen Verzeichnis des Befehlsfensters ein Verzeichnis, das dem Paketnamen des Service entspricht, den Sie gerade entwickeln.
2. Alternativ können Sie auch Workbench-Tools verwenden, die die Entwicklung von Web-Services unterstützen. In den Entwicklungstasks des Beispiels werden Microsoft Visual Studio 2008, Eclipse IDE for Java EE Developers und WebSphere Application Server Community Edition verwendet.

Informationen zu diesem Vorgang

An den bestehenden Web-Services müssen zur Unterstützung von WebSphere Transport for SOAP keine Änderungen vorgenommen werden. Mit den mitIBM MQ Transport for SOAP bereitgestellten Tools kann einWeb-Service implementiert und mit einem IBM MQ SOAP-Empfangsprogramm ausgeführt werden. Außerdem können mit diesen Tools WSDL, .NET-Client-Stubs und `.java`-Proxy-Klassen für die Entwicklung von IBM MQ Transport for SOAP-Clients generiert werden.

Führen Sie die folgenden Schritte aus, um einen Service zu erstellen und für die Implementierung und die Generierung von Clients vorzubereiten. Unter den entsprechenden Tasks finden Sie auch Informationen zur Erstellung eines Service mit Eclipse oder Microsoft Visual Studio 2008.

Vorgehensweise

1. Entwickeln Sie den Service in Ihrer normalen Entwicklungsumgebung.
2. Testen Sie den Service mit einem HTTP-Web-Service-Client.
3. Führen Sie zur Vorbereitung des Implementierungsverzeichnisses die folgenden Schritte aus:
   - Für Java
     a. Kopieren Sie die `.java`-Datei mit der Definition der Serviceschnittstelle in das Implementierungsverzeichnis.
     b. Kopieren Sie alle `.class`-Dateien für den Service in das Verzeichnis mit dem Paketnamen.
     c. Vergewissern Sie sich, dass der Klassenpfad alle erforderlichen Klassen umfasst, und kompilieren Sie dann die `.java`-Datei des Service mit **`javac`**.
   - Für .NET
     a. Kopieren Sie die `.asmx`-Datei mit der Definition des Service in das Implementierungsverzeichnis.
     b. Wenn Sie das Code-Behind-Modell verwenden, kopieren Sie alle `.dll` -Dateien in ein Verzeichnis *`deployment directory`*`\bin` .

.NET 1- oder 2-Service für IBM MQ Transport for SOAP mithilfe von Microsoft Visual Studio 2012 entwickeln

Entwicklung des Web-Service 'SampleStockQuote' für .NET 1 oder .NET 2 mithilfe von Microsoft Visual Studio 2012.

Informationen zu diesem Vorgang

Erstellen Sie unter Verwendung von Microsoft Visual Studio 2012 den Service 'StockQuote' mit einer 'Code-Behind'-Implementierung.

Vorgehensweise

1. Erstellen Sie eine Vorlage für den Service und stellen Sie sicher, dass sie unter HTTP ausgeführt werden kann.
   a) Starten Sie Visual Studio 2012 und klicken Sie auf > **Datei** > **Neu** > **Projekt**. Wählen Sie **C#** Projekttyp, **,NET Framework 2** und **ASP.NET-Web-Service-Anwendung** aus. Geben Sie **Name:** und **Lösungsname:** `StockQuoteDotNet` ein und klicken Sie auf > **OK**
   b) Klicken Sie im **Solution Explorer** mit der rechten Maustaste auf **Service1.asmx** und klicken Sie dann auf > **Umbenennen** > `StockQuote.asmx`.
   c) Ändern Sie das Codefragment `public class Service1` in `public class StockQuote`.
   d) Klicken Sie mit der rechten Maustaste auf **StockQuote.asmx** im **Solution-Explorer** > **Öffnen mit ...** > **XML-Editor**. Ändern Sie `Class="StockQuoteDotNet.Service1"` in `Class="StockQuoteDotNet.StockQuote"`.
   e) Ändern Sie das Codefragment `[WebService(Namespace = "http://tempuri.org/")]` in `[WebService(Namespace = "http://stock.samples/")]`.

f) Entfernen Sie die Codezeile `[ToolboxItem(false)]`.

g) Überprüfen Sie alles auf seine Richtigkeit: **Fehler beheben** > **Fehlerbehebung starten (F5)**. Prüfen Sie die Ausgabe in Explorer.

2. Fügen Sie die Methoden aus Beispiel `SQDNNonInline.asmx.cs` hinzu und testen Sie den Service unter HTTP.

   a) Öffnen Sie *MQ_INSTALLATION_PATH* `\tools\soap\samples\dotnet\SQDNNonInline.asmx.cs` und ersetzen Sie die Methode `HelloWorld` mit den vier `Quote`-Methoden; siehe Abbildung 167 auf Seite 1386. *MQ_INSTALLATION_PATH* steht für das Verzeichnis, in dem IBM MQ installiert ist.

   b) Klicken Sie für die Lösung auf **Erstellen** > **Erneut erstellen**. > Klicken Sie mit der rechten Maustaste auf einen der fehlerhaften **Threads** und klicken Sie dann auf > **Beheben** > Mit **System.Threading**.

   c) Drücken Sie die Taste F5, um die Fehlerbehebung zu starten.

   Der Service ist nicht konform mit WS-I Basic Profile v1.1. Sie können entweder die WebMethod-Anmerkung von `[SoapRpcMethod]` in `[SoapDocumentMethod]` ändern oder die Anmerkung `[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]` entfernen.

   d) Drücken Sie die Taste F5, um die Implementierung mittels HTTP zu prüfen.

3. Generieren Sie die WSDL und die Clients und führen Sie den Service mittels IBM MQ Transport for SOAP aus.

   a) Öffnen Sie ein Befehlsfenster in der Projektverzeichnisstruktur, in der `StockQuote.asmx` gespeichert ist.

   b) (Optional) Generieren Sie Artefakte mit 'amqwdeployWMQService'. Der Warteschlangenmanager muss gestartet werden:

```
amqswdeployWMQService -f StockQuote.asmx
-u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

   Alle Artefakte werden in der Verzeichnisstruktur `./generated` erstellt.

   c) (Optional) Generieren Sie nur die WSDL für den Aufruf des Service mithilfe des IBM MQ-Transports for SOAP.

```
amqswsdl -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

   a) Führen Sie den .NET-Listener aus. Verwenden Sie entweder die Datei `.\generated\server\startWMQNListener.cmd` oder geben Sie folgenden Befehl ein:

```
amqSOAPNETListener -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
```

4. Testen Sie den Service mithilfe eines von der WSDL generierten Clients oder mit den von **amqwdeployWMQService** generierten Clients.

Beispielcode

Der .NET -Beispiel-Web-Service `StockQuoteDotNet`wird in *MQ_INSTALLATION_PATH* `\tools\soap\samples\dotnet`installiert. *MQ_INSTALLATION_PATH* ist das Verzeichnis, in dem IBM MQ installiert ist. Die Web-Service-Bindung der veröffentlichten Beispiele weicht minimal von der in der Task verwende-

ten Bindung ab. Die Task verwendet die Standardeinstellungen und -werte von Microsoft Visual Studio 2012.

Es gibt zwei Beispiele für Web-Services von .NET Framework 1 und .NET Framework 2. `StockQuote-DotNet.asmx` ist ein Inline-Service. `SQDNNoninline.asmx` ist ein Code-Behind-Web-Service, der von `SQDNNoninline.asmx.cs`implementiert wird.

`StockQuoteDotNet` verwendet vier Methoden:

1. `float getQuote(String symbol)`
2. `void getQuoteOneWay(String symbol)`.
3. `int asyncQuote(int delay)`
4. `float getQuoteDOC(String symbol)`

```
<%@ WebService Language="C#" Class="StockQuoteDotNet" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;
[WebService (Namespace="http://stock.samples")]
public class StockQuoteDotNet {
     [WebMethod] [ SoapRpcMethod(OneWay=true) ]
     public void getQuoteOneWay(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
          System.Console.WriteLine("getQuoteOneWay was invoked.");
     }
     [WebMethod] [SoapRpcMethod]
     public float getQuote(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
          return 88.88F;
     }
     [WebMethod] [SoapRpcMethod]
     public int asyncQuote(int delay) {
          Thread.Sleep(delay);
          return delay;
     }
     [WebMethod]
     public float getQuoteDOC(String symbol) {
          return 77.77F;
     }
}
```

Abbildung 167. Integrierter Service: `StockQuoteDotNet.asmx`

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

Abbildung 168. Code-Behind: Gestaltung `SQDNNonInline.asmx`

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService(Namespace = "http://stock.samples")]
public class SQDNNonInline : System.Web.Services.Protocols.SoapHttpClientProtocol
{

    [WebMethod]
    [SoapRpcMethod(OneWay = true)]
    public void getNonInlineQuoteOneWay(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getNonInlineQuoteOneWay was invoked.");
    }

    [WebMethod]
    [SoapRpcMethod]
    public float getNonInlineQuote(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }

    [WebMethod]
    [SoapRpcMethod]
    public int asyncNonInlineQuote(int delay)
    {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getNonInlineQuoteDOC(String symbol)
    {
        return 77.77F;
    }
}
```

Abbildung 169. Code-Behind: Implementierung: SQDNNonInline.asmx.cs

Zugehörige Tasks

JAX-WS-EJB-Web-Service für W3C SOAP over JMS entwickeln
Ein Web-Service, gebunden an die W3C-Kandidatenempfehlung für SOAP over JMS, muss im EJB-Container eines JEE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 2 beim Verbinden eines Axis2-Web-Service-Clients und eines Web-Service, implementiert im WebSphere-Anwendungsserver, mit dem W3C SOAP over JMS-Protokoll dar.

JAX-WS-EJB-Web-Service für W3C SOAP over JMS entwickeln

Ein Web-Service, gebunden an die W3C-Kandidatenempfehlung für SOAP over JMS, muss im EJB-Container eines JEE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 2 beim Verbinden eines Axis2-Web-Service-Clients und eines Web-Service, implementiert im WebSphere-Anwendungsserver, mit dem W3C SOAP over JMS-Protokoll dar.

Vorbereitende Schritte

Erstellen Sie den EJB-Web-Service mit Rational Application Developer. Der Web-Service-Assistent in Rational Application Developer ermöglicht optional, einen Web-Service mit der W3C-Kandidatenempfehlung für die SOAP over JMS-Bindung zu erstellen. Rational Application Developer 7.54 ist erforderlich. Die Übung verwendete Rational Application Developer, enthalten in Rational Software Architect für WebSphere Software v7.5.5.1.

Als Teil dieser Task wird die EJB mit Rational Application Developer in WebSphere Application Server implementiert.

Zur Erstellung der in der Task verwendeten WSDL müssen Sie zunächst das Liberty-Profil einrichten. Danach können Sie die WSDL entweder aus dem dynamischen Webprojekt im Eclipse Galileo-Arbeitsbereich oder aus dem aktiven, im Liberty-Profil implementierten HTTP-Web-Service importieren.

WebSphere Application Server wird möglicherweise noch ausgeführt. Andernfalls können Sie das Programm aus der Serveransicht in RAD starten.

Informationen zu diesem Vorgang

In dieser Task stellen Sie den Service `StockQuoteAxis`, der als JAX-RPC-Axis-Service des **`SimpleJavaListener`** ausgeführt wird und IBM MQ-Transport für SOAP verwendet, erneut als einen JAX-WS-Service bereit, der in WebSphere Application Server mit dem Protokoll W3C SOAP over JMS ausgeführt wird.

Die Migration des Service aus dem **`SimpleJavaListener`** nach WebSphere Application Server wird in zwei Schritten ausgeführt:

1. Generierung des Web-Service aus der WSDL für den Service mit dem Top-down-EJB-Web-Service-Assistenten in Rational Application Developer.
2. Implementierung des Service durch Importieren des IBM MQ SOAP-Beispiels `StockQuoteAxis.java`.

Alternativ hätte der Service auch nach dem 'Bottom-up'-Prinzip, d. h. von unten nach oben, aus der `StockQuoteAxis.java` generiert werden können. Um jedoch sicherzustellen, dass die Schnittstelle zum migrierten Service identisch ist, ist die Top-down-Methode (Von oben nach unten) besser geeignet, da sie dieselbe WSDL verwendet.

Der Web-Service ist für den EJB-Container und nicht für den Web-Container entwickelt, da die JMS-Unterstützung Teil des EJB-Containers ist.

Vorgehensweise

1. Starten Sie Rational Application Developer und stellen Sie sicher, dass WebSphere Application Server ausgeführt wird.
   a) Starten Sie Rational Application Developer in einem neuen Arbeitsbereich.
   b) Öffnen Sie die Java EE-Perspektive.
   c) Öffnen Sie die Registerkarte **Server** und stellen Sie sicher, dass WebSphere Application Server ausgeführt wird.
      - Wenn die Ansicht keine WebSphere Application Server 7.0 enthält, klicken Sie mit der rechten Maustaste auf die Ansicht > **Neu** > **Server**. Folgen Sie den Optionen im Assistenten, um eine WebSphere Application Server 7.0 -Instanz zu erstellen.
      - Ist der Server vorhanden, wurde aber noch nicht gestartet, klicken Sie zum Starten auf die Pfeilspitze.
      - Um die Eigenschaften zu prüfen und einen schnellen Zugriff auf die Serverprotokolle zu erhalten, klicken Sie mit der rechten Maustaste auf **WebSphere Application Server 7.0 at localhost** > **Eigenschaften** > **WebSphere Application Server**.
      - Um den Server zu verwalten, verwenden Sie entweder einen externen Browser und öffnen Sie die URL `http://localhost:9061/ibm/console/unsecureLogon.jsp`oder klicken Sie mit der rechten Maustaste auf **WebSphere Application Server 7.0 at localhost** > **Administrationskonsole ausführen**.
      - Standardmäßig werden Veröffentlichungen automatisch ausgeführt. Viele Benutzer ziehen es vor, Server-Updates manuell zu implementieren. Klicken Sie doppelt auf **WebSphere Application Server 7.0 at localhost**und erweitern Sie das Pfeilsymbol **Veröffentlichung** im Fenster **Übersicht** . Klicken Sie auf **Nie automatisch veröffentlichen**.
      - Sie können auch eine weitere Standardeinstellung ändern: Deaktivieren Sie das Kontrollkästchen **Server beim Beenden der Workbench beenden** im Fenster **Übersicht**.
2. Erstellen Sie die JEE-Projekte.

Sie müssen ein Enterprise Application-Projekt (EAR) und ein Enterprise Java Bean-Projekt (EJB) erstellen.

a) Klicken Sie auf **Datei** > **Neu** > **EAR-Projekt**. Nennen Sie das Projekt `W3CJMSEAR` und klicken Sie auf **Fertigstellen**.

   Die Standardeinstellungen müssen WebSphere Application Server 7.0 als Ziellaufzeit und EAR-Version 5.0angeben. Die Standardkonfiguration muss ausgewählt werden.

b) Klicken Sie auf **Datei** > **Neu** > **EJB-Projekt**. Benennen Sie das Projekt `W3CJMSEJB`. Wählen Sie `W3CEARJMS` als **EAR-Projektname** > **Weiter**aus.

   Die Standardversion des EJB-Moduls lautet 3.0 und die Standardkonfiguration wird erneut verwendet.

c) Deaktivieren Sie das Kontrollkästchen **EJB-Client-JAR-Modul erstellen** und klicken Sie auf **Fertigstellen**.

3. Generieren und implementieren Sie den EJB-Web-Service aus der WSDL `StockQuoteAxis`.

a) Klicken Sie auf **Ausführen** > **Web Services Explorer starten**.

b) Wählen Sie über die Symbole im Fenster **Web Services Explorer** die WSDL-Seite aus und klicken Sie im Navigator auf **WSDL Main**.

c) Geben Sie im Fenster **Aktionen** die WSDL-URL für `StockQuoteAxis.wsdl` ein bzw. suchen Sie in diesem Fenster nach ihr.

   Wenn Sie Liberty mit `StockQuoteAxis` als HTTP-Service ausführen, lautet die URL:

   ```
   http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
   ```

   Wenn Sie die WSDL im Dateisystem verwenden, lautet die URL möglicherweise:

   ```
   File:\Dirpath\StockQuoteAxis\WebContent\wsdl\StockQuoteAxis.wsdl
   ```

d) Klicken Sie in der Navigationsstruktur auf die Zeile mit der importierten URL.

   Diese Zeile folgt unmittelbar auf **WSDL Main**, wenn dies die erste WSDL ist, die Sie in Web Services Explorer importiert haben.

e) Klicken Sie im Fenster **Aktionen** auf **Web-Service-Assistenten starten** > **Web-Service-Entwurf** > **Start**.

f) Wählen Sie im Web-Service-Assistenten **Top down EJB-Web-Service** aus.

   Wählen Sie die Konfiguration aus oder prüfen Sie sie anhand der Informationen in Tabelle 190 auf Seite 1389. Wählen Sie **Dateien ohne Warnung überschreiben** > **Weiter** aus.

Tabelle 190. Top-Down-Konfiguration des EJB-Web-Service

Feld	**Wert**
Server	`WebSphere Application Server 7.0`
Web-Service-Laufzeit	`IBM WebSphere JAX-WS`
Serviceprojekt	`W3CJMSEJB`
Service-EAR-Projekt	`W3CJMSEAR`
Konfiguration:	`No client generation`

g) Aktivieren Sie auf der Seite **Optionen zum Erstellen eines WebSphere-JAX-WS-EJB-Top-Down-Web-Service** das Kontrollkästchen **Zu JMS-Bindung wechseln**. Aktivieren Sie zudem **Wrapper-Stil aktivieren**, **WSDL in Projekt kopieren** und **Web-Service-Implementierungsdeskriptor generieren** aus und klicken Sie auf **Weiter**.

h) Aktivieren Sie auf der Seite **Konfiguration der WebSphere-JAX-WS-JMS-Bindung** das Kontrollkästchen **SOAP/JMS-Interoperabilitätsprotokoll verwenden** und geben Sie die Werte aus Tabelle 191 auf Seite 1390 ein. Lassen Sie die anderen Felder leer und klicken Sie auf **Weiter**.

Tabelle 191. Konfiguration der WebSphere-JAX-WS-JMS-Bindung	
Feld	**Wert**
JMS-Ziel	queue
JNDI-Name des Ziels:	requestaxis
JMS-Verbindungsfactory	qm1
Zielname für Antwort	W3CJMSEAR
Konfiguration:	replyaxis

a) Geben Sie auf der Seite **WebSphere-JAX-WS-Routerprojektkonfiguration** im Feld **JNDI-Name der Aktivierungsspezifikation** den Namen qm1as ein und klicken Sie auf **Weiter**.

RAD generiert und implementiert das Projekt innerhalb von 30 bis 60 Sekunden.

b) Ignorieren Sie die Optionen auf der Seite **Web-Service-Veröffentlichung** und klicken Sie auf **Fertigstellen**.

4. Prüfen Sie die generierte WSDL.

Die servicespezifische WSDL sollte generiert und im Projekt gespeichert werden.

a) Öffnen Sie im Enterprise-Explorer-Navigator den Ordner **W3CJMSEJB** > **ejbmodule** > **META-INF** > **wsdl**. Doppelklicken Sie auf die Datei StockQuoteAxis.wsdl, um sie im WSDL-Editor zu öffnen.

Überprüfen Sie die Bindung; folgende JMS-URL wird angezeigt:

```
jms:jndi:requestaxis?jndiConnectionFactoryName=qm1&targetService=StockQuoteAxis
```

5. Optional: Binden Sie die EJB mittels JAX-WS an SOAP over HTTP.

Durch die Bereitstellung von zwei Bindungen lassen Sie den Clients die Wahl, den Web-Service auch über die SOAP-Bindung aufzurufen. Zudem sind Clients so in der Lage, den Web-Server mittels HHTP abzufragen, um seine WSDL zu ermitteln.

Die Schritte zum Binden einer EJB an SOAP over HTTP sind nicht in der Task aufgeführt.

6. Implementieren Sie StockQuoteAxis und implementieren Sie den Service erneut mit dem Beispiel StockQuoteAxis.java.

a) Öffnen Sie im Enterprise Explorer-Navigator den Ordner **W3CJMSEJB** > **Services** Doppelklick StockQuoteAxisService , um die Implementierungsklasse in einem Java -Editor zu öffnen.

b) Öffnen Sie das Beispielprogramm StockQuoteAxis.java im Ordner *WebSphere MQ Installation directory*\tools\soap\samples\java\server > Wählen Sie alle Methoden aus, aber nicht den Klassennamen > **Kopieren**.

c) Wählen Sie in StockQuoteAxisSoapBindingImpl.java alle Methoden, aber nicht den Klassennamen aus, und fügen Sie die Methoden aus StockQuoteAxis.java ein.

d) Fügen Sie der WebSphere Application Server-Konsole eine auszugebende Druckanweisung hinzu, wenn der Service aufgerufen wird.
Ändern Sie die Methode 'getQuote(String symbol)':

```
public float getQuote(String symbol) {
System.out.println("StockQuoteAxisSoapBindingImpl called with symbol: "
+ symbol);
return ((float) 55.25);
```

e) Korrigieren Sie die Importe, indem Sie auf **Quelle** > **Importe organisieren** > **Speichern** klicken.

f) Korrigieren Sie die drei Fehler, die aufgetreten sind, da die Implementierung nicht mit der Schnittstelle übereinstimmt.

Die Fehler treten auf, da drei der Methoden in StockQuoteAxis.java Ausnahmen auslösen und die WSDL für den Service keine Fehlernachrichten enthält. Das Problem wird als Abweichung zwischen den Methodensignaturen und den Methodenanmerkungen des Web-Service diagnostiziert.

Versehen Sie entweder die Methoden mit Anmerkungen wie '@WebFault' und generieren Sie die WSDL erneut oder lassen Sie die Schnittstelle unverändert und entfernen Sie die Ausnahmen.

Damit die Schnittstelle unverändert bleibt, entfernen Sie die drei ausgelösten Ausnahmen (`throws exception`) aus den Methodensignaturen und klicken Sie auf **Speichern**.

Nächste Schritte

„Implementierung in einen Axis2-Client mit W3C SOAP over JMS” auf Seite 1432

Zugehörige Tasks

.NET 1- oder 2-Service für IBM MQ Transport for SOAP mithilfe von Microsoft Visual Studio 2012 entwickeln

Entwicklung des Web-Service 'SampleStockQuote' für .NET 1 oder .NET 2 mithilfe von Microsoft Visual Studio 2012.

IBM MQ-Web-Service-Clients für IBM MQ-Transport für SOAP entwickeln

Mit Ihrer normalen Entwicklungsumgebung entwickeln Sie Web-Service-Clients zur Verwendung des IBM MQ-Transports für SOAP.

Vorbereitende Schritte

Erstellen Sie den Service. Dazu können Sie eines der Beispielprogramme in „Web-Services für IBM MQ Transport for SOAP entwickeln” auf Seite 1383 verwenden.

Legen Sie fest, wie Sie die Client entwickeln, implementieren und verwenden möchten und wo Sie die WSDL für die Client-Generierung anfordern.

Legen Sie fest, wie Sie Clients und Services für IBM MQ Transport for SOAP entwickeln möchten.

Es gibt zwei Methoden.

1. Sie verwenden die Standardentwicklungstool, entwickeln einen HTTP-Service und -Client und verwenden dann die URL für WebSphere MQ Transport for SOAP.
2. Sie verwenden die Tools und Beispiele in IBM MQ Transport for SOAP.

Wenn Sie die HTTP-Methode verwenden, können Sie den Service sowohl auf einem HTTP-Server als auch mit IBM MQ Transport for SOAP ausführen. Zur Ausführung mit IBM MQ Transport for SOAP konfigurieren Sie den entsprechenden IBM MQ-Listener für SOAP und legen die Pfade und Bereitstellungsdeskriptoren für die Ausführung des Service fest. Die Tools in IBM MQ Transport for SOAP führen dann die Konfiguration aus. Alternativ können Sie die Umgebung für die Ausführung der Listener konfigurieren.

Die Tools in IBM MQ Transport for SOAP unterstützen Sie bei der Implementierung des Transports, wenn Sie darin noch keine Erfahrung haben. Für Produktionsarbeiten sind die Standardtools und die Implementierung desselben Service für verschiedene SOAP-Transporte vorteilhaft.

Zu entwickelnden Client-Typ festlegen

Sie müssen entscheiden, welchen Web-Service-Client-Typ Sie entwickeln möchten. Dies hängt davon ab, ob Sie die Serviceschnittstelle und die Serviceadresse kennen.

Ist die Schnittstelle bekannt, verwenden Sie Axis- oder .NET-Tools, um die Proxy-Client-Klassen in der Serviceschnittstelle zu generieren. Die Proxy-Client-Klassen erleichtern das Schreiben eines Clients zum Aufrufen des Service. Wenn Sie bei der Entwicklung des Clients den Speicherort des Service kennen, verwenden Sie die statische Proxy-Schnittstelle. Wenn sich der Speicherort des Service ändert, beispielsweise, wenn der Service auf einem Produktionsserver erneut implementiert wird, verwenden Sie die dynamische Proxy-Schnittstelle.

Wenn die Serviceschnittstelle zum Zeitpunkt der Client-Entwicklung nicht bekannt ist, können Sie unter Axis einen Dynamic Invocation Interface-Client (DII-Client) für Axis 1.4 erstellen. Eine DII-Client ruft alle Services über eine generische Schnittstelle auf. Um Parameter ordnungsgemäß an einen bestimmten Service zu übermitteln, müssen Sie die entsprechende Serviceschnittstelle programmge-

stützt erstellen – entweder im Client oder indem Sie die WSDL für den Service in den Client laden. Unter Axis2 können Sie einen Dispatch-Client erstellen. Der Dispatch-Client beschreibt die Clientanforderung mit einem Dokumentmodell, der DII-Client verwendet hingegen ein Klassenmodell. Beide erstellen die Anforderung dynamisch.

WSDL für den Service anfordern

Um einen Web-Service-Client zu erstellen, müssen Sie zunächst die Service-WSDL anfordern (außer bei programmgestützt erstellten Serviceschnittstellen). Die Service-WSDL kann von drei verschiedenen Quellen angefordert werden:

1. Direkt aus der Web-Service-Implementierung mithilfe eines Tools wie z. B. **java2wsdl** (Axis) oder **disco** (.NET).
2. Durch Abfrage des Web-Service über die URL: *Web service http url* ?wsdl.
3. Aus einer Datei, entweder in einem Dateisystem, oder aus einer Registry wie z. B. UDDI oder WebSphere-Service-Registry und Repository.

Anmerkung: Kann über HTTP nicht auf den Service zugegriffen werden, funktioniert die WSDL-Abfrage nicht. Der Service selbst ist unter Umständen nur unter Verwendung des IBM MQ-Transports für SOAP verfügbar.

Die mit **amqwdeployWMQService** generierte WSDL ist nicht identisch mit der WSDL, die mit **java2wsdl** oder **disco** generiert wurde. Zudem unterscheidet sich die generierte WSDL von jeder WSDL, mit der Sie möglicherweise den Service "Top Down" erstellt haben. Unter Axis ordnet der Bereitstellungsdeskriptor `server-config.wsdd` die von einem Client erstellte SOAP-Nachricht einer Operation und einem Service zu. **amqwdeployWMQService** generiert einen anderen Bereitstellungsdeskriptor in Eclipse.

Mit welcher WSDL Sie Clients erstellen, hängt davon ab, wie der Service implementiert ist:

Bereitstellung mit `amqwdeployWMQService`
Verwenden Sie die mit **amqwdeployWMQService** generierte WSDL. Geben Sie das Flag -w an und wählen Sie die WSDL `rpcLiteral` aus. Aus Gründen der Kompatibilität können Sie die WSDL `rpcEncoded` auswählen. Die WSDL `rpcEncoded` funktioniert nur mit .NET-Clients und Clients in Axis 1.4.

Manuelle Implementierung mit `SimpleJavaListener`
Verwenden Sie eine der folgenden WSDL-Dateien:

1. WSDL, mit der der Service definiert wird oder die in einem Repository gespeichert ist.
2. WSDL, die vom Service mit **java2wsdl** generiert wurde.
3. WSDL, die über die URL *Web service http url* ?wsdlabgefragt wird, sofern sie von einem HTTP-Server verfügbar ist. Führen Sie ein Tool wie Web Services Explorer aus, um die Servicedefinition direkt in Eclipse zu importieren.

Unter Umständen müssen Sie den URI für den Service ändern. Ändern Sie ihn in der Adresse des HTTP-Service, indem Sie den URI für IBM MQ Transport for SOAP angeben.

Manuelle Bereitstellung mit `amqSOAPNETListener`.
Verwenden Sie eine der folgenden WSDL-Dateien:

1. WSDL, mit der der Service definiert wird oder die in einem Repository gespeichert ist.
2. WSDL aus der .NET -Serviceklasse (.asmx) abgerufen. mit **disco**.
3. WSDL, die mit der URL *web services http url* ?wsdlabgefragt wird, falls verfügbar. Führen Sie ein Tool wie Web Services Explorer aus, um die Servicedefinition direkt in Eclipse zu importieren.
4. Aus der Serviceklasse .NET (.asmx) mit **amqswsdl** abgerufene WSDL.

Unter Umständen müssen Sie den URI für den Service ändern. Ändern Sie ihn in der Adresse des HTTP-Service, indem Sie den URI für IBM MQ Transport for SOAP angeben.

Implementiert in Windows Communication Foundation
Rufen Sie die Service-WSDL über die URL *Web service http url* ?wsdlab. Der Service muss mit der Verhaltenskonfiguration `serviceMetaData` definiert sein.

Bereitstellung in eine andere Serverplattform.
Folgen Sie den plattformspezifischen Anweisungen zum Anfordern der richtigen Service-WSDL.

Informationen zu diesem Vorgang

Entwickeln Sie Clients mit den Standardentwicklungstool. Die nachfolgenden Tasks veranschaulichen, wie Sie Clients für .NET 1 und 2, Axis 1.4 (JAX-RPC) und Axis2 (JAX-WS) erstellen. Informationen zur Vorgehensweise für Windows Communication Foundation finden Sieunter den zugehörigen Task-Links.

JAX-RPC-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln

Entwicklung eines Axis 1.4-Web-Service-Clients mit dem IBM MQ-Transport für SOAP.

Vorbereitende Schritte

Der Service muss verfügbar sein. Stellen Sie sicher, dass ein Anwendungsserver in Eclipse ausgeführt wird, der Axis 1.4 Web-Services unterstützt. In dieser Task verwenden wir das frei verfügbare Liberty-Profil. Sie können auch Tomcat 6 verwenden, einen kleineren Open-Source-Anwendungsserver.

Informationen zu diesem Vorgang

Die Task veranschaulicht die Entwicklung von drei Clienttypen für den unter Windows ausgeführten Beispielservice 'StockQuoteAxis' mit Eclipse. Bei den Clients handelt es sich um einen statischen und einen dynamischen Client, die mit dem Client-Proxy entwickelt wurden, sowie um einen DII-Client.

Dargestellt werden zwei alternative Methoden zur Generierung der Client-Proxys in der WSDL:

1. Client-Proxys mit **amqwdeployWMQService** generieren.
2. WSDL in Eclipse importieren und die Client-Proxys mit dem Web-Service-Assistenten erstellen.

Vorgehensweise

1. Starten Sie die Eclipse-IDE für Java EE-Entwickler.
2. Erstellen Sie ein Java-Projekt namens `StockQuoteAxisClient`:
   a) Wechseln Sie zur Java-Perspektive und klicken Sie auf **Datei** > **Neu** > **Java-Projekt**. Geben Sie im Feld **Project name** der Seite **Java-Projekt erstellen** den Wert `StockQuoteAxisEclipseClient`ein. Stellen Sie sicher, dass die Ausführungsumgebung entweder **J2SE1-1.4** ist oder **J2SE-1.5** und klicken Sie auf **Weiter**.
   b) Wählen Sie auf der Seite **Java-Einstellungen** in der Registerkarte **Bibliotheken** die Option **Externe JAR-Dateien hinzufügen** aus.
   c) Navigieren Sie zum Ordner *MQ_INSTALLATION_PATH*`/java/lib`, wählen Sie alle JAR-Dateien (`.jar`) aus und klicken Sie auf **Öffnen**.
   MQ_INSTALLATION_PATH ist das Verzeichnis, in dem IBM MQ installiert ist.
   d) Navigieren Sie zum Ordner *MQ_INSTALLATION_PATH*`/java/lib/soap`, wählen Sie alle JAR-Dateien (`.jar`) aus und klicken Sie auf **Öffnen**. `axis.jar` muss aus dem IBM MQ-Installationsmedium in diesem Verzeichnis installiert worden sein.
   MQ_INSTALLATION_PATH ist das Verzeichnis, in dem IBM MQ installiert ist.
   e) Die Registerkarte **Bibliothek** verweist jetzt auf alle `.jar` -Dateien, die zum Erstellen des Clients erforderlich sind > **Fertigstellen**.
3. Verwenden Sie eine dieser beiden Methoden, um in Eclipse Proxys für den Beispiel-Web-Service 'StockQuoteAxis' zu erstellen:
   - Client-Proxys mit **amqwdeployWMQService** generieren.

a. Mit diesem Befehl wird ein Warteschlangenmanager erstellt. Erstellen Sie für diese Task `QM1` als Standardwarteschlange.

b. Erstellen Sie ein Arbeitsverzeichnis namens `samples`. Kopieren Sie das Beispielprogramm `StockQuoteAxis.java` in das Verzeichnis `samples/soap/server`.

c. Ändern Sie `amqwsetcp.cmd` in *MQ_INSTALLATION_PATH*`/bin`, um das aktuelle Verzeichnis im Klassenpfad einzuschließen. *MQ_INSTALLATION_PATH* ist das Verzeichnis, in dem IBM MQ installiert ist.

d. Öffnen Sie ein Befehlsfenster in `samples` und führen Sie den geänderten Befehl **amqwsetcp** aus.

e. Erstellen Sie WSDL für den Service 'StockQuoteAxis', indem Sie folgenden Befehl ausführen:

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genAxisWsdl
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

Hinweis: Verwenden Sie bei Java-Befehlen "`/`" anstatt "`.`" oder "`\`".

Tipp: Anstatt die generierten Proxys in Eclipse zu importieren, können Sie die generierte WSDL aus `.samples/generated` importieren. Die daraus resultierenden Proxys unterscheiden sich in zwei Aspekten:

i) Die Paketnamen sind unterschiedlich - dies können Sie refaktorieren.

ii) Die mit Eclipse generierten Proxys enthalten die zusätzliche Helper-Klasse `StockQuoteAxisProxy.java`.

f. Erstellen Sie Client-Proxys für den Service 'StockQuoteAxis', indem Sie folgenden Befehl ausführen:

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genProxiestoAxis
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

g. Importieren Sie die Client-Proxys in 'StockQuoteAxisClient':

i) Klicken Sie mit der rechten Maustaste auf **StockQuoteAxisClient\src** und wählen Sie dann nacheinander **Dateisystem** > **Weiter** > **Durchsuchen** aus. Suchen Sie den Ordner `.\samples\generated\client\remote\soap\server` und klicken Sie auf **OK**.

ii) Aktivieren Sie auf der Seite **Import** die Option **Server** und klicken Sie auf **Fertigstellen**.

h. Refaktorieren Sie den Paketnamen in `soap.server`.

i) Klicken Sie mit der rechten Maustaste auf das Paket mit den Client-Proxys und wählen Sie **Refaktorieren** > **Umbenennen** aus. Geben Sie **New name**ein: `soap.server` > übernehmen Sie die ausgewählten Standardwerte für die anderen Auswahlmöglichkeiten > **OK**. Alle Fehler werden korrigiert.

- Client-Proxys mit Eclipse generieren.

  Sie können die WSDL für den Service abrufen. In diesem Beispiel wird der Service über das Liberty-Profil zur Verfügung gestellt und Sie erhalten die WSDL vom Web-Server.

  a. Wechseln Sie in Eclipse zur Webperspektive und stellen Sie sicher, dass das Liberty-Profil ausgeführt wird und 'StockQuoteAxis' bereitgestellt und synchronisiert wurde.

  b. Importieren Sie die WSDL in Web Services Explorer:

  i) Klicken Sie in der Aktionsleiste auf das Symbol **Web Services Explorer** oder klicken Sie auf **Ausführen** > **Web Services Explorer starten**.

  ii) Klicken Sie im Web Services Explorer auf das WSDL-Seitensymbol, um zur WSDL-Seite zu wechseln.

  iii) Klicken Sie im Navigatorfenster von Web Services Explorer auf **WSDL Main**.

iv) Geben Sie die URL des Web-Service gefolgt von ?WSDL ein. Die URL für 'StockQuoteAxis', implementiert im Liberty-Profil, lautet:

```
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
```

c. Generieren Sie die Client-Proxys:

i) Klicken Sie im Web Services Explorer auf **http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl**.

ii) Klicken Sie im Fenster **Aktionen** auf **Web-Service-Assistent starten**. Lassen Sie **Web-Service-Client** ausgewählt und klicken Sie auf **Start**.

iii) Klicken Sie in der ersten Seite des Assistenten in der Konfiguration auf den Projektlink **Client**. Wählen Sie das Clientprojekt **StockQuoteAxisClient** aus und klicken Sie auf **OK**.

Tipp: Unter Umständen verliert das Assistentenfenster den Fokus. Stellen Sie ihn manuell wieder her.

iv) Die Web-Service-Laufzeit muss Apache Axis sein, um einen JAX-RPC-Client zu generieren.

v) Klicken Sie auf **Fertigstellen**.

vi) Ändern Sie die statische URL des Service, sodass sie für den Service 'StockQuoteAxis' auf die Adresse des IBM MQ-Transports für SOAP verweist. Sie können diesen Schritt überspringen, bis Sie den Client mit einem HTTP-Server getestet haben.

a) Öffnen Sie die Datei `StockQuoteAxisServiceLocator.java` und suchen Sie die Deklaration für `StockQuoteAxis_address`.

b) Ändern Sie die URL in

```
"jms:/queue?destination=REQUESTAXIS
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
&amp;connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

Tipp: Wenn Sie Zeichenfolgen kopieren und in `.java`-Code einfügen, transformiert Eclipse & automatisch in `&` und umgekehrt.

d. Erstellen Sie drei Java-Clientklassen (jede mit einer Hauptmethode):

i) Paket erstellen. Klicken Sie mit der rechten Maustaste auf **StockQuoteAxisClient/src** > **Neues Paket**. Nennen Sie das Paket `soap.client` und klicken Sie auf **Fertigstellen**.

ii) Wählen Sie **soap.client** > **Neu** > **Klasse** aus. Nennen Sie die Klasse `SQAStaticClient`. Aktivieren Sie das Kontrollkästchen **public static void main(string [] args)** und klicken Sie auf 'Fertigstellen'.

iii) Wiederholgen Sie den Vorgang, um `SQADynamicClient.java` und `SQADIIClient.java` zu erstellen.

e. Schreiben Sie den Client-Code.

In Abbildung 173 auf Seite 1399 bis Abbildung 177 auf Seite 1401 werden Beispiele der drei Arten von Client-Code gezeigt. Die Beispiele verwenden eine HTTP-URL, um den Client zu testen, der den in einen HTTP-Server implementierten Service 'StockQuoteAxis' verwendet. Um die Clients für den implementierten Service 'StockQuoteAxis' mit IBM MQ Transport for SOAP auszuführen, müssen Sie die URL wie folgt ändern:

```
"jms:/queue?destination=REQUESTAXIS
connectionFactory=(connectQueueManager(QM1)binding(auto))
initialContextFactory=com.ibm.mq.jms.Nojndi
targetService=soap.server.StockQuoteAxis.java
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
```

– Abbildung 173 auf Seite 1399 und Abbildung 175 auf Seite 1400 verwenden den in Eclipse generierten Proxy, der über die zusätzliche Helper-Klasse 'StockQuoteAxisproxy' verfügt, welche das Codieren etwas vereinfacht.

– Abbildung 174 auf Seite 1400 und Abbildung 176 auf Seite 1400 verwenden den mit **amqwdeployWMQService** generierten Proxy.
– Abbildung 177 auf Seite 1401 verwendet keine Proxy-Klassen.

Jeder Client ruft `com.ibm.mq.soap.Register.extension()` auf, um eine Verknüpfung zu IBM MQ Transport for SOAP herzustellen. Die Erweiterung ist im Clientbereitstellungsdeskriptor registriert. Eine Beschreibung der Clientbereitstellung in Axis 1.4 finden Sie unter „Web-Service-Client für Axis 1.4 zur Verwendung von IBM MQ Transport for SOAP implementieren" auf Seite 1427.

f. Führen Sie die Clients aus, indem Sie die SOAP-Anforderungen an 'StockQuoteAxis' auf dem WebSphere Application Server Community Edition-Server senden, der im Arbeitsbereich konfiguriert wurde.

i) Stellen Sie sicher, dass der Server ausgeführt wird und dass 'StockQuoteAxis' implementiert und synchronisiert wurde.

ii) Wählen Sie den Client aus bzw. öffnen Sie den Client, den Sie testen möchten, und klicken Sie dann in der Aktionsleiste auf **Ausführen** . Alternativ können Sie auf das grüne Symbol für 'Ausführen' klicken oder im Navigator mit der rechten Maustaste auf den Client klicken und dann **Ausführen als** > **Ausführungskonfigurationen** auswählen. Konfigurieren Sie die Parameter, die zum Ausführen des Clients erforderlich sind.

g. Führen Sie den Client mit IBM MQ Transport for SOAP aus.

Hierbei wird **amqwdeployWMQService** zur Service-Implementierung verwendet. Dies funktioniert nur mit dem Client, der die mit **amqwdeployWMQService** erstellte WSDL bzw. die damit erstellten Proxys verwendet. Um den Client mit der Original-WSDL oder mit Proxys auszuführen, die in Eclipse erstellt wurden, müssen Sie den Service mit seinem in Eclipse erstellten Bereitstellungsdeskriptor implementieren. Starten Sie **SimpleJavaListener** manuell, indem Sie den bindenden Service-Port-Namen als Zielservicenamen `targetServiceName` verwenden.

i) Befolgen Sie die Anweisungen unter „Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren" auf Seite 1420, um den Service im IBM MQ Simple Java SOAP Listener zu implementieren. Die Servicebereitstellung funktioniert nur für den Client, der die mit **amqwdeployWMQService** erstellte WSDL bzw. die damit erstellten Client-Proxys verwendet.

ii) Führen Sie in einem Befehlsfenster den Befehl **amqwclientconfig** aus, um die Datei mit dem Clientbereitstellungsdeskriptor `client-deploy.wsdd` zu erstellen.

iii) Importieren Sie `client-deploy.wsdd` in das Stammverzeichnis des Java-Projekts, das Sie mit IBM MQ Transport for SOAP testen möchten.

a) Klicken Sie mit der rechten Maustaste auf das Java -Projekt **StockQuoteAxisEclipseClient** > **Importieren** > **Dateisystem** > **Weiter** > **Durchsuchen ...**

b) Durchsuchen Sie das Verzeichnis mit der Datei `client-deploy.wsdd` und klicken Sie auf **Öffnen**. Wählen Sie auf der Assistentenseite **Importieren** das Verzeichnis aus und aktivieren Sie die Option `client-deploy.wsdd`.

c) Vergewissern Sie sich, dass im Feld **Into folder** (In Ordner) der Wert `StockQuoteAxisEclipseClient` angegeben ist, und klicken Sie auf **Fertigstellen**.

iv) Bestätigen Sie, dass `StockQuoteAxisEclipseClient` als Arbeitsverzeichnis zum Ausführen einer Java-Anwendung in diesem Projekt verwendet wird:

Klicken Sie mit der rechten Maustaste auf das Java Projekt **StockQuoteAxisEclipseClient** > **Ausführen als ...** > **Ausführungskonfigurationen ...** > Wählen Sie die Registerkarte **(x) = Argumente** aus. Prüfen Sie, ob im Arbeitsverzeichnis das Optionsfeld **Standard** ausgewählt ist und der Pfad `StockQuoteAxisEclipseClient`lautet. Nehmen Sie alternativ eine der folgenden Auswahlen vor, um einen anderen Speicherort oder eine andere Datei mit der Clientkonfiguration auszuwählen:

– Aktivieren Sie die Option **Sonstige** und geben Sie einen Verzeichnispfad Ihrer Wahl ein.

– Geben Sie im Fenster **VM-Argumente** `-Daxis.ClientConfigFile=` *`full path to client deployment descriptor file`* ein.

v) Stellen Sie sicher, dass die URL so konfiguriert ist, dass Sie auf den implementierten Service verweist, der IBM MQ Transport for SOAP verwendet. Führen Sie den Client wie in Schritt ii beschrieben aus.

Tipp: Typischerweise tritt unter Umständen einer der folgenden Fehler auf:

i) `Exception: No client transport named 'jms' found!.`

ii) Ein JMS-Verbindungsfehler.

iii) `Exception: The AXIS engine could not find a target service to invoke! targetService is soap.server.StockQuoteAxis.java`

iv) `Exception: java.lang.InstantiationException: soap.server.StockQuoteAxis`

Erläuterungen:

i) `client-config.wsdd` wurde nicht gefunden oder enthält die Zeile `<transport name="jms" pivot="java:com.ibm.mq.soap.transport.jms.WMQSender"/>` in `client-config.wsdd`.

ii) Möglicherweise ein Problem mit dem Erstellungspfad - keine JAR-Dateien in *`MQ_INSTALLATION_PATH`*`/java/lib` enthalten. *`MQ_INSTALLATION_PATH`* ist das Verzeichnis, in dem IBM MQ installiert ist.

iii) Problem bei der Servicebereitstellung, entweder mit `server-config-wsdd` oder mit den an den **`SimpleSoapListener`** übermittelten Parametern.

iv) Abweichung zwischen Bereitstellungsdeskriptor und der Bereitstellung des Service.

Sollten Probleme beim Ausführen des Clients in Eclipse auftreten, verwenden Sie ein Befehlsfenster:

i) Wechseln Sie in der Baumstruktur des Arbeitsbereichsverzeichnisses zum Verzeichnis `StockQuoteAxisEclipseClient\bin`.

ii) Führen Sie die Befehle **amqwsetcp** und **amqwclientconfig** aus.

iii) Führen Sie `java soap/client/SQAStaticClient` aus.

JAX-RPC-Web-Service-Beispielclients

Die Java-Web-Service-Beispielclients von IBM MQ sind im Verzeichnis *`MQ_INSTALLATION_PATH`*`\tools\soap\samples\java\clients` installiert. *`MQ_INSTALLATION_PATH`* ist das Verzeichnis, in dem IBM MQ installiert ist.

`SQAxis2Axis.java`
`SQAxis2Axis.java`, Abbildung 170 auf Seite 1398, ist ein dynamischer Proxy-Client zum Aufrufen des Service `StockQuoteAxis`. Sie können die URL des Service überschreiben, der in den dynamischen Proxy kompiliert wird, indem Sie in der Befehlszeile eine URL eingeben.

`SQAxis2DotNet.java`
`SQAxis2DotNet.java`, Abbildung 171 auf Seite 1398, ist ein dynamischer Proxy-Client zum Aufrufen des Service `StockQuoteDotNet`. Sie können die URL des Service überschreiben, der in den dynamischen Proxy kompiliert wird, indem Sie in der Befehlszeile eine URL eingeben.

`WsdlClient.java`
`WsdlClient.java`, Abbildung 172 auf Seite 1399, ist ein dynamischer Aufrufclient zum Aufrufen des Service `StockQuoteDotNet` bzw. `StockQuoteAxis`. Standardmäßig ruft der Client den Service `StockQuoteAxis` auf. Fügen Sie die Befehlszeilenoption '-D' ein, um den Service `StockQuoteDotNet` aufzurufen, und die Befehlszeilenoption '-w', um einen anderen Port bereitzustellen als der in `.\generated\StockQuoteDotNet_Wmq.wsdl` angegeben ist.

```
package soap.clients;
import java.net.URL;
import soap.server.*;
public class SQAxis2Axis {
   public static void main(String[] args) {
      com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
         StockQuoteAxis service = null;
         if (args.length == 0)
            service = locator.getSoapServerStockQuoteAxis_Wmq();
         else
            service = locator.getSoapServerStockQuoteAxis_Wmq(
                new java.net.URL(args[0]));
         System.out.println("Response: " + service.getQuote("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                            e.printStackTrace();
         System.exit(2);
      }
  }
```

Abbildung 170. SQAxis2Axis.java

```
public class SQAxis2DotNet {
public static void main(String[] args) {
   com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteDotNet locator = new StockQuoteDotNetLocator();
         StockQuoteDotNetSoap_PortType service = null;
         if (args.length == 0)
            service = locator.getStockQuoteDotNetSoap();
         else
            service = locator.getStockQuoteDotNetSoap(new java.net.URL(
                  args[0]));
         System.out.println("Response: " + service.getQuoteDOC("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                            e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Abbildung 171. SQAxis2DotNet.java

```
package soap.clients;
import com.ibm.mq.soap.*;
import org.apache.axis.utils.Options;
import java.net.URL;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
public class WsdlClient {
public static void main(String[] args) {
      String wsdlService, wsdlPort, namespace, wsdlSource, wsdlTargetURI, s;
      try {
         Register.extension();
         Options opts = new Options(args);
         if (opts.isFlagSet('D') != 0) {
            wsdlService = "StockQuoteDotNet";
            wsdlPort = "StockQuoteDotNetSoap";
            namespace = "http://stock.samples";
            wsdlSource = "file:generated/StockQuoteDotNet_Wmq.wsdl";
         } else {
            wsdlService = "StockQuoteAxisService";
            wsdlPort = "soap.server.StockQuoteAxis_Wmq";
            namespace = "soap.server.StockQuoteAxis_Wmq";
            wsdlSource = "file:generated/soap.server.StockQuoteAxis_Wmq.wsdl";
         }
         if (null != (s = (opts.isValueSet('w'))))
            wsdlPort = s;
         System.out.println("start WsdlClient demo, wsdl port " + wsdlPort
               + " resolving uri to ...");
         QName servQN = new QName(namespace, wsdlService);
         QName portQN = new QName(namespace, wsdlPort);
         Service service = ServiceFactory.newInstance().createService(
               new URL(wsdlSource), servQN);
         Call call = (Call) service.createCall(portQN, "getQuote");
         wsdlTargetURI = call.getTargetEndpointAddress().toString();
         System.out.println("  '" + wsdlTargetURI + "'");
         Object ret = call.invoke(new Object[] { "XXX" });
         System.out.println("Response: " + ret);
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING WsdlClient DEMO <<<\n");
         e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Abbildung 172. WsdlClient.java

The example clients used in this task:

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy();
      System.out.println("Static client synchronous result is:"
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Abbildung 173. Statischer Client, der einen mit Eclipse generierten Proxy verwendet

```
package soap.client;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq();
      System.out.println("Static client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Abbildung 174. Statischer Client, der einen mit 'amqwdeployWMQService' generierten Proxy verwendet

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Abbildung 175. Dynamischer Client, der einen mit Eclipse generierten Proxy verwendet

```
package soap.client;

import java.net.URL;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL sqaURL = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq(sqaURL);
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Abbildung 176. Dynamischer Client, der einen mit 'amqwdeployWMQService' generierten Proxy verwendet

```
package soap.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
public class SQADIIClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL wsdl = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl");
      Service SQAService = (ServiceFactory.newInstance()).createService(wsdl,
          new QName("http://server.soap", "StockQuoteAxisService"));
      Call SQACall = SQAService.createCall(new QName("http://server.soap",
          "StockQuoteAxis"), "getQuote");
      System.out.println("DII client synchronous result is "
          + SQACall.invoke(new Object[] { "ibm" }));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Abbildung 177. DII-Client (kein Proxy)

Zugehörige Tasks

JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln
Entwicklung eines Axis2-Web-Service-Clients zur Verwendung des IBM MQ-Transports für SOAP. Die mit IBM MQ Transport for SOAP bereitgestellten Axis2-Beispielclients und der Befehl **wsimport** zum Generieren von Proxys werden aufgeführt.

.NET 1- oder 2-Client für WebSphere Transport for SOAP mit Microsoft Visual Studio 2012 Visual Studio 2008 entwickeln
Entwicklung eines .NET 1- oder 2-Web-Service-Client für den IBM MQ-Transport für SOAP.

JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln

Entwicklung eines Axis2-Web-Service-Clients zur Verwendung des IBM MQ-Transports für SOAP. Die mit IBM MQ Transport for SOAP bereitgestellten Axis2-Beispielclients und der Befehl **wsimport** zum Generieren von Proxys werden aufgeführt.

Vorbereitende Schritte

Rufen Sie die Axis2-Bibliotheken ab und konfigurieren Sie eine Entwicklungs- und Testumgebung für die Ausführung des Clients.

Anmerkung: Die Benennung der von Axis verwendeten Versionen und Releases führt zu einigen Unklarheiten. Für gewöhnlich bezieht sich Axis 1.4 auf die JAX-RPC-Implementierung, während sich Axis2 auf die JAX-WS-Implementierung bezieht.

Axis 1.4 ist ein Versionsstand. Wenn Sie im Internet nach Axis 1.4 suchen, werden Sie zu http://ws.apache.org/axis/ geführt. Die Seite enthält eine Liste früherer Axis-Versionen (1.2, 1.3) und das letzte Release von Axis am 22. April 2006 (Version 1.4). Es gibt höhere Releases von Axis 1.4, in denen Fehler korrigiert wurden, die aber alle unter Axis 1.4 zusammengefasst sind. Eines dieser Releases mit Fehlerkorrekturen wird mit IBM MQ ausgeliefert. Verwenden Sie für Axis 1.4 die Version von `axis.jar`, die mit IBM MQ ausgeliefert wird, anstelle der Version, die über http://ws.apache.org/axis/ erhältlich ist.

Die Axis-Website verwendet auch den Begriff Axis 1.1 für alle Versionen dessen, was für gewöhnlich als Axis 1.4 bezeichnet wird. Axis 1.2 bezieht sich auf das Produkt, das normalerweise als Axis2 bezeichnet wird.

Axis 1.5 ist kein höheres Release von Axis 1.4, sondern ein Axis2-Release. Wenn Sie nach Axis 1.5 suchen, werden Sie zur Website http://ws.apache.org/axis2/ geleitet.https://ws.apache.org/axis2/download.cgi enthält eine Liste der Releaseversionen von Axis2 mit den Bezeichnungen 0.9 bis 1.5.1 (und darunter verwirrenderweise Version 1.4). Bei IBM MQ Transport for SOAP muss die Axis2-Releaseversion 1.4.1

verwendet werden. Laden Sie Axis2 1.4.1 von der Website http://ws.apache.org/axis2/download/1_4_1/download.cgi herunter.

Sie können Proxys für die Web-Service-Clients für den IBM MQ-Transport für SOAP entweder mit dem Befehl **wsimport** oder mit IDE-Tools generieren. Die Eclipse-IDE für Java EE-Entwickler 3.5 SR1 verwendet den Befehl **wsdl2java**. **wsimport** wird mit Java 6 bereitgestellt. Mit Java 5 können Sie Client-Proxys ausführen, die mit **wsimport** oder **wsdl2java**generiert wurden.

Die mit dem IBM MQ-Transport für SOAP bereitgestellten Axis2-Web-Service-Beispielclients wurden mithilfe von **wsimport** entwickelt; siehe „Axis2-Beispielclients” auf Seite 1407.

Die folgende Task veranschaulicht die Generierung und Verwendung der vom Web-Services-Assistenten aus der Eclipse-IDE für Java EE-Entwickler erstellten Proxys. Die Beispielclients veranschaulichen die Verwendung der Proxys, die von **wsimport** erstellt werden.

Wenn Sie den Web-Service-Assistenten verwenden möchten, müssen Sie der Workbench einen Anwendungsserver hinzufügen, der Axis2 unterstützt. Die Schritte zeigen, wie Sie das Liberty-Profil für die Unterstützung von Axis2 mit der Workbench konfigurieren können.

1. Konfigurieren Sie den in der Eclipse-IDE für Java EE-Entwickler verwendeten Anwendungsserver für die Unterstützung von Axis2. Konfigurieren Sie für dieses Beispiel das Liberty-Profil.
   a. Öffnen Sie die Arbeitsbereichsvorgaben für die Konfiguration des Servers: Öffnen Sie **Fenster** > **Einstellungen**.
   b. Vergewissern Sie sich, dass die installierte Java Runtime Environment (JRE) `Java50` ist: Klicken Sie auf **Installierte JREs**.
   c. Fügen Sie das Liberty-Profil als Server hinzu:
   d. Fügen Sie Axis2 hinzu: Klicken Sie auf **Web-Services** > **Axis2-Einstellungen**. Auf der Registerkarte **Axis2 Runtime** > **Durchsuchen ...** Öffnen Sie das Verzeichnis mit den vielen `Axis2` -JAR-Dateien > **Anwenden**.
   e. Verknüpfen Sie Liberty mit Axis2: Klicken Sie auf **Web-Services** > **Server und Runtime**. Wählen Sie unter **`Server` IBM Liberty Server auswählen**und unter **`Web service runtime` Apache Axis2** > **Anwenden** > **OK** aus.
   f. Starten Sie den Server: Öffnen Sie die Webperspektive und die Serveransicht. Klicken Sie mit der rechten Maustaste auf die Serveransicht und wählen Sie dann **Neu** > **Server** aus. Vergewissern Sie sich, dass **IBM Liberty Server** ausgewählt und konfiguriert ist, und klicken Sie dann auf **Fertigstellen**. Starten Sie den-Server.
2. Vergewissern Sie sich, dass Sie den Service 'StockQuoteAxis' in Liberty zur Ausführung des Web-Service-Assistenten implementiert haben.
3. Um den Service mit IBM MQ Transport for SOAP zu testen, stellen Sie den Service über einen IBM MQ-Listener mit Transport for SOAP für Axis 1.4 bereit. Weitere Informationen hierzu finden Sie im Abschnitt Liberty-Profil.

Informationen zu diesem Vorgang

Die Eclipse-IDE für Java EE-Entwickler verwendet Java50 und den Web-Services-Assistenten zum Generieren der Proxyklassen für den Service. Die Proxy-Klassen unterscheiden sich von den Klassen, die vom Tool **wsimport** erstellt werden, das mit Java 6 bereitgestellt wird. Eine alternative Methode besteht darin, die Proxy-Klassen mit **wsimport** zu generieren und die erstellten Pakete in Ihre Eclipse Java EE IDE for Web Developers zu importieren.

Der Web-Service-Assistent in der Eclipse-IDE für Java EE-Entwickler erstellt einen Web-Service-Client in einem Webprojekt. Sie können den Client als einfache Java-Anwendung ausführen; ein Anwendungsserver ist nicht erforderlich. Sie können auch den Code in ein Java-Projekt übertragen und den Erstellungspfad so konfigurieren, dass er die Axis2 JAR-Dateien einbezieht.

Vorgehensweise

1. Erstellen Sie ein Webprojekt in einem neuen Unternehmensprojekt:

a) Stellen Sie sicher, dass im Projektexplorer kein Eintrag ausgewählt ist. Klicken Sie mit der rechten Maustaste auf den Leerraum und wählen Sie dann **Neu** > **EAR-Projekt** aus. Nennen Sie das Projekt `StockQuoteAxis2EAR` und klicken Sie abschließend auf **Fertigstellen**. Antworten Sie mit No auf das Fenster, das Ihnen die Option zum Öffnen der Java EE-Perspektive bietet.

   Die Standardwerte sind so festgelegt, dass Liberty verwendet wird.

b) Klicken Sie mit der rechten Maustaste auf `StockQuoteAxis2EAR` und wählen Sie dann **Neu** > **Dynamisches Webprojekt** aus. Nennen Sie das Projekt `StockQuoteAxis2WebClient` und aktivieren Sie das Kontrollkästchen für EAR-Mitgliedschaft, um das Projekt zu **StockQuoteAxis2EAR** hinzuzufügen. Liberty ist als Ziellaufzeit ausgewählt.

c) Klicken Sie auf der Seite **Neues dynamisches Webprojekt** im Bereich 'Konfiguration' auf **Ändern** und aktivieren Sie die Projektfacette für für Axis2-Web-Services. **Dynamic Web Module 2.5**, **Java 6.0**und **Liberty** sind bereits überprüft. > **OK** > **Fertigstellen**. Antworten Sie mit No auf das Fenster, das Ihnen die Option zum Öffnen der Java EE-Perspektive bietet.

2. Importieren Sie die WSDL für den Service in den Arbeitsbereich und generieren Sie den Client-Proxy:

   In diesem Beispiel enthält das WSDL-Dokument die HTTP-Servicebindung und wird zum Ziel für den statischen Web-Client-Proxy. Sie können die URL in der Web-Servicebindung so ändern, dass sie auf die URL des IBM MQ-Transports für SOAP verweist, bevor Sie den Client-Proxy generieren. Der statische Web-Client-Proxy ist dann der Service, der in IBM MQ Transport for SOAP implementiert wird.

   a) Starten Sie Web Services Explorer, indem Sie entweder in der Aktionsleiste auf das entsprechende Symbol klicken oder **Ausführen** > **Web Services Explorer starten** auswählen.

   b) Wählen Sie den WSDL-Explorer aus, indem Sie im Fenster **Web Services Explorer** auf das WSDL-Symbol klicken. Wählen Sie im Navigatorfenster den Eintrag **WSDL Main** aus. Geben Sie die URL der WSDL-Datei für 'StockQuoteAxis' ein und klicken Sie auf **Start**.
   Fordern Sie in diesem Beispiel die WSDL direkt vom HTTP-Service an: `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl`

   c) Klicken Sie im Navigator auf die Zeile mit der URL des Web-Service. Klicken Sie im Fenster **Aktionen** auf **WSDL in Workbench importieren**. Wählen Sie den Eintrag **StockQuoteAxis2WebClient** als **Workbenchprojekt** aus. Geben Sie im Feld **WSDL-Dateiname** den Namen `StockQuoteAxisHTTP.wsdl` ein und klicken Sie auf **Start**.

   d) Klicken Sie mit der rechten Maustaste auf **StockQuoteAxisHTTP.wsdl** und wählen Sie dann **Web-Services** > **Client generieren** aus. Stellen Sie sicher, dass die Konfigurationsinformationen zur Web-Services-Seite des Assistenten wie folgt lauten: Server: IBM WASCE v2.1 Server, Web-Service-Laufzeit: Apache Axis2, Clientprojekt: StockQuoteAxis2WebClient, Client-EAR-Projekt: StockQuoteAxisEAR. Wenn Sie die Konfiguration korrigieren möchten, klicken Sie auf die Zeilen, die falsch sind.

   e) Klicken Sie auf **Weiter**. Überprüfen Sie die Codegenerierungseinstellungen und klicken Sie dann auf **Fertigstellen**.

      Beachten Sie, dass ein neues Paket (`soap.server`) erstellt wird, das die von Ihnen benötigten Proxys enthält.

3. Konfigurieren Sie das Projekt zum Ausführen des IBM MQ-Transports für SOAP als JMS-Transport.

   IBM MQ Transport for SOAP verfügt über die Option `transportSender`, jedoch nicht über die Option `transportReceiver`. IBM MQ Transport for SOAP unterstützt also Axis2-Clients. Derzeit werden die Axis2-Services nicht unterstützt.

   a) Klicken Sie im Projekt **StockQuoteAxis2WebClient** mit der rechten Maustaste auf `WebContent\WEB-INF\conf\axis2.xml` und wählen Sie dann **Öffnen mit** > **XML-Editor**.

   b) Suchen Sie nach dem letzten Eintrag des Typs `transportSender` (am Ende der Datei) und nach dem auskommentierten Eintrag für JMS `transportSender`. Klicken Sie mit der rechten Maustaste auf die Zeile **Hinzufügen vor** und wählen Sie **transportSender** aus.

   c) Klicken Sie mit der rechten Maustaste auf **transportSender** und wählen Sie **Attribut hinzufügen** > **Name** aus. Klicken Sie erneut auf mit der rechten Maustaste auf **transportSender** und wählen Sie **Attribut hinzufügen** > **Klasse** aus.

d) Klicken Sie mit der rechten Maustaste auf **Name** und wählen Sie **Attribut bearbeiten** aus. Geben Sie im Feld **Wert** Folgendes ein: `jms`.

e) Klicken Sie mit der rechten Maustaste auf **Klasse** > **Attribut bearbeiten** > Geben Sie den **Wert ein:** `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender`. > Speichern.

f) Fügen Sie `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender` zum Buildpfad hinzu: Klicken Sie mit der rechten Maustaste auf **StockQuoteAxis2WebClient** > **Buildpfad** > **Buildpfad konfigurieren ...** > Klicken Sie auf die Registerkarte **Bibliotheken** > **Externe JAR-Dateien hinzufügen ...**. Wählen Sie alle JAR-Dateien im Verzeichnis *MQ_INSTALLATION_PATH* `\java\lib` aus und klicken Sie auf **OK**.

MQ_INSTALLATION_PATH ist das Verzeichnis, in dem IBM MQ installiert ist.

4. Erstellen Sie einen synchronen statischen Client, testen Sie ihn mittels HTTP und konvertieren Sie dann den Proxy, um den statischen Client mit IBM MQ Transport for SOAP auszuführen.

a) Klicken Sie mit der rechten Maustaste auf **Java-Ressourcen: src** und wählen Sie dann **Neu** > **Paket** aus. Nennen Sie das Paket `soap.client`. Klicken Sie abschließend auf 'Fertigstellen'.

b) Klicken Sie mit rechten Maustaste auf **soap.client** und wählen Sie dann **Neu** > **Klasse** aus. Nennen Sie die Klasse `SQA2StaticClient` und klicken Sie abschließend auf **Fertigstellen**.

c) Ersetzen Sie die Klasse durch den folgenden Code und klicken Sie anschließend auf **Speichern**.

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2StaticClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub();
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("Response is: "
          + (stub.getQuote(request)).getGetQuoteReturn());
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

5. Testen Sie den Client mit dem über Liberty bereitgestellten Service 'StockQuoteAxis' und mit IBM MQ Transport for SOAP.

a) Klicken Sie im Projektexplorer mit der rechten Maustaste auf **SQA2StaticClient** und wählen Sie dann **Ausführen als** > **Java-Anwendung** aus.

In der Konsolenansicht wird das Ergebnis `Response is 55.25` angezeigt. Sie können auch das Liberty-Konsolenfenster in der Konsolenansicht auswählen und die Ausgabe auf dem Liberty-Server anzeigen: `StockQuoteAxis called with parameter: ibm.`

b) Der Proxy wurde mit der Serviceadresse `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis` erstellt, der statische Client ruft also den Service auf, der in HTTP ausgeführt wird. Sie können den statischen Client so ändern, dass der Service mit IBM MQ Transport for SOAP aufgerufen wird. In den folgenden Anweisungen wird die Serviceadresse ohne erneute Erstellung des Proxy in `StockQuoteAxisServiceStub.java` geändert und die `SQA2StaticClient`-Laufzeitparameter werden für das Laden von `axis2.xml` konfiguriert. `axis2.xml` konfiguriert Axis2 für die Verwendung von IBM MQ Transport for SOAP.

c) Öffnen Sie die Datei `StockQuoteAxisServiceStub.java`.
Ersetzen Sie die beiden Vorkommen von `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis` durch

```
jms:/queue?destination=REQUESTAXIS@QM1
&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

d) Wenn Sie SQA2StaticClient jetzt ausführen, wird eine Ausnahme ausgegeben, da kein für JMS konfigurierter transportSender gefunden wurde.
Ausnahme:

```
Exception: null java.lang.NullPointerException at
soap.server.StockQuoteAxisServiceStub.getQuote(StockQuoteAxisServiceStub.java:547)
at soap.client.SQA2StaticClient.main(SQA2StaticClient.java:11)
```

e) Klicken Sie im Projektexplorer mit der rechten Maustaste auf **SQA2StaticClient** und wählen Sie dann **Ausführen als** > **Ausführungskonfigurationen** aus. Rufen Sie die Registerkarte **(x)= Argumente** auf und geben Sie im Eingabebereich **VM-Argumente** den Pfad der Datei axis2.conf ein. Klicken Sie dann auf **Anwenden** > **Ausführen**.
Das VM-Argument lautet wie folgt: -Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml. Sie können auch einen Standardpfad für die Axis2-Konfigurationsdatei bereitstellen.

f) Führen Sie SQA2StaticClient erneut aus. Bei dieser Ausführung verwenden Sie IBM MQ Transport for SOAP. Überprüfen Sie dies, indem Sie sich vergewissern, dass die Liberty-Konsole keine neue Ausgabe enthält. Öffnen Sie das zugehörige Konsolen- oder Befehlsfenster des SimpleJavaListener. Es zeigt die Ausgabe StockQuoteAxis called with parameter: ibm.

6. Erstellen Sie einen dynamischen Client für HTTP und IBM MQ Transport for SOAP und testen Sie ihn.

a) Klicken Sie mit rechten Maustaste auf **soap.client** und wählen Sie dann **Neu** > **Klasse** aus. Nennen Sie die Klasse SQA2DynamicClient und klicken Sie abschließend auf **Fertigstellen**.

b) Ersetzen Sie die Klasse durch den folgenden Code und klicken Sie anschließend auf **Speichern**.

Abbildung 178. SQA2DynamicClient.java

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2DynamicClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
       System.out.println("JMS sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
     } catch (Exception e) {
       System.out.println("Exception: " + e.getMessage());
       e.printStackTrace();
     }
   }
 }
```

c) Erstellen Sie eine Ausführungskonfiguration für SQA2DynamicClient.java und fügen Sie den Pfad zu axis2.xml hinzu:
-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml

d) Führen Sie SQA2DynamicClient aus. Überprüfen Sie die Konsolenausgabe auf die Einträge SQA2DynamicClient, Liberty und **SimpleJavaListener**.

7. Erstellen Sie einen asynchronen Client und rufen Sie das Ergebnis in einem Callback-Handler und im Hauptprogrammthread auf.

Die asynchronen Client-Proxys, die vom Web-Service für Eclipse Java EE IDE für Webentwickler erstellt wurden, unterscheiden sich von mit **wsimport** erstellten Proxys. Die **wsimport** -Proxys verwenden die generischen Typen Future, Responseund AsyncHandler .

Der Web-Service-Assistent für die Eclipse Java EE-IDE für Webentwickler erstellt eine abstrakte Klasse namens `StockQuoteAxisServiceCallbackHandler`. Sie müssen `StockQuoteAxisService-CallbackHandler` erweitern und einen Callback-Handler erstellen.

a) Klicken Sie mit rechten Maustaste auf **soap.client** und wählen Sie dann **Neu** > **Klasse** aus. Nennen Sie die Klasse `SQA2CallbackHandler` und klicken Sie abschließend auf **Fertigstellen**.

b) Ersetzen Sie die Klasse durch den folgenden Code:

```
package soap.client;
import soap.server.StockQuoteAxisServiceCallbackHandler;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
public class SQA2CallbackHandler
    extends StockQuoteAxisServiceCallbackHandler {
  private boolean complete = false;
  SQA2CallbackHandler() {
    super();
    System.out.println("Callback constructor");
  }
  public void receiveResultgetQuote(GetQuoteResponse response) {
    System.out.println("Result in Callback " + response.getGetQuoteReturn());
    super.clientData = response;
    complete = true;
  }
  public boolean isComplete() {
    return complete;
  }
}
```

c) Klicken Sie mit rechten Maustaste auf **soap.client** und wählen Sie dann **Neu** > **Klasse** aus. Nennen Sie die Klasse `SQA2AsyncClient` und klicken Sie abschließend auf **Fertigstellen**.

d) Ersetzen Sie die Klasse durch den folgenden Code:

Abbildung 179. `SQA2AsyncClient.java`

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
import soap.server.StockQuoteAxisServiceCallbackHandler;
@SuppressWarnings("unused")
public class SQA2AsyncClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      SQA2CallbackHandler callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      do {
        System.out.println("Waiting for HTTP callback");
        Thread.sleep(2000);
      } while (!callback.isComplete());
      System.out.println("HTTP poll: "
          + ((GetQuoteResponse) (callback.getClientData()))
                .getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
      System.out.println("JMS Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      while (!callback.isComplete()) {
        System.out.println("Waiting for JMS callback");
        Thread.sleep(2000);
      }
      System.out.println("JMS poll: "
        + ((GetQuoteResponse) (callback.getClientData())).getGetQuoteReturn());
```

```
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        e.printStackTrace();
      }
    }
  }
```

Die Konsolenausgabe lautet wie folgt:

```
HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
```

Axis2-Beispielclients

Die Beispielproxys werden mit dem **wsimport** -Tool generiert, das im Lieferumfang von Java 6 enthalten ist. Es werden sechs Beispiele bereitgestellt:

1. DynamicProxyClientSync.java
2. DynamicProxyClientAsyncPolling.java
3. DynamicProxyClientAsyncCallback.java
4. DispatchClientSync.java
5. DispatchClientAsyncPolling.java
6. DispatchClientAsyncCallback.java

Die Clientbeispiele werden für den Beispielserver 'StockQuoteAxis' generiert. Generieren Sie die Web Services Description Language (WSDL) mit dem Befehl **amqwdepoyWMQServer** und geben Sie dabei den Switch -w an, um den Stil `rpcLiteral` auszuwählen. Verwenden Sie den folgenden Befehl, um die Proxys für die Beispiele zu generieren:

```
wsimport soap.server.StockQuoteAxis_Wmq.wsdl -d generated -keep -p com.ibm.mq.axis2.samples
```

Abbildung 180. `DynamicProxyClientSync.java`

```
package com.ibm.mq.axis2.samples;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientSync");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      service.getQuoteOneWay("48");
      System.out.println(" > getQuoteOneWay has returned");

      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      float result = service.getQuote("48");
      System.out.println(" > getQuote has returned result of " + result);

      System.out.println("End of sample");
    }
```

```
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Abbildung 181. DynamicProxyClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.CancellationException;

import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncPolling");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously by pol
ling...");
      Response<Float> response = service.getQuoteAsync("49");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** Retrieve the result **/
      try {
        Float result = response.get();
        System.out.println(" > getQuoteAsync call has returned result of " + result);
      }
      catch (CancellationException ce) {
        // processing was cancelled via response.cancel()
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
```

```
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Abbildung 182. DynamicProxyClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncCallback implements AsyncHandler<Float> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncCallback");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      DynamicProxyClientAsyncCallback handler = new DynamicProxyClientAsyncCallback();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously using a call
back...");
      Future<?> monitor = service.getQuoteAsync("50", handler);
      System.out.println(" > Invoke call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<Float> response) {
    try {
      Float result = response.get();
      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
```

```
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

Abbildung 183. DispatchClientSync.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientSync");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          + "&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteA
xis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq", "soap.server.StockQuoteA
xis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service **/
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /************************************
       * Create OneWay SOAPMessage request.
       ************************************/
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a OneWay SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
```

```
      SOAPElement operation = body.addChildElement("getQuoteOneWay", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      dispatch.invokeOneWay(request);
      System.out.println(" > getQuoteOneWay call has returned");

      /****************************************
       * Create Request Reply SOAPMessage request.
       ****************************************/
      mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a Request Reply SOAP Message");
      request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      part = request.getSOAPPart();
      env = part.getEnvelope();
      header = env.getHeader();
      body = env.getBody();

      /** Construct the message payload **/
      operation = body.addChildElement("getQuote", "ns1", "soap.server.StockQuoteAxis_Wmq");
      value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      SOAPMessage ans = dispatch.invoke(request);
      System.out.println(" > getQuote call has returned");

      /** Retrieve the result **/
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      System.out.println("Parsing SOAP response...");
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement) bodyElement.getChildElements(resultNa
me).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Abbildung 184. DispatchClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncPolling");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          + "&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteA
xis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq", "soap.server.StockQuoteA
xis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation asynchronously by pol
ling...");
      Response<SOAPMessage> response = dispatch.invokeAsync(request);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");
```

```
      /** retrieve the result **/
      SOAPMessage ans = response.get();
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement) bodyElement.getChildElements(resultNa
me).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");

    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Abbildung 185. DispatchClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncCallback implements AsyncHandler<SOAPMessage> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncCallback");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          + "&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteA
xis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq", "soap.server.StockQuoteA
```

```
xis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload. **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint. **/
      DispatchClientAsyncCallback handler = new DispatchClientAsyncCallback();

      System.out
          .println("Invoking getQuote Request Reply operation asynchronously using a call□
back...");
      Future<?> monitor = dispatch.invokeAsync(request, handler);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<SOAPMessage> response) {
    try {
      // retrieve the result
      SOAPMessage ans = response.get();
      SOAPPart part = ans.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPBody body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");
```

```
      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement) bodyElement.getChildElements(resultNa
me).next();
      String result = responseElement.getValue();

      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Zugehörige Tasks

JAX-RPC-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln
Entwicklung eines Axis 1.4-Web-Service-Clients mit dem IBM MQ-Transport für SOAP.

.NET 1- oder 2-Client für WebSphere Transport for SOAP mit Microsoft Visual Studio 2012 Visual Studio 2008 entwickeln
Entwicklung eines .NET 1- oder 2-Web-Service-Client für den IBM MQ-Transport für SOAP.

.NET 1- oder 2-Client für WebSphere Transport for SOAP mit Microsoft Visual Studio 2012 Visual Studio 2008 entwickeln

Entwicklung eines .NET 1- oder 2-Web-Service-Client für den IBM MQ-Transport für SOAP.

Vorbereitende Schritte

Wichtig: Zur Ausführung dieser Task ist .NET Framework 2.0 erforderlich. Stellen Sie dessen Installation sicher, bevor Sie mit dieser Task beginnen.

Sie können einen .NET 1- oder 2-Client auf verschiedene Arten entwickeln:

1. Generieren Sie Client-Stubs mit **amqwdeployWMQService** aus einem Web-Service und importieren Sie sie in Microsoft Visual Studio.
2. Verwenden Sie **java2wsdl** , um WSDL aus einer Java -Implementierung eines Web-Service zu generieren, und verwenden Sie dann `wsdl.exe`, das im Lieferumfang von .NETenthalten ist, um Client-Stubs zu generieren.
3. Generieren Sie die WSDL aus einer .NET `.asmx`-Implementierung des Service mittels **amqswsdl** und verwenden Sie anschließend die Datei `wsdl.exe`.
4. Wenn Sie den Service für HTTP entwickelt und implementiert haben, verwenden Sie **Webreferenz hinzufügen ...** . Assistent in Microsoft Visual Studio zum Konfigurieren des Clients für den Zugriff auf den HTTP-Service. Ändern Sie die URL so, dass sie auf den für den IBM MQ-Transport für SOAP implementierten Service verweist.

Die Task verwendet den in „.NET 1- oder 2-Service für IBM MQ Transport for SOAP mithilfe von Microsoft Visual Studio 2012 entwickeln” auf Seite 1384 entwickelten Service.

Informationen zu diesem Vorgang

Führen Sie die nachfolgenden Schritte aus, um einen .NET 1- oder 2-Client für HTTP und den IBM MQ-Transport für SOAP zu erstellen.

Vorgehensweise

1. Erstellen Sie die Anwendung für die Clientkonsole und passen Sie sie so an, dass sie den HTTP-Web-Service 'StockQuote' aufruft.
   a) Klicken Sie mit der rechten Maustaste auf **Lösung 'StockQuoteDotNet'** in **Solution Explorer** > Hinzufügen ... > Neues Projekt. Wählen Sie den Projekttyp **C#**, **.NET Framework 2.0** und **Konsolenanwendung** aus. Nennen Sie das Projekt `StockQuoteClientDotNet` und klicken Sie auf > **OK**.
   b) Klicken Sie mit der rechten Maustaste auf **Lösung 'StockQuoteDotNet'** in **Solution Explorer** > Hinzufügen ... > Neues Projekt. Wählen Sie den Projekttyp **C#**, **.NET Framework 2.0** und **Konsolenanwendung** aus. Nennen Sie das Projekt `StockQuoteClientDotNet` und klicken Sie auf > **OK**.
   c) Klicken Sie mit der rechten Maustaste auf **StockQuoteClientDotNet** und klicken Sie dann auf > **Als Startprojekt festlegen**.
   d) Klicken Sie mit der rechten Maustaste auf **StockQuoteClientDotNet** > **Webreferenz hinzufügen ...** > Durchsuchen Sie die Web-Services in dieser Lösung und wählen Sie **StockQuote** > **Referenz hinzufügen**aus. Sie haben eine Webreferenz zum lokalen Host und die neue Konfigurationsdatei `app.config` hinzugefügt.
   e) Ändern Sie in Solution Explorer den Namen der Konsolenanwendung von `Program.cs` in `StockQuoteClientDotNet.cs`. > Klicken Sie auf **OK**, um alle Verwendungen von `Program.cs` in `StockQuoteClientDotNet.cs` zu ändern.
   f) Ersetzen Sie den Inhalt von `StockQuoteClientDotNet.cs` mit dem Code in Abbildung 186 auf Seite 1416.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Abbildung 186. HTTP-Programm 'StockQuoteClientDotNet'

   g) Starten Sie 'StockQuoteClientDotNet' um Testen mit dem Service `StockQuote.asmx`:
      i) Drücken Sie die Taste **F5**, klicken Sie in der Aktionsleiste auf den grünen Pfeil oder auf **Fehler beheben** > **Fehlerbehebung starten (F5)**.

      Befindet sich das Projekt 'StockQuoteDOtNet' in derselben Lösung, wird es automatisch gestartet. Andernfalls müssen Sie zuerst den Service starten.

      Das Befehlsfenster mit den Ergebnissen wird hinter dem Arbeitsbereich geöffnet. Die Anweisung `Console.ReadLine();` verhindert, dass es geschlossen wird, bevor Sie die **Eingabetaste** drücken.

   Tipp: Stellen Sie sicher, dass `StockQuote.asmx` die Startseite im Projekt `StockQuoteDotNet` ist.

2. Ändern Sie 'StockQuoteClientDotNet', dass der Service `StockQuote.asmx` mit dem IBM MQ-Transport für SOAP aufgerufen wird.
   a) Fügen Sie dem Client die fettgedruckten Zeilen hinzu.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        IBM.WMQSOAP.Register.Extension();
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
        stockobj.Url = "jms:/queue?"
          + "initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&connectionFactory=()&destination=REQUESTDOTNET@QM1"
          + "&targetService=StockQuote.asmx";
        Console.WriteLine("jms reply is: "
          + stockobj.getNonInlineQuote("jms request"));
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Abbildung 187. Geändertes Programm 'StockQuoteClientDotNet'

   Alternativ können Sie die Standard-URL ändern. Öffnen Sie **StockQuoteClientDotNet** > **Eigenschaften** > **Settings.settings** und ändern Sie den Wert der Eigenschaft `StockQuoteClientDotNet_localhost_StockQuote` in die URL des IBM MQ-Transports für SOAP.

   b) Fügen Sie `amqsoap.dll` eine Referenz hinzu.

      i) Klicken Sie im Projekt **StockQuoteClientDotNet** im **Solution-Explorer**mit der rechten Maustaste auf **Referenzen** > **Referenz hinzufügen** > Klicken Sie auf die Registerkarte **Durchsuchen** und navigieren Sie zu *MQ_INSTALLATION_PATH*\bin > Wählen Sie **amqsoap.dll** > **OK**aus. *MQ_INSTALLATION_PATH* ist das Verzeichnis, in dem IBM MQ installiert ist.

3. Testen Sie den Client mit dem Service `StockQuote.asmx` mit dem IBM MQ-Transport für SOAP.
   a) Öffnen Sie ein Befehlsfenster im Projektverzeichnis 'StockQuoteDotNet': `.\StockQuoteDotNet\StockQuoteDotNet.` > Vergewissern Sie sich, dass die Datei `.bin\StockQuoteDotNet.dll` vorhanden ist. Ist dies nicht der Fall, erstellen Sie die Lösung erneut.
   b) Geben Sie den Befehl **amqwRegisterdotNet** ein.

      Sie brauchen **amqwRegisterdotNet** nur einmal pro Installation auszuführen.

   c) Wenn Sie **amqwdeployWMQServer** mit der Option `genAsmxWMQBits` ausgeführt haben, führen Sie den .NET-SOAP-Listener aus:
   `generated\server\startWMQNListener`
   d) Führen Sie alternativ den Listener direkt aus:

```
amqwSOAPNETListener -u "jms:/queue?
destination=REQUESTDOTNET@QM1
&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuote.asmx&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
-w C:\IBM\ID\StockQuoteDotNet\StockQuoteDotNet -n 10
```

4. Drücken Sie in Visual Studio 2012 die Taste **F5**, um 'StockQuoteClientDotNet' auszuführen.

Web-Service-Clients für .NET Framework 1 und .NET Framework 2

Die .NET -Beispielclients, die mit IBM MQ Transport for SOAP bereitgestellt werden, verwenden generierte Stubs, um die Axis-und .NET -Beispielservices aufzurufen.

Für .NET Framework 1- und .NET Framework 2-Clients stellt IBM MQ den Zugriff auf Web-Services bereit, die .NET-Clients verwenden. Der Befehl **amqwdeployWMQService** verfügt über die Option `genProxies-toDotNet`, die .NET Framework 1-oder .NET Framework 2-Client-Stubs für einen Web-Service generiert. Sie können aber auch von .NET **wsdl** oder von Microsoft Visual Studio 2012 generierte Client-Stubs verwenden.

Die .NET Framework 1-und .NET -Web-Service-Beispielclients sind in *MQ_INSTALLATION_PATH* `\tools\soap\samples\dotnet`installiert. *MQ_INSTALLATION_PATH* ist das Verzeichnis, in dem IBM MQ installiert ist.

SQVB2Axis.vb
SQVB2Axis.vb,Abbildung 188 auf Seite 1418, ist der Visual Basic-Client zum Aufrufen des Service **StockQuoteAxisService**.

SQVB2DotNet.vb
QVB2DotNet.vb,Abbildung 189 auf Seite 1418, ist der Visual Basic-Client zum Aufrufen des Service **StockQuoteDotNet**.

SQCS2Axis.cs
SQCS2Axis.cs,Abbildung 190 auf Seite 1419, ist der C#-Client zum Aufrufen des Service **StockQuoteAxisService**. Sie können die URL des Service überschreiben, indem Sie in der Befehlszeile eine URL eingeben.

SQCS2DotNet.cs
SQCS2DotNet.cs,Abbildung 191 auf Seite 1419, ist der C#-Client zum Aufrufen des Service **StockQuoteDotNet**. Sie können die URL des Service überschreiben, indem Sie in der Befehlszeile eine URL eingeben.

```
Module SQVB2Axis
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj As New StockQuoteAxisService()
      Dim res As Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2Axis: reply is: '{0}'", res)
   End Function
End Module
```

Abbildung 188. SQVB2Axis

```
Module SQVB2DotNet
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj as new StockQuoteDotNet()
      Dim res as Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2DotNet: reply is: '{0}'", res)
   End Function
End Module
```

Abbildung 189. SQVB2DotNet

```
using System;
class SQCS2Axis {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteAxisService stockobj = new StockQuoteAxisService();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("SQCS2Axis RPC reply is: " + res);
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2Axis DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Abbildung 190. SQCS2Axis

```
using System;
class SQCS2DotNet {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteDotNet stockobj = new StockQuoteDotNet();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("RPC reply is: " + res);
         if (args.GetLength(0) == 0) {
            res = stockobj.getQuoteDOC("XXX");
            Console.WriteLine("DOC reply is: " + res);
         }
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2DotNet DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Abbildung 191. SQCS2DotNet

Zugehörige Tasks

JAX-RPC-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln
Entwicklung eines Axis 1.4-Web-Service-Clients mit dem IBM MQ-Transport für SOAP.

JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln
Entwicklung eines Axis2-Web-Service-Clients zur Verwendung des IBM MQ-Transports für SOAP. Die mit IBM MQ Transport for SOAP bereitgestellten Axis2-Beispielclients und der Befehl **wsimport** zum Generieren von Proxys werden aufgeführt.

Web-Services mit Verbindung über IBM MQ Transport for SOAP implementieren

Implementieren Sie in einer der unterstützten Serverumgebungen einen Web-Service und stellen Sie über IBM MQ Transport for SOAP eine Verbindung zu diesem her.

Vorbereitende Schritte

Entwickeln Sie einen Web-Service und testen Sie ihn mit SOAP over HTTP in der Zielumgebung.

Informationen zu diesem Vorgang

Ein Web-Service, der mit IBM MQ-Transport für SOAP ausgeführt werden soll, kann in verschiedenen SOAP-Laufzeitumgebungen bereitgestellt werden. Unter Axis 1.4 kann ein Service mit der mit IBM MQ installierten Software bereitgestellt werden. Für andere Laufzeitumgebungen müssen Sie zusätzliche Software installieren.

Die Ausführung von IBM MQ-Transport für SOAP ist nicht auf die Server beschränkt, für die es Bereitstellungsanweisungen gibt. Die Anweisungen zeigen Ihnen jedoch, wie Sie einen Service in den aufgeführten Umgebungen bereitstellen.

Anmerkung: Einige der integrierten Umgebungen verwenden SOAP over JMS mit der vom W3C empfohlenen JMS-SOAP-Bindung wie auch die IBM MQ-Transport für SOAP-Bindung. Releases von IBM MQ bis einschließlich 7.0.1.2 unterstützen nur die IBM MQ-Transport für SOAP-Bindung. Ab Version 7.0.1.3 können Sie Axis2-Clients mit einer URI bereitstellen, die der Empfehlung des W3C für SOAP over JMS entspricht. Weitere Informationen finden Sie im Lernprogramm Entwickeln Sie eine SOAP/JMS JAX-WS-Web-Service-Anwendung mit WebSphere Application Server V7 und Rational Application Developer 7.5.

Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren

Implementieren Sie einen Axis 1.4-Service für IBM MQ-Transport für SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

Vorbereitende Schritte

1. Befolgen Sie die Installationsanweisungen für den IBM MQ-Transport für SOAP.
2. Prüfen Sie Ihre Installation und Umgebung mit dem Befehl **runivt**.
3. So implementieren Sie einen Service erneut:
   a. Löschen Sie das Unterverzeichnis `./generated` und seine sämtlichen Unterverzeichnisse.
   b. Entfernen Sie die Anforderungen aus der Zielwarteschlange und löschen Sie die Warteschlange.
   c. Fahren Sie mit den Anweisungen aus Schritt „2” auf Seite 1420 fort.

Informationen zu diesem Vorgang

Diese Anweisungen beziehen sich auf die erste Implementierung eines Axis 1.4-Service. Um einen Axis 1.4-Service erneut zu starten, führen Sie den Axis 1.4-SOAP-Listener erneut aus: Schritt „11” auf Seite 1421.

Führen Sie folgenden Anweisungen aus, um einen neuen Axis 1.4-Service für den IBM MQ-Transport für SOAP zu implementieren:

Vorgehensweise

1. Erstellen Sie das Verzeichnis *deployDir* für die Implementierungsdateien.
   Das Dienstprogramm für die Bereitstellung verlangt, dass jeder Service aus einem eigenen Verzeichnis implementiert wird.
2. Öffnen Sie ein Befehlsfenster unter Windowsoder eine Befehlsshell mit X Window System auf UNIX and Linux -Systemen in *deployDir* , um **amqwdeployWMQService**auszuführen.
3. Führen Sie **amqwsetcp** zur Festlegung des Klassenpfads aus.
   Die JRE und das JDK müssen in Version 5.0 oder höher im Klassenpfad vorhanden sein und den gleichen Versionsstand aufweisen.
4. Kopieren Sie die Klassenquelle *className*`.java`in das Verzeichnis *deployDir*
5. Kopieren Sie alle Java -Quellendateien in demselben Paket wie *className* in *deployDir*/*packageName*, wobei *packageName* eine Verzeichnisstruktur ist, die dem Paketnamen entspricht.
6. **javac** *packageName*`.` *className*`.` ausführen

Möglicherweise müssen Sie einen Pfad zum aktuellen Verzeichnis " . "oder zum Verzeichnis *packageName* für **javac** hinzufügen, um die anderen Klassen zu finden.

7. Erstellen Sie die Axis-WSDL für den Service:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWsdl
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

8. Erstellen Sie die IBM MQ-Ressourcen für den Service:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

Tipp:

Wenn Sie einen neuen Warteschlangenmanager und die erforderlichen Ressourcen einrichten möchten, führen Sie zum Entwickeln und Testen den Befehl **setupWMQSOAP** aus.

Wenn Sie den neuen Warteschlangenmanager als Standardwarteschlangenmanager einrichten möchten, nehmen Sie eine Kopie von **setupWMQSOAP** aus dem Verzeichnis *WMQ installation directory*\tools\soap\samples und fügen Sie der Zeile den Parameter -q hinzu.

```
call :try -q crtmqm %QMGR%
```

9. Erstellen Sie den Axis-Listener und implementieren Sie den Service:

```
amqwdeployWMQService -f packageName.className.java -c AxisDeploy
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

10. Wenn Sie die Web Services Description Language (WSDL) für den Service generieren müssen, erstellen Sie Client-Stubs oder Client-Proxys. Führen Sie dazu **amqwdeployWMQService** mit einem der folgenden Parameter aus:

    - genAsmxWsdl
    - genAxisWsdl
    - genProxiesToDotNet
    - genProxiestoAxis

    Anmerkung: Die WSDL muss vor dem Generieren der Proxys generiert werden. Die Option AllAxis schlägt fehl, wenn CLASSPATH nicht so konfiguriert ist, dass alle zur Kompilierung importierten Klassen gefunden werden *className*.java. Wenn das Paket mit *className*.javamehrere Java -Dateien enthält, müssen Sie diese zuerst mit **javac**kompilieren. **amqwdeployWMQService** -f *packageName*. *className*.java -c CompileJava kompiliert nur *className*.java.

11. Starten Sie den generierten Axis-Listener.

```
.\generated\server\startWMQJListener.cmd
```

Zugehörige Tasks

.NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Bereitstellungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für CICS Transaction Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen

IBM MQ Transport for SOAP ist in die Web-Service-Unterstützung von CICS Transaction Server 4.1 integriert.

Service für WebSphere Application Server zur Verwendung von WebSphere Transport for SOAP implementieren
IBM MQ Transport for SOAP ist in WebSphere Application Server in den Service Integration Bus integriert.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen
IBM MQ-Transport für SOAP wird von WebSphere Enterprise Service Bus (ESB) und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

.NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren

Sie können einen .NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Bereitstellungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Vorbereitende Schritte

1. Befolgen Sie die Installationsanweisungen für den IBM MQ-Transport für SOAP.
2. Prüfen Sie Ihre Installation und Umgebung mit dem Befehl **runivt**.
3. Der Pfad zu den .NET Framework-Dateien `wsdl.exe` und `csc.exe` muss festgelegt sein. Die durch die Variable PATH angegebenen Kopien der Dateien `wsdl.exe` und `csc.exe` müssen in der gleichen .NET Framework-Version vorliegen. Falls Sie mehrere Versionen von .NET Framework installiert haben oder Visual Studio verwenden, sollten Sie die Variable PATH sorgfältig prüfen.
4. So implementieren Sie einen Service erneut:
   a. Löschen Sie das Unterverzeichnis `./generated` und seine Unterverzeichnisse.
   b. Entfernen Sie die Anforderungen aus der Zielwarteschlange und löschen Sie die Warteschlange.
   c. Fahren Sie mit den Anweisungen aus Schritt „2” auf Seite 1422 fort.

Informationen zu diesem Vorgang

Diese Anweisungen zeigen, wie ein .NET-Service zum ersten Mal implementiert wird. Zum Neustarten eines .NET-Service führen Sie das .NET SOAP-Empfangsprogramm erneut aus (Schritt „9” auf Seite 1423).

Führen Sie zur Implementierung eines neuen .NET Framework 1- oder .NET Framework 2-Service für IBM MQ Transport for SOAP das folgende Verfahren aus:

Vorgehensweise

1. Erstellen Sie das Verzeichnis *deployDir* für die Implementierungsdateien.
   Das Dienstprogramm für die Bereitstellung verlangt, dass jeder Service aus einem eigenen Verzeichnis implementiert wird.
2. Öffnen Sie im Verzeichnis *deployDir* ein Befehlsfenster für die Ausführung von **amqwdeployWMQService**.

```
C:\IBM\ID\QuoteClient>
```

3. Führen Sie **amqwsetcp** zur Festlegung des Klassenpfads aus.
   Der Klassenpfad wird nur für Axis-Clients benötigt.
4. Kopieren Sie den .NET -Service *className*`.asmx`in das Verzeichnis *deployDir* .
5. Erstellen Sie die Serviceimplementierung in einer Bibliothek (`.dll`).
   Die Inline-Serviceimplementierung befindet sich in der Datei *className*`.asmx`. Die CodeBehind-Serviceimplementierung kann sich zum Beispiel in der Datei *className*`.asmx.cs` befinden.

Abbildung 192 auf Seite 1423 zeigt ein Befehlsbeispiel für die Erstellung eines .NET Framework V2-Service als Bibliothek.

```
c:\WINDOWS\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.configuration.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.Services.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll
/debug:pdbonly /filealign:512 /optimize+
/out:obj\Quote.dll /target:library Properties\AssemblyInfo.cs Quote.asmx.cs
```

Abbildung 192. Build-Befehl für .NET Framework V2-Service

6. Kopieren Sie die Datei *className*.dll in das Verzeichnis *deployDir**bin*.
7. Richten Sie die IBM MQ-Ressourcen ein und erstellen Sie das Empfangsprogramm für den Service:

```
amqwdeployWMQService -f className.asmx -c genAsmxWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))
&targetService=className.asmx"
```

8. Wenn Sie die Web Services Description Language (WSDL) für den Service generieren müssen, erstellen Sie Client-Stubs oder Client-Proxys. Führen Sie dazu **amqwdeployWMQService** mit einem der folgenden Parameter aus:
   - genAsmxWsdl
   - genAxisWsdl
   - genProxiesToDotNet
   - genProxiestoAxis

   Anmerkung: Die WSDL muss vor dem Generieren der Proxys generiert werden.
9. Starten Sie das generierte .NET-Empfangsprogramm.

```
.\generated\server\startWMQNListener.cmd
```

Zugehörige Tasks

Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren
Implementieren Sie einen Axis 1.4-Service für IBM MQ-Transport für SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

Service für CICS Transaction Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen
IBM MQ Transport for SOAP ist in die Web-Service-Unterstützung von CICS Transaction Server 4.1 integriert.

Service für WebSphere Application Server zur Verwendung von WebSphere Transport for SOAP implementieren
IBM MQ Transport for SOAP ist in WebSphere Application Server in den Service Integration Bus integriert.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen
IBM MQ-Transport für SOAP wird von WebSphere Enterprise Service Bus (ESB) und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

Service für CICS Transaction Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen

IBM MQ Transport for SOAP ist in die Web-Service-Unterstützung von CICS Transaction Server 4.1 integriert.

Vorbereitende Schritte

Verwenden Sie zur Entwicklung eines Clients oder eines Service für IBM MQ die gleichen Tools wie für HTTP. CICS verwendet die folgenden Tools (diese entsprechen **Java2wsdl** und **wsdl2Java**):

- **DFHWS2LS** verwendet eine Web-Service-Beschreibung als Ausgangspunkt. Es verwendet die Nachrichtenbeschreibungen und die in diesen Nachrichten verwendeten Datentypen zur Erstellung von Datenstrukturen in einer höheren Programmiersprache. Diese Strukturen können Sie in Anwendungsprogrammen verschiedener Programmiersprachen verwenden.
- **DFHLS2WS** verwendet eine in einer höheren Programmiersprache vorliegende Datenstruktur als Ausgangspunkt. Aus dieser Struktur erstellt es eine Web-Service-Beschreibung mit Nachrichtenbeschreibungen sowie Nachrichtenschemas.

Folgen Sie den Anweisungen im Abschnitt Web-Service erstellen in der CICS-Produktdokumentation, um einen Web-Service zu erstellen.

Informationen zu diesem Vorgang

Folgen Sie den Anweisungen im Abschnitt CICS für Verwendung des IBM MQ-Transports konfigurieren in der CICS-Produktdokumentation. Anhand dieser Anweisungen können Sie den Web-Service für IBM MQ Transport for SOAP implementieren.

Zugehörige Tasks

Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren
Implementieren Sie einen Axis 1.4-Service für IBM MQ-Transport für SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

.NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Bereitstellungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für WebSphere Application Server zur Verwendung von WebSphere Transport for SOAP implementieren
IBM MQ Transport for SOAP ist in WebSphere Application Server in den Service Integration Bus integriert.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen
IBM MQ-Transport für SOAP wird von WebSphere Enterprise Service Bus (ESB) und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

Service für WebSphere Application Server zur Verwendung von WebSphere Transport for SOAP implementieren

IBM MQ Transport for SOAP ist in WebSphere Application Server in den Service Integration Bus integriert.

Vorbereitende Schritte

Entwickeln Sie Ihren Web-Service mit Rational Application Developer, WebSphere Integration Developer oder einem anderen Web-Services-Toolkit.

Informationen zu diesem Vorgang

Verwenden Sie die folgenden Anweisungen, um einen Service mit IBM MQ Transport for SOAP als SOAP-Transport unter WebSphere Application Serverzu implementieren.

Vorgehensweise

1. Konfigurieren Sie IBM MQ als JMS-Messaging-Provider für den Service Integration Bus unter WebSphere Application Server.
2. Konfigurieren Sie die vom Service benötigten IBM MQ-Ressourcen.
3. Folgen Sie den Anweisungen im Abschnitt JMS-Ressourcen für den synchronen SOAP over JMS-Endpunktlistener konfigurieren der WebSphere Application Server Network Deployment-Produktdokumentation.

   Für die anderen WebSphere Application Server-Plattformen stehen vergleichbare Anweisungen bereit.
4. Passen Sie die Service-URI an die IBM MQ Transport for SOAP-URI an.
5. Implementieren Sie den Service unter WebSphere Application Server.

Nächste Schritte

Implementieren Sie den Service mit HTTP als Transportmethode, so dass Clients den Service abfragen können, um die entsprechende WSDL zurückzuerhalten.

Zugehörige Tasks

Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren

Implementieren Sie einen Axis 1.4-Service für IBM MQ-Transport für SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

.NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren

Sie können einen .NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Bereitstellungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für CICS Transaction Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen

IBM MQ Transport for SOAP ist in die Web-Service-Unterstützung von CICS Transaction Server 4.1 integriert.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen

IBM MQ-Transport für SOAP wird von WebSphere Enterprise Service Bus (ESB) und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen

IBM MQ-Transport für SOAP wird von WebSphere Enterprise Service Bus (ESB) und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

Informationen zu diesem Vorgang

WebSphere Integration Developer stellt eine SOAP-Datentransformation bereit, die Sie an den IBM MQ JMS-Export binden können, um einen benutzerdefinierten IBM MQ JMS SOAP-Export zu erstellen.

Folgen Sie zum Erstellen eines benutzerdefinierten Exports für den Empfang von SOAP-Anforderungen über IBM MQ-Transport für SOAP den folgenden Anweisungen.

Vorgehensweise

1. Lesen Sie die Abschnitte "Overview of imports and exports" und "How to connect to IBM MQ" in der Produktdokumentation zu WebSphere Process Server for Multiplatforms.

Siehe Overview of imports and exports und How to connect to IBM MQ.

2. Folgen Sie der Task "Generating an MQ JMS export binding" in der Produktdokumentation zu WebSphere Integration Developer.

   Siehe Generating an MQ JMS export binding. Verwenden Sie zur Formatierung der SOAP-Nachricht die im Abschnitt Prepackaged JMS data format transformations beschriebene vordefinierte SOAP-Datenbindung.

Zugehörige Tasks

Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren
Implementieren Sie einen Axis 1.4-Service für IBM MQ-Transport für SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

.NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für IBM MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Bereitstellungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für CICS Transaction Server zur Verwendung von WebSphere-Transport für SOAP bereitstellen
IBM MQ Transport for SOAP ist in die Web-Service-Unterstützung von CICS Transaction Server 4.1 integriert.

Service für WebSphere Application Server zur Verwendung von WebSphere Transport for SOAP implementieren
IBM MQ Transport for SOAP ist in WebSphere Application Server in den Service Integration Bus integriert.

Web-Service-Clients für die Verwendung von IBM MQ Transport for SOAP implementieren

Implementieren Sie in einer der unterstützten Clientumgebungen einen Web-Service-Client und stellen Sie über IBM MQ Transport for SOAP eine Verbindung zu einem Serviceher.

Vorbereitende Schritte

Entwickeln Sie den Web-Service und implementieren Sie ihn so, dass er IBM MQ Transport for SOAP verwendet.

Informationen zu diesem Vorgang

Ein Web-Service-Client, der mit IBM MQ-Transport für SOAP ausgeführt werden soll, kann in verschiedenen Clientumgebungen bereitgestellt werden. Unter Axis 1.4 kann ein Java-Client mit der mit IBM MQ installierten Software bereitgestellt werden. Für andere Clientumgebungen müssen Sie zusätzliche Software installieren.

Die Ausführung von WebSphere-Transport für SOAP ist nicht auf die Clientumgebungen beschränkt, für die es Bereitstellungsanweisungen gibt. Die Anweisungen zeigen Ihnen jedoch, wie Sie einen Client in den unterstützten Umgebungen bereitstellen.

Anmerkung: Einige der integrierten Umgebungen verwenden SOAP over JMS mit der vom W3C empfohlenen JMS-SOAP-Bindung wie auch die IBM MQ-Transport für SOAP-Bindung. Releases von IBM MQ bis einschließlich 7.0.1.2 unterstützen nur die IBM MQ-Transport für SOAP-Bindung. Ab Version 7.0.1.3 können Sie Axis2-Clients mit einer URI bereitstellen, die der Empfehlung des W3C für SOAP over JMS entspricht. Weitere Informationen finden Sie im Lernprogramm Entwickeln Sie eine SOAP/JMS JAX-WS-Web-Service-Anwendung mit WebSphere Application Server V7 und Rational Application Developer 7.5.

Web-Service-Client für Axis 1.4 zur Verwendung von IBM MQ Transport for SOAP implementieren

Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie die Client-Proxys und die Clientklasse bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Vorbereitende Schritte

Tipp: Implementieren Sie den Service für HTTP, entwickeln und testen Sie den Client für HTTP und passen Sie den Client für IBM MQ Transport for SOAP dann wie folgt an:

1. Fügen Sie dem Client den Aufruf Register.extension() hinzu.
2. Geben Sie als statische Web-Service-Adresse in der Locator-Klasse des Client-Proxys die URI für IBM MQ Transport for SOAP an.

Informationen zu diesem Vorgang

Die Bereitstellung eines Axis 1.4-Client für die Verwendung des IBM MQ-Transports für SOAP erfordert gegenüber der Bereitstellung eines HTTP-Clients einen zusätzlichen Schritt. Sie müssen den Clientbereitstellungsdeskriptor `client-config.wsdd` erstellen, in dem Sie den Transport `jms:` der Senderklasse `com.ibm.mq.soap.transport.jms.WMQSender` zuordnen.

Wenn Sie zur Erstellung von Client-Proxys den Befehl **amqwdeployWMQService** verwenden, können Sie den Client in den vom Befehl generierten Verzeichnissen implementieren.

Vorgehensweise

1. Erstellen Sie das Verzeichnis *deployDir* für die Clientbereitstellungsdateien.
2. Öffnen Sie in *deployDir* ein Befehlsfenster (unter Windows) bzw. eine Befehlsshell mit X Window System (unter UNIX and Linux).
3. Führen Sie den Befehl **amqwsetcp.cmd** aus, um den CLASSPATH festzulegen.
4. Führen Sie den Befehl **amqwclientconfig.cmd** aus, um den Clientbereitstellungdeskriptor `client-config.wsdd` für Axis 1.4 in *deployDir* zu erstellen.
5. Stellen Sie sicher, dass sich die Klassen des Clientpakets, die Client-Proxy-Klassen und die vom Client verwendeten Bibliotheken im CLASSPATH befinden.

   amqwdeployWMQService platziert die .NET -Client-Proxys in `./generated/server/soap/client/remote/dotnetService` und die Axis 1.4 Proxys in `./generated/server/soap/client/remote/`*`client package`*.

Beispiel

Das Beispiel zeigt die Konfiguration und die Ausgabe (Abbildung 195 auf Seite 1428) eines Axis 1.4 Java-Clients. Der Client (Abbildung 194 auf Seite 1428) ruft einen Web-Service auf, der die Eingabeparameter zurückmeldet. Die Servicedefinition (Abbildung 193 auf Seite 1428) zeigt die aus der Service-WSDL entnommene URI.

```
<wsdl:service name="QuoteSOAPImplService">
    wsdl:port binding="intf:org.example.www.QuoteSOAPImplBindingSoap"
                      name="org.example.www.QuoteSOAPImpl_Wmq">
       <wsdlsoap:address location="jms:/queue?destination=REQUESTAXIS
           &connectionFactory=(connectQueueManager(QM1)binding(server))
           &initialContextFactory=com.ibm.mq.jms.Nojndi
           &targetService=org.example.www.QuoteSOAPImpl.java
           &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" />
    </wsdl:port>
</wsdl:service>
```

Abbildung 193. Servicedefinition

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                  + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

Abbildung 194. Axis 1.4 Java-Client

```
C:\IBM\ID\Test>dir /s /b
C:\IBM\ID\Test\client-config.wsdd
C:\IBM\ID\Test\org
C:\IBM\ID\Test\org\example
C:\IBM\ID\Test\org\example\www
C:\IBM\ID\Test\org\example\www\GetQuoteFaultMsg.class
C:\IBM\ID\Test\org\example\www\OrgExampleWwwQuoteSOAPImplBindingSoapStub.class
C:\IBM\ID\Test\org\example\www\QuoteClient.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImpl.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplService.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplServiceLocator.class

C:\IBM\ID\Test>amqwsetcp
C:\IBM\ID\Test>java org.example.www.QuoteClient.class
Response = IBM
```

Abbildung 195. Clientkonfiguration und Ausgabe

Nächste Schritte

1. Wenn Sie den Client als IBM MQ-Client implementieren, müssen Sie den Client- und Serververbindungskanal konfigurieren.
2. Wenn Sie den Client auf einem anderen Warteschlangenmanager als dem des Service implementieren, müssen Sie für den Client die Zielwarteschlange bereitstellen. Konfigurieren Sie die Zielwarteschlange auf dem Warteschlangenmanager des Service als eine Clusterwarteschlange bzw. auf dem Warteschlangenmanager des Clients als eine ferne Warteschlange.

Zugehörige Tasks

Web-Service-Client in Axis2 zur Verwendung des IBM MQ-Transports für SOAP implementieren
Bereiten Sie ein Bereitstellungsverzeichnis und eine Axis2-Konfigurationsdatei für den Client vor. Stellen Sie die Client-Proxys und Clientklassen bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS
Ein Web-Service, gebunden an die W3C-Kandidatenempfehlung für SOAP over JMS, muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients mit einem Web-Service dar, der über das Protokoll W3C SOAP over JMS in WebSphere Application Server bereitgestellt wird. Ändern Sie die URL im für IBM MQ Transport for SOAP entwickelten Axis2-Client, dass die W3C-Kandidatenempfehlung für SOAP over JMS verwendet wird.

Web-Service-Client für .NET Framework 1 und 2 zur Verwendung von IBM MQ Transport for SOAP implementieren
Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie den Client-Proxy und die Clientklasse bereit. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client in Axis2 zur Verwendung des IBM MQ-Transports für SOAP implementieren
Bereiten Sie ein Bereitstellungsverzeichnis und eine Axis2-Konfigurationsdatei für den Client vor. Stellen Sie die Client-Proxys und Clientklassen bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Vorbereitende Schritte

Tipp: Implementieren Sie den Service in HTTP. Entwickeln und testen Sie den Client für HTTP und ändern Sie anschließend die URL, damit auf den Service verwiesen wird, der IBM MQ Transport for SOAP verwendet.

Die Task zeigt, wie ein nicht verwalteter Axis2-Client in Java Standard Edition implementiert wird. Sie können einen Axis2-Client in einen Web-Container implementieren. In „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln" auf Seite 1401 haben Sie einen Client in einem Web-Container entwickelt und ihn in WebSphere Application Server Community Edition implementiert. Bei der Serverkonfiguration haben Sie die Axis2-Facette aktiviert und sie in die Konfiguration des Web-Containers einbezogen. Informationen zum Konfigurieren von Web-Containern auf anderen Anwendungsservern finden Sie in der Axis2 -Dokumentation https://ws.apache.org/axis2/1_4_1/installationguide.html#servlet_containeroder in der Dokumentation, die mit dem Web-Server bereitgestellt wird.

Anmerkung: Axis2 verwendet den Begriff 'Servlet-Container'. Ein Servlet-Container entspricht einem Web-Container.

Informationen zu diesem Vorgang

Sie implementieren einen Axis2-Client zur Verwendung des IBM MQ-Transports für SOAP genauso wie einen Axis2-Client zur Verwendung von HTTP. Zusätzliche Schritte sind erforderlich, um den IBM MQ-JAR-Dateien einen Klassenpfad bereitzustellen und die Axis2-Konfigurationsdatei zu ändern. Die Axis2-Konfigurationsdatei erfordert einen zusätzlichen Eintrag für JMS. Dieser Eintrag verweist auf die JAR-Datei des IBM MQ-Transports für SOAP, die den JMS `transportSender` implementiert.

Axis2 stellt ein Script (`axis2.bat` bzw. `axis2.sh`) zur Verfügung, um die Clientbereitstellung zu vereinfachen; siehe Beispiele in Abbildung 199 auf Seite 1431 und Abbildung 200 auf Seite 1432.

Anmerkung:

1. `axis2.bat` enthält einen Programmfehler, der korrigiert werden muss. Die Zeichenfolge `-Djava.ext.dirs="%AXIS2_HOME%\lib\"` muss in `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"` geändert werden.
2. In `axis2.bat` und `axis2.sh` wird `-Djava.ext.dirs` zur schnellen Referenz auf alle Axis2-JAR-Dateien verwendet, statt sie alle einzeln dem Klassenpfad hinzuzufügen. Leider ist diese Lösung fehlerhaft und funktioniert nur bei einigen JREs. Mit den JREs von IBM funktioniert sie nicht.

   Mit dem JVM-Parameter `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"` werden die Axis-JAR-Dateien der JVM zur Verfügung gestellt. Die JVM versucht, Instanzen von einigen der Axis-JAR-Dateien

zu erstellen, was einen Fehler verursacht. Die genauen Einzelheiten hängen dabei von der JVM ab. Normalerweise könnte eine der folgenden Zeilen im Stack-Trace enthalten sein:

```
org.apache.axiom.om.util.UUIDGenerator.getInitialUUID(UUIDGenerator.java:76)
```

oder `org.apache.axis2.deployment.DeploymentException: java.security.NoSuchAlgorithmException: MD5 MessageDigest not available`

Um einen nicht verwalteten Axis2-Client korrekt auszuführen, müssen Sie die Axis2-JAR-Dateien dem Klassenpfad hinzufügen. Der Klassenpfad steht nur der Clientanwendung und nicht der JVM zur Verfügung.

Die Prozedur beschreibt die allgemeinen Schritte, um einen nicht verwalteten Axis2-Client ohne das axis2-Script auszuführen. Die Beispiele in Abbildung 197 auf Seite 1431 und Abbildung 198 auf Seite 1431 sind Scripts für Windows und Linux.

Vorgehensweise

1. Laden Sie Axis2 1.4.1 von https://ws.apache.org/axis2/download/1_4_1/download.cgi herunter und entpacken Sie in einem Ordner, `Axis2-1.4.1`.
2. Aktualisieren Sie `axis2.xml` in `Axis2-1.4.1\conf`.
   a) Aktualisieren Sie `axis2.xml` in `Axis2-1.4.1\conf`. Fügen Sie IBM MQ Transport for SOAP als einen `transportSender` hinzu:

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender"/>
   ```

   b) Falls erforderlich, ändern Sie den Standardwert 10 der Größe des Verbindungspools.

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender">
   <parameter name="ResourcePoolCapacity">20</parameter>
   </transportSender>
   ```

   `ResourcePoolCapacity` definiert, wie viele Serviceendpunkteinträge zwischengespeichert werden. Der Wert muss mindestens 1 sein. Wenn die Anzahl der Serviceendpunkteinträge die Cachegröße überschreitet, werden Einträge gelöscht, um Platz für neue Einträge zu schaffen. Die Größe eines Endpunkteintrags variiert. Legen Sie eine Zahl fest, die groß genug ist, um eine Überlastung des Zwischenspeichers zu vermeiden.

   Informationen hierzu finden Sie unter „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln" auf Seite 1401 in Schritt 3.
3. Erstellen Sie das Verzeichnis *Implementierungsverzeichnis*. Kopieren Sie die Ordnerstruktur mit dem Client und den Client-Proxys in dieses Verzeichnis. *deployDir* entspricht dem Ordner *project*`\bin` in einem Eclipse Java -Projekt.
4. Öffnen Sie unter Windows ein Befehlsfenster oder im *Implementierungsverzeichnis* eine Befehlsshell mit X Window System auf UNIX and Linux-Systemen.
5. Aktualisieren Sie den Klassenpfad, um die Axis2-JAR-Dateien `com.ibm.mqjms.jar` und `com.ibm.mq.axis2.jar` des aktuellen Verzeichnisses einzuschließen.

   `com.ibm.mqjms.jar` verweist auf alle anderen erforderlichen IBM MQ-JAR-Dateien.
6. Starten Sie das Clientprogramm mit dem **Java**-Befehl.

Beispiele

Vier Beispiele zum Ausführen eines Axis2-Clients finden Sie in Abbildung 198 auf Seite 1431 bis Abbildung 200 auf Seite 1432. Abbildung 196 auf Seite 1431 zeigt die Ausgabe der Ausführung des in Abbildung 179 auf Seite 1406 aufgeführten asynchronen Clients.

```
cd C:\IBM\ID\Workspaces\Axis2docs\StockQuoteAxis2PojoClient\bin>
runpojo soap/client/SQA2AsyncClient

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS: Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
Press any key to continue . . .
```

Abbildung 196. Ausgabe der Ausführung von 'SQA2AsyncClient'

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

setlocal EnableDelayedExpansion
set CLASSPATH=
set AXIS2_CLASS_PATH=
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

"%JAVA_HOME%\bin\java" -Daxis2.repo="%AXIS2_HOME%\repository"
-Daxis2.xml="%AXIS2_HOME%\conf\axis2.xml" -cp
".;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar;%AXIS2_CLASS_PATH%" %1

pause
```

Abbildung 197. runpojo.bat: Windows mit Verwendung eines Klassenpfads

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm
# update classpath
AXIS2_CLASSPATH=""
for f in "$AXIS2_HOME"/lib/*.jar
do
  AXIS2_CLASSPATH="$AXIS2_CLASSPATH":$f
done
AXIS2_CLASSPATH="$AXIS2_HOME":"$JAVA_HOME/lib/tools.jar":"$AXIS2_CLASSPATH":"$CLASSPATH"
java -cp /home/alex/dev/sandbox/Soap/axis2:/opt/mqm/java/lib/com.ibm.mqjms.jar:
/opt/mqm/java/lib/com.ibm.mq.axis2.jar:$AXIS2_CLASSPATH
-Daxis2.xml=/home/alex/dev/sandbox/axis2-1.4.1/conf/axis2.xml %1
```

Abbildung 198. runpojo.sh: Linux mit Verwendung eines Klassenpfads.

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
pause
```

Abbildung 199. runaxis2.bat: Windows mit Verwendung von 'axis2.bat'

Hinweis

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
```

Abbildung 200. runaxis2.sh: Linux mit Verwendung von 'axis2.sh'

Hinweis

Zugehörige Tasks

Web-Service-Client für Axis 1.4 zur Verwendung von IBM MQ Transport for SOAP implementieren
Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie die Client-Proxys und die Clientklasse bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS
Ein Web-Service, gebunden an die W3C-Kandidatenempfehlung für SOAP over JMS, muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients mit einem Web-Service dar, der über das Protokoll W3C SOAP over JMS in WebSphere Application Server bereitgestellt wird. Ändern Sie die URL im für IBM MQ Transport for SOAP entwickelten Axis2-Client, dass die W3C-Kandidatenempfehlung für SOAP over JMS verwendet wird.

Web-Service-Client für .NET Framework 1 und 2 zur Verwendung von IBM MQ Transport for SOAP implementieren
Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie den Client-Proxy und die Clientklasse bereit. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS

Ein Web-Service, gebunden an die W3C-Kandidatenempfehlung für SOAP over JMS, muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients mit einem Web-Service dar, der über das Protokoll W3C SOAP over JMS in WebSphere Application Server bereitgestellt wird. Ändern Sie die URL im für IBM MQ Transport for SOAP entwickelten Axis2-Client, dass die W3C-Kandidatenempfehlung für SOAP over JMS verwendet wird.

Vorbereitende Schritte

Führen Sie zunächst die Task „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln” auf Seite 1401 durch, um **`SimpleJavaListener`** mit einem Axis2-Client und dem Protokoll des IBM MQ-Transports für SOAP aufzurufen.

Zudem müssen Sie zuvor den Web-Service erstellt sowie IBM MQ und WebSphere Application Server konfiguriert haben:

Weitere Informationen finden Sie im Abschnitt IBM MQ und WebSphere Application Server gemeinsam verwenden.

Außerdem müssen Sie die Task ausgeführt haben, die im Abschnitt „JAX-WS-EJB-Web-Service für W3C SOAP over JMS entwickeln” auf Seite 1387 beschrieben ist.

In der Task wird der Client in Eclipse Galileo ausgeführt. Sie können den Client auch aus der Befehlszeile ausführen, indem Sie die mit Axis2 mitgelieferte Datei `Axis2.bat` ändern.

Informationen zu diesem Vorgang

Die einzige Änderung, die Sie am vorhandenen statischen Axis2-Client `StockQuoteAxis` vornehmen müssen, um den von WebSphere Application Server gehosteten Service 'StockQuoteAxis' aufzurufen,

besteht darin, die an den Client übermittelte URL zu ändern. Da die WSDL nicht geändert wurde, können Sie dieselben Proxy-Klassen im Paket `soap.server` verwenden.

Die an den Client zu übermittelnde URL kann auf zwei Arten definiert werden. Sie können dieselbe URL wie in der generierten Datei `StockQuoteAxis.wsdl` verwenden. Sie müssen die Parameter `jndiInitialContextFactory` und `jndiURL` hinzufügen, um auf das WebSphere Application Server JNDI-Verzeichnis zuzugreifen. Alternativ können Sie die URL ändern und dem Client direkten Zugriff auf die Warteschlangen `REQUESTAXIS` und `REPLYAXIS` in `QM1` zu gewähren, ohne eine JNDI-Suche zu verwenden.

Die Verbindungsparameter, die Sie in der an den Axis2-Client übermittelten URL definieren, werden zur Verbindungsherstellung zum Warteschlangenmanager und zu den Warteschlangen von IBM MQ verwendet, die zum Senden und Empfangen von SOAP-Nachrichten erforderlich sind. Die an den Axis2-Client übermittelten Verbindungsparameter werden nicht zwingend vom Service verwendet. Mithilfe der verteilten Steuerung von Warteschlangen von IBM MQ können Sie den Client und den Service von der Verwendung desselben Warteschlangenmanagers oder desselben Name-Servers entkoppeln.

Vorgehensweise

1. Speichern Sie die URL der generierten Datei `StockQuoteAxis.wsdl` und schließen Sie Rational Application Developer, um Speicherplatz zu sparen.

   Wenn Sie die Serverkonfiguration nicht geändert haben, wird der Anwendungsserver durch das Schließen von Rational Application Developer gestoppt. Starten Sie den Server in diesem Fall mit folgendem Befehl:

   ```
   startserver server1
   ```

2. Öffnen Sie Eclipse Galileo im Arbeitsbereich mit dem Axis2-Clientprojekt.
3. Öffnen Sie `SQA2StaticClient.java`.

   Informationen hierzu finden Sie unter SQA2StaticClient.java.
4. Rufen Sie den Service mit der `queue`-Variante des URI auf.
   a) Ändern Sie die URL.
   Der neue URI lautet:

   ```
   jms:queue:REQUESTAXIS
             ?replyToName=REPLYAXIS
             &connectionFactory=connectQueueManager(QM1)Bind(Server)
             &targetService=StockQuoteAxis;
   ```

   Vergleichen Sie diese URL mit der URL von `StockQuoteAxis.wsdl`:

   ```
   jms:jndi:requestaxis
            ?jndiConnectionFactoryName=qm1
            &targetService=StockQuoteAxis
   ```

   Abbildung 201. URL von `StockQuoteAxis.wsdl`

   - `REQUESTAXIS` ist nun großgeschrieben, da es sich um einen Warteschlangennamen und nicht um einen JNDI-Namen handelt.
   - Die Verbindung zu `QM1` ist einfach.
   - Der URI enthält nicht den Namen der Zielwarteschlange für Antworten. Der Client muss die Warteschlange definieren, in der er Antworten erwartet.

   b) Führen Sie `SQA2StaticClient.java` mit demselben Wert für **Ausführen als ...** aus. Konfiguration, wie Sie in der Task „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln” auf Seite 1401ausgeführt haben.
5. Rufen Sie den Service mit der `jndi`-Variante des URI auf. Verwenden Sie WebSphere Application Server als Naming Server.

a) Verwenden Sie die URL von `StockQuoteAxis.wsdl`, Abbildung 201 auf Seite 1433, und geben Sie die fehlenden Parameter an, um den Naming Service in WebSphere Application Server zu verwenden.

Die fehlenden Parameter und Werte, die Sie angeben müssen, lauten wie folgt:

Tabelle 192. Zusätzliche JNDI-Parameter

Parameter	In diesem Beispiel verwendeter Wert	Beschreibung
`&jndiURL`	`iiop://localhost:2810` oder `corbaname:iiop:localhost:2810`	URI des Naming Provider. WebSphere Application Server verwendet den Standardwert 2809. Dieser ist auch bekannt als Portnummer des RMI-Konnektors und als Bootstrap-Port. Der Wert wird in der Datei `SystemOut.log` aufgeführt. `00000000 NameServerImp A NMSV0018I: Name server available on bootstrap port 2810`
`&jndiInitialContextFactory`	`com.ibm.websphere.naming.WsnInitialContextFactory`	Der Name der von WebSphere Application Server verwendeten Ausgangskontextfactory.
`&replyToName`	`replyaxis`	JNDI-Name der Warteschlange REPLYAXIS.

```
jms:jndi:requestaxis?
         &jndiURL=iiop//localhost:2810
         &jndiConnectionFactoryName=qm1
         &jndiInitialContextFactory=com.ibm.websphere.naming.WsnInitialContextFactory
         &targetService=StockQuoteAxis
         &replyToName=replyaxis;
```

b) Fügen Sie die für die JNDI-Suche erforderlichen JAR-Dateien hinzu.

In dieser Konfiguration wurden dem Erstellungspfad die folgenden JAR-Dateien hinzugefügt, um die Task mit der `jndi`-Variante der JMS-URL auszuführen:

- `com.ibm.jaxws.thinclient_7.0.0.jar` aus *`Rational installation directory`*`\SDP\runtimes\base_v7\runtimes`.
- `com.ibm.ws.runtime.jar` in *`Rational installation directory`*`\SDP\runtimes\base_v7\plugins`

Für einen anderen JNDI-Provider sind andere JAR-Dateien erforderlich.

Die anderen JAR-Dateien im Erstellungspfad sind:

i) Alle JAR-Dateien in *`WebSphere MQ Installation directory`*`\java\lib`.

ii) Alle JAR-Dateien in *`Axis2-1.5.1`*`\lib`.

iii) Java 6.0 JRE.

c) Führen Sie `SQA2StaticClient.java` mit demselben Wert für **Ausführen als ...** aus. Konfiguration, wie Sie in der Task „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln” auf Seite 1401ausgeführt haben.

Ergebnisse

In beiden Fällen wird die Antwort des Service in der Clientkonsolenansicht angezeigt.

Zugehörige Tasks

Web-Service-Client für Axis 1.4 zur Verwendung von IBM MQ Transport for SOAP implementieren
Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie die Client-Proxys und die Clientklasse bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client in Axis2 zur Verwendung des IBM MQ-Transports für SOAP implementieren
Bereiten Sie ein Bereitstellungsverzeichnis und eine Axis2-Konfigurationsdatei für den Client vor. Stellen Sie die Client-Proxys und Clientklassen bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client für .NET Framework 1 und 2 zur Verwendung von IBM MQ Transport for SOAP implementieren
Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie den Client-Proxy und die Clientklasse bereit. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client für .NET Framework 1 und 2 zur Verwendung von IBM MQ Transport for SOAP implementieren

Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie den Client-Proxy und die Clientklasse bereit. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Vorbereitende Schritte

Tipp: Entwickeln und testen Sie den Service und den Client mit Visual Studio. Passen Sie den Client dann für IBM MQ Transport for SOAP an.

1. Wenn Sie einen Service mit .NET Framework 1 oder 2 implementieren, erstellen Sie den Service als Bibliothek (`.dll`). Implementierung mit IBM MQ Transport for SOAP.
2. Fügen Sie dem Client den Aufruf Register.Extension() hinzu.
3. Fügen Sie eine Referenz zu amqsoap.dllhinzu, die sich in *MQ_Install*\binbefindet.
4. Ändern Sie die statische Eigenschaft URL im Konstruktor der Client-Proxy-Klasse in den URI `jms:/` für IBM MQ Transport for SOAP.

Informationen zu diesem Vorgang

Für die Implementierung eines Web-Service-Clients für .NET Framework 1 oder 2 zur Verwendung von IBM MQ Transport for SOAP ist ein zusätzlicher Implementierungsschritt erforderlich. Sie müssen die Bibliothek `amqsoap.dll` bei .NET Framework registrieren. `amqsoap.dll` wird zwar bei der Installation des IBM MQ-Transports für SOAP automatisch registriert, möglicherweise müssen Sie die Registrierung aber wiederholen.

Wenn Sie zur Erstellung von Client-Proxys den Befehl **amqwdeployWMQService** verwenden, können Sie den Client in den vom Befehl generierten Verzeichnissen implementieren.

Vorgehensweise

1. Erstellen Sie das Verzeichnis *deployDir* für die Clientbereitstellungsdateien.
2. Öffnen Sie im Verzeichnis *deployDir* ein Befehlsfenster.
3. Führen Sie **amqwsetcp** zur Festlegung von CLASSPATH aus, wenn der Service unter Axis 1.4 ausgeführt werden soll.
4. Führen Sie gegebenenfalls **amqwRegisterDotNet** aus, um die Bibliothek `amqsoap.dll` bei .NET Framework zu registrieren.

Beispiel

Das Beispiel zeigt die Konfiguration und die Ausgabe (Abbildung 204 auf Seite 1436) eines .NET Framework V2-Clients. Der Client (Abbildung 203 auf Seite 1436) ruft einen Web-Service auf, der die Eingabeparameter zurückmeldet. Die statische URL-Definition (Abbildung 202 auf Seite 1436) zeigt den Konstruktor für den Client-Proxy.

```
public Quote() {
    this.Url = "jms:/queue?destination=REQUESTDOTNET
               &connectionFactory=(connectQueueManager(QM1)binding(server))
               &initialContextFactory=com.ibm.mq.jms.Nojndi
               &targetService=Quote.asmx
               &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE";
}
```

Abbildung 202. Statischer Konstruktor für Client-Proxy

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
            } catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
            }
        }
    }
}
```

Abbildung 203. Clientprogramm

```
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>dir /s /b
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram\QuoteClientProgram.exe
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>quoteclientprogram
Response is: IBM
```

Abbildung 204. Konfiguration und Ausgabe

Nächste Schritte

1. Wenn Sie den Client als IBM MQ MQI client implementieren, müssen Sie den Client- und Serververbindungskanal konfigurieren.
2. Wenn Sie den Client auf einem anderen Warteschlangenmanager als dem des Service implementieren, müssen Sie für den Client die Zielwarteschlange bereitstellen. Konfigurieren Sie die Zielwarteschlange auf dem Warteschlangenmanager des Service als eine Clusterwarteschlange bzw. auf dem Warteschlangenmanager des Clients als eine ferne Warteschlange.

Zugehörige Tasks

Web-Service-Client für Axis 1.4 zur Verwendung von IBM MQ Transport for SOAP implementieren
Bereiten Sie für den Client ein Bereitstellungsverzeichnis und einen Bereitstellungsdeskriptor vor. Stellen Sie die Client-Proxys und die Clientklasse bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client in Axis2 zur Verwendung des IBM MQ-Transports für SOAP implementieren

Bereiten Sie ein Bereitstellungsverzeichnis und eine Axis2-Konfigurationsdatei für den Client vor. Stellen Sie die Client-Proxys und Clientklassen bereit und richten Sie den CLASSPATH ein. Konfigurieren Sie die IBM MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS
Ein Web-Service, gebunden an die W3C-Kandidatenempfehlung für SOAP over JMS, muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients mit einem Web-Service dar, der über das Protokoll W3C SOAP over JMS in WebSphere Application Server bereitgestellt wird. Ändern Sie die URL im für IBM MQ Transport for SOAP entwickelten Axis2-Client, dass die W3C-Kandidatenempfehlung für SOAP over JMS verwendet wird.

Verbindung eines Axis2-Clients mit einem JAX-WS-Service unter Verwendung von W3C SOAP over JMS und WebSphere Application Server

Wenn Sie diese Task abgeschlossen haben, haben Sie einen JAX-WS-Web-Service, der in WebSphere Application Server ausgeführt wird, aus einem Axis2-Client aufgerufen. Der Axis2-Client und WebSphere Application Server verwenden die W3C-Kandidatenempfehlung für das SOAP over JMS-Protokoll, das unter IBM MQ ausgeführt wird. Erstellen Sie den Web-Service-Client bzw. den Web-Service mit Eclipse Galileo bzw. Rational Application Developer.

Vorbereitende Schritte

Für diese Task sind Version 7 der Rational-Softwareentwicklungsumgebung und WebSphere Application Server erforderlich. Die Task wurde mit Rational Application Developer erstellt, das im Paket von Rational Software Architect for WebSphere Software 7.5.5.1und WebSphere Application Server 7.0 Test Environment 7.0.0.9 Update 1 enthalten ist. Sie benötigen außerdem IBM MQ 7.0.1.3oder höher.

Die Task baut auf den beiden Tasks Liberty-Profil und „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln" auf Seite 1401 auf. Für die Ausführung dieser Tasks hat Ihre Entwicklungsumgebung bereits Eclipse Galileo, das Liberty-Profil, das Eclipse-Plug-in für Liberty sowie Axis2 1.4.1 installiert.

Einige Schritte sind komplex. Bei den Schritten wird vorausgesetzt, dass Sie mit der Entwicklung von Web-Service-Anwendungen für WebSphere Application Server mit Rational Application Developer vertraut sind. Die Task belegt sehr viel Prozessor- und Speicherplatz. Die Task wurde auf einer virtuellen Maschine mit VMWare Windows 7 SP1 und 1,8 GB Speicher ausgeführt.

Installieren Sie die gesamte Software, bevor Sie mit der Task beginnen. Je nach Bandbreite benötigt die Software ungefähr einen Tag für den Download und einen Tag für die Installation. Der Zeitaufwand für die Task beträgt mindestens einen halben Tag.

Informationen zu diesem Vorgang

Das Szenario für diese Task lautet wie folgt: Mithilfe des Open-Source-Tools Eclipse Galileo haben Sie einen Web-Service für Aktiennotierungen namens `StockQuoteAxis` entwickelt. `StockQuoteAxis` wird mit SOAP over HTTP implementiert, das auf einem Open-Source-Server, Liberty-Profil, ausgeführt wird.

Sie möchten die Web-Services binden, die Sie für einen standardisierten Messaging-Transport wie z. B. SOAP over JMS oder für Web-Services Reliable Messaging sowie für SOAP over HTTP. Sowohl der Client als auch der Service sollen standardisierte Schnittstellen verwenden. Obwohl Ihre zukünftigen Projektentwicklungsteams eine Lösung implementiert haben, die IBM MQ-Transport für SOAP verwendet, haben Sie die Produktion noch nicht realisiert.

Mit dem Axis2-Client wurde das Problem gelöst, dass der SOAP-Client für den IBM MQ-Transport für SOAP eine Änderung vom HTTP-Client verlangte. Das Problem, dass der über den IBM MQ-Transport für SOAP verbundene Service von einem besonderen Listener gehostet wird, der von IBM MQ: `SimpleJavaListener` bereitgestellt wird, bleibt bestehen.

Mit dem W3C SOAP over JMS-Standard im Kandidatenempfehlungsstatus unterstützen manche Hersteller W3C SOAP over JMS. Die Unterstützung ermöglicht Ihnen, einen Web-Service für einen Anwendungs-

server zu implementieren und über eine Vielzahl von Verbindungsprotokollen eine Verbindung zum selben Service herzustellen. Durch die Unterstützung von WebSphere Application Server v7 wurde das Problem, den Web-Service separat hosten zu müssen, um einen nachrichtenbasierten SOAP-Transport zu verwenden. Wenn Sie eine standardisierte Nachrichtentransportschnittstelle JMS verwenden, können Sie Lösungen unter Verwendung von Tools unterschiedlicher Anbieter entwickeln. Sie hoffen, dass die Web-Service-Tools in Eclipse in Zukunft SOAP over JMS-Bindungen einschließen.

Die meisten der Schritte werden mithilfe von Eclipse oder mit den Verwaltungstools der WebSphere-Produkte ausgeführt. Beschrieben werden die Schritte für eine Windows-Umgebung. Mit leichten Änderungen an manchen Befehlen können Sie die Schritte auf anderen Plattformen ausführen.

Die vorläufigen Schritte zum Erstellen des HTTP-Web-Service und zu dessen Verbindungsherstellung mithilfe von Axis2 sind aufgeführt. Mit dem Client und der WSDL aus diesen Schritten erstellen Sie die Lösung.

Vorgehensweise

1. Stellen Sie mithilfe eines Axis2-Clients und mit IBM MQ-Transport for SOAP eine Verbindung zum Web-Service 'StockQuoteAxis' her.
   a) Verwenden Sie das Liberty-Profil.
   b) „JAX-WS-Client für WebSphere Transport for SOAP mithilfe von Eclipse entwickeln” auf Seite 1401
   c) „Web-Service-Client in Axis2 zur Verwendung des IBM MQ-Transports für SOAP implementieren” auf Seite 1429
2. Stellen Sie mithilfe eines Axis2-Clients und der W3C-Kandidatenempfehlung für SOAP over JMS eine Verbindung zum Web -Service 'StockQuoteAxis' her.
   a) Konfigurieren Sie IBM MQ-Ressourcen.
   b) Konfigurieren Sie WebSphere Application Server-Ressourcen.
   c) „JAX-WS-EJB-Web-Service für W3C SOAP over JMS entwickeln” auf Seite 1387
   d) „Implementierung in einen Axis2-Client mit W3C SOAP over JMS” auf Seite 1432

Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder andere Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb von Fremdprodukten, Fremdprogrammen und Fremdservices liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieser Dokumentation ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Europe
IBM Europe, Middle East and Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Défense
U.S.A.

Bei Lizenzanforderungen zu Double-Byte-Information (DBCS) wenden Sie sich bitte an die IBM Abteilung für geistiges Eigentum in Ihrem Land oder senden Sie Anfragen schriftlich an folgende Adresse:

Lizenzierung von geistigem Eigentum

IBM Japan, Ltd.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die Angaben in dieser Veröffentlichung werden in regelmäßigen Zeitabständen aktualisiert. Die Änderungen werden in Überarbeitungen oder in Technical News Letters (TNLs) bekanntgegeben. IBM kann jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

© Copyright IBM Corp. 2007, 2023

IBM Europe, Middle East and Africa
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in diesen Informationen beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Die in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Um diese so realistisch wie möglich zu gestalten, enthalten sie auch Namen von Personen, Firmen, Marken und Produkten. Sämtliche dieser Namen sind fiktiv. Ähnlichkeiten mit Namen und Adressen tatsächlicher Unternehmen oder Personen sind zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musterprogramme, die in Quellensprache geschrieben sind. Sie dürfen diese Musterprogramme kostenlos (d. h. ohne Zahlung an IBM) kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten.

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farbabbildungen.

Informationen zu Programmierschnittstellen

Die bereitgestellten Informationen zur Programmierschnittstelle sollen Sie bei der Erstellung von Anwendungssoftware für dieses Programm unterstützen.

Dieses Handbuch enthält Informationen über vorgesehene Programmierschnittstellen, die es dem Kunden ermöglichen, Programme zu schreiben, um die Services von WebSphere MQzu erhalten.

Diese Informationen können jedoch auch Angaben über Diagnose, Bearbeitung und Optimierung enthalten. Die Informationen zu Diagnose, Bearbeitung und Optimierung sollten Ihnen bei der Fehlerbehebung für die Anwendungssoftware helfen.

Wichtig: Verwenden Sie diese Diagnose-, Änderungs-und Optimierungsinformationen nicht als Programmierschnittstelle, da sie Änderungen unterliegen.

Marken

IBM, das IBM Logo, ibm.com, sind Marken der IBM Corporation in den USA und/oder anderen Ländern. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein.

Microsoft und Windows sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Dieses Produkt enthält Software, die von Eclipse Project (http://www.eclipse.org/) entwickelt wurde.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Teilenummer:

(1P) P/N: